Eduardo Sany Laber
Claudson Bornstein
Loana Tito Nogueira
Luerbio Faria (Eds.)

LNCS 4957

LATIN 2008: Theoretical Informatics

8th Latin American Symposium
Búzios, Brazil, April 2008
Proceedings

Springer

Eduardo Sany Laber   Claudson Bornstein
Loana Tito Nogueira   Luerbio Faria (Eds.)

LATIN 2008: Theoretical Informatics

8th Latin American Symposium
Búzios, Brazil, April 7-11, 2008
Proceedings

Springer

Volume Editors

Eduardo Sany Laber
PUC-RIO
Departamento de Informática
RDC ANDAR 5, Rua Marques de São Vicente, Gávea, Brazil
E-mail: laber@inf.puc-rio.br

Claudson Bornstein
Universidade Federal do Rio de Janeiro
Departamento de Ciência da Computação
21941-590, Rio de Janeiro, RJ, Brazil
E-mail: claudson@ufrj.br

Loana Tito Nogueira
Universidade Federal Fluminense
Instituto de Computação
Rua Passo da Patria 156, 24210-240, São Domingos, Niterói, RJ, Brazil
E-mail: loana@ic.uff.br

Luerbio Faria
Universidade Estadual do Rio de Janeiro
Rua Dr. Francisco Portela, 24435-005, São Gonçalo, RJ, Brazil
E-mail: luerbio@cos.ufrj.br

Library of Congress Control Number: 2008922953

CR Subject Classification (1998): F.2, F.1, E.1, E.3, G.2, G.1, I.3.5, F.3, F.4

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN	0302-9743
ISBN-10	3-540-78772-0 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-78772-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper   SPIN: 12244744   06/3180   5 4 3 2 1 0

Preface

The Latin American Theoretical INformatics Symposium (LATIN) is becoming a traditional and high-quality conference on the Theory of Computing. Previous conferences have been organized twice in Brazil: São Paulo (1992) and Campinas (1998); twice in Chile: Valparaíso (1995) and Valdivia (2006); once in Uruguay: Punta del Este (2000); once in Mexico: Cancun (2002); and once in Argentina: Buenos Aires (2004).

This volume contains the proceedings of the 8th Latin American Theoretical INformatics Symposium (LATIN 2008), which was held in Búzios, Rio de Janeiro, Brazil, April 7–11, 2008.

A total of 242 papers were reviewed by the program committee. Among them, 66 were selected for presentation at the conference. The selection was based on the papers' originality, quality and relevance to Theoretical Computer Science.

This volume also contains the extended abstract associated with the invited talk of Wojciech Szpankowski. We also had 4 invited talks by Cláudio Leonardo Lucchesi, Eva Tardos, Moni Naor and Robert Tarjan.

We would like to thank all members of the PC for their thorough work, which resulted in a good selection of papers; the members of the Steering Committee for their insightful advice and for sharing their experience with us; and the members of our community who served as referees.

In addition, we thank all our sponsors, Microsoft, UOL, IFIP, HP, Yahoo!, FAPERJ, CNPq and CAPES, and Springer for their continuous support.

April 2008

Eduardo Laber
Claudson Bornstein
Loana Nogueira
Luerbio Faria

Organization

Program Committee

Michael Bender	Stony Brook U., USA
Leo Bertossi	Carleton U., Canada
Claudson Bornstein (vice-chair)	U. F. Rio de Janeiro, Brazil
Ferdinando Cicalese	U. of Salermo, Italy
José Correa	U. Adolfo Ibáñez, Chile
Cristina G. Fernandes	U. de São Paulo, Brazil
David Fernández-Baca	Iowa State U., USA
Fedor Fomin	U. Bergen, Norway
Joachim von zur Gathen	U. of Bonn, Germany
Andrew Goldberg	Microsoft Research Silicon Valley, USA
Venkatesan Guruswami	U. of Washington, USA
Alejandro Hevia	U. Chile, Chile
John Iacono	Polytechnic U., USA
Eduardo Laber (chair)	PUC-Rio, Brazil
Alejandro López-Ortiz	U. of Waterloo, Canada
Arnaldo Mandel	U. de São Paulo, Brazil
Guilhermo Matera	U. Nacional de General Sarmiento, Argentina
Flávio Miyazawa	Unicamp, Brazil
Mike Molloy	U. of Toronto, Canada
Ojas Parekh	Emory U., USA
Boaz Patt-Shamir	Tel Aviv U., Israel
Artur Pessoa	U. F. Fluminense, Brazil
Jean-Éric Pin	U. of Paris 7, France
Satish Rao	U. of Berkeley, USA
R. Ravi	Carnegie Mellon U., USA
Andrea Richa	Arizona State U., USA
Miklós Ruszinkó	Computer and Automation Res. Inst., Hungary
Gelasio Salazar	U. Autónoma de San Luis Potosi, Mexico
Jayme L. Szwarcfiter	U. F. Rio de Janeiro, Brazil
Tamir Tassa	The Open U., Israel
Jorge Urrutia	U. Nacional Autónoma de Mexico, Mexico
Ugo Vaccaro	U. of Salerno, Italy
Vijay Vazirani	Georgia Tech, USA
Alfredo Viola	U. de la República, Uruguay
Renato Werneck	Microsoft Research Silicon Valley, USA
Frances Yao	City U. of Hong Kong, Hong Kong

Plenary Speakers

Cláudio Lucchesi, Unicamp, Brazil
Moni Naor, Weizmann Institute, Israel
Wojciech Szpankowski, Purdue University, USA
Éva Tardos, Cornell U., USA
Robert Tarjan, Princeton U., USA

Steering Committee

Martin Farach-Colton, Rutgers U., USA
Marcos Kiwi, U. Chile, Chile
Yoshiharu Kohayakawa, U. de São Paulo, Brazil
Daniel Panario, Carleton U., Canada
Sergio Rajsbaum, U. Nacional Autónoma de México, Mexico
Gadiel Seroussi, HP Labs, USA

Executive Committee

Conference Chair	Eduardo Laber (PUC-Rio, Brazil)
Conference Vice-chair	Claudson Bornstein (U. F. Rio de Janeiro, Brazil)
Co-organizer	Luerbio Faria (U. E. Rio de Janeiro, Brazil)
Co-organizer	Loana T. Nogueira (U. F. Fluminense, Brazil)

Sponsoring Institutions

Brazilian Research Agencies:

- CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico
- CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
- FAPERJ - Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro

Corporate Sponsors:

- UOL
- Microsoft
- Yahoo!
- HP

Computer Science Societies:

- IFIP - International Federation for Information Processing
- SBC - Sociedade Brasileira de Computação

Referees

Edward Haeusler
Hermann Haeusler
Iftach Haitner
M.T. Hajiaghayi
Angele M. Hamel
Russell Harmer
Nick Harvey
Meng He
Joos Heintz
Glenn Hickey
Petr Hlineny
Pieter Hofstra
Carlos Hoppen
Doug Howe
Nicole Immorlica
Riko Jacob
Svante Janson
Michael Kaltenbach
Marcin Kaminski
Haim Kaplan
Chinmay Karande
Jarkko Kari
Louis H. Kauffman
Ken-ichi Kawarayabashi
David Kempe
Walter Kern
Alex Kesselman
Zoltán Király
Ákos Kisvölcsey
Marcos Kiwi
Pang Ko
Yoshiharu Kohayakawa
Goran Konjevod
Guy Kortsarz
Evangelos Kranakis
Dieter Kratsch
Dalia Krieger
Michael Krivelevich
Alexander S. Kulikov
Gábor Kun
Alair Pereira do Lago
Zeph Landau
Michael Langberg
Gregoire Lecerf
Orlando Lee
Sangjin Lee
Erik Jan van Leeuwen
Christos Levcopoulos
Asaf Levin
Ming Li
Min Chih Lin
Zsuzsanna Liptak
Yi-Kay Liu
Errol Lloyd
Daniel Loebenberger
Martin Loebl
Sylvain Lombardy
Zvi Lotker
Q. Louveaux
Gabor Lugosi
Marco Macchetti
Frederic Maffray
Veli Makinen
Toufik Mansour
Roberto Mantaci
Gianluca De Marco
Javier Marenco
Mauricio Marín
Daniel Martin
Ryan Martin
Conrado Martinez
Fabio Martinez
Carlos A. Martinhon
Walter Mascarenhas
Martin Matamala
Nicole Megow
Marta Mejail
Manor Mendel
Criel Merino
Pascal Michel
Martin Milanic
Alexandre Miquel
David Mitchell
Thomas Moelhave
Cristopher Moore
Tal Moran
Eduardo Moreno
Rob Morris
Nicolas Stier Moses
Haiko Müller
Wolfgang Mulzer
Viswanath Nagarajan
Jorge Nakahara Jr.
Sergio Nesmachnow
C.F.X. de Mendonça Neto
Loana T. Nogueira
Michael Nüsken
Zeev Nutov
Nicolas Ollinger
Melih Onus
Alois Panholzer
Rina Panigrahy
Rohit Parikh
Andrzej Pelc
Todd Philips
Cynthia A. Phillips
Teo Chung Piaw
Christian Pich
Jose Coelho de Pina
Wojciech Plandowski
David Pritchard
Helmut Prodinger
Guido Proietti
Fabio Protti
Jens Putzka
Artem Pyatkin
Jose A.A. Qitzau
S.P. Radziszowski
Prasad Raghavendra
Anup Rao
Dieter Rautenbach
S.S. Ravi
Dror Rawitz
Igor Razgon
Daniel Reichman
Nora Reyes
Pedro J. de Rezende
Diego Rial
Rosiane de F. Rodrigues
Peter Rossmanith
Nicolau Saldanha
Alejandro Salinger
N. Deniz Sarier
Saket Saurabh

Carla Savage
Gabriel Scalosub
Nicolas Schabanel
Christian Scheideler
Claus Peter Schnorr
Michael Schnupp
Eric Schost
Michael Segal
Noa Segal-Agmon
Gil Segev
Luc Segoufin
Géraud Sénizergues
Gadiel Seroussi
Olivier Serre
Hovav Shacham
Alexander Sherstov
Yaoyun Shi
Maise D. Silva
Marcel Silva
Mohit Singh
Rene Sitters
Martin Skutella
Jose Soares
Yasmin Rios Solis
Motti Sorani
Cid Carvalho de Souza
Srinath Sridhar
R. Sritharan
Cliff Stein
Jorge Stolfi
Howard Straubing
K. Subramani
Zoya Svitkina
Kavitha Telikepalli
Eduardo Tengan
Guillaume Theyssier
Dimitrios Thilikos
Alex Thomo
Srikanta Tirthapura
Eric Tressler
Eduardo Uchoa
Ryuhei Uehara
Edgardo Ugalde
Jean-Marie Vanherpe
S. Venkatasubramanian
Adrian Vetta
Alfredo Viola
Ariel Waissbein
Yoshiko Wakabayashi
Lusheng Wang
Carola Wenk
Udi Wieder
Ryan Williams
David Wood
Eduardo Candido Xavier
Donglin Xia
Sergey Yekhanin
Xu Yichen
Uri Zwick
Grazyna Zwozniak

Table of Contents

Profile of Tries 1
[illegible]

Random 2-XORSAT at the Satisfiability Threshold 12
[illegible]

On Dissemination Thresholds in Regular and Irregular Graph Classes 24
[illegible]

How to Complete a Doubling Metric 36
[illegible]

Sorting and Selection with Random Costs 48
[illegible]

Guided Search and a Faster Deterministic Algorithm for 3-SAT 60
[illegible]

Comparing and Aggregating Partially Resolved Trees 72
[illegible]

Computing the Growth of the Number of Overlap-Free Words with Spectra of Matrices 84
[illegible]

On Stateless Multihead Automata: Hierarchies and the Emptiness Problem 94
[illegible]

Myhill-Nerode Theorem for Recognizable Tree Series Revisited 106
[illegible]

The View Selection Problem for Regular Path Queries 121
[illegible]

Optimal Higher Order Delaunay Triangulations of Polygons 133
[illegible]

Coloring Geometric Range Spaces 146
[illegible]
[illegible]

Local Algorithms for Dominating and Connected Dominating Sets of Unit Disk Graphs with Location Aware Nodes 158
[illegible]

Spanners of Complete k-Partite Geometric Graphs 170
[illegible]

Minimum Cost Homomorphisms to Reflexive Digraphs 182
[illegible]

On the Complexity of Reconstructing H-free Graphs from Their Star Systems 194
[illegible]

Optimization and Recognition for K_5-minor Free Graphs in Linear Time 206
[illegible]

Bandwidth of Bipartite Permutation Graphs in Polynomial Time 216
[illegible]

The Online Transportation Problem: On the Exponential Boost of One Extra Server 228
[illegible]

Average Rate Speed Scaling 240
[illegible]

Geometric Aspects of Online Packet Buffering: An Optimal Randomized Algorithm for Two Buffers 252
[illegible]

Maximizing the Minimum Load for Selfish Agents 264
[illegible]

Approximate Polynomial *gcd*: Small Degree and Small Height Perturbations 276
[illegible]

Pseudorandom Graphs from Elliptic Curves 284
[illegible]

Speeding-Up Lattice Reduction with Random Projections (Extended Abstract) 293
[illegible]

Sparse Approximate Solutions to Semidefinite Programs 306
[illegible]

On the Facets of Mixed Integer Programs with Two Integer Variables and Two Constraints 317
[illegible]

A Polyhedral Investigation of the LCS Problem and a Repetition-Free Variant 329
[illegible]
[illegible]

Competitive Cost Sharing with Economies of Scale 339
[illegible]

Emergency Connectivity in Ad-Hoc Networks with Selfish Nodes 350
[illegible]

Fully-Compressed Suffix Trees 362
[illegible]

Improved Dynamic Rank-Select Entropy-Bound Structures 374
[illegible]

An Improved Algorithm Finding Nearest Neighbor Using Kd-trees 387
[illegible]

List Update with Locality of Reference 399
[illegible]

Approximating Steiner Networks with Node Weights 411
[illegible]

Approximating Minimum-Power Degree and Connectivity Problems 423
[illegible]

Energy Efficient Monitoring in Sensor Networks 436
[illegible]
[illegible]

Approximation Algorithms for k-Hurdle Problems 449
[illegible] ffi [illegible]

Approximating Crossing Minimization in Radial Layouts 461
[illegible]

New Upper Bound on Vertex Folkman Numbers 473
[illegible]

Ptolemaic Graphs and Interval Graphs Are Leaf Powers 479
[illegible]

A Representation Theorem for Union-Difference Families and Application (Extended Abstract) 492
[illegible]

Algorithms to Locate Errors Using Covering Arrays 504
[illegible]

On Injective Colourings of Chordal Graphs 520
[illegible]

Spanning Trees with Many Leaves in Graphs without Diamonds and Blossoms 531
[illegible]

On 2-Subcolourings of Chordal Graphs 544
[illegible]

Collective Additive Tree Spanners of Homogeneously Orderable Graphs (Extended Abstract) 555
[illegible]

The Generalized Median Stable Matchings: Finding Them Is Not That Easy 568
[illegible]

Stateless Near Optimal Flow Control with Poly-logarithmic Convergence 580
[illegible]

The Least-Unpopularity-Factor and Least-Unpopularity-Margin Criteria for Matching Problems with One-Sided Preferences 593
[illegible]

Randomized Rendez-Vous with Limited Memory 605
[illegible]

Origami Embedding of Piecewise-Linear Two-Manifolds 617
[illegible]

Simplifying 3D Polygonal Chains Under the Discrete Fréchet Distance 630
[illegible]

Weighted Rectilinear Approximation of Points in the Plane 642
[illegible]

Paths with no Small Angles 654
[illegible]

Simpler Constant-Seed Condensers 664
[illegible]

Parallel Repetition of the Odd Cycle Game 676
[illegible]

I/O-Efficient Point Location in a Set of Rectangles 687
[illegible]

Finding Heavy Hitters over the Sliding Window of a Weighted Data Stream 699
[illegible]

Fixed-Parameter Algorithms for Cluster Vertex Deletion 711
[illegible]

Paths and Trails in Edge-Colored Graphs 723
[illegible]

Efficient Approximation Algorithms for Shortest Cycles in Undirected Graphs 736
[illegible]

Domination in Geometric Intersection Graphs 747
[illegible]

An Efficient Quantum Algorithm for the Hidden Subgroup Problem in Nil-2 Groups 759
[illegible]

Quantum Property Testing of Group Solvability 772
[illegible]

Solving NP-Complete Problems with Quantum Search 784
[illegible]

Author Index 793

Profile of Tries

G. Park[1], H.-K Hwang[2], P. Nicodème[3], and W. Szpankowski[4]

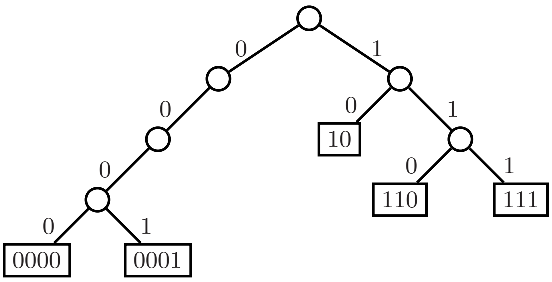
[1] Department of Computer Science, State University of New York at Geneseo, Geneseo, 14554, USA
park@geneseo.edu
[2] Institute of Statistical Science, Academia Sinica, 11529 Taipei, Taiwan
hkhwang@stat.sinica.edu.tw
[3] Laboratory LIX, École polytechnique, 91128 Palaiseau Cedex, France
nicodeme@lix.polytechnique.fr
[4] Department of Computer Sciences, Purdue University, 250 N. University Street, West Lafayette, Indiana, 47907-2066, USA
spa@cs.purdue.edu

Abstract. The *profile* of a trie, the most popular data structures on words, is a parameter that represents the number of nodes (either internal or external) with the same distance to the root. Several, if not all, trie parameters such as height, size, depth, shortest path, and fill-up level can be uniformly analyzed through the (external and internal) profiles. The analysis of profiles is surprisingly arduous but once it is carried out it reveals unusually intriguing and interesting behavior. We present a detailed study of the distribution of the profiles in a trie built over strings generated by a memoryless source (extension to Markov sources is possible). Our results are derived by methods of analytic algorithmics such as generating functions, Mellin transform, Poissonization and de-Poissonization, the saddle-point method, singularity analysis and uniform asymptotic analysis.

1 Introduction

Tries are prototype data structure useful for many indexing and retrieval purposes. Due to their simplicity and efficiency, tries found widespread use in diverse applications ranging from document taxonomy to IP addresses lookup and distributed hashing, from data compression to dynamic hashing, from partial-match queries to speech recognition, from leader election algorithms to distributed hashing tables (see [3,7,8,15]). In this paper, we are concerned with probabilistic properties of the profiles of tries, where the *profile* of a tree is the sequence of numbers each counting the number of nodes with the same distance to the root. We discover several new phenomena in the profiles of tries built over strings generated by a random memoryless source, and develop asymptotic tools to describe them.

Tries are natural choice of data structures when the input records involve a notion of alphabets or digits. They are often used to store such data so that future retrieval can be made efficient. Given a sequence of n words over the alphabet $\{a_1, \ldots, a_m\}$, $m \geq 2$, we can construct a trie as follows. If $n = 0$, then the trie is empty. If $n = 1$, then a single (external) node holding the word is allocated. If $n \geq 1$, then the trie consists of a root (internal) node directing words to the m subtrees according to the first symbol of each word, and words directed to the same subtree are themselves tries (see [7,8,15] for more details). For simplicity, we deal only with binary tries in this paper. In Figure 1 we plot a binary trie of 5 strings.



$B_{n,0} = 0, I_{n,0} = 1$

$B_{n,1} = 0, I_{n,1} = 2$

$B_{n,2} = 1, I_{n,2} = 2$

$B_{n,3} = 2, I_{n,3} = 1$

$B_{n,4} = 2, I_{n,4} = 0$

Fig. 1. A trie of $n = 5$ records and its profiles: the circles represent internal nodes and rectangles holding the records are external nodes

Throughout the paper, we write $B_{n,k}$ to denote the number of external nodes (leaves) at distance k from the root; the number of internal nodes at distance k from the root is denoted by $I_{n,k}$. For simplicity, we will refer to $B_{n,k}$ as the [illegible] and $I_{n,k}$ the [illegible]. Figure 1 shows a trie and its profiles.

In this paper we study the profiles of a trie built over n binary strings generated by a memoryless source. More precisely, we assume that the input is a sequence of n independent and identically distributed random variables, each being composed of an infinite sequence of Bernoulli random variables with mean p, where $0 < p < 1$ is the probability of a "1" and $q := 1 - p$ is the probability of a "0". The typical behaviors under this simple model often hold under more general models such as Markovian or dynamical sources, although the technicalities are usually more involved.

The motivation of studying the profiles is multifold. First, there are fine shape measures closely connected to many other cost measures on tries; some of them are indicated below. Second, at least for the first moment, they are also asymptotically close to the profiles of suffix trees, which in turn have a direct combinatorial interpretation in terms of words; see [9,15] for more information and another interpretation in terms of urn models. Third, not only the analytic problems are mathematically challenging, but the diverse new phenomena they exhibit are highly interesting and unusual. Fourth, our findings imply several new results on other shape parameters. Finally, most properties of random tries have also a prototype character and are expected to hold for other varieties of digital search trees (and under more general random models), although the proofs are generally more complicated.

Due to the usefulness of tries, many cost measures, discussed below, on random tries have been studied in the literature since the early 1970's, and most of these measures can be expressed and analyzed through the profiles studied in this paper. The [illegible] of a trie is the distance from the root to a randomly selected node; its distribution is given by the expected external profile divided by n. The [illegible], the sum of distances between nodes and the root, is defined as $\sum_j jI_{n,j}$. The [illegible] of a trie is the total number of internal nodes, or $\sum_j I_{n,j}$. The [illegible] of a trie is the length of the longest path from the root, or $\max\{j : B_{n,j} > 0\}$. The [illegible], the length of the shortest path from the root to an external node, is given by $\min\{j : B_{n,j} > 0\}$. The largest full level of a trie, called a [illegible], is given by $\max\{j : I_{n,j} = 2^j\}$.

We observe that by assumption of the model, the probability generating function $P_{n,k}(y) := \mathbb{E}(y^{B_{n,k}})$ of the external profile satisfies the recurrence

$$P_{n,k}(y) = \sum_{0\le j\le n} \binom{n}{j} p^j q^{n-j} P_{j,k-1}(y) P_{n-j,k-1}(y) \qquad (n \ge 2; k \ge 1), \tag{1}$$

with the initial conditions $P_{n,k}(y) = 1 + \delta_{n,1}\delta_{k,0}(y-1)$ when either $n \le 1$ and $k \ge 0$ or $k = 0$ and $n \ge 0$, where $\delta_{a,b}$ is the Kronecker symbol. Observe that this recurrence depends on two parameters n and k, which makes the analysis quite challenging, as we will demonstrate in this paper. The probability generating functions of the internal profile satisfy the same recurrence (1) but with different initial conditions.

From (1), the moments of $B_{n,k}$ and $I_{n,k}$ (centered or not) are seen to satisfy a recurrence of the form

$$x_{n,k} = a_{n,k} + \sum_{0\le j\le n} \binom{n}{j} p^j q^{n-j} \left(x_{j,k-1} + x_{n-j,k-1}\right),$$

with suitable initial conditions, where $a_{n,k}$ are known (either explicitly or inductively). A standard approach is to consider the Poisson generating function $\tilde{f}_k(z) := e^{-z}\sum_n x_{n,k} z^n/n!$, which in turn satisfies the functional equation

$$\tilde{f}_k(z) = \tilde{g}_k(z) + \tilde{f}_{k-1}(pz) + \tilde{f}_{k-1}(qz),$$

with a suitable $\tilde{g}_k(z)$. This equation can be solved explicitly by a simple iteration argument and asymptotically by using the Mellin transform ([2,15]). The final step is to invert from the asymptotics of the Poisson generating function $\tilde{f}_k(z)$ to recover the asymptotics of $x_{n,k}$. This last step is guided by the [illegible], which roughly states that

[illegible] $\{x_n\}_n$ [illegible] $x_n \sim e^{-n}\sum_{j\ge 0} x_j n^j/j!$ (2)

where $x_n \sim y_n$ if $\lim_{n\to\infty} x_n/y_n = 1$. By means of the Poisson heuristic (2), we expect that the expected profile $\mu_{n,k} := \mathbb{E}(B_{n,k})$ satisfies $\mu_{n,k} \sim e^{-n}\sum_{j\ge 0} \mu_{j,k} n^j/j!$. However, as we will see, such a heuristic holds in our case when $q^{2k}n \to 0$ but fails otherwise. The reason is that $\mu_{n,k}$ is too small in this range. We should mention that we need uniformity for our asymptotic approximations in k (varying with n) and in z (in some region in the complex plane) in order to invert the results to obtain $x_{n,k}$ by suitable complex analysis.

As far as probabilistic properties of the profiles of random tries are concerned, very little is known in the literature. Since the distribution of the depth D_n in random tries is given by $\mathbb{P}(D_n = k) = \mu_{n,k}/n$, where we recall $\mu_{n,k} := \mathbb{E}(B_{n,k})$, the asymptotics of the expected profile $\mu_{n,k}$ for $n \to \infty$ and varying $k = k(n)$ can be regarded as local limit theorems for D_n. Although many papers addressed the limiting behaviors of the depth, none dealt with the local limit theorem of D_n and the asymptotics of $\mu_{n,k}$ for varying k. Our result implies an unusual type of local limit theorem for D_n.

On the other hand, Pittel [12] showed that the distribution of the number of pairs of input-strings having a common prefix of length at least k is asymptotically Poisson when k is close to the height. Devroye [1] showed that

$$\begin{aligned}
&\text{if} \quad \frac{\mathbb{E}(B_{n,k})}{\sqrt{n}} \to \infty \quad \text{then} \quad \frac{B_{n,k}}{\mathbb{E}(B_{n,k})} \to 1 \quad \text{in probability;}\\
&\text{if} \quad \mathbb{E}(I_{n,k}) \to \infty \quad \text{then} \quad \frac{I_{n,k}}{\mathbb{E}(I_{n,k})} \to 1 \quad \text{in probability,}
\end{aligned}$$

under very general assumptions on the underlying models. These represent known results concerning profiles. We will see that convergence in probability in both cases holds as long as the variance tends to infinity.

Our results are described in details in the next section. In a brief summary, we show that for $k \le (1-\varepsilon)\log n/\log(1/q)$ the average profile $\mu_{n,k}$ is exponentially small, where $\varepsilon > 0$ is small. When k increases and lies in the range $(\log n - \log\log\log n + O(1))/\log(1/q)$, then $\mu_{n,k}$ decays to zero logarithmically until $k > k^*$ for a specific threshold k^* in this range beyond which $\mu_{n,k}$ suddenly grows unbounded in a logarithmic rate. The rate becomes polynomial $\Theta(n^{\upsilon})$ for some $0 < \upsilon \le 1$ when

$$\frac{1}{\log(1/q)}(1+\varepsilon)\log n \le k \le \frac{2}{\log(1/(p^2+q^2))}(1-\varepsilon)\log n.$$

Surprisingly enough, for this range of k an oscillating factor emerges in the expected profile behavior, that is, $\mathbb{E}(B_{n,k}) \approx G(\log_{p/q} p^k n) n^{\upsilon}/\sqrt{\log n}$, where G is a bounded periodic function. Such a behavior is a consequence of an infinite number of saddle-points appearing in the integrand of the associated Mellin integral transform. This was first observed for the internal profile by Nicodème [9]. For larger values of k, these oscillations disappear since the behavior of the expected profile is dominated by a polar singularity. We also show that a central limit theorem holds for both external and internal profiles for a wide range of k; furthermore, we also show that for k near the height the limiting distribution of the profiles becomes Poisson. Some of these results were already anticipated in [10] and the full version of this paper is [11].

2 Summary of Main Results

We summarize in this section our main results. Crucial to our analysis of the profiles is the asymptotics of the expected profiles. Not only are the results fundamental and highly interesting, but also the analytic methods we used are of certain generality.

From (1), we see that the expected external profile $\mu_{n,k} := \mathbb{E}(B_{n,k})$ satisfies the following recurrence

$$\mu_{n,k} = \sum_{0\le j\le n} \binom{n}{j} p^j q^{n-j} (\mu_{j,k-1} + \mu_{n-j,k-1}), \tag{3}$$

for $n \ge 2$ and $k \ge 1$ with the initial values $\mu_{n,0} = 0$ for all $n \ne 1$ and 1 for $n = 1$. Furthermore, $\mu_{0,k} = 0, k \ge 0$ and $\mu_{1,k} = 0$ for $k \ge 1$ and equal to 1 when $k = 0$.

We solve asymptotically (3) for various ranges of k when $p \ne q$; a crude description of the asymptotics of $\mu_{n,k}$ is as follows.

$$\frac{\log \mu_{n,k}}{\log n} \to \begin{cases} 0, & \text{if } \alpha \le \alpha_1; \\ -\rho + \alpha \log(p^{-\rho} + q^{-\rho}), & \text{if } \alpha_1 \le \alpha \le \alpha_2; \\ 2 + \alpha \log(p^2 + q^2), & \text{if } \alpha_2 \le \alpha \le \alpha_3; \\ 0, & \text{if } \alpha \ge \alpha_3, \end{cases} \tag{4}$$

where

$$\alpha_1 := \frac{1}{\log(1/q)}, \quad \alpha_2 := \frac{p^2+q^2}{p^2\log(1/p) + q^2\log(1/q)}, \quad \text{and} \quad \alpha_3 := \frac{2}{\log(1/(p^2+q^2))} \tag{5}$$

are delimiters of $\alpha := \lim_n k/\log n$ $(k = k(n))$, and

$$\rho := \frac{1}{\log(p/q)} \log\left(\frac{1-\alpha\log(1/p)}{\alpha\log(1/q) - 1}\right).$$

We also define $\alpha_0 := 2/(\log(1/p) + \log(1/q))$. Note that $\alpha_1 \le \alpha_2$; see Figure 2. The limiting estimate (4) gives a rough picture of $\mu_{n,k}$ as follows: $\mu_{n,k}$ is of polynomial growth rate when $\alpha_1 + \varepsilon \le \alpha \le \alpha_3 - \varepsilon$, and is smaller than any polynomial powers when $0 \le \alpha \le \alpha_1 - \varepsilon$ and $\alpha \ge \alpha_3 + \varepsilon$. Near the two boundaries α_1 and α_3, the behaviors of $\mu_{n,k}$ will undergo phase-changes from being sub-polynomial to being polynomial or the other way around.

To derive more precise asymptotics of $\mu_{n,k}$ than the phase transitions (4) of the polynomial order of $\mu_{n,k}$, we divide all possible values of k into four overlapping ranges.

(I) [illegible] : $1 \le k \le \alpha_1(\log n - \log\log\log n + O(1))$;
(II) [illegible] : $\alpha_1(\log n - \log\log\log n + K_n) \le k \le \alpha_2(\log n - K_n\sqrt{\log n})$;
(III) [illegible] : $k = \alpha_2 \log n + o((\log n)^{2/3})$;
(IV) [illegible] : $k \ge \alpha_2 \log n + K_n\sqrt{\log n}$,

where, throughout this paper, $K_n \ge 1$ represents a (generic) sequence tending to infinity.

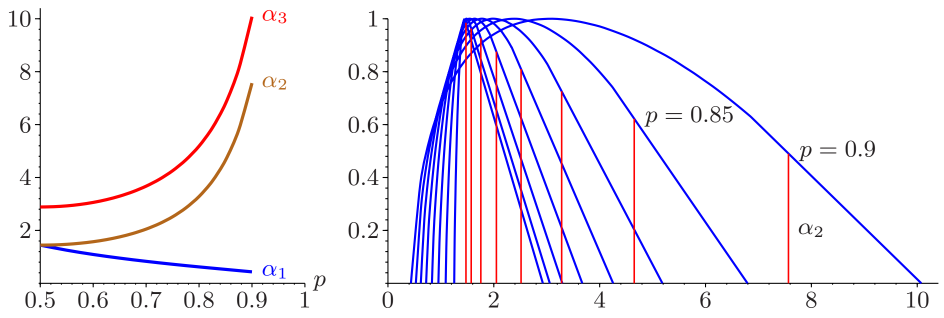


Fig. 2. *Left: A plot of α_1, α_2, and α_3 (defined in (5)) as functions of p. Right: The (non-zero) limiting order of $\log \mu_{n,k}/\log n$ plotted against $\alpha = \lim_n k/\log n$ for $p = 0.55, 0.6, \ldots, 0.9$ (the spans of the curves increase as p grows). The vertical lines represent the positions of α_2 (to the right of which the curves are straight lines); see (4).*

For convenience, we also write

$$L_n := \log n, \quad LL_n := \log\log n, \quad LLL_n := \log\log\log n.$$

The generic symbol ε is always used to represent a suitably small constant whose value may vary from one occurrence to another, and K_n denotes any sequence tending to infinity. The symbol $f(n) = \Theta(g(n))$ means that there are positive constants C and C' such that $C|g(n)| \le |f(n)| \le C'|g(n)|$.

Theorem 1 below precisely characterize the profile in range (I). Define

$$k_m := \alpha_1 \left(L_n - LLL_n + \log\left(\frac{p}{q} - 1\right) + m \log \frac{p}{q} \right) \qquad (m \ge 0), \tag{6}$$

$$S_{n,k,j} := \binom{k}{j} p^j q^{k-j} n \left(1 - p^j q^{k-j}\right)^{n-1} \qquad (0 \le j \le k).$$

Also, define $k_{-1} = 0$.

Theorem 1 (Asymptotics of $\mu_{n,k}$ in Range (I)). $m \ge 0$

$$k_{m-1} + \frac{\alpha_1 K_n}{LL_n} \le k \le k_m - \frac{\alpha_1 K_n}{LL_n}, \tag{7}$$

$$\mu_{n,k} = S_{n,k,m} \left(1 + O((m+1)e^{-K_n})\right). \tag{8}$$

$k = k_m + \alpha_1 x / LL_n$ $x = o(\sqrt{LL_n})$

$$\mu_{n,k} = S_{n,k,m} \left(1 + \frac{p\alpha_1 e^x}{q(m+1)}\right) \left(1 + O\left(x^2 LL_n^{-1} + (m+1) L_n^{-(1-q/p)}\right)\right). \tag{9}$$

Roughly speaking, for k lying in range (I) the expected external profile $\mu_{n,k}$ decays first exponentially fast (asymptotic to $q^k n(1-q^k)^{n-1}$). Then, when k is around $\alpha_1(\log n - \log\log\log n + \log(p/q-1) + m\log(p/q))$ for some integer $m \geq 0$,

$$\mu_{n,k} \sim \frac{k^m}{m!} p^m q^{k-m} n e^{-np^m q^{k-m}},$$

which is of order

$$\mu_{n,k} = O\left(\frac{\log\log n}{\log^{\xi-m} n}\right),$$

for some ξ. Thus, for $m < \xi$ the expected external profile decays only logarithmically, but for $m \geq \xi$ it increases logarithmically.

The behavior of $\mu_{n,k}$ in range (II) is described in Theorem 2 below. This is the most interesting region.

Theorem 2 (Asymptotics of $\mu_{n,k}$ in Range (II)). [illegible] k [illegible]

$$\mu_{n,k} = G_1\left(\rho; \log_{p/q} p^k n\right) \frac{n^{-\rho}\left(p^{-\rho}+q^{-\rho}\right)^k}{\sqrt{2\pi\beta_2(\rho)k}} \left(1 + O\left(\frac{1}{k(p/q)^\rho} + \frac{1}{k(\rho+2)^2}\right)\right), \tag{10}$$

[illegible] $\rho = \rho(n,k) > -2$ [illegible]

$$\begin{cases} \dfrac{\partial}{\partial\rho}\left(\rho^\rho e^{-\rho} n^{-\rho}(p^{-\rho}+q^{-\rho})^k\right) = 0, & [illegible]\ \rho \geq 1; \\ \dfrac{\partial}{\partial\rho}\left(n^{-\rho}(p^{-\rho}+q^{-\rho})^k\right) = 0, & [illegible]\ \rho \leq 1, \end{cases} \tag{11}$$

[illegible]

$$\beta_2(\rho) := \frac{p^{-\rho}q^{-\rho}\log(p/q)^2}{(p^{-\rho}+q^{-\rho})^2}, \tag{12}$$

$$G_1(\rho; x) = \sum_{j\in\mathbb{Z}} g(\rho+it_j)\Gamma(\rho+1+it_j)e^{-2j\pi ix} \qquad (t_j := 2j\pi/\log(p/q))$$

[illegible] $g(s) = 1 - 1/(p^{-s}+q^{-s})$ [illegible] $G_1(\rho, x)$ [illegible] 1, [illegible]

[illegible] □

As clearly spelled out in the above findings, the situation in range (II) becomes highly nontrivial and interesting. More precisely, for $\alpha_1(1+\varepsilon)\log n \leq k \leq \alpha_2(1-\varepsilon)\log n$, we find that

$$\mu_{n,k} \sim G_1\left(\rho; \log_{p/q} p^k n\right) \frac{p^\rho q^\rho (p^{-\rho}+q^{-\rho})}{\sqrt{2\pi\alpha_{n,k}}\log(p/q)} \times \frac{n^{\upsilon_1}}{\sqrt{\log n}},$$

where ($\alpha_{n,k} := k/\log n$)

$$\upsilon_1 = -\rho + \alpha_{n,k}\log(p^{-\rho}+q^{-\rho}),$$

$$\rho = -\frac{1}{\log(p/q)}\log\left(\frac{-1-\alpha_{n,k}\log q}{1+\alpha_{n,k}\log p}\right),$$

and $G_1(\rho; x)$ is a periodic function. We plot in Figures 3 the periodic parts of $G_1(-1, x)$ for a few values of p. These oscillations are consequences of an infinite number of saddle-points appearing in the integrand of the associated Mellin transform of the expected profile.

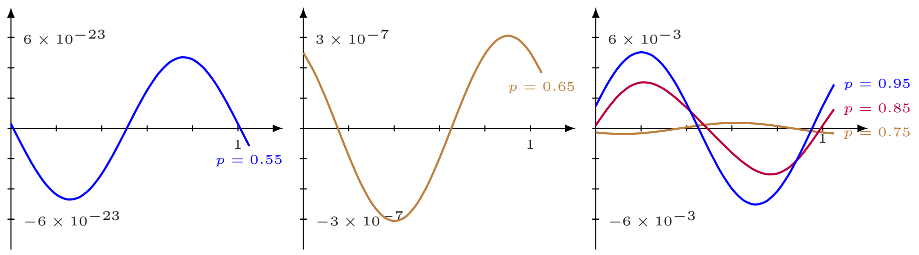


Fig. 3. *The fluctuating part of the periodic function* $G_1(-1; x)$ *for* $p = 0.55, 0.65, \ldots, 0.95$ *and for* x *in the unit interval; its amplitude tends to zero when* $p \to 0.5^+$

Finally, in ranges (III) and (IV) the expected profiles behaves as described in the following two theorems.

Theorem 3. [illegible]

$$k \geq \alpha_2 \left(L_n + K_n \sqrt{\alpha_2 \beta_2(-2) L_n} \right), \tag{13}$$

[illegible] β_2 [illegible] fi [illegible] [illegible]

$$\mu_{n,k} = 2pqn^2(p^2+q^2)^{k-1} \left(1 + O\left(K_n^{-1} e^{-K_n^2/2 + O(K_n^3/\sqrt{L_n})}\right)\right), \tag{14}$$

[illegible] $1 \ll K_n = o(\sqrt{L_n})$

Theorem 4. [illegible]

$$k = \alpha_2 \left(L_n + \xi \sqrt{\alpha_2 \beta_2(-2) L_n} \right), \tag{15}$$

[illegible] $\xi = \xi_{n,k} = o(L_n^{1/6})$ [illegible]

$$\mu_{n,k} = |g(-2)| \Phi(\xi) n^2 \left(p^2+q^2\right)^k \left(1 + O\left(\frac{1+|\xi|^3}{\sqrt{L_n}}\right)\right), \tag{16}$$

[illegible] ξ [illegible] $\Phi(\xi) = (2\pi)^{-1/2} \int_{-\infty}^{\xi} e^{-t^2/2} \mathrm{d}t$

Roughly speaking, in Theorem 3 we prove that for k in range (IV)

$$\mu_{n,k} \sim \frac{2pq}{p^2+q^2} n^{\upsilon_2},$$

where $\upsilon_2 = 2 + \alpha_{n,k} \log(p^2 + q^2)$, and the periodic function disappears. In this region, the asymptotic behavior of the expected profile is dictated by the expected number of pairs (of input-strings) having common prefixes of length at least k. This property is analytically reflected by a polar singularity in the associated Mellin transform. Asymptotics of $\mu_{n,k}$ in range (III) for $k = \alpha_2 \log n + o(\log^{2/3} n)$ is presented in Theorem 4. In this transitional range, the saddle-point coalesces with the polar singularity, so we use the Gaussian integral to describe the behavior of $\mu_{n,k}$.

In summary, our results roughly state that $\mu_{n,k} \to 0$ when $1 \le k \le k^*$ for some k^* close to $\alpha_1(\log n - \log\log\log n + O(1))$, then $\mu_{n,k}$ tends [illegible] to infinity at a logarithmic rate when $k > k^*$. Such an abrupt change has already been observed before in the literature for the shortest path and the fill-up level (see [6,12]), but not much is known for $\mu_{n,k}$ beyond that. Then we show that $\mu_{n,k}$ grows polynomially when k lies in the range $\alpha_1(1+\varepsilon)\log n \le k \le \alpha_3(1-\varepsilon)\log n$, reaching the peak where it is of order $n/\sqrt{\log n}$; it decays in a slower rate afterwards until it tends to zero again when $k \ge \alpha_3(\log n + K_n)$. A salient feature here is the presence of an oscillating function in the asymptotic approximation when $p \ne q$[1].

The expected value of the internal profile $\mathbb{E}(I_{n,k})$ is analyzed in a similar fashion. In particular, the expected internal profile is asymptotically equivalent to 2^k for $k \le \alpha_0(\log n - K_n\sqrt{\log n})$, where $\alpha_0 := 2/(\log(1/p) + \log(1/q))$. When $k \ge \alpha_2(\log n + K_n\sqrt{\log n})$, then $\mathbb{E}(I_{n,k}) \sim (p^2+q^2)\mathbb{E}(B_{n,k})/pq$. Between these two ranges, it is again the infinite number of saddle-points that yields the dominant asymptotic approximation. Unlike $\mu_{n,k}$, an additional phase transition appears in the asymptotics of the $\mathbb{E}(I_{n,k})$ when $k = \alpha_0 \log n + O(\sqrt{\log n})$, reflecting the structural change of the internal nodes from being asymptotically full to being of the same order as the number of external nodes.

Next we deal with variances of the external and internal profiles. Theorem 5 describes their the asymptotic behaviors.

Theorem 5. (i) [illegible] $1 \le k \le \alpha_1(1+o(1))L_n$ [illegible]

$$\sigma^2_{n,k} \sim \mu_{n,k}. \tag{17}$$

(ii) [illegible] $\alpha_1(L_n - LLL_n + K_n) \le k \le \alpha_2(L_n - K_n\sqrt{L_n})$ [illegible]

$$\sigma^2_{n,k} = G_2\left(\rho; \log_{p/q} p^k n\right) \frac{n^{-\rho}\left(p^{-\rho}+q^{-\rho}\right)^k}{\sqrt{2\pi\beta_2(\rho)k}} \left(1 + O\left(\frac{1}{k(p/q)^\rho} + \frac{1}{k(\rho+2)^2}\right)\right), \tag{18}$$

[illegible] $\rho = \rho(n,k) > -2$ [illegible]

$$G_2(\rho; x) = \sum_{j\in\mathbb{Z}} h(\rho + it_j)\Gamma(\rho+1+it_j)e^{-2j\pi i x} \qquad (t_j := 2j\pi/\log(p/q)).$$

(iii) [illegible] $k \ge \alpha_2(1-o(1))L_n$ [illegible]

[1] The expected values of many shape characteristics of random tries often exhibit the asymptotic pattern: $\sim F(\log_c n)n$ if $\log p/\log q$ is rational for some periodic function F and constant c expressible in terms of p, and $\sim Cn$ if $\log p/\log q$ is irrational; see [5,14,15]

$$\sigma_{n,k}^2 \sim 2\mu_{n,k}. \tag{19}$$

We observe that asymptotically the variance of the profile turns out to be of the same order as the expected value for all ranges of $k \geq 1$, namely, $\mathbb{V}(B_{n,k}) = \Theta(\mathbb{E}(B_{n,k}))$. In fact, we show that $\mathbb{V}(B_{n,k}) \sim \mathbb{E}(B_{n,k})$ in range (I), while $\mathbb{V}(B_{n,k}) \sim 2\mathbb{E}(B_{n,k})$ in range (IV), and in range (II) (polynomial growth) the variance and the expected profile differ only by the oscillating functions. The variance of the internal profile behaves almost identically to the variance of the external profile; roughly, $\mathbb{V}(I_{n,k}) = \Theta(\mathbb{V}(B_{n,k}))$ for all k.

Finally, we establish some distributional results as shown below.

Theorem 6. (i) [illegible] $\sigma_{n,k} \to \infty$ [illegible]

$$\frac{B_{n,k} - \mu_{n,k}}{\sigma_{n,k}} \xrightarrow{d} \mathcal{N}(0,1), \tag{20}$$

[illegible] $\mathcal{N}(0,1)$ [illegible] $\xrightarrow{d}$ [illegible] (ii) [illegible] $\sigma_{n,k} = \Theta(1)$ [illegible]

$$\begin{cases} \mathbb{P}\left(B_{n,k} = 2m\right) = \dfrac{\lambda_0^m}{m!}\, e^{-\lambda_0} + o(1), \\ \mathbb{P}\left(B_{n,k} = 2m+1\right) = o(1), \end{cases} \tag{21}$$

[illegible] fi[illegible] $m \geq 0$ [illegible] $\lambda_0 := pqn^2(p^2+q^2)^{k-1}$

Similarly, for the internal profile we have the following result.

Theorem 7. (i) [illegible] $\mathbb{V}(I_{n,k}) \to \infty$ [illegible]

$$\frac{I_{n,k} - \mathbb{E}(I_{n,k})}{\sqrt{\mathbb{V}(I_{n,k})}} \xrightarrow{d} \mathcal{N}(0,1)$$

(ii) [illegible] $\mathbb{V}(I_{n,k}) = \Theta(1)$ [illegible] $\lambda_1 := n^2(p^2+q^2)/2$

$$\mathbb{P}(I_{n,k} = m) = \frac{\lambda_1^m}{m!}\, e^{-\lambda_1} + o(1), \tag{22}$$

[illegible] $m \geq 0$

References

1. Devroye, L.: Laws of large numbers and tail inequalities for random tries and PATRICIA trees. Journal Computational and Applied Mathematics 142 (2002)
2. Flajolet, P., Gourdon, X., Dumas, P.: Mellin transforms and asymptotics: harmonic sums. Theoretical Computer Science 144, 3–58 (1995)
3. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press, Cambridge (1997)
4. Hwang, H.-K.: Profiles of random trees: plane-oriented recursive trees (preprint submitted for publication, 2005)

5. Jacquet, P., Szpankowski, W.: Analytical depoissonization and its applications. Theoretical Computer Science 201, 1–62 (1998)
6. Knessl, C., Szpankowski, W.: On the number of full levels in tries. Random Structures and Algorithms 25, 247–276 (2004)
7. Knuth, D.E.: The Art of Computer Programming, Volume III: Sorting and Searching, 2nd edn. Addison Wesley, Reading (1998)
8. Mahmoud, H.M.: Evolution of Random Search Trees. John Wiley & Sons, New York (1992)
9. Nicodème, P.: Average profiles, from tries to suffix-trees. In: Martínez, C. (ed.) 2005 International Conference on Analysis of Algorithms. Discrete Mathematics and Theoretical Computer Science, pp. 257–266 (2005)
10. Park, G., Szpankowski, W.: Towards a complete characterization of tries. In: SIAM-ACM Symposium on Discrete Algorithms, Vancouver, pp. 33–42 (2005)
11. Park, G., Hwang, H.K., Nicodeme, P., Szpankowski, W.: Profile of Tries (preprint, 2006)
12. Pittel, B.: Paths in a random digital tree: limiting distributions. Advances in Applied Probability 18, 139–155 (1986)
13. Prodinger, H.: How to select a loser. Discrete Mathematics 120, 149–159 (1993)
14. Schachinger, W.: Asymptotic normality of recursive algorithms via martingale difference arrays. Discrete Mathematics and Theoretical Computer Science 4, 363–397 (2001)
15. Szpankowski, W.: Average Case Analysis of Algorithms on Sequences. Wiley, New York (2001)

Random 2-XORSAT at the Satisfiability Threshold

Hervé Daudé[1] and Vlady Ravelomanana[2]

[1] Laboratoire d'Analyse, Topologie et Probabilités
UMR CNRS 6632 – Université de Provence
39, rue Joliot Curie, 13453 Marseille Cedex 13, France
[2] Laboratoire d'informatique de Paris Nord
UMR CNRS 7030 – Institut Galilée - Université Paris Nord
99, Av. Clément, 93430 Villetaneuse, France

Abstract. We consider the random 2-XOR satisfiability problem, in which each instance is a formula that is a conjunction of m Boolean equations of the form $x \oplus y = 0$ or $x \oplus y = 1$. Random formulas on n Boolean variables are chosen uniformly at random from among all $\binom{n(n-1)}{m}$ possible choices. This problem is known to have a coarse transition as n and m tends to infinity in the ratio $m/n \to c$ in particular the probability $p(n, cn)$ that a random 2-XOR formula is satisfiable tends to zero when c reaches $c = 1/2$. We determine the rate $n^{-1/12}$ at which this probability approaches zero inside the scaling window $m = \frac{n}{2}(1 + \mu n^{-1/3})$. This main result is based on a first exact enumeration result about the number of connected components of some constrained family of random edge-weighted (0/1) graphs, namely those without cycles of odd weight. This study relies on the symbolic method and analytical tools coming from generating function theory which enable us to describe the evolution of $n^{1/12}\ p(n, \frac{n}{2}(1 + \mu n^{-1/3}))$ as a function of μ. Our results are in accordance with those obtained by statistical physics methods, their tightness points out the benefit one could get in developping generating function methods for the investigation of phase transition associated to Constrained Satisfaction Problems.

1 Introduction

1.1 Context

The last decade has seen a growth of interest in the phase transition for Boolean Satisfiability (SAT) and more generally for Constraint Satisfaction Problems (CSP). For any $k \geq 2$, the random version of the famous k-SAT problem is known to exhibit a sharp [12] phase transition in the sense that, as the density c of formulas (where the number of clauses is c times the number of variables) is increased, formulas abruptly change from being satisfiable to being unsatisfiable at a critical threshold point. For general CSP, determining the nature of the phase transition (sharp or coarse), locating it, determining a precise scaling window and better understanding the structure of the space of solutions turn out to be very challenging tasks, which have aroused a lot of interest in different communities, namely mathematics, computer science and statistical physics (see e.g. [11], [1]). It is well known that a random 2-SAT formula with density $c < 1$ is satisfiable with probability tending to 1 as the number n of variables tends to infinity, while for $c > 1$, the probability of satisfiability tends to 0 as n tends to infinity [9], [13]. Indeed there is now, [2], a detailed picture of the transition yielding a scaling window of size $\Theta(n^{2/3})$. For greater values of k much less is known about the precise behaviour of random k-SAT near the threshold point whose exact location is still an open problem.

The difficulty of identifying transition factors and of performing theoretical explorations of the SAT transition has incited many researchers to turn to a variant of the SAT problem: the k-XORSAT problem. One is given a linear system over n Boolean variables, composed of m equations modulo 2, each involving exactly $k \geq 2$ variables. This problem introduced in [5] has contributed to develop or validate techniques, thus revealing typical behaviors of both random instances and their space of solutions for SAT-type problems (see, e.g., [4,10,23]). Particularly 2-XORSAT appears to be a seed of coarseness for the transition of a wide class of CSP [7]. Our main goal is to give a precise description of the scaling window associated to the critical ratio $c = \frac{1}{2}$, the zero point of satisfiability for 2-XORSAT. In using the so-called symbolic method from generating function theory [19], we give enumerative and analytic results related to random 2-XORSAT formula that make possible a rigorous verification of the prediction made by statistical physics technique.

Let us note that the satisfiability of 2-XORSAT formulas is strongly related to the existence of cycles in graphs. Such a formula s of density c can be represented by a weighted graph $G(s)$ with n vertices (one for each variable) and cn weighted (0 or 1) edges. For each equation $x_i \oplus x_j = \varepsilon$, we add the edge $\{x_i, x_j\}$ weighted ε. Then, observe that formula s is satisfiable if and only if $G(s)$ does not contain any cycle of odd weight. From this, one can deduce, see [6], that the probability $p(n, m)$ that a random 2-XORSAT formula over n variables with m equations is satisfiable verifies:

$$\lim_{n\to+\infty} p(n, cn) = \exp(c/2) \cdot (1 - 2c)^{1/4} \text{ for } 0 \leq c \leq 1/2 \text{ and } 0 \text{ otherwise.}$$

Figure 1 shows the evolution of $p(n, cn)$ as function of the density c, and for various sizes n, as well as the asymptotic limiting curve i.e. the threshold distribution function of the monotone property 2-XORSAT. As noticed in [27], a spectacular change of finite-size effects appears when c gets closer to $c = \frac{1}{2}$, the zero satisfiability point. We will give a precise characterization of this phenomenon, in particular we will show that the probability of satisfiability is of order of magnitude $\Theta(n^{-1/12})$ as the number of variables n gets large and whenever the number of equations satisfies $m = \frac{n}{2}(1 + \mu n^{-1/3})$, μ being a fixed real number.

1.2 Main Result and Outline of Proof

Whenever the tuned parameter μ is fixed, our main finding can be stated precisely as follows :

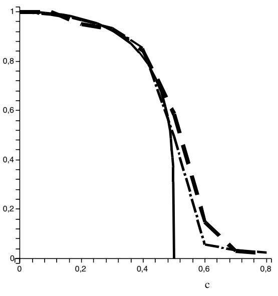


Fig. 1. Probability $p(n, cn)$ that a random 2-XOR formula with cn equations and n variables is satisfiable as a function of the ratio c, for various size of n (dashed $n = 1000$, dotted $n = 2000$) and the asymptotic threshold function (bold full line)

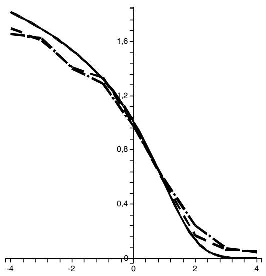


Fig. 2. Rescaled probability at the zero threshold point $c = 1/2$: $n^{1/12}\, p(n, n/2 + \mu n^{-1/3})$ as a function of μ , for $n = 1000$ (dashed), $n = 2000$ (dotted) with the theoretical limit (bold full line)

Theorem 1. *Let μ be any real constant. The probability $p(n, m)$ that a random 2-XORSAT formula with n variables and $m = \frac{n}{2}(1 + \mu n^{-1/3})$ equations is satisfiable verifies :*

$$\lim_{n\to\infty} n^{1/12} p(n, m) = \left(\sum_{r=0}^{\infty} \frac{\sqrt{2\pi}\, e^{1/4} e_r}{2^r}\, A(3r + 1/4, \mu) \right) , \tag{1}$$

where the sequence $(e_r)_{r\in\mathbb{N}}$ satisfies

$$\sum_{r=0}^{\infty} e_r x^r = \exp\left(\sum_{r=1}^{\infty} \frac{(6r)!}{2^{5r-1} 3^{2r} (3r)!\,(2r)!}\, x^r \right) \tag{2}$$

and

$$A(y, \mu) = \frac{e^{-\mu^3/6}}{3^{(y+1)/3}} \sum_{k\ge 0} \frac{\left(\frac{1}{2} 3^{2/3} \mu\right)^k}{k!\, \Gamma\big((y + 1 - 2k)/3\big)} \,. \tag{3}$$

Outline of the proof. For any $r \ge 0$, let $p_r\,(n,\, m)$ be the probability that an edge-weighted graph with $m = \frac{1}{2} n \left(1 + \mu n^{-1/3}\right)$ weighted (0/1) edges and n vertices has no cycle of odd weight and exactly $n - m + r$ acyclic components (r is the total excess of the graph see for example [21]). We have:

$$p(n, m) = \sum_{r\ge 0} p_r\,(n,\, m) \ .$$

Then, the two following facts show that our main result is a direct consequence of the dominated convergence theorem.

Fact (i): For all integer $r \geq 0$

$$n^{1/12} p_r\,(n,\, m) \sim \frac{\sqrt{2\pi}\, e^{1/4} e_r}{2^r}\, A(3r + 1/4, \mu)\,. \tag{4}$$

Fact (ii): There exits R, C, $\epsilon > 0$ such that for all $r \geq R$ and all n:

$$n^{1/12}\ p_r\,(n,\, m) \leq C\ e^{-\epsilon\, r}\,. \tag{5}$$

In contrast with the above mentionned results on random 2-SAT, we obtain sharper characterization in the scaling window. The accuracy of our results can be compared to the one on the finite size scaling for the core of large random hypergraphs in [14] expressed in terms of the Airy function. This function has been encountered in the physics of the classical random graphs [21] and is shown in [17,18] related to $A(y, \mu)$ appearing in our formula (1). In Figure 2, we give empirical results corresponding to random 2-XORSAT formulas with $n = 1000$ (resp. $n = 2000$) variables around the point phase transition $m = \frac{n}{2} + \mu\frac{n^{2/3}}{2}$ with $\mu \in [-4, 4]$. The figure illustrates the accuracy of formula (1).

1.3 Organization of the Paper and Further Results as $|\mu|$ is Large

In this paper, we embark on 2-XORSAT phase transition study with the powerful tools developped by analytic combinatorics. We show that methods developped on simple graphs [29,30,17,21,24,25] are also well suited for constrained weighted graphs associated to random 2-XORSAT formulas. In Section 2, Theorem 2 gives an explicit differential recurrence between Exponential Generating Functions C_ℓ of ℓ-connected components of weighted graph without cycle of odd weight. Then we deduce inequalities on C_ℓ involving simpler generating functions that concentrate essential informations about the asymptotical behaviour of C_ℓ's coefficients. In Section 3 we give the extended proof of our main theorem that is the one of facts (i) and (ii) mentionned above. To end this introductory section, we mention without proof more estimates that might be obtained within the same framework.

When $|\mu|$ grows with n (say for example $\mu = +n^{1/15}$ or $\mu = -\log n^2$), the results presented here can be extended to show that the probability $p\left(n, \frac{n}{2} + \frac{\mu n^{2/3}}{2}\right)$ is about $\sim e^{1/4}|\mu|^{1/4} n^{-1/12}$ whenever μ tends to $-\infty$. Similarly, $p\left(n, \frac{n}{2} + \frac{\mu n^{2/3}}{2}\right)$ is of order of magnitude $O\left(\mu^{-7/8} \exp\left(-\mu^3/6\right)\, n^{-1/12}\right)$ when μ tends to $+\infty$ with n but $\mu = O(n^{1/12})$. The proofs rely on the same principles except that instead of (4) we have (6) and (7) :

- $\forall r \in \mathbb{N}$ and $\mu(n)$ s. t. $|\mu(n)| = O(n^{1/12})$ and $\lim_{n\to\infty} \mu(n) = -\infty$

$$\lim_{n\to\infty} n^{1/12} |\mu(n)|^{3r-1/4} p_r(n, m) = \frac{e^{1/4} e_r}{2^r}. \tag{6}$$

•• $\forall r \in \mathbb{N}$ and $\mu(n)$ s. t. $\mu(n) \gg 1$ but $\mu(n) = O(n^{1/12})$

$$\lim_{n\to\infty} n^{1/12} \exp\left(\frac{\mu(n)^3}{6}\right) \mu(n)^{7/8-3r/2}\, p_r(n,m) = \frac{1}{2^{3r/2+1/8}\Gamma(3r/2+1/8)} . \tag{7}$$

2 Exact Enumeration

Recall that 2-XORSAT formula is satisfiable if and only if the associated weighted graphs have no cycle of odd weight. In this paragraph, we investigate the EGF of these combinatorial structures.

2.1 Generating Functions

We adapt the definitions on simple graphs [17,21] to simple (without self-loops nor multiple edges), undirected and labeled graphs with weighted edges 0 or 1.

Let $t(z)$ be the exponential generating function (EGF for short) of rooted labeled trees, we know from [3] that:

$$t(z) = ze^{t(z)} = \sum_{n=1}^{\infty} n^{n-1} \frac{z^n}{n!} . \tag{8}$$

Recall that a tree on n vertices has $n-1$ edges thus, in our case, the EGF of rooted labeled weighted trees with edge weight from $\{0, 1\}$ is:

$$T(z) = \sum_{n=1}^{\infty} (2n)^{n-1} \frac{z^n}{n!} = \frac{t(2z)}{2} = ze^{2T(z)} . \tag{9}$$

The differential operator $\vartheta_z = z\frac{\partial}{\partial z}$ corresponds to marking a vertex. To consider a pair of distinct vertices we will use the differential operator $\frac{\vartheta_z^2 - \vartheta_z}{2}$. From (9) we get:

$$\vartheta_z(T) = \frac{T}{1-2T} \quad \vartheta_z^2(T) = \frac{T}{(1-2T)^3} . \tag{10}$$

For sake of simplicity, throughout the rest of this paper $T \equiv T(z)$ which is given by (9).

We say that a connected graph has *excess* ℓ if it has $\ell(\geq -1)$ more edges than vertices. A connected component of excess ℓ is also called ℓ*-component*. If $\ell > 0$, ℓ-components are called *complex*. Let C_ℓ be the EGF of ℓ-components without cycles of odd weight. Observe that C_{-1} is the EGF of unrooted weighted trees thus $\vartheta_z(C_{-1}) = T$, it follows that:

$$C_{-1}(z) = T - T^2 = \sum_{n=1}^{\infty} n^{n-2} 2^{n-1} \frac{z^n}{n!} \tag{11}$$

When the random graph [15,16] evolves, it has many sparse ℓ-components $(-1 \leq \ell)$. Let us define the *total excess* of a graph as follows (see also [21, Section 13]) :

the total excess of a graph is the number of edges plus the number of acyclic components, minus the number of vertices. Let $E_r(z)$ be the EGF of all complex graphs (connected or not) without cycles of odd weight with a positive total excess r. By definition, we have $E_0(z) = 1$, $E_1(z) = C_1(z)$ and the relations

$$E_r(z) = [x^r] \exp\left(\sum_{i=1}^{\infty} x^i C_i(z)\right) \qquad\qquad rE_r = \sum_{k=1}^{r} kC_k\, E_{r-k}\,. \tag{12}$$

(The last equality is obtained by differentiating $\sum x^r E_r(z)$ w.r.t. x and equating the coefficients of x^{r-1}.)

A *smooth* graph is one without vertices of degree one. Smooth graphs can be obtained by recursively pruning the vertices of degree one, i.e. by cutting off repeatedly any vertex of degree 1 in the graph. Conversely, given any smooth graph, we obtain all graphs that prune down to it by simply sprouting a (rooted) tree from each reduced vertex. Accordingly, let $\tilde{C}_\ell$ be the EGF of smooth ℓ-components without cycles of odd weight, it follows that the EGFs C_ℓ, $\tilde{C}_\ell$ and T satisfy for any $\ell \geq 0$:

$$C_\ell(z) = \tilde{C}_\ell\left(T(z)\right). \tag{13}$$

From $n(\geq 3)$ labeled vertices, one can form $\frac{2^n n!}{2n}$ distinct elementary weighted cycle of length n. There is only one weighted (odd) cycle of length 2 and for greater length as many cycles of odd weight as cycles of even weight. Thus, we obtain

$$\tilde{C}_0(z) = \frac{1}{4}\sum_{n=3}^{\infty} 2^n \frac{z^n}{n!} = \frac{1}{4}\log\left(\frac{1}{1-2z}\right) - \frac{z}{2} - \frac{z^2}{2}\,.$$

From (13) we then have

$$C_0 = \frac{1}{4}\log\left(\frac{1}{1-2T}\right) - \frac{T}{2} - \frac{T^2}{2}\,. \tag{14}$$

In order to study random 2-XORSAT formula by means of enumerative approach, we have to compute the EGFs C_ℓ.

2.2 Differential Recurrence for EGFs

Observe that our combinatorial structures are constrained since the considered weighted graphs are without cycles of odd weights. As shown by the longstanding open problem of enumerating exactly triangle-free graphs [20], it is in general extremely difficult to derive EGFs of such structures [26]. However, the following Theorem gives exact enumerative results for the EGFs C_ℓ for all $\ell \geq -1$.

Theorem 2. *Let C_ℓ be the EGF of ℓ-components without cycles of odd weight. We have,*

$$C_{-1} = T - T^2, \quad C_0 = \frac{1}{4}\log\left(\frac{1}{1-2T}\right) - \frac{T}{2} - \frac{T^2}{2}$$

and for $\ell \geq 0$, $C_{\ell+1}(z)$ verifies the differential equation

$$(1-2T(z))\,\vartheta_z C_{\ell+1}(z) + (\ell+1)C_{\ell+1}(z) = \frac{\left(\vartheta_z^2 - 3\vartheta_z - 2\ell\right)}{2} C_\ell(z) + \sum_{p=0}^{\ell} \left(\vartheta_z C_p(z)\right)\,\left(\vartheta_z C_{\ell-p}(z)\right)\,, \tag{15}$$

where $\vartheta_z = z\frac{\partial}{\partial z}$ and T is given by (9).

Proof. We shall use multivariate exponential generating functions (EGF's) to study family of graphs with labeled vertices and edges weighted in $\{0, 1\}$. If $\mathcal{F}$ is such a family the associated EGF is the formal power series

$$F(w,u,z) = \sum_{M\in\mathcal{F}} w^{m_1(M)} u^{m_0(M)} \frac{z^{n(M)}}{n(M)!}\,, \tag{16}$$

where $n(M)$ is the number of vertices of a graph M and $m_0(M)$ (resp. $m_1(M)$) denotes the number of edges of weight 0 (resp. 1) of M. Also, a graph M is of *odd* (resp. *even*) *weight* iff $m_1(M)$ is odd (resp. even).

To prove (15), we show the following

$$(\vartheta_w + \vartheta_u)C_{\ell+1} + (u+w)\left(\frac{\vartheta_z^2 - \vartheta_z}{4}\right)C_\ell = w\left(\frac{\vartheta_z^2-\vartheta_z}{2} - \vartheta_w\right)C_\ell + u\left(\frac{\vartheta_z^2-\vartheta_z}{2} - \vartheta_u\right)C_\ell + \frac{(w+u)}{2}\sum_{p=-1}^{\ell+1}\left(\vartheta_z C_p\right)\,\left(\vartheta_z C_{\ell-p}\right)\,, \quad (\ell \geq 0)\,. \tag{17}$$

which gives more lights on the combinatorial interpretations of (15).

The second term, i.e. the summation, in the RHS of (17) reflects the choices of one vertex v_p from a connected p-component ($p \in [-1\,, \ell+1]$) and another vertex $v_{\ell-p}$ from an $(\ell-p)$-component (both components without cycles of odd weight) in order to add a distinguished edge of weight 1 (resp. 0), viz. $v_p \overset{\mathbf{1}}{-} v_{\ell-p}$ (resp. $v_p \overset{\mathbf{0}}{-} v_{\ell-p}$), between the two previously disconnected components.

In the RHS of (17), the two terms before the summation correspond to adding a distinguished edge (respectively of weight 1 or 0) to an existing ℓ-component without cycles of odd weight. To add this last edge, we may choose a pair of distinct vertices not already connected by an edge of weight 1 (resp. 0) thus the operator $\frac{\vartheta_z^2-\vartheta_z}{2} - \vartheta_w$ (resp. $\frac{\vartheta_z^2-\vartheta_z}{2} - \vartheta_u$). We observe that this last edge (independently of its weight) can introduce cycles of odd weight.

Next, the first term in the LHS of (17) corresponds to starting with a connected component of $\mathcal{C}_{\ell+1}$ having a distinguished edge.

Finally, the second term in the LHS of (17) take into account the constructions containing cycles of odd weight with a marked edge whose deletion will suppress all the cycles of odd weight. For this last term, let us first remark that if we choose any pair of distinguished vertices in a connected component without cycles of odd weight there is exactly one possibility to insert a weighted edge between these two vertices in order to create an odd cycle. Second, observe that the number of such constructions obtained in adding an edge of weight 0 is equal to the one when adding an edge of weight 1, thus the operator $\frac{\vartheta_z^2-\vartheta_z}{4}$.

Observe that univariate EGF can be obtained from multivariate EGF by setting $w = u = 1$, thereby ignoring the number of edges and their respective weights. Now (15) appears as a direct consequence of (17).

Note that equations (10) and (15) allow us to compute the sequence of EGFs $(C_\ell)_{\ell\geq -1}$ and the first of them are given by: $C_1 = \frac{2\,T^4(3-2T)}{3(1-2\,T)^3}$ and $C_2 = \frac{T^4\left(1+28\,T-46\,T^2+36\,T^3-8\,T^4\right)}{3(1-2\,T)^6}$. When performing partial fraction decomposition w.r.t. T, it yields

$$C_1 = \frac{5}{48}\,\frac{1}{(1-2\,T)^3} - \frac{19}{48}\frac{1}{(1-2\,T)^2} + \frac{13}{24}\frac{1}{(1-2\,T)} - \frac{1}{4} - \frac{1}{8}T + \frac{1}{12}T^2\,.$$

2.3 Inequalities for EGFs

For our purpose, we need the following notations :

Definition 1. *If $A(z) = \sum_n a_n z^n$ and $B(z) = \sum_n b_n z^n$ are two formal power series, we write $A(z) \preceq B(z)$ if and only if $\forall n \in \mathbb{N}$ and $n \geq 3$, $a_n = [z^n]\,A(z) \leq b_n = [z^n]\,B(z)$.*

Expanding the series $(1-2\,T)\vartheta_z A(z) + (\ell+1)A(z)$ we have:

$$n!\;[z^n]\left(\left((1-2\,T\right)\vartheta_z A(z) + (\ell+1)A(z)\right) = (n+\ell+1)a_n - \sum_{k=1}^{n}\binom{n}{k}2^k k^{k-1}(n-k)a_{n-k} \tag{18}$$

This proves the following result :

Lemma 1. *Let $A(z) = \sum_{n\geq 0} a_n \frac{z^n}{n!}$, and $\ell \in \mathbb{N}$. If $\left(1-2\,T\right)\vartheta_z A(z)+(\ell+1)A(z) \succeq 0$ then $A(z) \succeq 0$.*

Theorem 3. *Let $b_1 = \frac{5}{48}$, $c_1 = 19/48$. For $\ell \geq 1$*

$$\begin{aligned} 2(\ell+1)b_{\ell+1} &= 3\ell(\ell+1)b_\ell + 6\sum_{p=1}^{\ell-1} p(\ell-p)b_p b_{\ell-p} \;\; \textit{and} \\ 2(3\ell+2)c_{\ell+1} &= 8(\ell+1)b_{\ell+1} + 3\ell b_\ell + (3\ell-1)(3\ell+2)c_\ell \\ &\quad + 12\sum_{p=1}^{\ell-1} p(3\ell-3p-1)b_p c_{\ell-p}\,. \end{aligned} \tag{19}$$

Then, for $\ell \geq 1$

$$\frac{b_\ell}{(1-2T)^{3\ell}} - \frac{c_\ell}{(1-2T)^{3\ell-1}} \preceq C_\ell(z) \preceq \frac{b_\ell}{(1-2T)^{3\ell}}\,. \tag{20}$$

Proof. By induction on ℓ. Due to place limitation, the proof is omitted.

As a consequence of Theorem 3, we have:

Corollary 1. *We have $E_0(z) = 1$ and $E_1(z) = C_1(z)$. For $\ell \geq 1$, the EGFs E_ℓ satisfies*

$$\frac{e_\ell}{(1-2T)^{3\ell}} - \frac{f_\ell}{(1-2T)^{3\ell-1}} \preceq E_\ell \preceq \frac{e_\ell}{(1-2T)^{3\ell}}\,, \tag{21}$$

where the sequences (b_ℓ), (c_ℓ) and (e_ℓ) are defined respectively by (19) and by $e_\ell = \sum_{k=1}^{\ell} k\, b_k\, e_{\ell-k}$; $f_1 = c_1 = \frac{19}{48}$ and

$$f_\ell = c_\ell + \frac{1}{\ell}\sum_{k=1}^{\ell-1}(kc_k e_{\ell-k} + kb_k f_{\ell-k})\,, \quad (\ell \geq 2)\,. \tag{22}$$

Proof. The bounds are derived by induction.

3 Proof of Theorem 1

In this paragraph, we focus on the studies of random 2-XORSAT formula inside the critical window. Recall that we consider random formulas on n Boolean variables, chosen uniformly at random from among all $\binom{n(n-1)}{m}$ possible choices. The satisfiability of such formulas is the same as the probability that our random weighted graphs have no cycles of odd weight. Such graphs with exactly n vertices, m edges and a total excess r ($r \geq 0$) have exactly $n-m+r$ tree components. Thus, they are enumerated by the following EGF:

$$\frac{C_{-1}(z)^{n-m+r}}{(n-m+r)!} E_r(z) \exp\left(C_0(z)\right)$$

Therefore, the probability $p_r\,(n,\,m)$ that a graph with $m = \frac{1}{2}n\left(1+\mu n^{-1/3}\right)$ edges and n vertices has total excess r with all components without cycles of odd weights is exactly

$$p_r\,(n,\,m) = \frac{n!}{\binom{n(n-1)}{m}}\,[z^n]\,\frac{\left(T-T^2\right)^{n-m+r}}{(n-m+r)!} E_r(z) \frac{e^{-T/2-T^2/2}}{(1-2T)^{1/4}}\,. \tag{23}$$

We used (11) and (14) for respectively C_{-1} and C_0.

3.1 Proof of Fact (i)

The formula (24) below based on techniques introduced in [17] and following [21, Lemma 3], is a key tool for the computation of such probabilities. Let $r \in \mathbb{N}$ be fixed. If $m = \frac{1}{2}n\left(1+\mu n^{-1/3}\right)$ and if y is any real constant, we have

$$\frac{e^{-\mu^3/6-n}}{2^{2m-n-2r}}\,[z^n]\,\frac{\left(T-T^2\right)^{n-m+r}\, e^{-T/2-T^2/2}}{(1-2T)^y} \sim e^{-3/8}\, A(y,\mu)\, n^{y/3-2/3} \tag{24}$$

where $A(y, \mu)$ is given by (3). To prove (24), we first use Cauchy's integral formula and after the substitution $\tau = \frac{z}{2}e^{-z}$, so that $T(\tau) = \frac{z}{2}$. We then obtain

$$[z^n] \frac{\left(T - T^2\right)^{n-m+r} e^{\left(-\frac{T}{2}-\frac{T^2}{2}\right)}}{(1-2T))^y} = \frac{e^n 2^{2m-n-2r-1}}{\pi i} \times \oint (1-z)^{1-y} e^{-z/4-z^2/8} e^{nh(z)} \frac{dz}{z}\,, \quad (25)$$

where

$$h(z) = z - 1 - \log z - \left(1 - \frac{m}{n}\right) \log \frac{1}{1-(z-1)^2}\,. \quad (26)$$

The proof of (24) can now be completed following the one of [21, Lemma 3], by choosing the path of integration $z = e^{-(\alpha+it)n^{-1/3}}$ where t runs from $-\pi n^{1/3}$ to $\pi n^{1/3}$ and α is the positive solution of $\mu = \frac{1}{\alpha} - \alpha$. Note that h defined in (26) is exactly the same as in [21, equation (10.12)] satisfying $h(1) = h'(1) = 0$ and if $m = \frac{n}{2}$ $h''(1) = 0$.
Next, using Stirling approximation it yields

$$\frac{n!}{\binom{n(n-1)}{m}} \frac{1}{(n-m+r)!} \sim \pi^{1/2} \frac{2^{-\mu n^{2/3}+r+1/2}}{n^{r-1/2}} e^{-\mu^3/6-n+5/8}\,. \quad (27)$$

The '$n^{y/3}$' in (24) tells us that we have only to consider the term $\frac{e_r}{(1-2T)^{3r}}$ from E_r (the other term in the lower-bound of E_r in the inequalities of Corollary 1 can be neglegted). Taking $y = 3r + 1/4$ after multiplying (24), (27) and e_r, we obtain the result announced by (4).

3.2 Sketch of the Proof of Fact (ii)

By Corollary 1, (23) and using the change of variable $\tau = \frac{z}{2}e^{-z}$, the probability $p_r\,(n,\, m)$ verifies

$$p_r\,(n,\, m) \le \frac{n!}{\binom{n(n-1)}{m}} \frac{e_r}{(n-m+r)!} \frac{e^n 2^{2m-n-r-1}}{\pi i} \times \oint \left(\frac{z(2-z)}{1-z}\right)^r (1-z)^{3/4-2r} e^{-z/4-z^2/8} e^{nh(z)} dz\,, \quad (28)$$

with $h(z)$ as in (26) and where the contour is a circle $z = \rho e^{i\theta}$ with $0 < \rho < 1$.
Using complex analysis, it can be proved (details omitted) that

$$p_r\,(n,\, m) < \alpha_2\, r^{-5/12}\, n^{-1/12} \exp\left(-\frac{\mu^3}{6} + \mu r^{2/3} + \left(\log 3 - 2\log 2 - \frac{1}{2}\right) r\right) \quad (29)$$

for some constant $\alpha_2 > 0$. Note that, the quantity $(\log 3 - 2\log 2 - \frac{1}{2}) = -.787\cdots < 0$. Therefore, for r sufficiently large we have (5).

4 Conclusion

We have shown that the generating function approach is well suited to make precise the scaling window of a particular Contraint Satisfaction Problem: 2-XORSAT. This problem is governed by the behaviour of cycles in random weighted graphs thus our analysis is a first step towards a fine description of the scaling window associated to other random CSPs like 2-Colourability, i.e. bipartitness, or to random quantified XOR-formulas [8].

References

1. Achlioptas, D., Moore, C.: Random k-SAT: Two Moments Suffice to Cross a Sharp Threshold. SIAM Journal of Computing 36, 740–762 (2006)
2. Bollobás, B., Borgs, C., Chayes, J.T., Kim, J.H., Wilson, D.B.: The scaling window of the 2-SAT transition. Random Structures and Algorithms 18, 201–256 (2006)
3. Cayley, A.: A Theorem on Trees. Quart. J. Math. Oxford Ser. 23, 376–378 (1889)
4. Cocco, S., Dubois, O., Mandler, J., Monasson, R.: Rigorous decimation-based construction of ground pure states for spin glass models on random lattices. Phys. Rev. Lett. 90, 047205 (2004)
5. Creignou, N., Daudé, H.: Satisfiability threshold for random XOR-CNF formulas. Discrete Applied Mathematics 96-97, 41–53 (1999)
6. Creignou, N., Daudé, H.: Coarse and sharp thresholds for random k-XOR-CNF satisfiability. Informatique théorique et applications/Theoretical Informatics and Applications 37(2), 127–147 (2003)
7. Creignou, N., Daudé, H.: Coarse and sharp transitions for random generalized satisfiability problems. In: Proceedings of the third colloquium on mathematics and computer science, Vienna, Austria, pp. 507–516. Birkhäuser, Basel
8. Creignou, N., Daudé, H., Egly, U.: Phase Transition for Random Quantified XOR-Formulas. Journal of Artificial Intelligence Research 28, 1–17 (2007)
9. Chvátal, V., Reed, B.: Mick gets some (the odds are on his side). In: Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, pp. 620–627. IEEE, Los Alamitos (1992)
10. Dubois, O., Mandler, J.: The 3-XOR-SAT threshold. In: Proceedings of the 43th Annual IEEE Symposium on Foundations of Computer Science, pp. 769–778. IEEE, Los Alamitos (2002)
11. Dubois, O., Monasson, R., Selman, B., Zecchina, R.: Editorial. Theoretical Computer Science 265(1-2)
12. Friedgut, E., Bourgain, J.: Sharp thresholds of graph properties, and the k-sat problem. Journal of the A.M.S. 12(4), 1017–1054
13. Goerdt, A.: A threshold for unsatisfiability. Journal of of Computer and System Sciences 53(3), 469–486
14. Dembo, A., Montanari, A.: Finite size scaling for the core of large random hypergraphs. Ann. of App. Prob. (to appear, 2008)
15. Erdös, P., Rényi, A.: On random graphs. Publ. Math. Debrecen 6, 290–297 (1959)
16. Erdös, P., Rényi, A.: On the evolution of random graphs. Publ. Math. Inst. Hung. Acad. Sci. 5, 17–61 (1960)
17. Flajolet, P., Knuth, D.E., Pittel, B.: The first cycles in an evolving graph. Discrete Math. 75, 167–215 (1989)

18. Flajolet, P., Salvy, B., Schaeffer, G.: Airy phenomena and analytic combinatorics of connected graphs. The Electronic Journal of Combinatorics 11(1), R34 (2004)
19. Flajolet, P., Sedgewick, R.: Analytic Combinatorics. Forthcoming book (chapters are avalaible as INRIA research reports at Philippe Flajolet's home page), http://algo.inria.fr/flajolet/Publications/books.html
20. Harary, F., Palmer, E.: Graphical Enumeration. Academic Press, New York and London (1973)
21. Janson, S., Knuth, D.E., Luczak, T., Pittel, B.: The birth of the giant component. Random Structures and Algorithms 4, 233–358 (1993)
22. Sloane, N.J.A., Plouffe, S.: Encyclopedia of Integer Sequences. Academic Press, London, http://www.research.att.com/~njas/sequences/
23. Mézard, M., Ricci-Tersenghi, F., Zecchina, R.: Alternative solutions to diluted p-spin models and XORSAT problems. J. Stat. Phys. 111, 505 (2003)
24. Ravelomanana, V.: The Average Size of Giant Components between the Double-Jump. Algorithmica 46(3-4), 529–555 (2006)
25. Ravelomanana, V.: Another Proof of Wright's inequalities. Inf. Proc. Letters 104(1), 36–39 (2007)
26. Ravelomanana, V., Thimonier, L.: Forbidden subgraphs in connected graphs. Theor. Comput. Sci. 314(1-2), 121–171 (2004)
27. Rémi, M.: Introduction to phase transitons in random optimization problems. Personnal communication (2007)
28. Wright, E.M.: Asymptotic relations between enumerative functions in graph theory. Proc. London Math. Soc. 20, 558–572 (1970)
29. Wright, E.M.: The Number of Connected Sparsely Edged Graphs. Journal of Graph Theory 1, 317–330 (1977)
30. Wright, E.M.: The Number of Connected Sparsely Edged Graphs. III. Asymptotic results. Journal of Graph Theory 4, 393–407 (1980)

On Dissemination Thresholds in Regular and Irregular Graph Classes*

I. Rapaport[1], K. Suchan[1,2], I. Todinca[3], and J. Verstraete[4]

[1] Departamento de Ingeniería Matemática and Centro de Modelamiento Matemático, Universidad de Chile
rapaport@dim.uchile.cl

[2] Faculty of Applied Mathematics, AGH - University of Science and Technology, Cracow, Poland
karol@suchan.info

[3] LIFO, Université d'Orléans, France
Ioan.Todinca@univ-orleans.fr

[4] University of California, San Diego, California, USA
jverstra@math.ucsd.edu

Abstract. We investigate the natural situation of the dissemination of information on various graph classes starting with a random set of informed vertices called active. Initially active vertices are chosen independently with probability p, and at any stage in the process, a vertex becomes active if the majority of its neighbours are active, and thereafter never changes its state. We show that in any cubic graph, with high probability, the information will not spread to all vertices in the graph if $p < \frac{1}{2}$. We give families of graphs in which information spreads to all vertices with high probability for relatively small values of p.

1 Introduction

Let $G = (V, E)$ be a simple undirected graph. A *configuration* C of G is a function that assigns to every vertex in V a value in $\{0, 1\}$. The value 1 means that the corresponding vertex is *active* while the value 0 represents *passive* vertices.

We investigate the natural situation in which a vertex v needs a *strong majority* of its neighbours, namely strictly more than $\frac{1}{2}d(v)$ neighbours, to be active in order to become an active vertex. Therefore, consider the following rule of *dissemination* that acts on configurations: a passive vertex v whose strict majority of neighbours are active becomes active; once active, a vertex never changes its state. The initial configuration of a dissemination process is called an *insemination.* Since the set of active vertices grows monotonically in a finite set V, a fixed point has to be reached after a finite number of steps. If the fixed point is such

* Authors acknowledge the support of CONICYT via Anillo en Redes ACT08 (I.R., K.S.), ECOS-CONICYT (I.R., I.T.), Fondap on Applied Mathematics (I.R.) and an Alfred P. Sloan Fellowship (J.V.).

that all vertices have become active, then we say that the initial configuration *overruns* the graph G. A *community* [10] (also called an *alliance*) in G is a subset of nodes $X \subseteq V$ each of which has at least as many neighbours in X as in $V \setminus X$, i.e. for every $v \in X$, $|N(v) \cap X| \geq |N(v) \cap (V \setminus X)|$. Notice that a configuration overruns G if and only if it contains no community of passive vertices.

Dissemination has been intensively studied in the literature, using various dissemination rules (see e.g. [17] for a survey). Among other types of rules we can cite models in which a vertex becomes active if the total weight of its active neighbours exceeds a fixed value [12], or symmetric majority voting rules, for which an active vertex may also become passive if the number of passive neighbours outweights the number of active neighbours [17]. One of the main questions for each of these models is to find small sets of active vertices which overrun the network. Several authors considered the problem of finding small communities in arbitrary graphs or special graph classes [8,10].

In this work we consider a probabilistic framework. A random configuration in which each vertex is active with probability p and passive with probability $1-p$ is called a *p-insemination*. We are interested in the probability $\theta_p(G)$ that a p-insemination overruns G. It is clear that $\theta_p(G)$ is a monotonic increasing function of p. We investigate the majority dissemination process starting with a p-insemination for various graph classes. Such random dissemination processes, with different types of dissemination rules, have been studied in the literature in the context of cellular automata or in bootstrap percolation [11].

One of the basic questions is to determine the ratio of active vertices (in other words, the critical value of p) one needs in order to overrun the whole graph with high probability. Without any restriction on the structure of the underlying graph, it appears to be difficult to determine this ratio. It is therefore more instructive to consider whole classes of graphs. If $\mathcal{G}$ is a class of graphs, let $\boldsymbol{G} = (G_n)_{n\in\mathbb{N}}$ denote a generic sequence of graphs $G_n \in \mathcal{G}$ such that $|V(G_n)| < |V(G_{n+1})|$ for all $n \in \mathbb{N}$. We define *dissemination half-thresholds* p_c^+ and p_c^- of class $\mathcal{G}$ by

$$p_c^+(\mathcal{G}) = \inf\{p \mid \exists \boldsymbol{G} : \lim \theta_p(G_n) = 1\}$$
$$p_c^-(\mathcal{G}) = \sup\{p \mid \forall \boldsymbol{G} : \lim \theta_p(G_n) = 0\}$$

In words, for $p < p_c^-$ and any increasing sequence $\boldsymbol{G}$ in $\mathcal{G}$, the probability that a random p-insemination overruns the graph tends to zero.

For example, for the class $\mathcal{K}$ of all complete graphs, it is straightforward to see that $p_c^+(\mathcal{K}) = p_c^-(\mathcal{K}) = \frac{1}{2}$. If for a class $\mathcal{G}$ the two half-thresholds are equal, we say that $p_c(\mathcal{G}) = p_c^+(\mathcal{G}) = p_c^-(\mathcal{G})$ is the *dissemination threshold* of class $\mathcal{G}$. It is convenient to introduce the following terminology: throughout the paper, if $(A_n)_{n\in\mathbb{N}}$ is a sequence of events in a probability space such that $\lim_{n\to\infty} \mathbb{P}[A_n] = 1$, we write A_n a.a.s (asymptotically almost surely). For example, if $p < p_c^-(\mathcal{G})$ then a.a.s $G_n \in \mathcal{G}$ is not overrun by a p-insemination.

In this paper, we consider dissemination on regular graphs and particular classes of irregular graphs. First we consider regular graphs, for which we give simple lower bounds for the dissemination half-threshold p_c^-, and we prove that the threshold p_c is exactly $\frac{1}{2}$ for cubic graphs. In the second part, we give simple explicit constructions of graph classes with relatively small dissemination half-threshold $p_c^+(\mathcal{G})$. This counters the naive intuition that one should need about half of the vertices to overrun the whole graph.

Regular graphs. The dissemination process, as we have mentioned, has been studied for specific families of graphs, such as integer lattices, hypercubes, and so on, all of which are regular graphs. More generally, let $\mathcal{G}_r$ be the family of r-regular graphs. We observe that $p_c(\mathcal{G}_2) = 1$, since a p-insemination overruns a cycle if and only if there are no two consecutive passive vertices. A more interesting case is the class $\mathcal{Q}$ of hypercube graphs: these are regular graphs but with growing degrees. Following from more general results on families of regular graphs with growing degrees, Balogh, Bollobás and Morris [5] showed $p_c(\mathcal{Q}) = \frac{1}{2}$. Balogh and Pittel [6] considered dissemination on random r-regular graphs. Consider $G_{n,r}$, a graph chosen uniformly at random from the family of all r-regular graphs on n vertices, so $\mathcal{G}(r) = \{G_{n,r} : n \in \mathbb{N}\}$. It turns out that $p_c(\mathcal{G}(r))$ a.a.s. exists and equals

$$p_r := 1 - \inf_{y\in(0,1)} \frac{y}{F(r-1, 1-y)}$$

where $F(r, y)$ is the probability of obtaining at most $r/2$ successes in r independent trials with the success probability equal y. This leaves the determination of the dissemination threshold for $\mathcal{G}_r$ for fixed $r > 3$ as an open question. Let us have a look at the values of p_r for small r, though:

r	3	4	5	6	7
p_r	0.5	0.667	0.275	0.397	0.269

We conjecture that the dissemination thresholds $p_c(\mathcal{G}_r)$ exist and equal p_r. Towards this conjecture, we show the following modest result:

Theorem 1. *For all positive integers r, $p_c^-(\mathcal{G}_r) \le p_r$ and*

$$p_c^-(\mathcal{G}_r) \ge \begin{cases} \frac{1}{r} & \text{if } r \text{ is odd} \\ \frac{2}{r} & \text{if } r \text{ is even} \end{cases}$$

We will prove the conjecture in the case $r = 3$:

Theorem 2. $p_c(\mathcal{G}_3) = \frac{1}{2}$

Irregular graphs. It is natural to search for graph classes $\mathcal{G}$ for which $p_c^+(\mathcal{G})$ is small. If, as we conjecture, regular graphs behave like random regular graphs, then regular graphs cannot have very low thresholds. One should consider graphs whose vertices have varying degrees – we refer to these loosely as irregular graphs. To this end, we consider the class of wheels and toroidal graphs. Let C_n denote the cycle on n vertices and C_n^2 denote the toroidal grid on n^2 vertices. Notice that C_n^2 is, indeed, the cartesian square of C_n. In general, let C_n^k denote the k-dimensional torus. Let $u * C_n^k$ denote the k-dimensional torus augmented with a single universal vertex u. We will consider the class of wheels – i.e. the family $\mathcal{W} = \{u * C_n \mid n \in \mathbb{N}\}$ – and the class of toroidal grids plus a universal vertex – i.e. $\mathcal{T} = \{u * C_n^2 \mid n \in \mathbb{N}\}$. Our main result is that for both classes the dissemination threshold is small:

Theorem 3. *For the class $\mathcal{W}$, we have $p_c^+(\mathcal{W}) = 0.4030...$, where $0.4030...$ is the unique root in the interval $[0, 1]$ of the equation $p + p^2 - p^3 = \frac{1}{2}$. For the class $\mathcal{T}$ of toroidal grids plus a universal vertex, we have $0.35 \le p_c^+(\mathcal{T}) \le 0.372$.*

Since our goal is to find graph classes with small dissemination thresholds, clearly the second result is stronger than the first. Nevertheless, we shall present their proofs in parallel. For establishing the bounds on toroidal grids plus a universal vertex we need (a small amount of) computer-aided computations, while on wheels all computations are easy to check by hand.

The results of Balogh and Pittel on 7-regular graphs imply the existence of graph classes with half-threshold $p_c^+ < 0.27$. Although this bound is smaller than in our case, our result has the advantage of giving explicit constructions of graph classes with small half-threshold p_c^+. We also believe that our proof techniques might give new tools for constructing classes with even smaller values of p_c^+. Let us remark that computer simulations for higher dimension tori with a universal vertex $u * C_n^k$ indicate even lower thresholds. In simulations, a random p-insemination overruns $u * C_n^2$ the graph a.a.s. already with $p = 0.37$, which fits within the bounds shown in this paper. For k equal 3, 4 and 5 the graph $u * C_n^k$ is a.a.s. overrun by a random p-insemination already with p equal 0.35, 0.32 and 0.3, respectively. We leave the following as an open problem: Is there a family of graphs on which any p-insemination overruns the graph a.a.s for any $p > 0$?

2 Regular Graphs

In this section we outline the proof of Theorem 1. Balogh and Pittel [6] showed that for the class of random r-regular graphs, the dissemination threshold is a constant p_r a.a.s. where $p_3 = \frac{1}{2}$, $p_4 = \frac{2}{3}$ and so on. This establishes the upper bound in Theorem 1. For the lower bound, we use the following easy observation. The average degree of a graph G is $2e(G)/|V(G)|$.

Lemma 1. *Let G be a graph of average degree more than $2k - 2$, where $k \in \mathbb{N}$. Then G has a subgraph of minimum degree at least k.*

Proof. Let G be such a graph. We recursively remove vertices of degree at most $k - 1$. Each step this removes at most $k - 1$ edges, thus at the end of this process we must obtain a non-empty subgraph of G. This subgraph has the required property. $\square$

Let I be the set of active vertices of a p-insemination of $G \in \mathcal{G}_r$, and $I^c = V(G) \setminus I$. Then

$$\mathbb{E}[|I^c|] = (1-p)n \quad \text{and} \quad \mathbb{E}[e(I^c)] = \frac{r}{2}(1-p)^2 n$$

where $e(I^c)$ is the number of edges of G with both ends in I^c. Note that $|I^c|$ is a binomial random variable, in particular the Chernoff Bound [2] implies:

$$|I^c| \sim (1-p)n \quad \text{a.a.s.} \tag{2.1}$$

We also need to prove that

$$e(I^c) \sim \frac{r}{2}(1-p)^2 n \quad \text{a.a.s.} \tag{2.2}$$

This is proved using the Independent Bounded Differences (IBD) inequality (see [14]).

Theorem 4 ([14]). *Let $X = (X_1, X_2, \ldots, X_q)$ be a family of independent random variables with X_i taking values in a set A_i for each i. Suppose that the real-valued function f defined on ΠA_i satisfies*

$$|f(x) - f(x')| \leq c_i$$

whenever vectors x and x' only differ on the ith coordinate. Let μ be the expected value of $f(X)$. Then for any $t \geq 0$,

$$\mathbb{P}(|f(X) - \mu| \geq t) \leq 2e^{-2t^2/\sum c_i^2}.$$

Note that $e(I^c)$ can be considered as a function of the independent variables X_v, for all vertices v of the graph, where $X_v = 1$ if v is active in the initial configuration, and $X_v = 0$ if v is initially passive. By changing the value of only one variable X_v, we simply move vertex v from I to I^c or vice-versa. Thus the value of $e(I^c)$ changes by at most r since $G \in \mathcal{G}_r$. By applying Theorem 4 to $e(I^c)$, we obtain (2.2). If $p < 1/r$ for r odd and $p < 2/r$ for r even, by (2.1) and (2.2), we have

$$e(I^c) > \left(\left\lceil \frac{r}{2} \right\rceil - 1\right) |I^c| \quad \text{a.a.s.}$$

Lemma 1 with $k = \lceil r/2 \rceil$ implies that the graph $G[I^c]$ induced by I^c a.a.s has a subgraph of minimum degree at least $\lceil r/2 \rceil$, and so I^c a.a.s contains a community. This gives $\theta_p(G) \to 0$, as required.

3 Cubic Graphs

In this section, we prove Theorem 2, which determines the dissemination threshold for cubic graphs. We observe that a community in a cubic graph contains a cycle, and therefore the obstruction to a p-insemination overrunning a cubic graph is a cycle of passive vertices.

3.1 Random Cubic Graphs

In this section, we outline the proof of Theorem 2. To prove that $p_c(\mathcal{G}_3) \leq \frac{1}{2}$ we shall find a family of cubic graphs G such that $\theta_p(G) \to 1$ as $|V(G)| \to \infty$ for all $p > \frac{1}{2}$. Note that the existence of such a family is implied by the work of Balogh and Pittel [6]. Nevertheless, our proof is short, self-contained and can be easily turned into an explicit construction of such a family. This family of cubic graphs is generated by considering cubic graphs chosen at random from all cubic graphs, and then showing that such a random graph has the required properties. A survey of random regular graphs is found in Wormald [15]. The specific property we shall require of such graphs G is that the length of the shortest cycle in G tends to infinity as $|V(G)|$ tends to infinity, and G contains no more than 2^i cycles of length i for every $i \leq |V(G)|$. We call such graphs *cycle-sparse*. The following fundamental result on short cycles in random regular graphs was proved by Bollobás [3]:

Proposition 1. *Let X_i denote the number of cycles of length i in a random cubic graph on n vertices, for $i \leq n$. Then, for any fixed integer $g > 3$,*

$$\lim_{n\to\infty} \mathbb{P}[\forall i \leq g : X_i = 0] = \exp\left(-\sum_{i=1}^{g} i^{-1}2^{i-1}\right).$$

This result was recently extended to longer cycles in random cubic graphs by Garmo [9]. Omitting technical details, the results of Garmo show that for any $i \leq n$, $\mathbb{P}[X_i > 2^i] = O(i^{-2})$. Since the Euler sum converges, we deduce that with positive probability $X_i \leq 2^i$ for all i. A few more technical considerations show that we can ensure that with positive probability, $X_i = 0$ for $i \leq g$ and $X_i \leq 2^i$ for $i > g$, no matter what constant value of g we prescribe. It follows that there are infinitely many cycle-sparse cubic graphs.

To finish the proof that $p_c(\mathcal{G}_3) \leq \frac{1}{2}$, we fix $p > \frac{1}{2}$ and apply the Harris-Kleitman inequality [2]. For this inequality we consider the probability space $\mathbb{Q}_n$, whose underlying sample space is the n-dimensional Boolean lattice $\{0,1\}^n$ endowed with the natural product probability measure

$$\mathbb{P}(\omega) := \prod_{i=1}^{n} p^{\omega_i}(1-p)^{1-\omega_i} \quad \text{for } \omega \in \{0,1\}^n.$$

We may consider $\omega \in \{0,1\}^n$ as the incidence vector of a subset of $\{1,2,\ldots,n\}$. Taking this stance, a downset in $\mathbb{Q}_n$ is an event $A \subset \{0,1\}^n$ such that if $\omega \in A$ and $\omega' \subseteq \omega$, then $\omega' \in A$. An event is an upset if its complement is a downset.

Proposition 2. *Let $A_1, A_2, \ldots, A_r$ be downsets in $\mathbb{Q}_n$. Then*

$$\mathbb{P}[A_1 \cap A_2 \cap \cdots \cap A_r] \geq \prod_{i=1}^{r} \mathbb{P}[A_i].$$

The same holds if the events are all upsets.

In the current context, we take a p-insemination of a cycle-sparse n-vertex cubic graph G_n (seen as a $\{0,1\}^n$ vector), and observe that the events A_C that all vertices of a cycle $C \subset G_n$ are passive are downsets in Q_n. By the Harris-Kleitman inequality,

$$\mathbb{P}[\bigcap_{C \subset G_n} \overline{A}_C] \;\geq\; \prod_{C \subset G_n} \mathbb{P}[\overline{A}_C]$$

where the products and intersections are over all cycles $C \subset G$. Observe that $\overline{A}_C$ has probability $(1-(1-p)^{\ell})$ if C has length ℓ. Using the cycle-sparse property of G_n, we see

$$\prod_{C \subset G_n} \mathbb{P}[\overline{A}_C] \geq \prod_{i > g} (1-(1-p)^i)^{2^i}.$$

Since $p > \frac{1}{2}$, $1-(1-p)^i > e^{-2(1-p)^i}$. Consequently,

$$\prod_{C \subset G_n} \mathbb{P}[\overline{A}_C] > \exp\Big(2\sum_{i>g}(2(1-p))^i\Big) > \exp\Big(-\frac{2(2(1-p))^g}{1-2p}\Big).$$

We conclude that for any $p > \frac{1}{2}$ and any constant g,

$$\limsup_{n\to\infty} \theta_p(G_n) \leq 1 - \lim_{n\to\infty} \exp\Big(-\frac{2(2p)^g}{1-2p}\Big).$$

Since g was an arbitrary constant,

$$\lim_{n\to\infty} \theta_p(G_n) = 1$$

and this shows $p_c(\mathcal{G}_3) \leq \frac{1}{2}$.

3.2 $p_c(\mathcal{G}_3) \geq \frac{1}{2}$

With high probability, the existence of many short vertex-disjoint cycles in a cubic graph prevents a p-insemination from overrunning the graph. Therefore, to prove $p_c(\mathcal{G}_3) \geq \frac{1}{2}$, it is enough to consider cubic graphs which have very few short disjoint cycles – after some technical details, we may assume that we have an infinite sequence $\boldsymbol{G}$ where an n-vertex cubic graph G_n in $\boldsymbol{G}$ has no cycles of length at most $2g$ where $g = \frac{1}{8}\log n$. These details will be presented in the full version of the paper. We now outline the proof that for any $p < \frac{1}{2}$ and any increasing sequence of graphs G_n, $\theta_p(G_n) \to 0$ as $n \to \infty$.

Let $C_\lambda(G_n)$ denote the number of sets of λ vertices of G_n through which G_n contains a cycle of length λ – we shall call these cyclic sets. Note that, in general, $C_\lambda(G_n)$ is less than the number of cycles of length λ in G_n. The key idea in showing $\theta_p(G_n) \to 0$ is the following technical proposition:

Proposition 3. *For some λ satisfying $\lambda = \Theta(\log n)$,*

$$C_\lambda(G_n) = \Omega(\lambda^{-4} 2^{\lambda}).$$

An intuitive way to see this is via eigenvalues: the number of closed walks of length k in G_n is exactly $n\sum_{i=1}^{n}\lambda_i^k$, where λ_i is the ith largest eigenvalue of the adjacency matrix of G_n. Since G_n is cubic, $\lambda_1 = 3$. Now it is possible, although fairly detailed, to show by subtracting walks on trees, that about $\Omega(2^k/k)$ of these walks contain cycles provided k is a large enough constant times $\log n$. A similar computation is carried out in [13] (see Proposition 4.2). Putting $k = \lambda$, and using the girth condition, one arrives at the bound on $C_\lambda(G)$ in Proposition 3. We also observe that in a random cubic graph, the expected number of cycles of length λ is roughly $2^\lambda/\lambda$, so in the sense of counting cycles, G_n is close to a random cubic graph, and these were discussed in the last section. We consider the events A_X that all vertices in a cyclic set X of size λ are passive. The Harris-Kleitman Inequality – Proposition 2 – gives a lower bound on the probability that no A_X occurs, whereas we require an upper bound. The requisite inequality for such an upper bound is Janson's Inequality [16]:

Proposition 4. *Let $A_1, A_2, \dots, A_r$ be downsets in the probability space $\mathbb{Q}_n$, and define*

$$\triangle = \sum_{i\sim j}\mathbb{P}[A_i \cap A_j]$$

where $i \sim j$ means the events A_i and A_j are dependent and μ is the expected number of A_i which occur. Then

$$\mathbb{P}[\bigcap_{i=1}^{r}\overline{A}_i] \leq e^{-\mu^2/2\triangle}.$$

Showing $\theta_p(G_n) \to 0$ is equivalent to showing that some A_X occurs a.a.s., and we shall establish this with Janson's Inequality by showing that for the events A_X, $\mu^2/\triangle \to \infty$.

To prove this, note that from Proposition 3,

$$\mu = (1-p)^\lambda C_\lambda(G_n) = \Omega\Big(\frac{(2-2p)^\lambda}{\lambda^4}\Big).$$

It is trickier to estimate $\triangle$, and this relies heavily on the assumption that G_n has no cycles of length at most $2g$. To estimate $\triangle$, we fix a cyclic set X and ask, for each $i \in \mathbb{N}$, for the number $\triangle_i(X)$ of cycles C of length λ for which $|X \cap V(C)| = i$. It turns out that

$$\triangle_i(X) = \lambda^{O(1)}2^{\lambda-i-g} \qquad \text{for } 1 \leq i < \lambda - g$$

and $\triangle_i(X) = 0$ otherwise. This allows us to estimate $\triangle$:

$$\begin{aligned}\triangle &\leq C_\lambda(G_n) \sum_{i=1}^{\lambda-g-1} (1-p)^{2\lambda-i} \triangle_i(G_n) \\ &= O(\mu^2) \cdot \lambda^{O(1)} \sum_{i=1}^{\lambda-g-1} (1-p)^{-i} 2^{-i-g} \\ &= O(\mu^2) \lambda^{O(1)} 2^{-g}.\end{aligned}$$

Here we used the fact that $p < \frac{1}{2}$. By the choice of g, $\lambda^{O(1)} 2^{-g} \to 0$, and we are done: $\triangle/\mu^2 \to 0$. In words, some λ-cycle is passive a.a.s by Janson's Inequality, and therefore $\theta_p(G_n) \to 0$.

4 Wheels and Toroidal Grids

We prove here Theorem 3: wheels and toroidal grids plus a universal vertex u have (relatively) small dissemination half-thresholds p_c^+. One of the main observations is that, for any probability $p > 0$, if the universal vertex becomes active during the dissemination process, then the graph is overrun a.a.s. Thus, for any value p such that p-inseminations contaminate a.a.s. more than half of the vertices of the cycle or of the toroidal grid, we deduce that the whole graph is overrun.

There has been much research on dissemination on the k-dimensional torus and grid graphs. The considered rules were the l-neighbours rule, which are more general than the majority rule: in this setting a vertex becomes active if at least l of its neighbours already are active. In particular, Aizenman and Lebowitz [1] studied the 2-neighbours dissemination on P_n^2 and their results extend to C_n^2. Notice that the majority dissemination on C_n^2 is the 3-neighbours dissemination, since C_n^2 is a four-regular graph.

Our approach is based on the observation that once the universal vertex u becomes active, the majority dissemination in the C_n^k part of $u * C_n^k$, in fact, follows the *weak majority* rule restricted to C_n^k. In the weak majority rule a vertex becomes active if at least half of its neighbours are active. If the p-insemination of $u * C_n^k$ is such that half plus one vertex of C_n^k become active, then u becomes active as well. Moreover, for any $p > 0$, the weak majority rule dissemination process for C_n^k will almost surely overrun the whole graph (the result is trivial for cycles, and due to Aizenman and Lebowitz for toroidal grids):

Lemma 2 (see [1]). *Let $O_p^w(G)$ be the random event that a p-insemination overruns G under the* weak majority *rule, and let us denote $o_p^w(G)$ the corresponding probability. Then for any $p > 0$ and any $k \in \{1, 2\}$,*

$$\lim_{n\to\infty} o_p^w(C_n^k) = 1$$

Therefore, for any probability $p > 0$ on graphs of type $u * C_n^k$, if the dissemination contaminates the vertex u it will almost surely overrun the whole graph.

Lemma 3. *Denote by $F_p(G)$ the number of active vertices obtained by the p-dissemination process on G. For every class of graphs $\mathcal{G}$ of type $u * C_n^k$, $p_c^+(\mathcal{G}) =$ $\inf\{p \in [0,1]$ over all values p such that there exists an increasing sequence $u * C_{n_i}^k$ satisfying* $\lim_{i\to\infty} \mathbb{P}(F_p(C_{n_i}^k) > n_i^k/2) = 1$.

From now on we only consider the p-dissemination process in cycles and toroidal grids, under the strong majority rule. Recall that $F_p(G)$ is the random variable counting the number of active vertices in the final state, after a p-dissemination process in G. We give upper and lower bounds for the expected value of F_p for cycles and toroidal grids. Moreover, we shall see that, with very high probability, the value of $F_p(C_n^k)$ is very close to its expectation, when $n \to \infty$. Therefore, it is sufficient to see for which values of p this quantity $\mathbb{E}(F_p(C_n^k))$ is strictly bigger than $n^k/2$, and for which values it is strictly smaller than $n^k/2$. According to Lemma 3, the dissemination threshold for the class $u * C_n^k$ lies between the two values.

Since we are unable to give an exact formula for $F_p(C_n^k)$, we give upper and lower bounds for this quantity. Consider a window $D^d(v)$ formed by all vertices at distance at most d from v in C_n^k. Let $S_p^d(v)$ be a random variable equal to 1 if v becomes active when we replace, in the original p-insemination, all vertices outside the window $D^d(v)$ by passive vertices, and equal to 0 otherwise. Let $s_p^d(C_n^k)$ be the probability that $S_p^d(v) = 1$ (by symmetry this probability is the same for all vertices). Dually, let $W_p^d(v) = 1$ if v becomes active when, in the initial p-insemination, all vertices outside $D^d(v)$ are transformed into active vertices, and $W_p^d(v) = 0$ otherwise. The probability that $W_p^d(v) = 1$ is denoted $w_p^d(C_n^k)$. Finally, let $S_p^d(G) = \sum_v S_p^d(v)$ and $W_p^d(G) = \sum_v W_p^d(v)$[1].

Clearly, we have

Lemma 4. *For any constant d and any $k \geq 1$,*

$$S_p^d(C_n^k) \leq F_p(C_n^k) \leq W_p^d(C_n^k)$$

For any fixed values of k and d, the probabilities $s_p^d(C_n^k)$ and $w_p^d(C_n^k)$ can be expressed as polynomials on p.

Lemma 5

1. *For any $n \geq 3$,*

$$s_p^1(C_n) = w_p^1(C_n) = p + p^2 - p^3.$$

[1] In the case of cycles, it is easy to see that the dissemination process stops in exactly one step: a passive vertex becomes active iff both neighbours are active, therefore $S_p^d(C_n) = F_p(C_n) = W_p^d(C_n)$ for any $n \geq 3$ and any $d \geq 1$.

2. *For any $n \geq 5$, $s_p^3(C_n^2)$ and $w_p^3(C_n^2)$ are polynomials of degree 25 on p. Their exact formula has been computed by a program.*

Proof. Let us prove the first part of the lemma. Let v be a vertex of the cycle and assume that all vertices at distance at least 2 from v are passive. Then v will be active if and only if initially v is already active (which occurs with probability p) or initially v is passive and both his neighbours are active (which occures with probability $(1-p)p^2$. Therefore the probability that u becomes active is $p + p^2 - p^3 = s_p^1$. Now if we configure all non-neighbours of v to be active, the situation is exactly the same: v will be active iff it was active since the begining, or if it was initially passive and both neighbours were active.

For the second part of the proof, the polynomials corresponding to s_p^3 and w_p^3 have been computed by a program. The program considers the window $D^3(v)$ formed by the 25 vertices of distance at most 3 from vertex v in C_n^2. For each number i, with $0 \leq i \leq 25$, we count the number of configurations with exactly i active vertices and such that v belongs to a passive community. (We consider both settings, when vertices outside the window are all active, respectively all passive.) We find e.g. 1 community with 0 active vertices, 24 communities with one active vertex, 276 communities with 2 active vertices, etc. The probability of such a configuration being $p^i(1-p)^{25-i}$, we obtain the required polynomials. $\square$

The expectation of the variable $S_p^d(C_n^k)$ (respectively $W_p^d(C_n^k)$) is $n^k s_p^d(C_n^k)$ (respectively $n^k s_p^d(C_n^k)$). Moreover, we have:

$$S_p^d(C_n^k) \sim n^k s_p^d(C_n^k) \quad \text{and} \quad W_p^d(C_n^k) \sim n^k w_p^d(C_n^k) \quad \text{a.a.s.} \tag{4.1}$$

For proving that the two quantities are very close to their expectations we use again the Independent Bounded Differences inequality (Theorem 4). Consider $S_p^d(C_n^k)$ and $W_p^d(C_n^k)$ as real functions on all possible initial configurations of C_n^k (so their domain is $\{0,1\}^{n^k}$). For each vertex v of C_n^k, let X_v be the random variable s.t. $X_v = 1$ if v is active in the initial configuration, and $X_v = 0$ if v is initially passive. Clearly the variables X_v are independent. Recall that $S_p^d(C_n^k) = \sum_w S_p^d(w)$, where $S_p^d(w)$ is the boolean random variable corresponding to the event "vertex w becomes active if we replace, in the original p-insemination, all vertices at distance larger that d from w by passive vertices". If in the initial configuration we only change the value of one vertex v, this only changes the values $S_p^d(w)$ for vertices w at distance at most d from v. Hence the value of $S_p^d(C_n^k)$ is modified by at most a constant value. By similar arguments, the value of $W_p^d(C_n^k)$ also changes by at most a constant. Therefore we can apply Theorem 4 to both functions, and deduce Equation 4.1.

We are now able to prove our Theorem 3. Consider the case of wheels. For any $p > 0.4030...$, we have $s_p^1(C_n) = p + p^2 - p^3 > 1/2$. By Lemma 4 and Equation 4.1, we have that $F_p(C_n) > n/2$ a.a.s. Therefore $p_c^+(\mathcal{W}) \leq p$, for any $p > 0.4030....$ by Lemma 3. Symmetricaly, for any $p < 0.4030...$, $w_p^1(C_n) < 1/2$

and thus $F_p(C_n) < n/2$ a.a.s. We deduce by Lemma 3 that $p_c^+(\mathcal{W}) \geq 0.4030...$, which proves the first part of Theorem 3.

The same kind of arguments can be applied to toroidal grids plus one vertex. For any $p \geq 0.372$ (resp. any $p \leq 0.35$), the polynomial $s_p^3(C_n^2)$ (resp. $w_p^3(C_n^2)$, see Lemma 5) has value strictly greater (resp. smaller) than $1/2$. We conclude by Lemma 3 that $0.35 \leq p_c^+(\mathcal{T}) \leq 0.372$.

References

1. Aizenman, A., Lebowitz, J.: Metastability effects in bootstrap percolation. J. Phys. A: Math. Gen. 21, 3801–3813 (1988)
2. Alon, N., Spencer, J.: The Probabilistic Method, 2nd edn. Wiley, Chichester (1992–2000)
3. Bollobás, B.: Random graphs, 2nd edn. Academic Press, Cambridge University Press (1985–2001)
4. Balogh, J., Bollobás, B.: Bootstrap percolation on the hypercube. Probab. Theory Related Fields 134(4), 624–648 (2006)
5. Balogh, J., Bollobás, B., Morris, J.: Majority bootstrap percolation on the hypercube. (manuscript, 2007)
6. Balogh, J., Pittel, B.: Bootstrap percolation on the random regular graph. Random Structures Algorithms 30(1-2), 257–286 (2007)
7. Bollobás, B., Szemerédi, E.: Girth of sparse graphs. J. Graph Theory 39(3), 194–200 (2002)
8. Carvajal, B., Matamala, M., Rapaport, I., Schabanel, N.: Small alliances in graphs. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 218–227. Springer, Heidelberg (2007)
9. Garmo, H.: The asymptotic distribution of long cycles in random regular graphs. Random Struct. Algorithms 15(1), 43–92 (1999)
10. Haynes, T.W., Hedetniemi, S.T., Henning, M.A.: Global deffensive alliances in graphs. Electronic J. Comb. 10, 139–146 (2003)
11. Holroyd, A.: Sharp Metastability Threshold for Two-Dimensional Bootstrap Percolation. Probability Theory and Related Fields 125(2), 195–224 (2003)
12. Kempe, D., Kleinberg, J., Tardos, E.: Maximizing the Spread of Influence through a Social Network. In: Proceedings of KDD 2003, pp. 137–146 (2003)
13. Lubotsky, A., Phillips, R., Sarnak, R.: Ramanujan graphs. Combinatorica 8, 261–278 (1988)
14. Habib, M., McDiarmid, C., Ramirez-Alfonsin, J., Reed, B.: Probabilistic Methods for Algorithmic Discrete Mathematics. Series: Algorithms and Combinatorics, vol. 16. Springer, Heidelberg (1998)
15. Wormald, N.: Models of random regular graphs. In: Lamb, J.D., Preece, D.A. (eds.) Surveys in Combinatorics. London Mathematical Society Lecture Note Series, vol. 276, pp. 239–298. Cambridge University Press, Cambridge (1999)
16. Janson, S., Łuczak, T., Rucinski, A.: Random graphs. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley-Interscience, New York (2000)
17. Peleg, D.: Local majorities, coalitions and monopolies in graphs: a review. Theoretical Computer Science 282, 231–257 (2002)

How to Complete a Doubling Metric

Anupam Gupta[1,⋆] and Kunal Talwar[2]

[1] Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213
[2] Microsoft Research, Silicon Valley Campus, Mountain View, CA 94043

Abstract. In recent years, considerable advances have been made in the study of properties of metric spaces in terms of their doubling dimension. This line of research has not only enhanced our understanding of finite metrics, but has also resulted in many algorithmic applications. However, we still do not understand the interaction between various graph-theoretic (topological) properties of graphs, and the doubling (geometric) properties of the shortest-path metrics induced by them. For instance, the following natural question suggests itself: *given a finite doubling metric (V, d), is there always an* $\underline{\textit{unweighted}}$ *graph (V', E') with $V \subseteq V'$ such that the shortest path metric* $\overline{d' \text{ on } V' \text{ is}}$ *still doubling, and which agrees with d on V.* This is often useful, given that unweighted graphs are often easier to reason about.

A first hurdle to answering this question is that subdividing edges can increase the doubling dimension unboundedly, and it is not difficult to show that the answer to the above question is negative. However, surprisingly, allowing a $(1 + \varepsilon)$ distortion between d and d' enables us bypass this impossibility: we show that for any metric space (V, d), there is an *unweighted* graph (V', E') with shortest-path metric $d' : V' \times V' \to \mathbb{R}_{\geq 0}$ such that

– for all $x, y \in V$, the distances $d(x, y) \leq d'(x, y) \leq (1 + \varepsilon) \cdot d(x, y)$, and
– the doubling dimension for d' is not much more than that of d, where this change depends only on ε and not on the size of the graph.

We show a similar result when both (V, d) and (V', E') are restricted to be trees: this gives a simple proof that doubling trees embed into constant dimensional Euclidean space with constant distortion. We also show that our results are tight in terms of the tradeoff between distortion and dimension blowup.

1 Introduction

The algorithmic study of finite metrics has become a central theme in theoretical computer science in recent years. Of particular interest has been the study of the geometry of metrics—embeddings into Minkowski spaces have been the most obvious example, accompanied by the study of notions of metric dimension which

⋆ This research was partly supported by the NSF CAREER award CCF-0448095, and by an Alfred P. Sloan Fellowship.

have allowed us to partially quantify geometric properties that make metrics tractable for several algorithmic problems.

Given these advances in our understanding of the geometric properties of abstract metric spaces, it is worth remarking that our comprehension of the *topological* properties of metric spaces—and of the relationship between topology and geometry has lagged behind: we do not yet have a good comprehension of how the structure of a graph interacts with the dimensionality of the shortest-path metric induced by it. One such example shows up in a paper [8], where a fairly simple algorithm is given for low-distortion Euclidean embeddings of unweighted trees whose shortest-path metric is doubling—however, extending the result to embed *weighted* trees (also with doubling shortest-path metrics) requires significantly more work. This raises the natural question: *given a doubling tree metric $M = (V, d)$, is there an unweighted tree $G = (V', E')$ whose shortest-path metric is also doubling, and contains M as a submetric?* In fact, the situation is even more embarrassing: we do not know the answer even if we drop the requirement that G be a tree, and look for any unweighted graph!

An immediate obstacle to answering these question is the observation that subdividing the edges of a weighted tree to convert it into an unweighted tree can increase the dimension unboundedly. For example, take a star $K_{1,n}$, and set the length of the i^{th} edge $\{v_0, v_i\}$ to be 2^i. It is easy to check that the metric d_G has constant doubling dimension; however, subdividing the i^{th} edge into 2^i parts to make it unit-weighted creates a new graph with n points at unit distance from each other, which has a doubling dimension $\log n$ that is unbounded. On the positive side, it is easy to show that this metric can be embedded into the real line with distortion 2 (e.g., the map $v_i \mapsto 2^i$), which we can subdivide without altering the doubling dimension. In this paper, we show that this positive result is not an aberration: any tree metric can be represented as a submetric of an unweighted tree metric which has almost the same doubling dimension. We show a similar result for arbitrary graphs as well, and show that our tradeoff between distortion and the dimension blowup is asymptotically optimal.

Formal Definitions: To define the problems we study, let us define the *convex closure* of a graph, which is an extension of the notion of subdividing edges. Given a graph $G = (V, E)$ with edge lengths $\ell : E \to \mathbb{R}_{\geq 0}$, assume that the names of the vertices in V belong to some total order $(V, \prec)$. Let $\overline{V}_G$ be the uncountably infinite set of points $V \cup \{e[x] \mid e \in E, x \in (0, \ell(e))\}$ obtained by considering each edge as a continuous segment of length $\ell(e)$. Let $M_G = (V, d_G)$ be the shortest-path metric of the graph G: we can define a natural metric on the set $\overline{V}_G$ as

$$\begin{aligned}\overline{d}_G(e[x], e'[y]) = \min\{&x + d(u, u') + y, \quad x + d(u, v') + (\ell(e') - y), \\ &(\ell(e) - x) + d(u', v) + y, \quad (\ell(e) - x) + d(u', v') + (\ell(e') - y)\},\end{aligned}$$

if $e = \{u, v\}$ (with $u \prec v$) and $e' = \{u', v'\}$ (with $u' \prec v'$). We now define the *convex closure* of the graph G to be the metric space $\mathsf{conv}(G) \doteq \overline{M}_G = (\overline{V}_G, \overline{d}_G)$. Note the metric obtained by subdividing edges of G is a sub-metric of the convex closure of G, and hence it suffices to study the doubling dimension of this convex closure $\mathsf{conv}(G)$.

1.1 Our Results

The example of $K_{1,n}$ with exponential edge weights shows that even if the shortest-path metric M_G of a graph G is doubling, its convex closure $\overline{M}_G$ may not be doubling. The goal of this paper is to show that despite this, there is a "close-by" graph G' whose convex closure $\overline{M}_{G'}$ is indeed doubling. In particular, the main theorem is the following:

Theorem 1 (Main Theorem). *Given a graph $G = (V, E)$ with specified edge-lengths, we can efficiently find a graph $G' = (V, E')$ (also with non-negative edge-lengths) such that*
- *The distances in G and G' are within a multiplicative factor of $(1 + \varepsilon)$ of each other, and*
- *If $\dim(M_G) = k$, then $\dim(M_{G'}) = O(k)$, and $\dim(\mathsf{conv}(G')) = O(k \log \varepsilon^{-1})$.*

Since Theorem 1 does not give any guarantees about the topology of the graph G', we prove an analogous result about tree metrics, with improved guarantees on the dimension:

Theorem 2. *Given a tree $T = (V, E)$ with specified edge-lengths, we can efficiently find a tree $T' = (V', E')$ with $V \subseteq V'$ (and with non-negative edge-lengths) such that*
- *For $x, y \in V$, the distance between them in T and T' are within a multiplicative factor of $(1 + \varepsilon)$ of each other, and*
- *If $\dim(M_T) = k$, then $\dim(M_{T'}) = O(k)$, and $\dim(\mathsf{conv}(T')) = O(k + \log \log \varepsilon^{-1})$.*

As a corollary of this result, we obtain an independent proof of the following result about embeddings of doubling tree metrics into ℓ_p spaces:

Corollary 3 ([8]). *Every (weighted) doubling tree metric embeds into ℓ_p with constant distortion and constant dimension.*

(Another proof of this embedding result for doubling trees appears in [21], using completely different techniques.)

In addition, we show that the tradeoff between the distortion and the dimension of the convex closure shown in Theorem 2 is asymptotically optimal:

Theorem 4. *There exists a tree metric $T = (V, E)$ with $\dim(M_T) = O(1)$ such that for any tree metric $T' = (V', E')$ with $V \subseteq V'$, the following holds. If $d_T(u, v) \le d_{T'}(u, v) \le (1 + \varepsilon) d_T(u, v)$ for all $u, v \in V$, then $\dim(\mathsf{conv}(T'))$ must be $\Omega(\log \log \varepsilon^{-1})$.*

For general graphs, we show that our tradeoff is asymptotically optimal, under the restriction that the graph G' is defined on the same vertex set as G, i.e. we do not use any steiner points.

Theorem 5. *There exists a metric $G = (V, E)$ with $\dim(M_G) = O(1)$ such that for any graph $G' = (V, E')$, the following holds. If $d_G(u, v) \le d_{G'}(u, v) \le (1 + \varepsilon) d_G(u, v)$ for all $u, v \in V$, then $\dim(\mathsf{conv}(G'))$ must be $\Omega(\log \varepsilon^{-1})$.*

Proofs of Theorems 4 and 5 are omitted from this extended abstract and can be found in the full version [10].

1.2 Related Work

The notion of doubling dimension was introduced by Assouad [1] and first used in algorithm design by Clarkson [5]. The properties of doubling metrics and their algorithmic applications have since been studied extensively, a few examples of which appear in [8,18,19,25,11,2,6,12,16,17].

Somewhat similar in spirit to our work is the 0-extension problem [14,3,7]. Given a graph G, the 0-extension (*cf.* Lipschitz Extendability [13,23,20]) problem deals with extending a (Euclidean) embedding of the vertices of the graph to an embedding of the convex closure of the graph, while approximately preserving the Lipschitz constant of the embedding. Our results can be interpreted as analogues to the above where the goal is to approximately preserve the doubling dimension.

A number of papers have dealt with geometric implications of topological properties of the graph inducing the metric, e.g. when the graph is planar [15,24], outer-planar [9], series-parallel [9], or a tree [22].

2 Preliminaries and Notation

Given a graph G, the shortest path metric on it is denoted by d_G and we shall use $B_G(x,r)$ to denote the "ball" $\{y \in V_G : d_G(x,y) < r\}$. We will often omit the subscript G when it is obvious from context. There are several ways of defining the doubling constant λ and the doubling dimension dim for a metric space, all of them within a constant factor of each other: here is the one that will be most useful for us.

Definition 6 (Doubling Constant and Doubling Dimension). *A metric space* (X,d) *has* doubling constant λ *if for each* $x \in X$ *and* $r \geq 0$*, given the ball* $B(x,2r)$*, there is a set* $S \subseteq X$ *of size at most* λ *such that* $B(x,2r) \subseteq \cup_{y\in S} B(y,r)$*. The* doubling dimension $\dim((X,d)) = \log_2 \lambda$*.*

Fact 7 (Subset Closed). *Let metric* $M = (V,d)$ *have doubling dimension* k*. If* $X' \subseteq X$*, and* $d' = d|_{X'\times X'}$*, then* (X',d') *has doubling dimension at most* k*.*

Fact 8 (Small Uniform Metrics). *If a metric* $M = (V,d)$ *has doubling dimension* k *then there exists a point* x *and a radius* r *such that the ball* $B(x,r)$ *contains at least* 2^k *points with interpoint distances at least* $r/2$*.*

Given a metric (X,d), an r*-packing* is a subset $P \subseteq X$ such that any two points in P are at least distance r from each other. An r*-covering* is a subset $C \subseteq X$ such that for each point $x \in X$, there is a point $c \in C$ at distance $d(x,c) \leq r$. An r*-net* is a subset $N \subseteq X$ that is both an r-packing and an r-covering.

Fact 9 ("Small" Nets). *Let metric* $M = (V,d)$ *have doubling dimension* k*, and* N *is an* r*-net of* M*, then for any* $x \in V$ *and radius* R*, the set* $B(x,R) \cap N$ *has size at most* $(4R/r)^k$*.*

3 A Structure Theorem

In this section, we show how to characterize the dimension of the convex closure of a graph H in terms of some easier-to-handle parameters of the graph.

Definition 10 (Long Edges). *Given a graph $H = (V, E)$, a vertex $u \in V$ and a radius $r \geq 0$, call an edge $e = \{v, w\}$ a* long edge *with respect to u, r if one endpoint of e is at distance at most r from u, and $l(e) > r$.*

Let the set of long edges with respect to u, r be denoted by $L_u(r)$. The following structure theorem gives us a characterization of the doubling dimension of $\mathsf{conv}(H)$ in terms of the number of long edges. The proof is omitted from this extended abstract and can be found in the full version [10].

Theorem 11 (Structure Theorem). *There exist constants c_1 and c_2 such that the following holds. Consider any graph $H = (V, E)$, and any $k \geq \dim_H$: if the number of long edges $|L_u(r)| \leq 2^k$ for every $u \in V$ and every $r \geq 0$, then the doubling dimension of the convex closure $\mathsf{conv}(H)$ is at most $c_1 k$. Moreover, if the doubling dimension of the convex closure $\mathsf{conv}(H)$ is at most k, then for every vertex $u \in V$ and every radius $r \geq 0$, the number of long edges $|L_u(r)| \leq 2^{c_2 k}$.*

4 Convex Completions for Graphs

In this section, we show how to take a graph $G = (V, E)$ and obtain a graph $G' = (V, E')$ on the same vertex set, which has (almost) the same distances as in G, but whose doubling dimension does not change by much under taking the convex closure. In particular, we use a bounded-degree spanner construction due to Chan et al. [4]: they give an algorithm that given a metric (V, d) with dimension $\dim = \dim(G)$ and a parameter $\varepsilon < 1/4$, outputs a spanner $G' = (V, E')$ such that $d(x, y) \leq d_{G'}(x, y) \leq (1 + \varepsilon)\, d(x, y)$ for all pairs $x, y \in V$, and moreover the degree of each vertex $x \in V$ is bounded by $\varepsilon^{-O(\dim_G)}$. We show that the convex closure of this spanner has doubling dimension of $O(\dim_G \log \varepsilon^{-1})$.

4.1 The Spanner Construction

We start with a graph G and carry out a series of transformations to obtain graph G'. Let $\varepsilon < \frac{1}{4}$ be given and let $\tau = 6 + \lceil \log(\frac{1}{\varepsilon}) \rceil$. Without loss of generality, the smallest pairwise distance in G is at least 2^τ. We start with some more definitions.

Definition 12 (Hierarchical Tree). *A hierarchical tree for a set V is a pair $(\mathbb{T}, \phi)$, where $\mathbb{T}$ is a rooted tree, and ϕ is a labeling function $\phi : \mathbb{T} \to V$ that labels each node of $\mathbb{T}$ with an element in V, such that the following conditions hold.*

1. *Every leaf is at the same depth from the root.*
2. *The function ϕ restricted to the leaves of $\mathbb{T}$ is a bijection into V.*

3. *If u is an internal node of $\mathbb{T}$, then there exists a child v of u such that $\phi(v) = \phi(u)$. This implies that the nodes mapped by ϕ to any $x \in V$ form a connected subtree of $\mathbb{T}$.*

Definition 13 (Net-Tree). *A* net tree *for a metric (V, d) is a hierarchical tree $(\mathbb{T}, \phi)$ for the set V such that the following conditions hold.*

1. *Let N_i be the set of nodes of $\mathbb{T}$ that have height i. (The leaves have height 0.) Let $r_0 = 1$, and $r_{i+1} = 2r_i$, for $i \geq 0$. (Hence, $r_i = 2^i$.) Then, for $i \geq 0$, $\phi(N_{i+1})$ is an r_{i+1}-net for $\phi(N_i)$.*
2. *Let node $u \in N_i$, and its parent node be p_u. Then, $d(\phi(u), \phi(p_u)) \leq r_{i+1}$.*

It is easy to see that net-trees exist for all metrics, and Har-Peled and Mendel show how to construct a net-tree efficiently [11].

To construct their bounded-degree spanner, Chan et al. [4] define the following: suppose we are given a graph $G = (V, E)$, whose shortest-path metric (V, d_G) has doubling dimension $\dim_G$. Let $\varepsilon > 0$ and $(\mathbb{T}, \phi)$ be any net tree for M. For each $i > 0$, let

$$E_i := \left\{ \{u, v\} \mid u, v \in \phi(N_i), d_G(u, v) \leq (4 + \frac{32}{\varepsilon}) \cdot r_i \right\} \setminus \bigcup_{j \leq i-1} E_j, \qquad (4.1)$$

where E_0 is the empty set. (Here the parameters N_i, r_i are as in Definition 13.) Letting C_ε denote $(4 + \frac{32}{\varepsilon})$, we note that all edges in E_i have length in $(C_\varepsilon r_{i-1}, C_\varepsilon r_i]$.

While the graph $\widehat{G} = (V, \widehat{E} = \cup_i E_i)$ is a $(1+\varepsilon)$-spanner for the original metric with few edges, obtaining a bounded-degree spanner requires some modifications to the basic construction. First, the edges in $\widehat{E}$ are directed (merely for the purposes of the algorithm, and the proof). For each $v \in V$, define $i^*(v) := \max\{i | v \in \phi(N_i)\}$. For each edge $(u, v) \in \widehat{E}$, we direct the edge from u to v if $i^*(u) < i^*(v)$. If $i^*(u) = i^*(v)$, the edge can be directed arbitrarily. Chan et al. show that each vertex $x \in V$ has *out-degree* bounded by $\beta = \varepsilon^{-O(\dim_G)}$. Then, the following steps are performed:

– Consider any vertex x, and all the edges that are directed *into* x. These edges come from various sets E_i: let us denote by $F_i = F_i(x)$ the subset of edges directed into x that belong to E_i.
– Suppose the non-empty subsets are $F_{i_1}, F_{i_2}, \ldots, F_{i_t}$, where $i_j < i_{j+1}$. We do nothing to the first $7 \log \varepsilon^{-1}$ of these edge sets; these contribute $\varepsilon^{-O(\dim_G)}$ to the final degree of x.
– Consider a value of $j > 7 \log \varepsilon^{-1}$: from the set $F_{i_{(j-7\log\varepsilon^{-1})}}$ of edges directed *into* x, we choose an arbitrary one $\{u, x\}$. We replace edges of the form $\{y, x\} \in F_{i_j}$ by edges $\{y, u\}$—and refer to these (at most $\varepsilon^{-O(\dim_G)}$) edges as edges *donated* from x to u.

  Note that the length of the edge $\{u, x\}$ is at most $C_\varepsilon\, 2^{i_{(j-7\log\varepsilon^{-1})}} \leq C_\varepsilon\, \varepsilon^7 2^i$, whereas the length of any edge in $\{y, x\} \in F_{i_j}$ is at least $C_\varepsilon\, 2^{i-1}$; hence $d_G(u, x) \leq (\varepsilon^7/2) d_G(x, y) \leq \varepsilon^6 d_G(x, y)$, since $\varepsilon \leq 1/4$. By the triangle inequality, $d_G(u, y) \in (1 \pm \varepsilon^6) d_G(x, y)$.

Additionally, note that if x donates a long edge $(x, y) \in F_{i_j}$ to u, then $(u, x) \in F_{i_{j-7\log\varepsilon^{-1}}}$ so that $d_G(x, u)$ is at least $C_\varepsilon 2^{(i_{j-7\log\varepsilon^{-1}})-1}$.

Theorem 14 ([4]). *The spanner thus constructed has degree* $\varepsilon^{-O(\dim_G)}$ *and stretch* $(1 + \varepsilon)$*.*

From the construction of the bounded-degree spanner, note that each vertex $u \in V$ has the following edges incident to it:

- **Type-A edges.** These correspond to the $\varepsilon^{-O(\dim_G)}$ edges that were directed *away* from u.
- **Type-B edges.** These correspond to the edges directed *into* u that belong to the smallest $7\log\varepsilon^{-1}$ levels; this gives another $(\varepsilon^{-O(\dim_G)})$ edges in total.
- **Type-C edges.** For each edge $e = \{u, x\}$ of type-A incident to u, there are at most $(\varepsilon^{-O(\dim_G)})$ other edges incident to u that are not counted above. Each such edge $e' = \{y, u\}$ corresponds to some edge of the form $\{y, x\} \in E_i$ (for some i such that $x, y \in \phi(N_i)$), such that the edge was "donated" from x to u to maintain x's degree bound.

4.2 Bounding the Dimension of the Convex Closure

Simply by the distortion bound, it follows that the doubling dimension of the bounded-degree spanner G' is close to $\dim_G$. Of course, the bounded-degree does not imply that $\mathsf{conv}(G')$ has low doubling dimension: in this section, we use the Structure Theorem 11 to show this fact, and hence prove Theorem 1.

Lemma 15. *Given the graph G' defined as above, fix any vertex v and radius R, and $\varepsilon < \frac{1}{4}$. Then the number of long edges $|L_v(R)|$ with respect to v, R is at most $O(\varepsilon^{-O(\dim_G)})$.*

Proof. Recall that $L_v(R)$ is the set of edges that have one endpoint within the ball $B(v, R)$, and have length at least R. Define $\ell \in \mathbb{Z}_{\geq 0}$ such that $R \in (C_\varepsilon 2^{\ell-1}, C_\varepsilon 2^\ell]$.

By the spanner construction, any type-A or type-B edge that is long must belong to $\cup_{i\geq\ell} E_i$, and hence must have both endpoints in $\phi(N_\ell)$. Moreover, one endpoint of each such a long edge must lie in the ball $B(v, R) \subseteq B(v, C_\varepsilon 2^\ell)$; since the points in $\phi(N_\ell)$ are at distance at least 2^ℓ from each other, there can be at most $(C_\varepsilon)^{O(\dim_G)}$ many such endpoints within the ball. Moreover, each one of these endpoints has at most $\varepsilon^{-O(\dim_G)}$ type-A or type-B edges; multiplying them together, using the fact that $C_\varepsilon = O(\varepsilon^{-1})$, and simplifying gives an upper bound of $\varepsilon^{-O(\dim_G)}$ on the number of type-A and type-B edges in $L_v(R)$.

Let us now consider the edges in $L_v(R)$ that are of type-C with respect to their endpoint within $B(v, R)$. Recall that each type-C edge $\{u, y\}$ can be associated with some edge $\{x, y\} \in \widehat{E}$ (of almost the same length—up to a factor of $(1 \pm \varepsilon^6)$) such that x donates the edge to u. Let us fix one such long edge $e = \{u, y\}$ associated with $\{x, y\} \in E_i$—hence the distance $d_G(x, y) \in (C_\varepsilon 2^{i-1}, C_\varepsilon 2^i]$, and also $x, y \in \phi(N_i)$. By the construction of the type-C edges, the distance $d_G(u, x) \leq \varepsilon^6 \cdot d_G(x, y)$, and hence x lies in the ball $B(v, R + \varepsilon^6 C_\varepsilon 2^i)$.

Given any fixed level $i \geq \ell - 1$, the number of donor vertices is bounded by the number of points in $B(v, C_\varepsilon(2^\ell + \varepsilon^6 2^i))$ that are at least 2^i distance apart from each other, which can be loosely bounded by $\varepsilon^{-O(\dim_G)}$. Each such donor vertex could donate $\varepsilon^{-O(\dim_G)}$ edges, which would give us a total of $\varepsilon^{-O(\dim_G)}$ edges for the level i. Summing this over all levels would give us too many edges, so we use this bound only for levels i such that $\ell - 1 \leq i \leq \ell + O(\log \varepsilon^{-1})$.

Consider any level $i > \ell + 6 \log \varepsilon^{-1}$: any donor vertex for such a level must lie in the ball $B(v, C_\varepsilon(2^\ell + \varepsilon^6 2^i)) \subseteq B(v, C_\varepsilon \varepsilon^6 2^{i+1}) \subseteq B(v, C_\varepsilon\, \varepsilon^5\, 2^i)$. A little algebra shows that

$$\varepsilon^5 C_\varepsilon = \varepsilon^5 (4 + \frac{32}{\varepsilon}) \leq \varepsilon^5 \frac{33}{\varepsilon} \leq \varepsilon^4 \cdot 33 \leq \varepsilon,$$

and thus the donor vertex must be at distance at most $\varepsilon 2^i$ from v. However, since the donor vertex must belong to $\phi(N_i)$, it must be at distance at least 2^i from any other donor vertices. Now, if there were two donor vertices at distance $\varepsilon 2^i$ from v, they would be at distance $2\varepsilon 2^i < 2^i$ from each other—this implies that there can be at most one donor vertex for such a "high" level.

Finally, it remains to show that the total number of long edges donated by this donor vertex x to vertices in $B(v, R)$ is small. Let $i_1, i_2, \ldots, i_t$, $i_j < i_{j+1}$ be the levels for which x donates a long edge to vertices in $B(v, R)$; we shall show that t is at most $O(\log \varepsilon^{-1})$. Since the first edge is long, $R \leq C_\varepsilon 2^{i_1+1}$. Moreover, since x donates this edge to u, we conclude that $d_G(x, u_1) \leq \varepsilon^6 C_\varepsilon 2^{i_1}$, so that $d_G(v, x) \leq R + \varepsilon^6 C_\varepsilon 2^{i_1} \leq (2 + \varepsilon^6) C_\varepsilon 2^{i_1}$. Suppose that $t > 7 \log \varepsilon^{-1} + 3$. Then an edge in F_{i_t} is donated from x to u_t, and we have that $d_G(x, u_t) \geq C_\varepsilon 2^{i_4 - 1}$. On the other hand, since $u_t \in B(v, R)$, by triangle inequality, $d_G(x, u_t) \leq d_G(x, v) + d_G(v, u_t) \leq (3 + \varepsilon^6) C_\varepsilon 2^{i_1}$. Since $i_4 \geq i_1 + 3$, this gives us the desired contradiction. Thus $t \leq O(\log \varepsilon^{-1})$. Since there are at most $\varepsilon^{-O(\dim_G)}$ edges donated to $B(v, R)$ from each of these levels, the claim follows. □

Using Lemma 15 along with the Structure Theorem 11 implies that the dimension of $\mathsf{conv}(G')$ is bounded by $O(\dim_G \log \varepsilon^{-1})$, which proves Theorem 1.

5 Convex Completions for Trees

The construction of the previous section showed that given any graph G, we could construct a new graph G' such that distances in G and G' are within $(1 + \varepsilon)$ of each other, and $\mathsf{conv}(G')$ has low doubling dimension. However, since the construction starts with the shortest-path metric d_G and completely ignores the topological structure of G itself, it is not suited to proving Theorem 2 which seeks to start with a tree and end with another tree. In this section, we show a different approach that allows us to monitor the graph structure more closely.

5.1 The Construction for Trees

We give a procedure that takes a general graph G and outputs a graph G' (since the construction itself does not depend on G being a tree); we then show some properties that hold when G is a tree. The procedure takes a graph $G =$

(V,E), and constructs a new graph $G' = (V', E')$ with $V \subseteq V'$ (by way of an intermediate graph $\widehat{G}$) as follows. Define an *exponential tail* with k edges as a path $P = \langle v_0, v_1, v_2, \ldots, v_k \rangle$, where the length of the edge $\{v_{i-1}, v_i\}$ is 2^i. Without loss of generality, the smallest edge length in G is at least 2^τ, where $\tau = 6 + \lceil \log(\frac{1}{\varepsilon}) \rceil$.

We construct the graph G' in the following way:

– As in Section 4.1, we consider a net-tree $(\mathbb{T}, \phi)$ for the graph G. If N_i is the set of nodes in T at height i, then for $u \in V$ define $i^*(u)$ to be the largest i such that $u \in \phi(N_i)$. Attach to each $u \in V$ an exponential tail with $i^*(u)$ edges; refer to the j^{th} vertex on this path as $u_{[j]}$, with $u_{[0]} = u$. Let $\widehat{G}$ be this intermediate graph consisting of G along with the tails.
– Consider an edge $e = \{u, v\} \in E(G)$, and suppose its length lies in the interval $(C_\varepsilon 2^{i-1}, C_\varepsilon 2^i]$. Some leaf of T must be mapped by ϕ to $u \in V$: let the level-(i) ancestor of that node be mapped by ϕ to $\widehat{u}$; similarly, define $\widehat{v}$ be defined for v. We now make an edge $\{\widehat{u}_{[i]}, \widehat{v}_{[i]}\}$ of length ℓ_e in the graph G'.

Note that if we start off with a tree T, the above procedure adds exponential tails to T to get the intermediate graph $\widehat{T}$, and then "moves the edges up the tails" to get the final graph T'.

Proposition 16 (Distance Preservation). *Let $\varepsilon < 1/4$. If the input graph is a tree $T = (V, E)$, then the above procedure results in a connected tree $T' = (V', E')$ such that for any $x, y \in V$,*

$$(1+\varepsilon)^{-1} d_T(x,y) \le d_{T'}(x,y) \le (1+\varepsilon) d_T(x,y).$$

Proof. Let us consider performing the above-mentioned transformation for edges in increasing order of edge-length. Given $j \in \mathbb{Z}_{\ge 0}$, let T_j be the forest formed by deleting all edges of length more than $C_\varepsilon 2^j$ from T; also, let T'_j be the forest formed by deleting the corresponding edges in T'. We will prove by induction on j that for all x, y that lie in some connected component in T_j, their distance in T'_j will satisfy the desired stretch bound. The base case is trivial, since all components of T_0 have single nodes in them.

To prove the claim for j, we inductively assume it for $j-1$. Now consider taking some edge $e = \{u, v\}$ of length $\ell_e \in (C_\varepsilon 2^{j-1}, C_\varepsilon 2^j]$. In this case we find some nodes $\widehat{u}$ and $\widehat{v}$, and add an edge of length ℓ_e between $\widehat{u}_{[j]}$ and $\widehat{v}_{[j]}$. By the properties of the net-tree, the distance $d_T(u, \widehat{u}) \le 2^{j+1} - 2$. Since T_j already contains all edges of length at most $C_\varepsilon 2^{j-1}$, and $C_\varepsilon \ge 4$, the net point $\widehat{u}$ lies in the same component as u in T_j. By the induction hypothesis, $d_{T'}(u, \widehat{u}) \le (1+\varepsilon) 2^{j+1}$; note that this implicitly proves that u and $\widehat{u}$ are in the same component in T'_j. A similar claim holds for $d_T(v, \widehat{v})$. Hence the distance in T'_{j+1} between u and v is at most

$$\begin{aligned}
& d_{T'_j}(u, \widehat{u}) + d_{T'_j}(\widehat{u}, \widehat{u}_{[j]}) + \ell_e + d_{T'_j}(\widehat{v}_{[j]}, \widehat{v}) + d_{T'_j}(\widehat{v}, v) \\
& = 2 \times (1+\varepsilon) 2^{j+1} + 2 \times 2^{j+1} + \ell_e \\
& \le \ell_e \left(\tfrac{8(1+\varepsilon)+8}{C_\varepsilon} + 1 \right) \le (1+\varepsilon) \ell_e,
\end{aligned}$$

where we used the fact that $C_\varepsilon = (4 + \frac{32}{\varepsilon})$ and $\varepsilon < 1/4$. Since each of the edges of T are not stretched by more than $(1+\varepsilon)$, this implies that the stretch for all pairs is bounded by the same value.

We also need to show that the distances are not shrunk too much in T': to show this, we go via $\widehat{T}$. (Recall that $\widehat{T}$ was the original tree T along with the exponential tails.) First note that for any $u, v \in V$, $d_T(u,v) = d_{\widehat{T}}(u,v)$. We show that distance do not shrink in going from $\widehat{T}$ to T'. It suffices to show this for the edges of T'. For an edge $e' = (\widehat{u}_{[j]}, \widehat{v}_{[j]})$ that has length $\ell_e \geq C_\varepsilon 2^{j-1}$, we note that their distance in $\widehat{T}$

$$d_{\widehat{T}}(\widehat{u}_{[j]}, \widehat{v}_{[j]}) \leq d_{\widehat{T}}(\widehat{u}_{[j]}, \widehat{u}) + d_{\widehat{T}}(\widehat{u}, u) + \ell_e + d_{\widehat{T}}(v, \widehat{v}) + d_{\widehat{T}}(\widehat{v}, \widehat{v}_{[j]}) \leq 4(2^{j+1} - 2) + \ell_e \tag{5.2}$$

Since $C_\varepsilon > 32/\varepsilon$, this is at most $(1+\varepsilon)\ell_e$. Thus the contraction going from $\widehat{T}$ to T' is at most $(1+\varepsilon)$.

Finally, we note that we have shown that T' is connected, and the number of edges in T' is equal to the number of edges in $\widehat{T}$, which is a tree. Thus T' is a tree as well. □

5.2 Bounding the Dimension of the Convex Closure: The Tree Case

Finally, to show that the doubling dimension of $\mathsf{conv}(T')$ is small, we will again invoke Theorem 11. However, since we have added additional vertices in going from T to T', we first show that $\dim(T')$ is $O(\dim(T))$. Since we have already shown that distances are preserved in going from $\widehat{T}$ to T', it suffices to bound the doubling dimension of $\widehat{T}$ (proof omitted).

Lemma 17. *The doubling dimension of $\widehat{T}$ is at most $O(\dim(T))$.*

Finally, Theorem 2 follows from the following bound on the number of long edges in T'

Lemma 18 (Few Long Edges). *For any vertex $v \in T'$ and every radius R, the number of long edges in T' is bounded by $2^{O(\dim)} \log \varepsilon^{-1}$.*

Proof. First consider some $v \in V$, and $R \geq 0$, and define $\ell \in \mathbb{Z}_{\geq 0}$ such that $R \in (C_\varepsilon 2^{\ell-1}, C_\varepsilon 2^\ell]$. Every long edge incident on $B(v,R)$ must have length at least R. Further, edges longer than $2C_\varepsilon R$ are incident on a tail node further than R from its root, and hence such an edge cannot be incident on $B(v,R)$. For each of the length scales $(C_\varepsilon 2^{\ell+j-1}, C_\varepsilon 2^{\ell+j}) : 0 \leq j \leq \log C_\varepsilon$, we will bound the number of long edges in that length scale. Fix one such scale, and let $L(v,R,j) = \{(u_i, w_i) : 1 \leq i \leq |L(v,R,j)|\}$ be the set of long edges of length in $(C_\varepsilon 2^{\ell+j-1}, C_\varepsilon 2^{\ell+j})$, such that $d(v,u_i) \leq R$. Since each long edge has length more than R, there is a path from v to u_i that does not use any of the long edges. Consider the set of nodes $W = \{w_i : 1 \leq i \leq |L(v,R,j)|\}$. Clearly, for any $w, w' \in W$, $d(w,w')$ is at most $2R + 2C_\varepsilon 2^{\ell+j} \leq 4C_\varepsilon 2^{\ell+j}$. Moreover, since T is a tree, the symmetric difference of the v-w and v-w' paths gives the shortest path from w' to w. Since the long edges incident on w and w' are in this symmetric difference, we conclude that $d(w,w') \geq 2C_\varepsilon 2^{\ell+j-1}$. Thus from the bound on doubling dimension, we conclude that $|W| \leq 2^{O(\dim)}$. Adding the contribution of the $O(\log \varepsilon^{-1})$ distance scales, we get the desired bound.

We now extend the argument to a vertex $v_{[i]}$ on an exponential tail hanging off v. If $i \geq j$, then $B(v_{[i]}, R) = \{v_{[i]}\}$. All edges incident on v have, up to a factor of two, the same length, and thus their endpoints form a near uniform submetric. Thus we can bound the degree of $v_{[i]}$ by $2^{O(\dim)}$ and the claim follows. On the other hand, when $i < j$, $B(v_{[i]}, R) \subseteq B(v, 2R)$ and an argument analogous to the one for the case $v \in V$ above suffices. $\square$

Acknowledgments. We would like to thank Robi Krauthgamer and Ravishankar Krishnaswamy for discussions.

References

1. Assouad, P.: Plongements lipschitziens dans $\mathbf{R}^n$. Bull. Soc. Math. France 111(4), 429–448 (1983)
2. Beygelzimer, A., Kakade, S., Langford, J.: Cover trees for nearest neighbor. In: The 23rd International Conference on Machine Learning (ICML) (2006)
3. Călinescu, G., Karloff, H., Rabani, Y.: Approximation algorithms for the 0-extension problem. In: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, pp. 8–16. ACM Press, New York (2001)
4. Chan, H.T.-H., Gupta, A., Maggs, B.M., Zhou, S.: On hierarchical routing in DOubling metrics. In: Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 762–771 (2005)
5. Clarkson, K.L.: Nearest neighbor queries in metric spaces. Discrete Comput. Geom. 22(1), 63–93 (1999)
6. Cole, R., Gottlieb, L.-A.: Searching dynamic point sets in spaces with bounded doubling dimension. In: The thirty-eighth annual ACM symposium on Theory of computing (STOC) (2006)
7. Fakcharoenphol, J., Harrelson, C., Rao, S., Talwar, K.: An improved approximation algorithm for the 0-extension problem. In: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms, pp. 257–265. Society for Industrial and Applied Mathematics (2003)
8. Gupta, A., Krauthgamer, R., Lee, J.R.: Bounded geometries, fractals, and low–distortion embeddings. In: Proceedings of the 44th Symposium on the Foundations of Computer Science (FOCS), pp. 534–543 (2003)
9. Gupta, A., Newman, I., Rabinovich, Y., Sinclair, A.: Cuts, trees and ℓ_1-embeddings of graphs. Combinatorica 24(2), 233–269 (2004) (Preliminary version in 40th FOCS 1999)
10. Gupta, A., Talwar, K.: How to complete a doubling metric (2008), `http://arxiv.org/abs/0712.3331v1`
11. Har-Peled, S., Mendel, M.: Fast constructions of nets in low dimensional metrics, and their applications. In: Proceedings of the twenty-first annual symposium on Computational geometry, pp. 150–158 (2005)
12. Indyk, P., Naor, A.: Nearest neighbor preserving embeddings. In: ACM Transactions on Algorithms (to appear)
13. Johnson, W.B., Lindenstrauss, J., Schechtman, G.: Extensions of lipschitz maps into banach spaces. Israel J. Math. 54(2), 129–138 (1986)

14. Karzanov, A.: Minimum 0-extensions of graph metrics. European Journal of Combinatorics 19(1), 71–101 (1998)
15. Klein, P., Plotkin, S.A., Rao, S.B.: Excluded minors, network decomposition, and multicommodity flow. In: Proceedings of the 25th ACM Symposium on the Theory of Computing (STOC), pp. 682–690 (1993)
16. Konjevod, G., Richa, A.W., Xia, D.: Optimal-stretch name-independent compact routing in doubling metrics. In: The twenty-fifth annual ACM symposium on Principles of distributed computing (2006)
17. Konjevod, G., Richa, A.W., Xia, D.: Optimal scale-free compact routing schemes in doubling networks. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA) (2007)
18. Krauthgamer, R., Lee, J.R.: The intrinsic dimensionality of graphs. In: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing, pp. 438–447. ACM Press, New York (2003)
19. Krauthgamer, R., Lee, J.R.: Navigating nets: simple algorithms for proximity search. In: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, pp. 798–807. Society for Industrial and Applied Mathematics (2004)
20. Lee, J., Naor, A.: Absolute lipschitz extendability. Comptes Rendus de l'Académie des Sciences - Series I - Mathematics 338(11), 859–862 (2004)
21. Lee, J., Naor, A., Peres, Y.: Trees and Markov convexity. Geometric and Functional Analysis. Preliminary version in SODA (to appear, 2006)
22. Matoušek, J.: On embedding trees into uniformly convex Banach spaces. Israel Journal of Mathematics 114, 221–237 (1999); (Czech version in: Lipschitz distance of metric spaces, C.Sc. degree thesis, Charles University 1990).
23. Matoušek, J.: Extension of Lipschitz mappings on metric trees. Commentationes Mathematicae Universitatis Carolinae 31(1), 99–104 (1990)
24. Rao, S.B.: Small distortion and volume preserving embeddings for planar and Euclidean metrics. In: 15th Annual ACM Symposium on Computational Geometry, pp. 300–306 (1999)
25. Talwar, K.: Bypassing the embedding: Algorithms for low-dimensional metrics. In: Proceedings of the 36th ACM Symposium on the Theory of Computing (STOC), pp. 281–290 (2004)

Sorting and Selection with Random Costs

Stanislav Angelov[1], Keshav Kunal[1], and Andrew McGregor[2]

[1] Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA 19104, USA
{angelov,kkunal}@cis.upenn.edu
[2] Information Theory & Applications Center,
University of California, San Diego, CA 92093, USA
andrewm@ucsd.edu

Abstract. There is a growing body of work on sorting and selection in models other than the unit-cost comparison model. This work treats a natural stochastic variant of the problem where the cost of comparing two elements is a random variable. Each cost is chosen independently and is known to the algorithm. In particular we consider the following three models: each cost is chosen uniformly in the range $[0, 1]$, each cost is 0 with some probability p and 1 otherwise, or each cost is 1 with probability p and infinite otherwise. We present lower and upper bounds (optimal in most cases) for these problems. We obtain our upper bounds by carefully designing algorithms to ensure that the costs incurred at various stages are independent and using properties of random partial orders when appropriate.

1 Introduction

In the relatively recent area of priced information [5,6,4], there is a set of *facts* each of which can be *revealed* at some cost. The goal is to pay the least amount such that the revealed facts allow some inference to be made. A specific problem in this framework, posed by Charikar et al. [4], is that of sorting and selection where each comparison has an associated cost. Here we are given a set V of n elements and the cost of comparing two elements u and v is $c_{(u,v)}$. This cost is known to the algorithm. We wish to design algorithms for sorting and selection that minimize the total cost of the comparisons performed. Results can be found in [15,11,12] where the performance of the algorithms is measured in terms of competitive analysis. In all cases assumptions are made about the edge costs, e.g., that there is an underlying monotone structure [15,11] or metric structure [12].

A related problem that predates the study of priced information is the problem of *sorting nuts and bolts* [1,17]. This is a problem that may be faced by "any disorganized carpenter who has a mixed pile of bolts and nuts and wants to find the corresponding pairs of bolts and nuts" according to the authors of [1]. The problem amounts to sorting two sets, X and Y, each with n elements given that comparisons are only allowed between $u \in X$ and $v \in Y$. It can be shown that this problem can be generalized to the priced information problem in which comparison costs are either 1 or ∞.

In this paper we study a natural stochastic variant of the sorting problem. We consider each comparison cost to be chosen independently at random. Specifically, we consider the following three models:

(a) Uniform Costs: $c_{(u,v)}$ is chosen uniformly in the range $[0, 1]$,
(b) Boolean Costs: $c_{(u,v)} = 0$ with probability p and 1 otherwise,
(c) Unit and Infinite Costs: $c_{(u,v)} = 1$ with probability p and ∞ otherwise.

The first model is in the spirit of the work on calculating the expected cost of the minimum spanning tree [10]. The second and third models are related to the study of random partially ordered sets (see [3] for an overview) and linear extensions [9,13,2]. Specifically, in Model (b), the free comparisons define a partial order $(V, \preccurlyeq)$ that is chosen according to the *random graph model*. To sort V we need to do the minimum number of remaining comparisons to determine the linear extension, or total order. In Model (c) we have the problem of inferring properties of the random partial order $(V, \preccurlyeq)$ defined by the cost 1 edges.

1.1 A Motivation from Game Theory

The framework of priced information lends itself naturally to a game theoretic treatment where there are numerous sellers each owning one or more facts. Some facts will be, in a sense, more valuable than others. In the case of sorting, the value of a comparison (u, v) is inversely related to $|\{w : u < w < v \text{ or } v < w < u\}|$ because for each such w, the comparisons (u, w) and (w, v) together provide an alternative way of implying (u, v). How should sellers price their information in an effort to maximize their profit? Herein lies the dilemma — if the pricing of the facts is strictly monotonic with their value, the buyer can infer the sorted order from the prices themselves and by performing a single (cheapest) comparison! Yet, if there is no correlation, the seller is not capitalizing on the value of the information they have to sell. It seems likely that the optimum pricing of a fact will be a non-deterministic function of the value. While a treatment of the game theoretic problem seems beyond our reach at this time, we feel that a first step will be to find optimal buyer strategies when the price of each fact is chosen randomly and independently of the value of the fact.

1.2 Our Results

For $p = 1/2$, our results are summarized in Table 1. In general, we will present bounds in terms of both n and p. Note that rather than using a competitive analysis of our algorithms (as in [15,11,12]) we estimate the expected cost of our algorithms and the expected cost of the respective minimum certificate.

We would like to note that for the first three rows of Table 1, the expected cost of each comparisons is $1/2$ but the variance differs. For selection type problems the variance makes a big difference since there are many ways to certify the rank of an element. However for sorting there is only one (minimal) certificate for the sorted order. Nevertheless, a little bit of variance makes it possible to sort with only linear cost rather than $O(n \log n)$ cost.

Table 1. Comparison between the expected costs of our algorithms and the minimum certificates for sorting and selection for various cost functions when $p = 1/2$. The first row follows from standard algorithms and is given as a reference point for comparison. Also, in the case of $c_{(u,v)} \in \{1, \infty\}$ we consider finding all maximal/minimal elements.

	Max and Min		Selection		Sorting	
	Upper Bound	Min. Certificate	Upper Bound	Min. Certificate	Upper Bound	Min. Certificate
$c_{(u,v)} = 1/2$	$O(n)$	$\Omega(n)$	$O(n)$	$\Omega(n)$	$O(n \log n)$	$\Omega(n)$
$c_{(u,v)} \in [0, 1]$	$O(\log n)$	$\Omega(\log n)$	$O(\log^6 n)$	$\Omega(\log n)$	$O(n)$	$\Omega(n)$
$c_{(u,v)} \in \{0, 1\}$	$O(1)$	$\Omega(1)$	$O(\log n)$	$\Omega(1)$	$O(n)$	$\Omega(n)$
$c_{(u,v)} \in \{1, \infty\}$	$O(n \log n)$	$\Omega(n)$	–	–	–	–

One of the main challenges in the analysis of our algorithms is to ensure that the costs incurred at various stages of the algorithm are independent. We achieve this by carefully designing the algorithms and describing an alternative random process of cost assignment that we argue is equivalent to the original random process of cost assignment.

2 Preliminaries

We are given a set V of n elements, drawn from some totally ordered set. We are also given a non-negative symmetric function $c : V \times V \to \mathcal{R}^+$ which determines the cost of comparing two elements of V. Given V and c, we are interested in designing algorithms for sorting and selection that minimize the total cost of the performed comparisons.

The above setting is naturally described by the complete weighted graph on V, call it G, where the weight c_e of an edge e is determined by the cost function c. The direction of each edge (u, v) in G is consistent with the underlying total order and is unknown unless the edge e is *probed*, i.e., the comparison between u and v is performed, or it is implied by transitivity, i.e., a directed path between u and v is already revealed. In this case we call u and v *comparable*.

An algorithm for sorting or selection should reveal a *certificate* of the correctness of its output. In the case of sorting, the minimal certificate is unique, namely the Hamiltonian path in G between the largest and the smallest elements of V. In the case of selection, the certificate is a subgraph of G that includes a (single) directed path between the element of the desired rank and each of the remaining elements of V. In the special case of max-finding, the certificate is a rooted tree on V, the maximum element being the root. The *cost* of a certificate is defined as the total cost of the included edges.

In this paper we consider three different stochastic models for determining the cost function c (see Section 1). In Models (b) and (c), the graphs induced respectively by the cost 0 or 1 edges have natural analogue to random graphs with parameter p, denoted by $G_{n,p}$. Note that in Models (a) and (b), the maximum cost of a comparison is 1. When this is case, the following proposition will be useful and follows from a natural greedy strategy to find the maximum element in the standard comparison model.

Proposition 1. *Given a set V of n elements, drawn from a totally ordered set, where the cost of the comparison between any two elements is at most 1, we can find (and certify) the maximum element performing $n-1$ comparisons incurring a cost of at most $n-1$.*

We will measure the performance of our algorithms by comparing the expected total cost of the edges probed with the expected cost of a minimum certificate. Note that the cost of the minimum certificate is concentrated around the mean in most cases. Even when the minimum certificate cost is far from the mean, we can obtain good bounds on the expected ratio by using algorithms from [4] (Model (a)) or standard algorithms (Model (b) and (c)).

Finally, in the analysis, it would be often useful to number the elements of V, $v_1, \cdots, v_n$ such that $v_1 < \cdots < v_n$. We also define the *rank* of an element v with respect to a set $S \subseteq V$ to be $\mathrm{rk}_S(v) = |\{u : u \leq v, u \in S\}|$.

3 Uniform Comparison Costs

In this section we will assume that the cost of each comparison is chosen uniformly at random in the range $[0, 1]$. We consider the problems of finding the maximum or minimum elements, general selection, and sorting. The algorithms are presented in Fig. 1.

Theorem 1. *The expected cost of UniformFindMax is at most $2(H_n - 1)$ where $H_k = \sum_{i=1}^{k} 1/i$.*

Proof. We analyze a random process where we consider edges one by one in a non-decreasing order of their cost. Note that the costs of edges define a random permutation on the edges. If an edge is incident to two candidate elements, i.e., elements that have not lost so far a performed comparison, we probe the edge, otherwise we ignore the edge. Either way we say the edge is *processed*.

We divide the analysis in rounds. A round terminates when an edge is probed. After the end of a round, the number of candidates for the maximum decreases by one. Therefore after $n-1$ rounds the last candidate would be the maximum element. For $r \in \{1, \ldots, n-1\}$, let t_r denote the random variable which counts the number of edges processed in the rth round. Let $T_r = \sum_{i=1}^{r} t_i$ denote the rank of the edge (in the sorted by costs order) found in the rth round. Therefore, the expected cost of the performed comparison is $\mathbb{E}[T_r] / (\binom{n}{2} + 1)$.

It remains to show an upper bound on the value of $\mathbb{E}[T_r] = \sum_{i=1}^{r} \mathbb{E}[t_i]$. So far T_{r-1} edges have been processed. The probability that the next edge is between two candidate elements is $p = \binom{n-(r-1)}{2} / \left(\binom{n}{2} - T_{r-1}\right) \geq \binom{n-(r-1)}{2} / \binom{n}{2}$. Hence, for $r \in \{1, \ldots, n-2\}$, $\mathbb{E}[t_r] \leq 1/p \leq \binom{n}{2} / \binom{n-(r-1)}{2}$, and for $r = n-1$, we have

$\mathbb{E}[T_r] \le \binom{n}{2}$. We conclude that the total expected cost is at most,

$$\sum_{r=1}^{n-1} \frac{\mathbb{E}[T_r]}{(\binom{n}{2}+1)} \le 1 + \sum_{r=1}^{n-2}\sum_{i=1}^{r} \frac{1}{\binom{n-(i-1)}{2}} \le 1 + \sum_{r=1}^{n-2} \frac{2}{n-r+1} = 2(H_n - 1) \ .$$

Theorem 2. *The expected cost of the cheapest rank k certificate is $H_k + H_{n-k+1} - 2$.*

Proof. Consider v_i with $i < k$. Any certificate must include a comparison with at least one of $v_{i+1}, \ldots, v_k$. The expected cost of the minimum of these $k - i$ comparisons is $\frac{1}{k-i+1}$. Summing over i, $i < k$, yields $H_k - 1$. Similarly, now consider v_i with $i > k$. Any certificate must include a comparison with at least one of $v_k, \ldots, v_{i-1}$. The expected cost of the minimum of these $i-k$ comparisons is $\frac{1}{i-k+1}$. Summing over i, $n \ge i > k$, yields $H_{n-k+1} - 1$. The theorem follows.

Note that the theorem above also implies a lower bound of $\Omega(\log n)$ on the expected cost of the cheapest certificate for the maximum (minimum) element. To prove a bound on the performance of *UniformSelection* we need the following preliminary lemma.

Lemma 1. *Let $v \in V$ and perform each comparison (not just those involving v) with probability p . Then, with probability at least $1 - 1/n^4$ (assuming $p > 1/n^3$), for all u such that*

$$|\operatorname{rk}_V(u) - \operatorname{rk}_V(v)| \ge \frac{150 \log n\,(\log n + \log(1/p))}{p} \ ,$$

the relationship between u and v is certified by the comparisons performed.

Proof. Without loss of generality, $\operatorname{rk}_V(v) \ge n/2$. We will consider elements in $S = \{u : \operatorname{rk}_V(u) < \operatorname{rk}_V(v)\}$. The analysis for elements among $\{u : \operatorname{rk}_V(u) > \operatorname{rk}_V(v)\}$ is identical and the result follows by the union bound. Throughout the proof we will assume that n is sufficiently large.

Let D be the subset of S such that $u \in D$ is comparable to v. We partition S into sets,

$$B_i = \{u : \operatorname{rk}_V(v) - wi \le \operatorname{rk}_V(u) < \operatorname{rk}_V(v) - w(i-1)\} \ ,$$

where $w = \frac{12 \log n}{p(1-e^{-1})}$. Let $X_i = D \cap B_i$, that is, the elements from set B_i that are comparable to v. For the sake of notation, let $X_0 = \{v\}$. Let $D_i = \bigcup_{0 \le j \le i-1} X_j$. If we perform a comparison between an element of D_i and an element u of B_i then we certify that u is less than v. The probability that an element of B_i gets compared to an element of D_i is,

$$1 - (1-p)^{D_i} \ge 1 - e^{-pD_i} \ge (1 - e^{-1}) \max\{1, pD_i\} \ .$$

Let $(Y_i)_{1 \le i}$ be a family of independent random variables distributed as $\mathbf{Bin}(w, q)$ where $q = (1 - e^{-1}) \max\{pD_i, 1\}$. Note that $\mathbb{E}[Y_i] = 12 D_i \log n$ if $pD_i \le 1$.

1. For i such that $D_i < 1/p$. Using the Chernoff Bounds,

$$\mathbb{P}\left[X_i < D_i \log n\right] = \mathbb{P}\left[X_i < \frac{qw}{12}\right] \leq \mathbb{P}\left[Y_i < \frac{\mathbb{E}\left[Y\right]}{12}\right] \leq e^{-6(11/12)^2 D_i \log n} \leq n^{-5} \ .$$

   In other words, the number of comparable elements increases by at least a $\log n$ factor until $D_i \geq 1/p$. Hence, with probability at least $1 - \log(1/p)/n^5$, for all i, $D_i \geq \min\{1/p, (\log n)^{i-1}\}$. In particular, $D_{\log 1/p} \geq 1/p$.
2. Assume that $i > \log(1/p)$ and therefore $D_{\log 1/p} \geq 1/p$. Using Chernoff Bounds, we get

$$\mathbb{P}\left[X_i < \frac{1}{p}\right] \leq \mathbb{P}\left[Y_i < \frac{1}{p}\right] \leq \mathbb{P}\left[Y_i < \frac{\mathbb{E}\left[Y_i\right]}{12 \log n}\right] \leq e^{-(1-\frac{1}{12\log n})^2 6 D_i \log n} \leq n^{-5} \ .$$

Therefore with probability at least $1 - t/n^5$, $D_t \geq 6p^{-1} \log n$ where $t = 6 \log n + \log(1/p)$. Consider an element $u \in B_{t'}$ where $t' > t$. The probability that u is not in D is at most $(1-p)^{6 \log n/p} \leq 1/n^6$. Hence, with probability at least $1 - (1+t)/n^5$,

$$|S \setminus D| \leq wt \leq \frac{150 \log n \left(\log n + \log(1/p)\right)}{p} \ .$$

Lemma 2. *Consider the algorithm UniformFindRankCertificate called on a randomly chosen v. With probability at least $1 - n^{-3}$ the algorithm returns a certificate for the rank of v. The expected cost of the comparisons is $O(\log^5 n)$.*

Proof. Let V_i be the set of elements at the start of iteration i. Let $p_1 = \alpha/n$ be the probability that $c_e \in [0, \alpha/n]$. For $i > 1$, let $p_i = \alpha 2^{i-1}/n$ be the probability that $c_e \in [\alpha 2^{i-1}/n, \alpha 2^i/n]$. First we show that, with probability at least $1 - \frac{\log_2(n/\alpha)}{n^4}$, for all $1 \leq i \leq \log_2(n/\alpha)$, $|V_i| < n/2^{i-1}$. Assume that $|V_i| < n/2^{i-1}$. Appealing to Lemma 1, there are less than

$$\frac{300 \log |V_i| (\log |V_i| + \log(1/p))}{p} \leq \frac{600 \log^2 n}{\alpha 2^{i-1}/n} = |V_i|/2$$

elements in $V_{i+1} \setminus V_i$ and hence $|V_{i+1}| < n/2^i$. It remains to show that the cost per iteration is $O(\log^4 n)$. This follows since the expected number of comparisons is $O(V_i^2 \alpha 2^i/n) = O(\alpha n/2^i)$ and each comparison costs at most $\alpha 2^i/n$.

The following theorem can be proved using standard analysis of the appropriate recurrence relations and Lemma 2.

Theorem 3. *The algorithm UniformSelection can be used to select the kth element. The expected cost of the certificate is $O(\log^6 n)$. The algorithm UniformSort returns a sorting certificate with expected cost $O(n)$.*

Note that we can check if a certificate is a valid one without performing any additional comparisons. In the case when *UniformFindRankCertificate* fails, we can reveal all edges to obtain a certificate without increasing asymptotically the overall expected cost.

```
Algorithm UniformFindMax($V$)
1.  for $j = 1$ to $n - 1$
2.      do Perform cheapest remaining comparison
3.         Remove the smaller element of the performed comparison
4.  return remaining element

Algorithm UniformFindRankCertificate($V, v$)
1.  Let $\alpha = 1200 \log^2 n$
2.  Perform all comparisons $e$ such that $c_e \in [0, \alpha/n]$
3.  for $u \in V$
4.      do if $u$ is comparable with $v$
5.            then $V \leftarrow V \setminus \{u\}$
6.                 if $u < v$ then $V_1 \leftarrow V_1 \cup \{u\}$ else $V_2 \leftarrow V_2 \cup \{u\}$
7.  for $i = 1$ to $\log_2(n/\alpha)$
8.      do Perform all comparisons $e$ such that $c_e \in [\alpha 2^{i-1}/n, \alpha 2^i/n]$
9.         Repeat Steps 3-6
10. return $V_1, V_2$

Algorithm UniformSelection($V, k$)
1.  if $|V| = 1$ then return $V$
2.  Pick random pivot $v \in V$
3.  $(V_1, V_2) \leftarrow$ UniformFindRankCertificate($V, v$)
4.  $V \leftarrow V \setminus \{v\}$
5.  if $|V_1| > k$ then UniformSelection($V_1, k$) else UniformSelection($V_2, k - |V_1|$)

Algorithm UniformSort($V$)
1.  Pick random pivot $v \in V$
2.  $(V_1, V_2) \leftarrow$ UniformFindRankCertificate($V, v$)
3.  return (UniformSort($V_1$), $v$, UniformSort($V_2$))
```

Fig. 1. Algorithms for uniform comparison costs

Theorem 4. *The expected cost of the cheapest sorting certificate is* $(n-1)/2$.

Proof. For each $1 \leq i \leq n-1$ there must be a comparison between v_i and v_{i+1}. The expected cost of each is $1/2$. The theorem follows by linearity of expectation.

4 Boolean Comparison Costs

In this section we assume that comparisons are for free with probability p and have cost 1 otherwise. We consider the problems of finding the maximum or minimum elements, general selection, and sorting. The algorithms for maximum finding and selection are presented in Fig. 2. For sorting we use results from [2] and [15] to obtain a bound on the number of comparisons needed to sort the random partial order defined by the free comparisons.

Theorem 5. *The expected cost of BooleanFindMax is* $1/p - 1$ *as* $n \to \infty$.

```
Algorithm BooleanFindMax(V)
1.  Perform all free comparisons
2.  Find the maximum element among the elements that have not lost a comparison in Step 1 using cost 1 comparisons.

Algorithm BooleanSelection(V, k)
1.  Perform all free comparisons
2.  $w \leftarrow 3(\log n)/p^2$
3.  $S \leftarrow \{v : v$ wins at least $k-1-w$ comparisons and loses at least $n-k-w$ comparisons$\}$
4.  Find the minimum and maximum element of $S$ and determine their exact rank by comparing them to all elements whose relation to them is unknown.
5.  $r_{min} \leftarrow \mathrm{rk}_V$(minimum element of $S$)
6.  $r_{max} \leftarrow \mathrm{rk}_V$(maximum element of $S$)
7.  $T \leftarrow \{v : r_{min} \le \mathrm{rk}_V(v) \le r_{max}\}$
8.  if $r_{min} \le k$ and $r_{max} \ge k$
9.      then return StandardSelection($T, k - r_{min}$)
10.     else return StandardSelection($V, k$)
```

Fig. 2. Algorithms for boolean comparison costs

Proof. Consider the ith *largest* element. The probability that there is no free comparison to a larger element is $(1-p)^{i-1}$. Hence, after performing all the free comparisons, the expected number of non-losers, in the limit as n tends to infinity, is

$$\lim_{n\to\infty}\sum_{i=1}^{n}(1-p)^{i-1} = \lim_{n\to\infty}\frac{1-(1-p)^n}{p} = 1/p \ .$$

Hence, by Proposition 1, the expected number of comparisons of cost 1 that are necessary is $1/p - 1$.

The theorem above leads to an immediate corollary:

Corollary 1. *The expected cost of the cheapest certificate for the maximum element and the element of rank k is $\Omega(1/p)$ as $n \to \infty$.*

Using Theorem 5, we obtain a sorting algorithm with expected cost of at most $(1/p-1)(n-1)$ by repeating $n-1$ times *BooleanFindMax*. We improve this result (for sufficiently small p) by observing that the free comparisons define a random partial order on the n elements, call it $G_{n,p}$. In [2], the expected number of linear extensions of $G_{n,p}$ was shown to be

$$p^{-1}\prod_{k=1}^{n} 1-(1-p)^k \le 1/p^{n-1} \ .$$

A conjecture, proposed by Kislitsyn [16], Fredman [9], and Linial [18], states that given a partial order P, there is a comparison between two elements such that the fraction of extensions of P where the first elements precedes the second one is between $1/3$ and $2/3$. Ignoring running time, this would imply sorting with cost $\log_{3/2} e(P)$, where $e(P)$ denotes the number of linear extensions of P. In [14], a weaker version of the conjecture was shown giving rise to an efficient, via randomization [8], sorting algorithm with cost $\log_{11/8} e(P)$. Taking a different approach, Kahn and Kim [13] described a deterministic polynomial time, $O(\log e(P))$ cost algorithm to sort any partial order P.

Combining the above results, and using Jensen's inequality, we obtain a sorting algorithm with expected cost at most,

$$\log_{11/8} e(G_{n,p}) \leq (\log_{11/8} p^{-1})(n-1) \ .$$

Note that for $p < 0.1389$, $\log_{11/8}(1/p) < 1/p - 1$. Combining the two sorting methods, we obtain the following theorem.

Theorem 6. *There is a sorting algorithm for the Boolean Comparison Model with expected cost of* $\min\{\log_{11/8} 1/p, 1/p-1\} \cdot (n-1)$.

The proof of the following theorem about the cheapest sorting certificate is nearly identical to that of Theorem 4.

Theorem 7. *The expected cost of the cheapest sorting certificate is* $(1-p)(n-1)$.

We next present our results for selection.

Theorem 8. *The algorithm BooleanSelection can be used to select the* kth *element. The expected cost of the algorithm is* $O(p^{-2}\log n)$.

Proof. We want to bound the size of set S as defined in the algorithm. Fix an element v_j. For an element v_i such that $i < j$, let $l = j - i - 1$. Consider the event that we can infer $v_i < v_j$ from the free comparisons because there exists an element $v_{i'}$ such that $v_i < v_{i'} < v_k$ and $c_{(v_i,v_{i'})} = c_{(v_{i'},v_j)} = 0$. The probability of this event is $1-(1-p^2)^l$ and hence with probability at least $1-1/n^3$ we learn $v_i < v_j$ if $l \geq w = 3(\log n)/p^2$. Therefore, with probability at least $1-1/n^2$, v_j wins at least $j-1-w$ comparisons. Similarly with probability at least $1-1/n^2$, v_j loses at least $n-j+w$ comparisons.

Hence, with probability $1-2/n^2$, every element from the set

$$S' = \{v : k - w \leq \mathrm{rk}_V(v) \leq k + w\} \ ,$$

belongs to the set S and in particular the element of rank k also belongs to S. Note that no element from outside S' can belong to S and hence $|S| \leq 2w$. By Proposition 1, it takes $O(w)$ comparisons to compute the minimum and maximum elements in S. There are at most $2w$ elements incomparable to the minimum (maximum) element with probability at least $1-2/n^2$ and hence the expected cost for determining the exact rank of minimum (maximum) element from S is bounded by

$$2w(1-2/n^2) + (n-1)2/n^2 \, = \, O(w)$$

```
Algorithm PosetFindMaximal
1.  Pick $v \in V$
2.  while $|V| > 0$
3.      do Perform cost 1 comparisons with $v$ until it loses (or is certified maximal)
4.          if $v$ wins all of its comparisons then return $v$ maximal
5.              else $V \leftarrow V \setminus \{v\}$ and set $v$ to the winner of the last comparison

Algorithm PosetFindAllMaximal
1.  for each $v \in V$
2.      do Perform, in a random order, cost 1 comparisons with $v$
3.          until $v$ loses or all such comparisons are performed
4.  return All elements that did not lose comparison
```

Fig. 3. Algorithms for $1/\infty$ comparison costs

in expectation. Since the size of T is also $O(w)$, step 5 takes $O(w)$ time if $v_k \in T$, which happens with probability at least $1 - 2/n^2$, and $O(n)$ otherwise. Similar to the previous step, the expected cost is $O(w)$.

Note that with a slight alteration to the *BooleanSelection* algorithm it is possible to improve upon Theorem 8 if p is much smaller than $1/\log n$. Namely, setting $w = 150p^{-1} \log n \log(n/p)$, and appealing to Lemma 1 in the analysis, gives an expected cost of $O\left(p^{-1} \log n \log(n/p)\right)$.

5 Unit and Infinite Comparison Costs

In this section we consider the setting where only a subset of the comparisons is allowed. More specifically, each comparison is allowed with probability p (has cost 1) and is not allowed otherwise (has infinite cost). Here, the underlying total order might not be possible to infer even if all comparisons are performed. This is because, for example, adjacent elements can be compared only with probability p. Hence, even the maximum element might not be possible to certify exactly. We therefore relax our goals to finding maximal elements and inferring the poset defined by the edges of cost 1. In what follows, we present algorithms for finding a maximal element as well as all maximal elements (see Fig. 3). We consider an element maximal if it wins (directly or indirectly) all allowed comparisons to its neighbors.

Theorem 9. *The expected cost of the cheapest certificate for all maximal elements is $\Omega\left(n(1-(1-p)^{n-1})\right)$.*

Proof. In this setting, each element that has no edges of cost 1 incident to it is a maximal element. In expectation, there are $n(1-p)^{n-1}$ such elements. For each of the remaining elements we need to do at least one comparison. Note that each comparison satisfies this requirement for two elements. Therefore, we need to do at least $\frac{1}{2}(n - n(1-p)^{n-1})$ comparisons in expectation.

Theorem 10. *The expected cost of PosetFindAllMaximal is $O(n \log n)$. The expected cost of PosetFindMaximal is at most $n-1$.*

Proof. We first analyze *PosetFindAllMaximal*. Fix an element v. Let $i = \mathrm{rk}_V(v)$. Consider the following equivalent random process that assigns costs (1 or ∞) to edges in the following way:

1. Pick t from a random variable T distributed as $\mathbf{Bin}(n-1,p)$.
2. Repeat t times: Assign cost 1 to a random edge adjacent to v whose cost has not yet been determined.
3. Declare the cost of all other edges adjacent to v to be ∞.
4. For each remaining graph edge assign cost 1 with probability p and ∞ otherwise.

We may assume that the algorithm probes the cost 1 edges in this order until v loses a comparison or until all cost 1 edges are revealed. If v has not lost a comparison, v loses the next performed comparison with probability at least $(i-1)/(n-1)$. Hence, the expected number of comparisons involving v is

$$\sum_t \mathbb{P}\left[T=t\right] \sum_{j=1}^{t} \frac{i-1}{n-1}\left(1-\frac{i-1}{n-1}\right)^{j-1} j \leq \sum_t \mathbb{P}\left[T=t\right] \frac{n-1}{i-1} \leq \frac{n-1}{i-1} \ .$$

Therefore, by linearity of expectation the total number of comparisons we expect to do is at most $(n-1)H_{n-1} + (n-1)$.

The second part of the theorem follows easily from Proposition 1. The algorithm *PosetFindMaximal* is given for completeness.

Recently, Daskalakis et al. [7] gave algorithms with $O(wn)$ cost for finding all maximal elements in a poset where w is the width or maximum size of incomparable elements in the poset. Note that for $p < 1/2$, $\mathbb{E}\left[w\right] = \Omega(\log n)$ but for higher values of p, their algorithm yields a cheaper solution. However, their results also apply in the worst case, not just the expected case.

6 Conclusions and Open Questions

We have presented a range of algorithms for finding cheap sorting/selection certificates in three different stochastic priced-information models. Most of our algorithms are optimal up to constants and the remaining algorithms are optimal up to poly-logarithmic terms (for constant values of the parameter p). Beyond improving the existing algorithms there are numerous ways to extend this work. In particular,

– What about the price model in which the comparison costs are chosen in an adversarial manner but the order of the elements is randomized?
– In this work we have compared expected cost of minimum certificates to expected cost of the algorithms presented. Is it possible to design algorithm which are optimal in the sense that the expected cost of the certificate found is minimal over all algorithms? Perhaps this would admit an information theoretic approach.

Finally, this work was partially motivated by the game theoretic framework described in Section 1.1. A full treatment of this problem was beyond the scope of the present work. However, the problem seems natural and deserving of further investigation.

References

1. Alon, N., Blum, M., Fiat, A., Kannan, S., Naor, M., Ostrovsky, R.: Matching nuts and bolts. In: SODA, pp. 690–696 (1994)
2. Alon, N., Bollobás, B., Brightwell, G., Janson, S.: Linear extensions of a random partial order. Annals of Applied Probability 4, 108–123 (1994)
3. Brightwell, G.: Models of random partial orders, pp. 53–83 (1993)
4. Charikar, M., Fagin, R., Guruswami, V., Kleinberg, J.M., Raghavan, P., Sahai, A.: Query strategies for priced information. J. Comput. Syst. Sci. 64(4), 785–819 (2002)
5. Cicalese, F., Laber, E.S.: A new strategy for querying priced information. In: Gabow, H.N., Fagin, R. (eds.) STOC, pp. 674–683. ACM, New York (2005)
6. Cicalese, F., Laber, E.S.: An optimal algorithm for querying priced information: Monotone boolean functions and game trees. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 664–676. Springer, Heidelberg (2005)
7. Daskalakis, C., Karp, R.M., Mossel, E., Riesenfeld, S., Verbin, E.: Sorting and Selection in Posets (2007), http://arxiv.org/abs/0707.1532
8. Dyer, M.E., Frieze, A.M., Kannan, R.: A random polynomial time algorithm for approximating the volume of convex bodies. In: STOC, pp. 375–381. ACM, New York (1989)
9. Fredman, M.L.: How good is the information theory bound in sorting? Theor. Comput. Sci. 1(4), 355–361 (1976)
10. Frieze, A.M.: Value of a random minimum spanning tree problem. J. Algorithms 10(1), 47–56 (1985)
11. Gupta, A., Kumar, A.: Sorting and selection with structured costs. In: FOCS, pp. 416–425 (2001)
12. Gupta, A., Kumar, A.: Where's the winner? Max-finding and sorting with metric costs. In: Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX 2005 and RANDOM 2005. LNCS, vol. 3624, pp. 74–85. Springer, Heidelberg (2005)
13. Kahn, J., Kim, J.H.: Entropy and sorting. J. Comput. Syst. Sci. 51(3), 390–399 (1995)
14. Kahn, J., Saks, M.: Balancing poset extensions. Order 1, 113–126 (1984)
15. Kannan, S., Khanna, S.: Selection with monotone comparison cost. In: SODA, pp. 10–17 (2003)
16. Kislitsyn, S.S.: A finite partially ordered set and its corresponding set of permutations. Mathematical Notes 4, 798–801 (1968)
17. Komlós, J., Ma, Y., Szemerédi, E.: Matching nuts and bolts in $O(n \log n)$ time. SIAM J. Discrete Math. 11(3), 347–372 (1998)
18. Linial, N.: The information-theoretic bound is good for merging. SIAM J. Comput. 13(4), 795–801 (1984)

Guided Search and a Faster Deterministic Algorithm for 3-SAT

Dominik Scheder*

Theoretical Computer Science, ETH Zürich
CH-8092 Zürich, Switzerland
dscheder@inf.ethz.ch

Abstract. Most deterministic algorithms for NP-hard problems are *splitting algorithms*: They split a problem instance into several smaller ones, which they solve recursively. Often, the algorithm has a choice between several splittings. For 3-SAT, we show that choosing wisely which splitting to apply, one can avoid encountering too many worst-case instances. This improves the currently best known deterministic worst case running time for 3-SAT from $\mathcal{O}(1.473^n)$ to $\mathcal{O}(1.465^n)$, n being the number of variables in the input formula.

1 Introduction

Most deterministic algorithms for NP-hard problems like k-SAT, k-colorability and Maximum Independent Set use the idea of splitting: A problem instance I is replaced by several smaller instances $I_1, \ldots, I_\ell$, which are solved recursively. Of course, we want ℓ to be small and the size of the instances I_i to be much smaller than the size of I—whatever *size* means in this context. Most algorithms use several branching rules, i.e. rules for replacing I by $I_1, \ldots, I_\ell$. Inevitably, not every rule will apply to every instance, and some rules will amount to higher running time and some to lower. Often, a single rule is responsible for the worst-case behavior of the algorithm. Imagine you have a "meta-rule" that tells you what branching rule to apply in order to avoid encountering too many worst-case instances. This will of course speed up your algorithm. For our 3-SAT-algorithm, we find such a meta-rule. The general idea is to run a preliminary search on a given instance I that simply aborts when a worst case instance is encountered. We pick *one* such worst-case instance I' and again start the preliminary search on I'. Repeating, we will find an instance I^* and a search tree for I^* that contains no worst-case instances. The trick is that one can show, for our particular algorithm, that this very search tree is also a search tree for I. We use the instance I^* as a *search guide* for I, always applying the branching rules that would have applied in the search tree for I^*. The algorithm to which we apply this idea is the deterministic local search algorithm for 3-SAT by Dantsin et al. [3], of which we improve the running time from the previously best known $\mathcal{O}(1.473^n)$ [2] to $\mathcal{O}(1.465^n)$ (here, n is the number of variables). The idea is in fact not limited to 3-SAT, it can be applied for general k-SAT, but the improvement over the original algorithm by [3] becomes smaller and smaller.

* Research is supported by the SNF Grant 200021-118001/1.

Previous Results

In recent years, a lot of research has been done in designing "modestly exponential" algorithms deciding 3-SAT, i.e. running in time $\mathcal{O}(a^n)$, for a considerably smaller than 2. The currently fastest *randomized* algorithm, given by Daniel Rolf [7], achieves a running time of $\mathcal{O}(1.32216^n)$.

The running times of *deterministic* algorithms for 3-SAT are much higher: Dantsin et al. [3] gave a deterministic algorithm based on local search, with a running time of $\mathcal{O}(1.481^n)$. Later, Brueggemann and Kern [2] further improved this algorithm and obtained a running time of $\mathcal{O}(1.473^n)$, which was the previously fastest known deterministic algorithm.

We apply the idea of *guided search* to the splitting algorithm in Dantsin et al. [3] and Brueggemann and Kern [2], thus avoiding encountering too many worst-case formulas and improving the running time of deterministic local search algorithms for 3-SAT from $\mathcal{O}(1.473^n)$ to $\mathcal{O}(1.465^n)$.

Notation

A CNF formula, or simply a CNF, is a conjunction (`and`) of clauses, and a clause is a disjunction (`or`) of literals. A literal is either a boolean variable x or its negation $\bar{x}$. We can assume that no clause contains both a variable and its negation. A k-CNF is a CNF in which every clause contains at most k literals, and k-SAT is the problem whether a given k-CNF is satisfiable. If γ is a *partial* truth assignment, then we denote by $F^{[\gamma]}$ the k-CNF obtained by setting the variables of F as described by γ. If γ does not set variable x, we may write $[\gamma, x \mapsto 1]$ (or $[\gamma, x \mapsto 0]$) to denote the partial assignment that behaves like γ, and in addition sets x to 1 (or to 0).

2 The Local Search Algorithm *k*-SAT

In [3], Dantsin et al. give a surprising approach to deciding k-SAT. Let F be a k-CNF and n be the number of variables in F. Let $\{0,1\}^n$ be the set of all possible truth assignments to these variables. We search for a satisfying assignment not in the whole cube $\{0,1\}^n$, but locally in some *Hamming ball* $B_r(\alpha) := \{\beta \in \{0,1\}^n : \ d(\alpha,\beta) \le r\}$ of radius r centered at some $\alpha \in \{0,1\}^n$. We say F is $B_r(\alpha)$-*satisfiable* if $B_r(\alpha)$ contains an assignment satisfying F. We will see below how this can be decided for k-CNFs in time $\mathcal{O}(k^r \operatorname{poly}(n))$. For certain values of r, k^r is much smaller than the volume of $B_r(\alpha)$. By choosing $N(n,r)$ many Hamming balls that together cover $\{0,1\}^n$, we can decide satisfiability of F in time $\mathcal{O}(N(n,r)k^r \operatorname{poly}(n))$. There is of course a trade-off between the radius r of the balls and the number of balls needed to cover $\{0,1\}^n$. Dantsin et al. [3] show how to choose r optimally such that if $B_r(\alpha)$-satisfiability can be decided in $\mathcal{O}(a^r \operatorname{poly}(n))$, satisfiability of F can be decided in $\mathcal{O}\left(\left(2 - \frac{2}{1+a}\right)^n \operatorname{poly}(n)\right)$.

Note that by the symmetry of $\{0,1\}^n$, $B_r(\alpha)$-satisfiability is basically the same problem for each $\alpha \in \{0,1\}^n$. Hence we will assume for the rest of the paper that $\alpha = (1,\ldots,1)$ and write B_r for $B_r\big((1,\ldots,1)\big)$. Algorithm 1, given in Dantsin et al. [3], decides B_r-satisfiability in $\mathcal{O}(k^r \mathrm{poly}(n))$ steps.

Algorithm 1. `searchball`(Formula F, depth r)

```
1: if $F$ contains no negative clause then
2:     return true
3: else if $\square \in F$ or $r \leq 0$ then
4:     return false
5: else
6:     pick some negative clause $\{\bar{x}_1,\ldots,\bar{x}_\ell\} \in F$
7:     return $\bigvee_{i=1}^{\ell}$ searchball($F^{[x_i \mapsto 0]}$, $r-1$)
8: end if
```

Here, a negative clause is a clause containing only negative literals (and thus is not satisfied by $\alpha = (1,\ldots,1)$). Let us see why this algorithm works. The first four lines should be clear: If F contains no negative clause, α satisfies F, and surely $\alpha \in B_r$. Otherwise, if $\square \in F$, then F is clearly unsatisfiable. Also, if α does not satisfy F, and $r = 0$, then F is clearly not B_r-satisfiable.

So much for the base cases. The interesting step is of course the recursion. Consider the negative clause $\{\bar{x}_1,\ldots,\bar{x}_\ell\}$. If there is some satisfying assignment $\alpha^* \in B_r$, it must set some x_i to 0. Let α_i^* be the assignment setting x_i to 1, but else agreeing with α^*. Since $d(\alpha^*,\alpha) \leq r$, it holds that $d(\alpha_i^*,\alpha) \leq r-1$. Note that α_i^* satisfies $F^{[x_i \mapsto 0]}$. Therefore, F' is B_{r-1}-satisfiable, and the recursive call `searchball`$(F^{[x_i \mapsto 0]}, r-1)$ will return `true`. Conversely, if some $F^{[x_i \mapsto 0]}$ is B_{r-1}-satisfiable, it is easy to see that F is B_r-satisfiable.

3 Branching Rules

We say in lines 6 and 7 `searchball` performs a branching. To be more precise, define branchings as follows:

Definition 3.1. *For a partial assignment γ, let $|\gamma|$ denote the number of variables γ sets to* 0. *A* branching for F *is a set $\Gamma = \{\gamma_1,\ldots,\gamma_\ell\}$ of partial assignments with $|\gamma_i| \geq 1$ for all $1 \leq i \leq \ell$.*

Note that we only count variables γ sets to 0. This has two reasons. First, almost no assignment we encounter sets any variable to 1. We will see an exception at the end of the paper, but this will not cause any trouble. Second, we want to measure how far a partial assignment takes us from $\alpha = (1,\ldots,1)$, and setting variables to 1 obviously does not increase the distance from α. The intuition behind the definition of branchings is that `searchball` first computes some branching Γ for F and then recurses on each of the formulas $F^{[\gamma]}$ for each $\gamma \in \Gamma$. The following definition and observation ensure that this is legal, i.e. will give a correct result.

Definition 3.2. *We define* valid *branchings inductively. Let F be a CNF. For every negative clause $\{\bar{x}_1, \ldots, \bar{x}_\ell\} \in F$ the branching*

$$\{[x_1 \mapsto 0], \ldots, [x_\ell \mapsto 0]\}$$

is valid for F. If some branching Γ is valid for F, and there is some $\gamma \in \Gamma$ and a branching $\Gamma' = \{\beta_1, \ldots, \beta_\ell\}$ that is valid for $F^{[\gamma]}$, then

$$\Gamma \setminus \gamma \cup \{\gamma\beta_1, \ldots, \gamma\beta_\ell\}$$

is valid for F.

Here, $\gamma\beta_i$ denotes the combination of both partial assignments. Note that this is well defined, as these two partial assignments refer to disjoint sets of variables. The following observation gives meaning to the previous definition.

Observation. If Γ is a branching valid for F, then F is B_r-satisfiable if and only if there exists some $\gamma \in \Gamma$ such that $F^{[\gamma]}$ is $B_{r-|\gamma|}$-satisfiable.

One might think that these definitions are overly formal, since the notion of branchings in the context of recursive algorithms is a familiar one. However, as our algorithm becomes more involved, it will become clear that it pays to have things defined formally. Using the definition of branchings, we can replace lines 6 and 7 by

```
6: apply some rule to obtain a valid branching $\{\gamma_1, \ldots, \gamma_\ell\}$ for $F$
7: return $\bigvee_{i=1}^{\ell}$ searchball($F^{[\gamma_i]}$, $r - |\gamma_i|$)
```

It is clear that for 3-CNFs, we always find a valid branching containing at most 3 partial assignments, thus `searchball` has a running time of $\mathcal{O}\left(3^r \text{poly}(n)\right)$. Our goal is to decrease the basis of the exponential to some $a < 3$. To achieve this, we first give four relatively simple branching rules for 3-CNFs.

3.1 Simple Branching Rules

Let $\text{Neg}(F)$ denote the set of all negative clauses in F, i.e. clauses without positive literals. Accordingly, the empty clause $\square$ is a negative clause, too. If $\text{Neg}(F)$ consists of pairwise disjoint clauses, we say F is Neg-*disjoint*. From now on, we assume that all negative 3-clauses in F are pairwise disjoint, i.e. that F is Neg-disjoint or contains some negative clause of size at most two. We will show at the end of the paper how to deal with intersecting negative 3-clauses. Let us state four simple rules the algorithm tries to apply. See Figure 1 for an illustration.

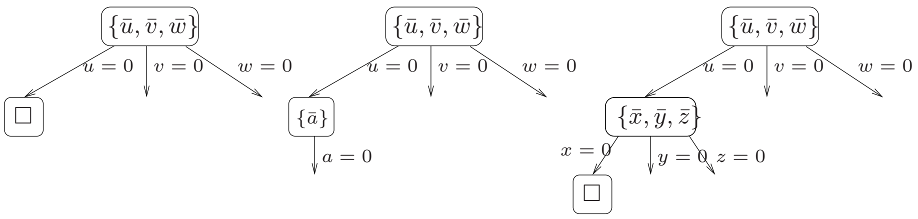
Rule 1. If there is some $\{\bar{x}_1, \ldots, \bar{x}_\ell\} \in \text{Neg}(F)$ with $\ell \leq 2$, then use the branching $\{[x_1 \mapsto 0], \ldots, [x_\ell \mapsto 0]\}$. This includes the case $\square \in F$.

Note that if Rule 1 does not apply, then by assumption F is Neg-disjoint. Clearly, any satisfying assignment needs to set at least $|\mathrm{Neg}(F)|$ variables to 0. Hence if $|\mathrm{Neg}(F)| > r$ at this point, the algorithm immediately returns "not B_r-satisfiable". We assume from now on that $|\mathrm{Neg}(F)| \le r$.

Rule 2. Suppose F contains two clauses of the form $\{u\}$ and $\{\bar{u}, \bar{v}, \bar{w}\}$. Use the branching $\{[v \mapsto 0], [w \mapsto 0]\}$. Note that the partial assignment $[u \mapsto 0]$ need not be part of the branching, because $F^{[u \mapsto 0]}$ contains the empty clause.

Rule 3. Suppose F contains clauses of the form $\{u, \bar{a}\}$ and $\{\bar{u}, \bar{v}, \bar{w}\}$. Use the branching $\{[u \mapsto 0, a \mapsto 0], [v \mapsto 0], [w \mapsto 0]\}$.

Rule 4. Suppose F contains clauses of the form $\{u, x\}$, $\{\bar{x}, \bar{y}, \bar{z}\}$ and $\{\bar{u}, \bar{v}, \bar{w}\}$, the latter two being distinct. Use the branching $\{[u \mapsto 0, y \mapsto 0], [u \mapsto 0, z \mapsto 0], [v \mapsto 0], [w \mapsto 0]\}$. Similarly to the case for Rule 2, the partial assignment $[u \mapsto 0, x \mapsto 0]$ is not part of the branching, since $\Box \in F^{[u \mapsto 0, x \mapsto 0]}$.



$\{\bar{u}, \bar{v}, \bar{w}\}$
$u = 0$
$v = 0$
$w = 0$
$\Box$
$\{\bar{u}, \bar{v}, \bar{w}\}$
$u = 0$
$v = 0$
$w = 0$
$\{\bar{a}\}$
$a = 0$
$\{\bar{u}, \bar{v}, \bar{w}\}$
$u = 0$
$v = 0$
$w = 0$
$\{\bar{x}, \bar{y}, \bar{z}\}$
$x = 0$
$y = 0$
$z = 0$
$\Box$

Fig. 1. Visualization of Rules 2, 3 and 4, respectively

It should be noted that there is nothing new about these branching rules. They all appear in [2], and some appear already in [3]. Each formula occurring in the computation of `searchball`(F,r) is of the form $F^{[\gamma]}$ for some partial assignment γ. In this sense, branching rules extend γ: For example, Rule 3 extends it to $[\gamma, u \mapsto 0, a \mapsto 0]$, $[\gamma, v \mapsto 0]$ and $[\gamma, w \mapsto 0]$ in the recursive calls.

4 Partial Exact Assignments and Guided Search

If some of Rules 1–4 applies to F, then `searchball` applies a branching Γ and calls itself recursively on $F^{[\gamma]}$ for each $\gamma \in \Gamma$. If none of these rules applies to F, we call F *reduced*. This is where the difficult part begins. The approach of [3] and [2] is (briefly) to define additional rules and then prove a non-trivial theorem that if these rules do not apply, there is some other way to decide quickly whether F is B_r-satisfiable. Unfortunately, the additional rule causes a higher running time. Our approach is not completely different, however, we do not introduce any additional rules. Observe that Rules 1–4 might give several valid branchings for the same formula. It turns out that, depending on which branching we decide to apply, we may encounter reduced formulas very often or very rarely. We show that we can find a "guide" formula that tells us which branchings to apply in order to avoid encountering too many reduced formulas. Central to our algorithm will be the following special type of partial assignments:

Definition 4.1. *Let F be a* Neg-*disjoint CNF. A partial assignment γ to the variables of F is called a* partial exact assignment *w.r.t. F, short* pea*, if*

– *it sets no variable to* 1*, and*
– *in each clause $C \in \text{Neg}(F)$, it satisfies at most one literal (i.e. it sets the corresponding variable to* 0*), and*
– *it does not set further variables.*

For example, if $F = \{\{\bar{u}, \bar{v}\}, \{\bar{x}, \bar{y}\}, \{x, \bar{a}\}\}$, then $\gamma = [u \mapsto 0, y \mapsto 0]$ is a pea for F, but $[u \mapsto 0, v \mapsto 0]$ and $[u \mapsto 0, a \mapsto 0]$ are not. Please note that though defined in more general terms, we will use the term pea w.r.t. F only if F is a reduced 3-CNFs. A crucial fact in our algorithm will be that Rules 2 and 4 behave "nicely" with respect to peas. This means, if γ is a pea w.r.t. some F, and Rule 2 or Rule 4 applies to $F^{[\gamma]}$, then the extensions of γ produced by the branching will be peas w.r.t. F, as well. This is because all variables set to 0 in Rule 2 and 4 occur in a 3-clause of $F^{[\gamma]}$. Since applying γ does not create new 3-clauses, these must already have been in F. However, Rules 1 and 3 can produce non-peas, as they set variables to 0 which do not necessarily occur in a 3-clause.

If we encounter a reduced formula F, we cannot apply any of Rules 1–4 and thus have to pick some $\{\bar{x}_1, \bar{x}_2, \bar{x}_3\} \in \text{Neg}(F)$ and recurse on the three formulas $F^{[x_i \mapsto 0]}, i = 1, 2, 3$. This branching rule, if applied over and over again, would amount to a running time of $\mathcal{O}(3^r \text{poly}(n))$. Having applied it once, we would like to make sure that we will not encounter any further reduced formulas in the subsequent recursive calls. This is too much to ask for, but what we definitely do not want is to encounter a reduced formula $F^{[\gamma]}$ where γ is a pea for F. Think of peas as being especially benign and well-behaved partial assignments. We surely do not want these nice peas to bring us into trouble.

Definition 4.2. *We call a computation of* `searchball`(F, r) good *if for any $F^{[\gamma]}$ occurring in the computation with γ being a pea w.r.t. F and $|\gamma| \geq 1$, the formula $F^{[\gamma]}$ is non-reduced.*

We will give a procedure that runs in reasonable time, and for every reduced F, finds either a satisfying assignment in B_r or a good computation. The benefit of a good computation is clear: As long as our branchings produce peas w.r.t. F, we will not encounter reduced formulas. Recall that Rules 2 and 4 never extend a pea γ to a non-pea. Rules 1 and 3 might, but these rules are so efficient that this compensates for the possibility of encountering a reduced formula afterwards.

We will compute a "guide" formula G for which we find a good computation, and then show that the same computation can be performed on F. Let us be more precise: We pick a negative clause $\{\bar{x}_1, \bar{x}_2, \bar{x}_3\}$ in $\text{Neg}(F)$, called the *starting clause* of F. For each $i = 1, 2, 3$, we try to extend the partial assignment $[x_i \mapsto 0]$ to a pea γ w.r.t. F such that $F^{[\gamma]}$ is reduced. We do this by recursively applying Rules 1–4, but aborting the recursion on formulas $F^{[\gamma]}$ if γ is not a pea w.r.t. F, or if γ is a pea and $F^{[\gamma]}$ is reduced. This procedure `searchball-prelim` is given in detail in Algorithm 2. The number of leaves visited by `searchball-prelim` is $\leq f_{r-1}$, where

$$f_i := \begin{cases} 2f_{i-1} + 2f_{i-2} & \text{if } i \geq 1, \\ 1 & \text{if } i \leq 0 \ . \end{cases} \tag{1}$$

Algorithm 2. `searchball-prelim`(Formula F, partial assignment γ, radius r)

```
1: if $\gamma$ is not a pea w.r.t. $F$ then
2:    return undefined
3: else if $F^{[\gamma]}$ is reduced then
4:    return $\gamma$
5: else if $r \leq 0$ then
6:    return undefined
7: else
8:    Apply one of Rules 1–4 and obtain a branching $\{\gamma_1, \ldots, \gamma_\ell\}$
9:    for $1 \leq i \leq \ell$ do
10:       $\gamma'$ := searchball-prelim($F$, $\gamma\gamma_i$, $r - |\gamma_i|$)
11:       if $\gamma' \neq$ undefined then
12:          return $\gamma'$
13:       end if
14:    end for
15:    return undefined
16: end if
```

This can easily be seen by induction on r: If Rule 4 is applied, it causes two calls with $r - 1$ and two with $r - 2$. Rules 1, 2 and 3 are clearly even better. So we call `searchball-prelim`(F, $[x_i \mapsto 0]$, $r - 1$) for each $i = 1, 2, 3$. Suppose each of these three calls returns `undefined`. Then `searchball-prelim` did not encounter any reduced $F^{[\gamma]}$ for γ being a pea, i.e. this was a good computation. Otherwise, let γ_1 be the returned pea and consider $F_2 := F^{[\gamma_1]}$. As for F, pick a negative clause $\{\bar{y}_1, \bar{y}_2, \bar{y}_3\} \in \text{Neg}(F_2)$ as starting clause of F_2 (if there is one). Call `searchball-prelim`(F_2, $[y_i \mapsto 0]$, $r-1$) for each $i = 1, 2, 3$. Either every call returns `undefined`, or some γ_2 is returned. In the latter case, define $F_3 := F_2^{[\gamma_2]}$ and we do for F_3 what we did for F_2. This creates a sequence $F = F_1, F_2, \ldots$ where $F_{i+1} = F_i^{[\gamma_i]}$, γ_i is a pea w.r.t. F_i returned by `searchball-prelim`, and every F_i is reduced. Since each F_{i+1} contains strictly fewer variables than F_i, this process terminates in some $F_m =: G$. Furthermore, it is not difficult to see that $\gamma_1\gamma_2 \ldots \gamma_{m-1}$ is a pea w.r.t. F, and thus $|\gamma_1\gamma_2 \ldots \gamma_{m-1}| \leq r$: Recall that F is Neg-disjoint, and we assume $|\text{Neg}(F)| \leq r$, hence every pea has size $\leq r$. See Figure 2.

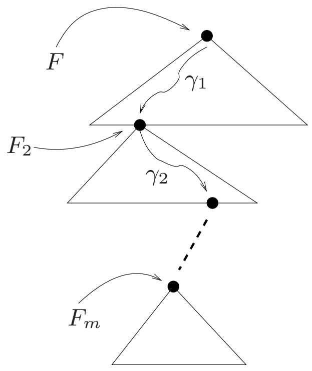


Fig. 2. Constructing a sequence F_1, F_2, $\ldots$, F_m of reduced formulas. The process terminates in F_m. Either $\text{Neg}(F_m) = \emptyset$, or for every γ in the tree of F_m that is a pea w.r.t. F_m, $F_m^{[\gamma]}$ is non-reduced.

There are two cases: First, the process above could terminate with $\text{Neg}(G) = \emptyset$. Then setting all variables in G to 1 satisfies it. Since $G = F^{[\gamma_1\gamma_2\cdots\gamma_{m-1}]}$ and $|\gamma_1\gamma_2\ldots\gamma_{m-1}| \leq r$, F is B_r-satisfiable.

Second, the process could terminate with G for which `searchball-prelim` returned `undefined`. Let us contemplate for a moment what this means. When reaching G in the process described above, we pick a starting clause $\{\bar{z}_1, \bar{z}_2, \bar{z}_3\} \in \text{Neg}(G)$ and call `searchball-prelim`(G, $[z_i \mapsto 0]$, $r-1$) for $i = 1, 2, 3$, and each call returns `undefined`. This means that for any pea γ that occurs in the computation of `searchball-prelim`, $G^{[\gamma]}$ is non-reduced. Therefore, one of Rules 1–4 applies to it, giving a branching Γ. We will show that Γ is valid for $F^{[\gamma]}$, as well, i.e. we can perform the same computation on F instead of G, which will be a good computation of `searchball` on F. We say we use G as a *search guide* for `searchball`(F, r) telling us which branching to apply. Here it is important that the branching in Line 8 of `searchball-prelim` is chosen among all possible branchings by some deterministic rule, such that when performing the same computation for F, we will get exactly the same branching again. This will be a *good* computation, and we will not encounter reduced formulas $F^{[\gamma]}$ as long as γ is a pea w.r.t. G. We need the following technical lemma to show that any branching which is valid for $G^{[\gamma]}$ is also valid for $F^{[\gamma]}$, if γ is a pea w.r.t. G.

Lemma 4.3. *Let F and G be reduced 3-CNFs. Let F_3, G_3 denote the set of all 3-clauses of F and G, respectively. If $G_3 \subseteq F_3$, and γ is a pea w.r.t. G, then* $\text{Neg}(G^{[\gamma]}) \subseteq \text{Neg}(F^{[\gamma]})$.

Proof. Consider any $C \in \text{Neg}(G^{[\gamma]})$. We will show that $C \in \text{Neg}(F^{[\gamma]})$. There is some clause $D \in G$ with $D^{[\gamma]} = C$. If $|D| = 3$, then by assumption $D \in F$ and $C = D^{[\gamma]} \in F^{[\gamma]}$, and we are done. Otherwise, $|D| \leq 2$, and thus D is not a negative clause, because G is reduced. Thus, D is either of the form $\{u\}$, $\{u, \bar{a}\}$ or $\{u, x\}$. If $D = \{u\}$ or $\{u, \bar{a}\}$, then $\gamma(u) = 0$, since $D^{[\gamma]}$ is a negative clause.

Since γ is a pea w.r.t. G, there is a clause $\{\bar{u}, \bar{v}, \bar{w}\}$ in G. Hence Rule 2 or 3 applies to G, contradicting the assumption that G is reduced. If $D = \{u, x\}$, then $C = \emptyset$, and $\gamma(u) = \gamma(x) = 0$. Therefore G contains distinct clauses $\{\bar{u}, \bar{v}, \bar{w}\}$ and $\{\bar{x}, \bar{y}, \bar{z}\}$, and Rule 4 applies. This is again a contradiction. It follows that $|D| = 3$ and $C \in F^{[\gamma]}$, thus $\mathrm{Neg}(G^{[\gamma]}) \subseteq \mathrm{Neg}(F^{[\gamma]})$. $\square$

Lemma 4.4. *Let F and G be reduced 3-CNFs and suppose that $G_3 \subseteq F_3$. If γ is a pea w.r.t. G and one of Rules 1–4 applies to $G^{[\gamma]}$ yielding a branching Γ, then Γ is valid for $F^{[\gamma]}$, as well.*

Proof. Let γ be a pea w.r.t. G. The idea of the proof is the same for each of the four rules, but for the sake of completeness, we will show all four cases.

Case 1. If Rule 1 applies to $G^{[\gamma]}$, then $G^{[\gamma]}$ contains a clause $C = \{\bar{x}_1, \ldots, \bar{x}_\ell\}$ with $\ell \leq 2$. Let $\{[x_1 \mapsto 0], \ldots, [x_\ell \mapsto 0]\}$, $\ell \leq 2$ be the branching. By Lemma 4.3, $C \in F^{[\gamma]}$, thus the branching is valid for $F^{[\gamma]}$.

Case 2. If Rule 2 applies, $G^{[\gamma]}$ contains clauses $C = \{\bar{u}, \bar{v}, \bar{w}\}$ and $D = \{u\}$. By Lemma 4.3, $C \in F^{[\gamma]}$. Note that $\square = D^{[u \mapsto 0]} \in G^{[\gamma, u \mapsto 0]}$. Since $[\gamma, u \mapsto 0]$ is a pea w.r.t. G, too, and we consider the empty clause to be a negative clause, Lemma 4.3 applies again, thus $\square \in F^{[\gamma, u \mapsto 0]}$, and the branching $\{[v \mapsto 0, w \mapsto 0]\}$ is valid for $F^{[\gamma]}$.

Case 3. If Rule 3 applies, there are clauses $\{u, \bar{a}\}$ and $\{\bar{u}, \bar{v}, \bar{w}\}$ in $G^{[\gamma]}$. By Lemma 4.3, $\{\bar{u}, \bar{v}, \bar{w}\} \in F^{[\gamma]}$, as well. Hence the branching $\{[u \mapsto 0], [v \mapsto 0], [w \mapsto 0]\}$ is valid for both $G^{[\gamma]}$ and $F^{[\gamma]}$. Since $[\gamma, u = 0]$ is a pea w.r.t. G, Lemma 4.3 applies and $\{\bar{a}\} \in \mathrm{Neg}(F^{[\gamma, u \mapsto 0]})$. Therefore, the branching $\{[u \mapsto 0, a \mapsto 0], [v \mapsto 0], [w \mapsto 0]\}$ is valid for $F^{[\gamma]}$.

Case 4. If Rule 4 applies, $G^{[\gamma]}$ contains clauses of the form $\{u, x\}$, $\{\bar{x}, \bar{y}, \bar{z}\}$ and $\{\bar{u}, \bar{v}, \bar{w}\}$. By Lemma 4.3, the latter two are in $F^{[\gamma]}$, as well. When applying Rule 4, we do not recurse on $G^{[\gamma, u \mapsto 0, x \mapsto 0]}$, because in this formula, $\{u, x\}$ has become an empty clause. Note that according to our definition, the empty clause is a negative clause, and since $[\gamma, u \mapsto 0, x \mapsto 0]$ is a pea w.r.t. G, Lemma 4.3 implies that $F^{[\gamma, u \mapsto 0, x \mapsto 0]}$ contains the empty clause, too. Therefore the branching $\{[u \mapsto 0, y \mapsto 0], [u \mapsto 0, z \mapsto 0], [v \mapsto 0], [w \mapsto 0]\}$ is valid for $F^{[\gamma]}$. $\square$

Let us summarize our algorithm. If F is not reduced, the algorithm applies one of Rules 1–4. Otherwise, it computes the search guide $G = F^{[\gamma_1 \gamma_2 \cdots \gamma_{m-1}]}$. Let $\{\bar{z}_1, \bar{z}_2, \bar{z}_3\}$ be the starting clause of G. We call `searchball`$(F^{[z_i \mapsto 0]}, r - 1)$ for $i = 1, 2, 3$. As each partial assignment $[z_i \mapsto 0]$ is a pea w.r.t. G, and `searchball-prelim`$(G, [z_i \mapsto 0], r - 1)$ returned `undefined`, $G^{[z_i \mapsto 0]}$ is not reduced, and hence one of Rules 1–4 applies to it, yielding a branching Γ. By Lemma 4.4, this branching is valid for $F^{[z_i \mapsto 0]}$, as well, hence `searchball` applies this very branching in the recursive call `searchball`$(F^{[z_i \mapsto 0]}, r - 1)$. The same argument holds for every subsequent recursive call `searchball`$(F^{[\gamma]}, r')$, as long as γ is a pea w.r.t. G. If `searchball` is called with some $F^{[\gamma]}$ and γ is not a pea w.r.t. G, we have to discard our search guide. In this case, it may happen that $F^{[\gamma]}$ is reduced again, and we would have to run `searchball-prelim` on $F^{[\gamma]}$, to find a new search guide.

We will now analyze the running time. It turns out that Rule 4 is the "worst case rule" that dominates the running time of the algorithm. However, we have to be careful in our calculations because we do a lot of additional work in `searchball-prelim`. We have to make sure that the running time is still dominated by f_r, defined in (1).

Theorem 4.5. *If F contains no intersecting negative 3-clauses, the number of leaves visited by* `searchball`(F, r) *is* $\leq 3(r+1)^2 f_{r-1}$.

Proof. We prove a stronger statement. We claim in addition that if F is reduced and G is used as a search guide for `searchball`, and γ is a pea w.r.t. G, then `searchball`$(F^{[\gamma]}, r)$ visits $\leq (r+1)^2 f_r$ leaves.

We use induction on r. For $r = 0$, the statement is trivial. If F is reduced, we compute a search guide formula G. Doing this, we call `searchball-prelim` $\leq 3r$ times, each time creating $\leq f_{r-1}$ leaves. Then we pick a clause $\{\bar{y}_1, \bar{y}_2, \bar{y}_3\} \in \mathrm{Neg}(G)$ and call `searchball`$(F^{[y_i \mapsto 0]}, r-1)$ for $i = 1, 2, 3$. Since each $[y_i \mapsto G]$ is a pea w.r.t. G, by induction each call causes $\leq r^2 f_{r-1}$ leaves. Together, this amounts to $\leq 3(r+1)^2 f_{r-1}$ calls.

If F is not reduced and we are not using a search guide, then one of Rules 1–4 applies, and it is straightforward to show that the number of leaves is $\leq 3(r+1)^2 f_{r-1}$.

The most interesting case is when `searchball` is called for $F^{[\gamma]}$, using G as a search guide, and γ is a pea w.r.t. G. If Rule 4 applies, we pick clauses $\{u, x\}$, $\{\bar{x}, \bar{y}, \bar{z}\}$ and $\{\bar{u}, \bar{v}, \bar{w}\} \in G^{[\gamma]}$ and use the branching $\{[u \mapsto 0, y \mapsto 0], [u \mapsto 0, z \mapsto 0], [v \mapsto 0], [w \mapsto 0]\}$. Recall that all four extended assignments are peas w.r.t. G, hence the recursive calls cause $\leq 2r^2 f_{r-1} + 2(r-1)^2 f_{r-2} \leq (r+1)^2 f_r$ leaves. For Rule 2, the argument is exactly the same. This is really the crucial point in this algorithm: Of all four rules, Rule 4 yields the worst running time. However, using our search guide, we can be sure not to encounter a reduced formula after applying Rule 4.

If Rule 3 applies, we pick $\{u, \bar{a}\}, \{\bar{u}, \bar{v}, \bar{w}\} \in G^{[\gamma]}$ and use the branching $\{[u \mapsto 0, a \mapsto 0], [v \mapsto 0], [w \mapsto 0]\}$. Note that $\gamma[v \mapsto 0]$ and $\gamma[w \mapsto 0]$ are peas w.r.t. G, hence these calls cause $\leq 2r^2 f_{r-1}$ leaves. However, $\gamma[u \mapsto 0, a \mapsto 0]$ is perhaps not a pea w.r.t. G, hence $F^{[u \mapsto 0, a \mapsto 0]}$ might be reduced, and this call causes $\leq 3(r-1)^2 f_{r-3}$ leaves. Altogether, this is surely $\leq 2(r+1)^2 f_{r-1} + 3(r+1)^2 f_{f-3}$, which is $\leq (r+1)^2 f_r$. If Rule 1 applies, we cause $\leq 2 \times 3r^2 f_{r-2} \leq 3(r+1)^2 f_{r-1}$ leaves. This completes the proof. □

To summarize, computing and using a search guide guarantees that reduced formulas might be encountered once, but in subsequent calls, they will be encountered only after Rule 1 or Rule 3 has been applied. These rules are so efficient that they compensate for the possibility of encountering a reduced formula afterwards. To complete our algorithm, we have to show finally what to do when F is not Neg-disjoint. We basically do the same as Brueggemann and Kern [2].

Theorem 4.6. *Let F be a 3-CNF. The number of leaves visited by* `searchball`(F, r) *is* $\leq 3(r+1)^2 f_{r-1}$.

Proof. If F contains a negative clause of size ≤ 2, we obtain by induction that `searchball`(F, r) causes $\leq 2 \times 3r^2 f_{r-2} \leq 3(r+1)^2 f_{r-1}$ leaves. Otherwise, all negative clauses are of size three. There are three cases:

First, there could be clauses $\{\bar{u}, \bar{v}, \bar{w}\}$, $\{\bar{u}, \bar{v}, \bar{z}\}$ intersecting in exactly two literals. We use the branching $\{[u \mapsto 0], [u \mapsto 1, v \mapsto 0], [u \mapsto 1, v \mapsto 1, w \mapsto 0, z \mapsto 0]\}$. Though not valid according to our definition, it still holds that F is B_r-satisfiable if and only if $F^{[\gamma]}$ is $B_{r-|\gamma|}$-satisfiable, for some γ in the branching. The claimed time bound follows after a short computation.

Second, if F contains two 3-clauses $\{\bar{u}, \bar{v}, \bar{w}\}$, $\{\bar{u}, \bar{y}, \bar{z}\}$ intersecting in exactly one literal, use the branching $\{[u \mapsto 0], [u \mapsto 1, v \mapsto 0, y \mapsto 0], [u \mapsto 1, v \mapsto 0, z \mapsto 0], [u \mapsto 1, w \mapsto 0, y \mapsto 0], [u \mapsto 1, w \mapsto 0, z \mapsto 0]\}$. Again, a short calculation shows that this causes $\leq 3(r+1)^2 f_{r-1}$ leaves.

Third, it could be that F does not contain intersecting negative clauses. Then Theorem 4.5 applies. This completes the proof. □

It is standard to show that $f_r \in \mathcal{O}(a^r)$, where $a \approx 2.74$ is the largest root of $x^2 - 2x - 2$. Therefore, B_r-satisfiability can be decided in time $\mathcal{O}(a^r \text{poly}(n))$, and thus, by the results of Danstin et al., we can decide 3-SAT in time $\mathcal{O}\left(1.465^n \text{poly}(n)\right)$.

5 Conclusions

Observe that Lemma 4.3, though looking innocent, is really the core reason why our algorithm works. It is also what causes trouble when ones tries to directly apply guided search to other backtracking algorithms. Take for example Beigel and Eppstein's Algorithm [1] for solving (3,2)-CSP. For this algorithm we cannot find and apply an equivalent of Lemma 4.3, because the algorithm uses some kind of resolution which introduces new constraints, whereas our application of Lemma 4.3 relies crucially on the fact that if G was created from F by steps of the algorithm, then G does not contain any 3-clauses that F does not contain.

For traditional backtracking algorithms for k-SAT, often called *DPLL algorithms*, after Davis, Putnam, Logemann and Loveland [4,5], there is an even simpler technique than guided search. Consider a backtracking algorithm that chooses a shortest clause $C = \{u_1, \ldots, u_i\} \in F$ and recurses on $F_1 := F^{[u_1 \mapsto 1]}$ and $F_0 := F^{[u_1 \mapsto 0]}$, where F_0 contains an $(i-1)$-clause. In this context, call a formula *reduced* if every clause in F has size exaclty k. It is clear that if F is a k-CNF and in the recursive computation of F, a reduced formula $F^{[\gamma]}$ occurs, then $F^{[\gamma]} \subset F$, and the two formulas are *SAT-equivalent* in the sense that one is satisfiable if and only if the other is. Hence all other open branches of the search tree can be pruned, and the algorithm only needs to recurse on $F^{[\gamma]}$. This is known as the *autark assignment rule* and was used by Monien and Speckenmeyer [6] to speed up their k-SAT algorithm.

It should be mentioned that we first tried to prove some kind of autarky result, i.e. that if F is reduced and $F^{[\gamma]}$ is reduced as well, then one formula is B_r-satisfiable if and only if the other is. Unfortunately, this is not true. One can, however, obtain a SAT-equivalence under certain conditions stronger than reducedness, which leads to a simpler proof of the $\mathcal{O}(1.473^n)$-bound of Brueggemann and Kern [2].

Acknowledgements

I want to thank Andreas Razen and Emo Welzl for fruitful and thorough discussions, which greatly helped to improve this paper. Further I am grateful for the comments and suggestions of the referees.

References

1. Beigel, R., Eppstein, D.: 3-coloring in time $O(1.3446^n)$: a no-MIS algorithm. In: Proc. 36th Symp. Foundations of Computer Science, October 1995, pp. 444–453. IEEE, Los Alamitos (1995)
2. Brueggemann, T., Kern, W.: An improved local search algorithm for 3-SAT. Memorandum 1709, Department of Applied Mathematics, University of Twente, Enschede (2004)
3. Dantsin, E., Goerdt, A., Hirsch, E.A., Kannan, R., Kleinberg, J., Papadimitriou, C., Raghavan, O., Schöning, U.: A deterministic $(2 - 2/(k + 1))^n$ algorithm for k-SAT based on local search. Theoretical Computer Science 289, 69–83 (2002)
4. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Comm. ACM 5, 394–397 (1962)
5. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. Assoc. Comput. Mach. 7, 201–215 (1960)
6. Monien, B., Speckenmeyer, E.: Solving satisfiability in less than 2^n steps. Discrete Applied Mathematics 10, 287–295 (1985)
7. Rolf, D.: Improved bound for the PPSZ/Schöning-algorithm for 3-SAT. Electronic Colloquium on Computational Complexity (ECCC) 159 (2005)

Comparing and Aggregating Partially Resolved Trees[⋆]

Mukul S. Bansal, Jianrong Dong, and David Fernández-Baca

Department of Computer Science, Iowa State University, Ames, IA, USA
{bansal,jrdong,fernande}@cs.iastate.edu

Abstract. We define, analyze, and give efficient algorithms for two kinds of distance measures for rooted and unrooted phylogenies. For rooted trees, our measures are based on the topologies the input trees induce on *triplets*; that is, on three-element subsets of the set of species. For unrooted trees, the measures are based on *quartets* (four-element subsets). Triplet and quartet-based distances provide a robust and fine-grained measure of the similarities between trees. The distinguishing feature of our distance measures relative to traditional quartet and triplet distances is their ability to deal cleanly with the presence of unresolved nodes, also called polytomies. For rooted trees, these are nodes with more than two children; for unrooted trees, they are nodes of degree greater than three.

Our first class of measures are parametric distances, where there is parameter that weighs the difference between an unresolved triplet/quartet topology and a resolved one. Our second class of measures are based on Hausdorff distance. Each tree is viewed as a set of all possible ways in which the tree could be refined to eliminate unresolved nodes. The distance between the original (unresolved) trees is then taken to be the Hausdorff distance between the associated sets of fully resolved trees, where the distance between trees in the sets is the triplet or quartet distance, as appropriate.

1 Introduction

Evolutionary trees, also known as phylogenetic trees or phylogenies, represent the evolutionary history of sets of species. Such trees have uniquely labeled leaves, corresponding to the species, and unlabeled internal nodes, representing hypothetical ancestors. The trees may be rooted, if the evolutionary origin is known, or unrooted, otherwise.

This paper addresses two related questions: (1) How does one measure how close two evolutionary trees are to each other? (2) How does one combine or *aggregate* the phylogenetic information from conflicting trees into a single *consensus tree*? Among the motivations for the first question is the growth of phylogenetic databases, such as TreeBase [19], with the attendant need for sophisticated querying mechanisms and for means to assess the quality of answers to queries. The second question arises from the fact that phylogenetic analyses — e.g., by parsimony — typically produce multiple evolutionary trees (often in the thousands) for the same set of species.

We address the above questions by defining appropriate *distance measures* between trees. While several such measures have been proposed before (see below), ours provide a feature that previous ones do not: The ability to deal elegantly with the presence

⋆ This work was supported in part by National Science Foundation AToL grant EF-0334832.

of *unresolved* nodes, also called *polytomies*. For rooted trees these are nodes with more than two children; for unrooted trees, they are nodes of degree greater than three. Polytomies cannot simply be ignored, since they arise naturally in phylogenetic analysis. Furthermore, they must be treated with care: A node may be unresolved because it truly must be so or because there is not enough evidence to break it up into resolved nodes — that is, the polytomies are either "hard" or "soft" [17].

Our Contributions. We define and analyze two kinds of distance measures for phylogenies. For rooted trees, our measures are based on the topologies the input trees induce on *triplets*; that is, on three-element subsets of the set of species. For unrooted trees, the measures are based on *quartets* (four-element subsets). Our approach is motivated by the observation that triplet and quartet topologies are the basic building blocks of rooted and unrooted trees, in the sense that they are the smallest topological units that completely identify a phylogenetic tree [21]. Triplet and quartet-based distances thus provide a robust and fine-grained measure of the differences and similarities between trees[1]. In contrast with traditional quartet and triplet distances, our two classes of distance measures deal cleanly with the presence of unresolved nodes.

The first kind of measures we propose are *parametric distances*: Given a triplet (quartet) X, we compare the topologies that each of the two input trees induces on X. If they are identical, the contribution of X to the distance is zero. If both topologies are fully resolved but different, then the contribution is one. Otherwise, the topology is resolved in one of the trees, but not the other. In this case, X contributes p to the distance, where p is a real number between 0 and 1. Parameter p allows one to make a smooth transition between hard and soft views of polytomy. At one extreme, if $p = 1$, an unresolved topology is viewed as different from a fully resolved one. At the other, when $p = 0$, unresolved topologies are viewed as identical to resolved ones. Intermediate values of p allow one to adjust for the degree of certainty one has about a polytomy.

The second kind of measures proposed here are based on viewing each tree as a set of all possible fully resolved trees that can be obtained from it by refining its unresolved nodes. The distance between two trees is defined as the Hausdorff distance between the corresponding sets, where the distance between trees in the sets is the triplet or quartet distance, as appropriate.

After defining our distance measures, we proceed to study their mathematical and algorithmic properties. We obtain exact and asymptotic bounds on expected values of parametric triplet distance and parametric quartet distance. We also study for which values of p, parametric triplet and quartet distances are metrics, *near-metrics* (in the sense of [15]), or non-metrics.

Aside from the mathematical elegance that metrics and near-metrics bring to tree comparison, there are also algorithmic benefits. We formulate phylogeny aggregation as a *median* problem, in which the objective is to find a consensus tree whose total distance to the given trees is minimized. We do not know whether finding the median tree relative to parametric (triplet or quartet) distance is NP-hard, but conjecture that it is. This is suggested by the NP-completeness of the maximum triplet compatibility problem (see [8]). However, by the results mentioned above and well-known facts about

[1] Biologically-inspired arguments in favor of triplet-based measures can be found in [11].

the median problem [26], there is a simple constant-factor approximation algorithm for aggregating rooted and unrooted trees relative to parametric distance: Simply return the input tree with minimum distance to the remaining input trees. We show that there are values of p for which parametric distance is a metric, but the median tree may not be fully resolved even if all the input trees are. However, beyond a threshold, the median tree is guaranteed to be fully resolved if the input trees are fully resolved.

We suspect that computing Hausdorff triplet (quartet) distance between two trees is NP-hard. However, we show that one can partially circumvent the issue by proving that, under a certain density assumption, Hausdorff distance is within a constant factor of parametric distance — that is, the measures are *equivalent* in the sense of [15].

Finally, we present a $O(n^2)$-time algorithm to compute parametric triplet distance and a $O(n^2)$ 2-approximate algorithm for parametric quartet distance. To our knowledge, there was no previous algorithm for computing the parametric triplet distance between two rooted trees, other than by enumerating all $\Theta(n^3)$ triplets. Two algorithms exist that can be directly applied to compute the parametric quartet distance. One runs in time $O(n^2 \min\{d_1, d_2\})$, where, for $i \in \{1, 2\}$, d_i is the maximum degree of a node in T_i [10]; the other takes $O(d^9 n \log n)$ time, where d is the maximum degree of a node in T_1 and T_2 [24].[2] Our faster $O(n^2)$ algorithm offers a 2-approximate solution when an exact value of the parametric quartet distance is not required. Additionally, our algorithm gives the exact answer when $p = \frac{1}{2}$.

Related Work. Several other measures for comparing trees have been proposed; we mention a few. A popular class of distances are those based on symmetric distance between sets of *clusters* (that is, on sets of species that descend from the same internal node in a rooted tree) or of *splits* (partitions of the set of species induced by the removal of an edge in an unrooted tree); the latter is the well-known Robinson-Foulds (RF) distance [20]. It is not hard to show that two rooted (unrooted) trees can share many triplet (quartet) topologies but not share a single cluster (split). Cluster- and split-based measures are also coarser than triplet and quartet distances.

One can also measure the distance between two trees by counting the number of *branch-swapping* operations needed to convert one of the trees into the other [2]. However, the associated measures can be hard to compute, and they fail to distinguish between operations that affect many species and those that affect only a few. An alternative to distance measures are *similarity* methods such as maximum agreement subtree (MAST) approach [16]. While there are efficient algorithms for computing the MAST, the measure is coarser than triplet-based distances.

There is an extensive literature on consensus methods for phylogenetic trees. A non-exhaustive list of methods based on splits or clusters includes strict consensus trees [18], majority-rule trees [3], and the Adams consensus [1]. For a thorough survey on the subject, see [9].

The fact that consensus methods tend to produce unresolved trees, with an attendant loss of information, has been observed before. An alternative approach is to cluster the input trees into groups using some distance measure, each of which is represented by a consensus tree, in such a way as to minimize some measure of information loss [25].

[2] Note that unresolved nodes seem to complicate distance computation: The quartet distance between a pair of *fully resolved* unrooted trees can be obtained in $O(n \log n)$ time [7].

In addition to consensus methods, there are techniques that take as input sets of quartet trees or triplet trees and try to find large compatible subsets or subsets whose removal results in a compatible set [5,22]. These problems are related to the *supertree problem*, which generalizes the consensus problem by allowing the leaves of the input trees to overlap only partially [6].

The consensus problem on trees exhibits parallels with the *rank aggregation problem* [14,15]. Here we are given a collection of rankings (that is, permutations) of n objects, and the goal is to find a ranking of minimum total distance to the input rankings. A distance between rankings of particular interest is *Kendall's tau*, defined as the number of pairwise disagreements between the two rankings. Like triplet and quartet distances, Kendall's tau is based on elementary ordering relationships. Rank aggregation under Kendall's tau is NP-complete even for four lists [14].

A permutation is the analog of a fully resolved tree, since every pairwise relationship between elements is given. The analog to a partially-resolved tree is a *partial ranking*, in which the elements are grouped into an ordered list of *buckets*, such that elements in different buckets have known ordering relationships, but elements within a bucket are not ranked [15]. Our definitions of parametric distance and Hausdorff distance are inspired by Fagin et al.'s *Kendall tau with parameter* p and their Hausdorff version of Kendall's tau, respectively [15]. We note, however, that aggregating partial rankings seems computationally easier than the consensus problem on trees. For example, while the Hausdorff version of Kendall's tau is easily computable [15], it is unclear whether the Hausdorff triplet or quartet distances are polynomially-computable for trees.

Organization of the Paper. Section 2 reviews basic notions in phylogenetics and distances. Our distance measures and the consensus problem are formally defined in Section 3. The basic properties of parametric distance are proved in Section 4. Section 5 studies the connection between Hausdorff and parametric distances. Section 6 gives efficient algorithms for computing parametric distance.

2 Preliminaries

Phylogenies. By and large, we follow standard terminology (i.e., similar to [21]). We write $[N]$ to denote the set $\{1, 2, \ldots, N\}$, where N is a positive integer.

Let T be a rooted or unrooted tree. We write $\mathcal{V}(T)$, $\mathcal{E}(T)$, and $\mathcal{L}(T)$ to denote, respectively, the node set, edge set, and leaf set of T. A *taxon* (plural *taxa*) is some basic unit of classification; e.g., a species. Let S be a set of taxa. A *phylogenetic tree* or *phylogeny* for S is a tree T such that $\mathcal{L}(T) = S$. Furthermore, if T is rooted, we require that every internal node have at least two children; if T is unrooted, every internal node is required to have degree at least three. We write $RP(n)$ and $P(n)$ to denote, respectively, the sets of all rooted and unrooted phylogenetic trees over $S = [n]$.

An internal node in a *rooted* phylogeny is *resolved* if it has exactly two children; otherwise it is *unresolved*. Similarly, an internal node in an *unrooted* phylogeny is *resolved* if it has degree three, and *unresolved* otherwise. Unresolved nodes in rooted and unrooted trees are also referred to as *polytomies* or *multifurcations*. A phylogeny (rooted or unrooted) is *fully resolved* if all its internal nodes are resolved.

A *contraction* of a phylogeny T is obtained by deleting an internal edge and identifying its endpoints. A phylogeny T_2 *refines* phylogeny T_1 if and only if T_1 can be obtained from T_2 through 0 or more contractions. T_2 is a *full refinement* of T_1 if T_2 is a fully resolved tree that refines T_1. $\mathcal{F}(T)$ denotes the set of all full refinements of T.

Let X be a subset of $\mathcal{L}(T)$ and let $T[X]$ denote the minimal subtree of T having X as its leaf set. The *restriction* of T to X, denoted $T|X$, is the phylogeny for X defined as follows. If T is unrooted, then $T|X$ is the tree obtained from $T[X]$ by suppressing all degree-two nodes. If T is rooted, $T|X$ is obtained from $T[X]$ by suppressing all degree-two nodes except for the root.

A *triplet* is a three-element subset of S; a *quartet* is a four-element subset of S. A triplet (quartet) X is said to be *resolved* in a phylogenetic tree T over S if $T|X$ is fully resolved; otherwise, X is *unresolved*.

Finally, we need some special notation for rooted trees T. We write *rt*(T) to denote the root node of T. Let v be a node in T. Then, *pa*(v) denotes the parent of v in T and *Ch*(v) is the set of children of v. Furthermore, $T(v)$ denotes the subtree of T rooted at v and $\overline{T(v)}$ denotes the tree obtained by deleting $T(v)$ from T, as well as the edge from v to its parent, if such an edge exists.

Distance Measures, Metrics, and Near-metrics. A *distance measure* on a set D is a binary function d on D satisfying the following three conditions: (i) $d(x, y) \geq 0$ for all $x, y \in D$; (ii) $d(x, y) = d(y, x)$ for all $x, y \in D$; and (iii) $d(x, y) = 0$ if and only if $x = y$. Function d is a *metric* if, in addition to being a distance measure, it satisfies the triangle inequality; i.e., $d(x, z) \leq d(x, y) + d(y, z)$ for all $x, y, z \in D$. Distance measure d is a *near-metric* if there is a constant c, independent of the size of D, such that d satisfies the *relaxed polygonal inequality*: $d(x, z) \leq c(d(x, x_1) + d(x_1, x_2) + \cdots + d(x_{n-1}, z))$ for all $n > 1$ and $x, z, x_1, \ldots, x_{n-1} \in D$ [15]. Two distance measures d and d' with domain D are *equivalent* if there are constants $c_1, c_2 > 0$ such that $c_1 d'(x, y) \leq d(x, y) \leq c_2 d'(x, y)$ for every pair $x, y \in D$ [15].

3 Distance Measures for Phylogenies

Let T_1 and T_2 be any two rooted (unrooted) phylogenies over the same taxon set S. We partition the set of triplets (quartets) over S into the following five sets.[3]

1. $\mathcal{S}(T_1, T_2)$: triplets (quartets) X that are resolved in T_1 and T_2, and $T_1|X = T_2|X$.
2. $\mathcal{D}(T_1, T_2)$: triplets (quartets) X that are resolved in T_1 and T_2, but $T_1|X \neq T_2|X$.
3. $\mathcal{R}_1(T_1, T_2)$: triplets (quartets) that are resolved in T_1, but not in T_2.
4. $\mathcal{R}_2(T_1, T_2)$: triplets (quartets) that are resolved in T_2, but not in T_1.
5. $\mathcal{U}(T_1, T_2)$: triplets (quartets) that are unresolved in both T_1 and T_2.

Let p be a real number in the interval $[0, 1]$. The *parametric triplet (quartet) distance between* T_1 *and* T_2 is defined as

$$d^{(p)}(T_1, T_2) = |\mathcal{D}(T_1, T_2)| + p\left(|\mathcal{R}_1(T_1, T_2)| + |\mathcal{R}_2(T_1, T_2)|\right). \tag{1}$$

[3] Note that the sets $\mathcal{S}(T_1, T_2)$ and $\mathcal{U}(T_1, T_2)$ are not used in this section, but are needed later.

Parameter p allows one to make a smooth transition from soft to hard views of polytomy: When $p = 0$, resolved triplets (quartets) are treated as equal to unresolved ones, while when $p = 1$, they are treated as being completely different. Intermediate values of p allow one to adjust for the amount of evidence required to resolve a polytomy.

Let d be a metric over fully resolved trees. Metric d can be extended to partially resolved trees via *Hausdorff distance*, as follows.

$$d_{\text{Haus}}(T_1, T_2) = \max\left\{ \max_{t_1 \in \mathcal{F}(T_1)} \min_{t_2 \in \mathcal{F}(T_2)} d(t_1, t_2), \max_{t_2 \in \mathcal{F}(T_2)} \min_{t_1 \in \mathcal{F}(T_1)} d(t_1, t_2) \right\} \quad (2)$$

When d is the triplet (quartet) distance, d_{Haus} is called the *Hausdorff triplet (quartet) distance*. In Equation (2), $\max_{t_1 \in \mathcal{F}(T_1)} \min_{t_2 \in \mathcal{F}(T_2)} d(t_1, t_2)$ gives the maximum distance between a full refinement of T_1 and the set of full refinements of T_2. Similarly, $\max_{t_2 \in \mathcal{F}(T_2)} \min_{t_1 \in \mathcal{F}(T_1)} d(t_1, t_2)$ is the maximum distance between a full refinement of T_2 and the set of full refinements of T_1. Therefore, T_1 and T_2 are at Hausdorff distance r of each other if every full refinement of T_1 is within distance r of a full refinement of T_2 and vice-versa.

Aggregating Phylogenies. Let k be a positive integer and S be a set of taxa. A *profile of length k* (or simply a *profile*) is a mapping $\mathcal{P}$ that assigns each $i \in [k]$ a phylogenetic tree $\mathcal{P}(i)$ over S. We refer to these trees as *input trees*. A *consensus rule* is a function that maps a profile $\mathcal{P}$ to some phylogenetic tree T over S called a *consensus tree*.

Let d be a distance measure whose domain is the set of phylogenies over S. We extend d to define a distance measure from profiles to phylogenies as $d(T, \mathcal{P}) = \sum_{i=1}^{k} d(T, \mathcal{P}(i))$. A consensus rule is a *median rule* for d if for every profile $\mathcal{P}$ it returns a phylogeny T^* of minimum distance to $\mathcal{P}$; such a T^* is called a *median*. The problem of finding a median for a profile with respect to a distance measure d is referred to as the *median problem* (relative d), or as the *aggregation problem*.

4 Properties of Parametric Distance

In what follows, unless mentioned explicitly, whenever we refer to parametric distance, we mean both its triplet and quartet varieties. We begin with a useful observation.

Proposition 1. *For every p, q such that $p, q \in (0, 1]$, $d^{(p)}$ and $d^{(q)}$ are equivalent.*

The proof of the next theorem is along the lines of an analogous result for aggregating partial rankings by Fagin et al. [15] and is omitted from this extended abstract.

Theorem 1. *(1) For $p = 0$, $d^{(p)}$ is not a distance measure. (2) For $0 < p < 1/2$, $d^{(p)}$ is a near-metric, but not a metric. (3) For $p \geq 1/2$, $d^{(p)}$ is a metric.*

Part (3) of Theorem 1 leads directly to approximation algorithms. Part (2) indicates that the measure degrades nicely, since constant factor approximation ratios are also achievable with near-metrics [15].

The next result establishes a threshold for p beyond which a collection of fully resolved trees give enough evidence to produce a fully resolved tree.

Theorem 2. *Let $\mathcal{P}$ be a profile of length k, such that for all $i \in [k]$, tree $\mathcal{P}(i)$ is fully resolved. Then, if $p \geq 2/3$, there exists median tree T for $\mathcal{P}$ relative to $d^{(p)}$ such that T is fully resolved.*

Proof (sketch). Suppose T is a median tree that contains an unresolved node v. The key idea is to show that there is a way to refine v into two nodes such that the number of input triplet (quartet) topologies with which the resulting tree disagrees is at most twice the number with which it agrees. The theorem follows by applying this refinement step repeatedly, until a fully resolved tree is obtained. □

We can, in fact, show that if $p > 2/3$ and the input trees are fully resolved, the median tree relative to $d^{(p)}$ *must* be fully resolved. On the other hand, it is easy to show that when $p \in [1/2, 2/3)$, there are profiles of fully resolved trees whose median tree is only partially resolved.

It is interesting to compare Theorem 2 with analogous results for partial rankings. Consider the variation of Kendall's tau for partial rankings in which a pair of items that is ordered in one ranking but is in the same bucket in the other contributes p to the distance, where $p \in [0, 1]$. This distance measure is a metric when $p \geq 1/2$ [15]. Furthermore, if $p \geq 1/2$ the median ranking relative to this distance is a full ranking if the input consists of full rankings [4]. In contrast, Proposition 1 and Theorem 2 show that, for $p \in [1/2, 2/3]$, parametric triplet or quartet distance are metrics, but the median tree is not guaranteed to be fully resolved even if the input trees are. This opens up a range of values for p wherein parametric triplet/quartet distance is a metric, but where one can adjust for the degree of evidence needed to resolve a node.

We now consider the expected value of parametric triplet and quartet distances.

Theorem 3. *Let $u(n)$ and $r(n)$ denote the probabilities that a given quartet is, respectively, unresolved or resolved in an unrooted phylogeny chosen uniformly at random from $P(n)$. Then,*

(i) $E(d^{(p)}(T_1, T_2)) = \binom{n}{4} \cdot \left(\frac{2}{3} \cdot r(n)^2 + 2 \cdot p \cdot r(n) \cdot u(n)\right)$, *if T_1 and T_2 are* unrooted *phylogenies chosen uniformly at random with replacement from $P(n)$, and*

(ii) $E(d^{(p)}(T_1, T_2)) = \binom{n}{3} \cdot \left(\frac{2}{3} \cdot r(n+1)^2 + 2 \cdot p \cdot r(n+1) \cdot u(n+1)\right)$, *if T_1 and T_2 are* rooted *phylogenies chosen uniformly at random with replacement from $RP(n)$.*

Part (i) of Theorem 3 follows directly from [13,23]. Part (ii) follows from part (i) and the relationship between rooted and unrooted trees [21]. Since $u(n) \sim \sqrt{\frac{\pi(2\ln 2 - 1)}{4n}}$ [23] and $r(n) = 1 - u(n)$, Theorem 3 implies that $E(d^{(p)}(T_1, T_2))$ is asymptotically $\frac{2}{3} \cdot \binom{n}{4}$ for unrooted trees and $\frac{2}{3} \cdot \binom{n}{3}$ for rooted trees.

5 Relationships Among the Metrics

We do not know whether the Hausdorff triplet or quartet distances are computable in polynomial time. Indeed, we suspect that, unlike its counterpart for partial rankings, this may not be possible. On the positive side, we show here that, in a broad range of cases, it is possible to obtain an approximation to the Hausdorff distance by exploiting its connection with parametric distance. As in the previous section, our results apply to both triplet and quartet distances.

Lemma 1. *For every two phylogenies T_1 and T_2 over S, $|\mathcal{D}(T_1, T_2)| + \frac{2}{3} \cdot \max\{|\mathcal{R}_1(T_1, T_2)|, |\mathcal{R}_2(T_1, T_2)|\} \leq d_{\text{Haus}}(T_1, T_2) \leq |\mathcal{D}(T_1, T_2)| + |\mathcal{R}_1(T_1, T_2)| + |\mathcal{R}_2(T_1, T_2)| + |\mathcal{U}(T_1, T_2)|$.*

Proof (sketch). The proof of the lower bound on d_{Haus} is in two steps. We first show that T_1 can be refined so that it disagrees with T_2 in at least two thirds of the triplets (quartets) in $\mathcal{R}_2(T_1, T_2)$. Next, we show the existence of an analogous refinement of T_2. Note that the triplets (quartets) in $\mathcal{D}(T_1, T_2)$ are resolved differently in any refinements of T_1 and T_2. This gives lower bounds for both arguments in the outer $\max$ of the definition of $d_{\text{Haus}}(T_1, T_2)$ (Equation 2) and yields the lemma.

The upper bound follows by assuming that T_1 and T_2 are refined so that the triplets (quartets) in $\mathcal{R}_1(T_1, T_2)$, $\mathcal{R}_2(T_1, T_2)$, and $\mathcal{U}(T_1, T_2)$ are resolved differently. □

It is instructive to compare Lemma 1 with the situation for partial rankings. In the Hausdorff version of Kendall's tau, each partial ranking is viewed as the set of all possible full rankings that can be obtained by refining it (that is, ordering elements within buckets). The distance is then the Hausdorff distance between the two sets, where the distance between two elements is Kendall's tau. Let L_1 and L_2 be two partial rankings. Re-using notation, let $\mathcal{D}(L_1, L_2)$ be the set of all pairs that are ordered differently in L_1 and L_2, $\mathcal{R}_1(L_1, L_2)$ be the set of pairs that are ordered in L_1 but in the same bucket in L_2, and $\mathcal{R}_2(L_1, L_2)$ be the set of pairs that are ordered in L_2 but in the same bucket in L_1. Then, $d_{\text{Haus}}(L_1, L_2) = |\mathcal{D}(L_1, L_2)| + \max\{|\mathcal{R}_1(L_1, L_2)|, |\mathcal{R}_2(L_1, L_2)|\}$ [12,15]. This expression leads to an efficient way to compute $d_{\text{Haus}}(L_1, L_2)$ and establishes an equivalence with the parametric version of Kendall's tau defined in Section 4 [15]. It seems unlikely that a similar simple expression can be obtained for Hausdorff triplet or quartet distance. There are at least two reasons for this. Let L_1 and L_2 be partial rankings. Then, it is possible to resolve L_1 so that it disagrees with L_2 in any pair in $\mathcal{R}_2(L_1, L_2)$. Similarly, there is a way to resolve L_2 so that it disagrees with L_1 in any pair in $\mathcal{R}_1(L_1, L_2)$. An analog for trees cannot be established for this property; hence, the $\frac{2}{3}$ factor in the lower bound of Lemma 1. The second reason is due to the properties of the set $\mathcal{U}(L_1, L_2)$. It can be shown that is one can refine L_1 and L_2 in such a way that pairs of elements that are unresolved in both rankings are resolved the same way in the refinements. This is, in general, impossible for trees and leads to the presence of $|\mathcal{U}(T_1, T_2)|$ in the upper bound of Lemma 1.

While the above observations are an obstacle to establishing equivalence between d_{Haus} and $d^{(p)}$, we *can* show equivalence when the number of triplets (quartets) that are unresolved in both trees is suitably small. The result below follows from Lemma 1.

Theorem 4. *Let β be a positive real number. Suppose we restrict ourselves to pairs of trees (T_1, T_2) such that $|\mathcal{U}(T_1, T_2)| \leq \beta(|\mathcal{D}(T_1, T_2)| + |\mathcal{R}_1(T_1, T_2)| + |\mathcal{R}_2(T_1, T_2)|)$. Then, Hausdorff distance and parametric distance are equivalent.*

6 Computing Parametric Distance

Let $R(T)$ and $U(T)$ denote the sets of all triplets (quartets) that are, respectively resolved and unresolved in T. We need the following fact, which holds for rooted and unrooted trees.

Proposition 2. *For any two phylogenies T_1, T_2 over the same set of taxa,*

$$\begin{aligned} d^{(p)}(T_1,T_2) = \; & |R(T_1)| - |\mathcal{S}(T_1,T_2)| + p\cdot(|U(T_1)| - |U(T_2)|) \\ & + (2p-1)\cdot|\mathcal{R}_1(T_1,T_2)|. \end{aligned} \quad (3)$$

Proof. It can be shown that $|\mathcal{R}_1(T_1,T_2)| + |\mathcal{U}(T_1,T_2)| = |U(T_2)|$, $|\mathcal{R}_2(T_1,T_2)| + |\mathcal{U}(T_1,T_2)| = |U(T_1)|$, and $|\mathcal{S}(T_1,T_2)|+|\mathcal{D}(T_1,T_2)|+|\mathcal{R}_1(T_1,T_2)| = |R(T_1)|$. These relationships, along with Equation (1), establish Equation (3). $\square$

6.1 Computing the Parametric Triplet Distance

Theorem 5. *The parametric triplet distance $d^{(p)}(T_1,T_2)$ for two rooted phylogenetic trees T_1 and T_2 over the same set of n taxa can be computed in $O(n^2)$ time.*

Proof (sketch). Our algorithm computes $d^{(p)}(T_1,T_2)$ via Equation (3). For this, it needs $|R(T_1)|$, $|U(T_1)|$, $|U(T_2)|$, $|\mathcal{S}(T_1,T_2)|$ and $|\mathcal{R}_1(T_1,T_2)|$. The first three values can easily be obtained in $O(n)$ time. Below we outline an algorithm that computes the remaining two values in $O(n^2)$ time. This gives a $O(n^2)$ parametric triplet distance algorithm.

Our algorithm relies on a preprocessing step that calculates and stores the following four quantities for every pair u, v such that u, v are internal nodes of T_1 and T_2, respectively: $|\mathcal{L}(T_1(u))\cap\mathcal{L}(T_2(v))|$, $|\mathcal{L}(T_1(u))\cap\mathcal{L}(\overline{T_2(v)})|$, $|\mathcal{L}(\overline{T_1(u)})\cap\mathcal{L}(T_2(v))|$, and $|\mathcal{L}(\overline{T_1(u)})\cap\mathcal{L}(\overline{T_2(v)})|$. All these $O(n^2)$ values can be computed in $O(n^2)$ time by visiting the pairs according to interleaved postorder traversals of T_1 and T_2, in which the set intersection sizes for each pair of nodes are computed by using the set intersection sizes computed for their children. We omit the details.

We need two definitions. Let T be a rooted phylogenetic tree. Let $X = \{x, y, z\}$ be a triplet. Suppose X is resolved in T. We say that X is *induced* by edge $(pa(v), v)$ in T if x, y are in $\mathcal{L}(T(v))$, and z is in $\mathcal{L}(\overline{T(v)})$. Note that X may be induced by multiple edges in T. Now suppose X is unresolved in T. We say that X is *associated* with the least common ancestor (lca) v of X in T. Observe that node v is unique and that it must be unresolved.

To compute $|\mathcal{S}(T_1,T_2)|$ we enumerate all pairs of internal edges $(pa(u), u) \in \mathcal{E}(T_1)$ and $(pa(v), v) \in \mathcal{E}(T_2)$ according to an order obtained by interleaving postorder traversals of T_1 and T_2. For each pair, we compute the number of common triplet topologies induced by the pair in $O(1)$ time by using the values $|\mathcal{L}(T_1(u)) \cap \mathcal{L}(T_2(v))|$, and $|\mathcal{L}(\overline{T_1(u)}) \cap \mathcal{L}(\overline{T_2(v)})|$ computed in the preprocessing step. Thus, each identically resolved triplet is counted at least once. Since a triplet may be induced by multiple edges, it is necessary to adjust for over counting. Indeed, among the triplets induced by the edges $(pa(u), u) \in T_1$ and $(pa(v), v) \in T_2$, the ones that have already been counted at an earlier step are exactly those that are either (i) induced by both edges $(pa(u), u)$ and (u, y) in T_1, for some $y \in Ch(u)$, and are induced by the edge $(pa(v), v)$ in T_2, or, (ii) induced by both edges $(pa(v), v)$ and (v, y) in T_2, for some $y \in Ch(v)$, and are induced by the edge $(pa(u), u)$ in T_1. Both the counting and the correction for over counting can be done in $O(|\,Ch(u)| + |\,Ch(v)|)$ per pair, for a total of $O(n^2)$ time.

To compute the value of $|\mathcal{R}_1(T_1, T_2)|$ we enumerate all pairs formed by picking an edge $e = (pa(u), u) \in \mathcal{E}(T_1)$ and an internal unresolved node $v \in \mathcal{V}(T_2)$ according to interleaved postorder traversals of T_1 and T_2. At each step, we count the number of triplets that are induced by e in T_1 and associated with v in T_2. Such triplets must be resolved in T_1 but unresolved in T_2. Let us say that a triplet X is *relevant* if it is induced by edge $(pa(u), u)$ in T_1, and $T_2[X]$ is a subtree of $T_2(v)$. There are $m = \binom{|P|}{2} \cdot |Q|$ relevant triplets, where $P = \mathcal{L}(T_2(v)) \cap \mathcal{L}(T_1(u))$ and $Q = \mathcal{L}(T_2(v)) \cap \mathcal{L}(\overline{T_1(u)})$. Out of these, we are interested in counting the number of triplets X whose lca in T_2 is v, and X is unresolved in T_2. Any such triplet X falls into one of three categories: (i) the lca of X in T_2 is not v, (ii) the lca of X in T_2 is v, X is resolved in T_1 and T_2, and $T_1|X = T_2|X$, (iii) the lca of X in T_2 is v, X is resolved in T_1 and T_2, but $T_1|X \neq T_2|X$. The sizes of these sets can be obtained in $O(|\,Ch(u)| \cdot |\,Ch(v)|)$ time — details are omitted. The number thus computed is then subtracted from m to get the quantity we need. The total time over all pairs u, v is $O(n^2)$. As in the computation of $|\mathcal{S}(T_1, T_2)|$, we must correct for over counting. Indeed any triplet induced by edge $(pa(u), u)$ and edge (u, y) in T_1, for some $y \in Ch(u)$, has already been counted in an earlier step of the interleaved traversals of T_1 and T_2. It can be shown that one can adjust for this over counting while keeping within the required time bound. □

6.2 An Approximation Algorithm for Parametric Quartet Distance

Theorem 6. *Let T_1 and T_2 be two unrooted phylogenetic trees on the same n leaves. Then, for $p = \frac{1}{2}$, $d^{(p)}(T_1, T_2)$ can be computed in $O(n^2)$ time. For $p \in (\frac{1}{2}, 1]$, a value x such that $d^{(p)}(T_1, T_2) \leq x \leq 2 \cdot d^{(p)}(T_1, T_2)$ can be computed in $O(n^2)$ time.*

Proof (sketch). Our algorithm first computes the values of $|\mathcal{S}(T_1, T_2)|$, $|R(T_1)|$, $|U(T_1)|$, and $|U(T_2)|$ — this can be done in $O(n^2)$ time [10]. If $p = \frac{1}{2}$, these values suffice to obtain $d^{(p)}(T_1, T_2)$ exactly, since the term involving $|\mathcal{R}_1(T_1, T_2)|$ in Equation (3) vanishes. For $p > \frac{1}{2}$, we also use Equation (3), but instead of $|\mathcal{R}_1(T_1, T_2)|$ we use a 2-approximation y to $|\mathcal{R}_1(T_1, T_2)|$; that is, y satisfies $|\mathcal{R}_1(T_1, T_2)| \leq y \leq 2|\mathcal{R}_1(T_1, T_2)|$. Below, we outline how to compute such a y in $O(n^2)$ time. As a result, we obtain an $O(n^2)$-time 2-approximate algorithm for $d^{(p)}(T_1, T_2)$.

Let (u, v) be an edge in tree T. We denote the subtree of $T - (u, v)$ that contains node u by $T(u \leftarrow v)$, and the one that contains v by $T(v \leftarrow u)$. Quartet $\{a, b, c, d\}$ is *induced* by edge (u, v) if $\{a, b\} \in \mathcal{L}(T(u \leftarrow v))$ and $\{c, d\} \in \mathcal{L}(T(v \leftarrow u))$. Note that every resolved quartet is induced by at least one edge. Quartet $\{a, b, c, d\}$ is *associated* with node v in T if the paths from v to a, v to b, v to c, and v to d are edge-disjoint. Note that each unresolved quartet is associated with exactly one node in T.

Our algorithm roots T_1 by adding a root node to an arbitrarily chosen edge in T_1. It then enumerates each edge $e = (pa(u), u) \in \mathcal{E}(T_1)$ according to a preorder traversal of T_1 and each internal node $v \in \mathcal{V}(T_2)$ of degree at least 4. For each pair, it counts the number of quartets that are induced by e in T_1 and associated with v in T_2. As in the rooted case (Theorem 5), we do this indirectly. We first obtain the number of *relevant* quartets; namely those induced by $(pa(u), u)$. This can be done efficiently with suitable preprocessing. To find the size of the subset of these quartets that are unresolved and associated with v (which is what we need), we count the number of all other quartets and subtract it from the number of relevant quartets. Each of these other quartets appears in one of the following five configurations in the tree T_2: (i) there exists a neighbor x of v in T_2, such that the quartet is completely contained in $T_2(x \leftarrow v)$, (ii) there exist two neighbors x, y of v in T_2, such that $T_2(x \leftarrow v)$ contains three leaves from the quartet and $T_2(y \leftarrow v)$ contains the other leaf, (iii) there exist two neighbors x, y of v in T_2, such that $T_2(x \leftarrow v)$ contains two leaves from the quartet and $T_2(y \leftarrow v)$ contains the other two leaves, and (iv) there exist three neighbors x, y, z of v in T_2, such that $T_2(x \leftarrow v)$ contains two leaves from the quartet, $T_2(y \leftarrow v)$ contains one leaf of the quartet, and $T_2(z \leftarrow v)$ contains the remaining leaf. Handling cases (i), (ii) and (iii) efficiently is relatively easy, but case (iv) requires computing first a combined value that counts each quartet from case (iii) twice and each quartet from (iv) once, and then deriving the value for case (iv). The time per pair $(pa(u), u) \in \mathcal{E}(T_1), v \in \mathcal{V}(T_2)$ is $O(|\mathit{Ch}(u)| \cdot |\mathit{adj}(v)|)$, where $\mathit{adj}(v)$ is the set of neighbors of v in T_2, for a total of $O(n^2)$ time.

Note that, as described, the above computation over counts some quartets. It is not clear how to correct for this while staying within a $O(n^2)$ time bound. However, within this time, we *can* guarantee that no quartet is counted at least once and more than twice. Thus, instead of computing $|\mathcal{R}_1(T_1, T_2)|$ exactly, we obtain a 2-approximation to its value. □

References

1. Adams III, E.N.: N-trees as nestings: Complexity, similarity, and consensus. J. Classification 3(2), 299–317 (1986)
2. Allen, B., Steel, M.: Subtree transfer operations and their induced metrics on evolutionary trees. Annals of Combinatorics 5, 1–13 (2001)
3. Barthélemy, J.P., McMorris, F.R.: The median procedure for n-trees. Journal of Classification 3, 329–334 (1986)
4. Bartholdi, J.J., Tovey, C.A., Trick, M.A.: Voting schemes for which it can be difficult to tell who won the election. Social Choice and Welfare 6, 157–165 (1989)
5. Berry, V., Jiang, T., Kearney, P.E., Li, M., Wareham, H.T.: Quartet cleaning: Improved algorithms and simulations. In: Nešetřil, J. (ed.) ESA 1999. LNCS, vol. 1643, pp. 313–324. Springer, Heidelberg (1999)
6. Bininda-Emonds, O.R.P. (ed.): Phylogenetic supertrees: Combining Information to Reveal the Tree of Life. Computational Biology Series, vol. 4. Springer, Heidelberg (2004)
7. Brodal, G.S., Fagerberg, R., Pedersen, C.N.S.: Computing the quartet distance in time $O(n \log n)$. Algorithmica 38(2), 377–395 (2003)
8. Bryant, D.: Building trees, hunting for trees, and comparing trees: Theory and methods in phylogenetic analysis. PhD thesis, Department of Mathematics, University of Canterbury, New Zealand (1997)
9. Bryant, D.: A classification of consensus methods for phylogenetics. In: Janowitz, M., Lapointe, F.-J., McMorris, F., Mirkin, B.B., Roberts, F. (eds.) Bioconsensus. Discrete Mathematics and Theoretical Computer Science, vol. 61, pp. 163–185. American Mathematical Society, Providence (2003)

10. Christiansen, C., Mailund, T., Pedersen, C.N., Randers, M., Stissing, M.S.: Fast calculation of the quartet distance between trees of arbitrary degrees. Algorithms for Molecular Biology 1(16) (2006)
11. Cotton, J.A., Slater, C.S., Wilkinson, M.: Discriminating supported and unsupported relationships in supertrees using triplets. Systematic Biology 55(2), 345–350 (2006)
12. Critchlow, D.E.: Metric Methods for Analyzing Partially Ranked Data. Lecture Notes in Statist, vol. 34. Springer, Berlin (1980)
13. Day, W.H.E.: Analysis of quartet dissimilarity measures between undirected phylogenetic trees. Systematic Zoology 35(3), 325–333 (1986)
14. Dwork, C., Kumar, R., Naor, M., Sivakumar, D.: Rank aggregation methods for the web. In: Tenth International World Wide Web Conference, Hong Kong, May 2001, pp. 613–622 (2001)
15. Fagin, R., Kumar, R., Mahdian, M., Sivakumar, D., Vee, E.: Comparing partial rankings. SIAM J. Discrete Math. 20(3), 628–648 (2006)
16. Finden, C.R., Gordon, A.D.: Obtaining common pruned trees. J. Classification 2(1), 225–276 (1985)
17. Maddison, W.P.: Reconstructing character evolution on polytomous cladograms. Cladistics 5, 365–377 (1989)
18. McMorris, F.R., Meronk, D.B., Neumann, D.A.: A view of some consensus methods for trees. In: Felsenstein, J. (ed.) Numerical Taxonomy, pp. 122–125. Springer, Heidelberg (1983)
19. Piel, W., Sanderson, M., Donoghue, M., Walsh, M.: Treebase (last accessed, February 2, 2007), http://www.treebase.org
20. Robinson, D.F., Foulds, L.R.: Comparison of phylogenetic trees. Mathematical Biosciences 53, 131–147 (1981)
21. Semple, C., Steel, M.: Phylogenetics. Oxford Lecture Series in Mathematics. Oxford University Press, Oxford (2003)
22. Snir, S., Rao, S.: Using max cut to enhance rooted trees consistency. IEEE/ACM Trans. Comput. Biol. Bioinformatics 3(4), 323–333 (2006)
23. Steel, M., Penny, D.: Distributions of tree comparison metrics — some new results. Systematic Biology 42(2), 126–141 (1993)
24. Stissing, M., Pedersen, C.N.S., Mailund, T., Brodal, G.S., Fagerberg, R.: Computing the quartet distance between evolutionary trees of bounded degree. In: Sankoff, D., Wang, L., Chin, F. (eds.) APBC. Advances in Bioinformatics and Computational Biology, vol. 5, pp. 101–110. Imperial College Press (2007)
25. Stockham, C., Wang, L.-S., Warnow, T.: Statistically based postprocessing of phylogenetic analysis by clustering. In: ISMB, pp. 285–293 (2002)
26. Vazirani, V.V.: Approximation Algorithms. Springer, Berlin (2001)

Computing the Growth of the Number of Overlap-Free Words with Spectra of Matrices[⋆]

Raphaël M. Jungers[1], Vladimir Yu. Protasov[2], and Vincent D. Blondel[1,⋆⋆]

[1] Department of Applied Mathematics, Université catholique de Louvain, 4 avenue Georges Lemaitre, B-1348 Louvain-la-Neuve, Belgium
{raphael.jungers,vincent.blondel}@uclouvain.be

[2] Department of Mechanics and Mathematics, Moscow State University, Vorobyovy Gory, Moscow, 119992, Russia
v-protassov@yandex.ru

Abstract. Overlap-free words are words over the alphabet $A = \{a, b\}$ that do not contain factors of the form $xvxvx$, where $x \in A$ and $v \in A^*$. We analyze the asymptotic growth of the number u_n of overlap-free words of length n. We obtain explicit formulas for the minimal and maximal rates of growth of u_n in terms of spectral characteristics (the lower spectral radius and the joint spectral radius) of one set of matrices of dimension 20. Using these descriptions we provide estimates of the rates of growth that are within 0.4% and 0.03% of their exact value. The best previously known bounds were within 11% and 3% respectively. We prove that u_n actually has the same growth for "almost all" n. This "average" growth is distinct from the maximal and minimal rates and can also be expressed in terms of a spectral quantity (the Lyapunov exponent). We use this expression to estimate it.

1 Introduction

Binary overlap-free words have been studied for more than a century. These are words over the binary alphabet $A = \{a, b\}$ that do not contain factors of the form $xvxvx$, where $x \in A$ and $v \in A^*$. For instance, the word *baabaa* is overlap free, but the word *baabaab* is not, since it can be written $xuxux$ with $x = b$ and $u = aa$. See [1] for a recent survey. Thue [19,20] proved in 1906 that there are infinitely many overlap-free words. Indeed, the well-known Thue-Morse sequence[1] is overlap-free, and so the set of its factors provides an infinite number

[⋆] This is a shortened version of a journal paper submitted for publication; see [7].

[⋆⋆] The research reported here was partially supported by the "Communauté francaise de Belgique - Actions de Recherche Concertées", by the EU HYCON Network of Excellence (contract number FP6-IST-511368), and by the Belgian Programme on Interuniversity Attraction Poles initiated by the Belgian Federal Science Policy Office. The scientific responsibility rests with its authors. Raphaël Jungers is a FNRS fellow (Belgian Fund for Scientific Research).

[1] The Thue-Morse sequence is the infinite word obtained as the limit of $\theta^n(a)$ for $n \to \infty$ with $\theta(a) = ab,\ \theta(b) = ba$; see [6].

of different overlap-free words. The asymptotics of the number u_n of such words of a given length n was analyzed in a number of subsequent contributions[2]. The number of factors of length n in the Thue-Morse sequence is proved in [4] to be larger than $3n$, thus providing a linear lower bound on u_n:

$$u_n \ \geq \ 3\,n.$$

The next improvement was obtained by Restivo and Salemi [17]. By using a certain decomposition result, they showed that the number of overlap-free words grows at most polynomially:

$$u_n \ \leq \ C\,n^{\,r},$$

where $r = \log(15) \approx 3.906$. This bound has been sharpened successively by Kfoury [10], Kobayashi [11], and finally by Lepisto [12] to the value $r = 1.37$. One could then suspect that the sequence u_n grows linearly. However, Kobayashi [11] proved that this is not the case. By enumerating the subset of overlap-free words of length n that can be infinitely extended to the right he showed that $u_n \ \geq C\,n^{\,1.155}$ and so we have

$$C_1\,n^{\,1.155} \leq u_n \leq C_2\,n^{\,1.37}.$$

It is worth noting that the sequence u_n is 2-regular, as shown by Carpi [5]. On Figure 1(a) we show the values of the sequence u_n for $1 \leq n \leq 200$ and on Figure 1(b) we show the behavior of $\log u_n / \log n$ for larger values of n. One can see that the sequence u_n is not monotonic, but is globally increasing with n. Moreover the sequence does not appear to have a polynomial growth since the value $\log u_n / \log n$ does not seem to converge. In view of this, a natural question arises: is the sequence u_n asymptotically equivalent to n^r for some r ? Cassaigne proved in [6] that the answer is negative. He introduced the lower and the upper exponents of growth:

$$\begin{aligned} \alpha &= \sup\bigl\{r \ \big|\exists C > 0, u_n \geq Cn^r\bigr\}, \\ \beta &= \inf\bigl\{r \ \big|\exists C > 0, u_n \leq Cn^r\bigr\}, \end{aligned} \tag{1}$$

and showed that $\alpha < \beta$. Cassaigne made a real breakthrough in the study of overlap-free words by characterizing in a constructive way the whole set of overlap-free words. By improving the decomposition theorem of Restivo and Salemi he showed that the numbers u_n can be computed as sums of variables that are obtained by certain linear recurrence relations. These relations are explicitly given in the next section. As a result of this description, the number of overlap-free words of length n can be computed in logarithmic time. For the exponents of growth Cassaigne has also obtained the following bounds: $\alpha < 1.276$ and $\beta > 1.332$. Thus, combining this with the earlier results described above, one has the following inequalities:

$$1.155 \ < \ \alpha \ < \ 1.276 \qquad \text{and} \qquad 1.332 \ < \ \beta \ < \ 1.37. \tag{2}$$

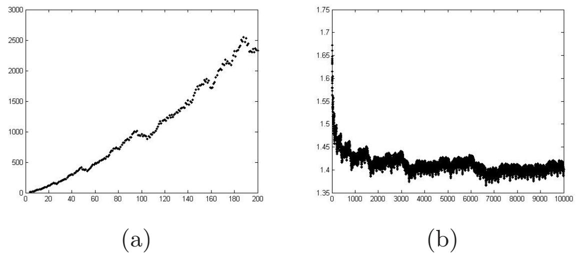


Fig. 1. The values of u_n for $1 \le n \le 200$ (a) and $\log u_n / \log n$ for $1 \le n \le 10000$ (b)

In this paper we develop a linear algebraic approach to study the asymptotic behavior of the number of overlap-free words of length n. Using the results of Cassaigne we show in Theorem 2 that u_n is asymptotically equivalent to the norm of a long product of two particular matrices A_0 and A_1 of dimension 20×20. This product corresponds to the binary expansion of the number $n-1$. Using this result we express the values of α and β by means of certain joint spectral characteristics of these matrices. We prove that $\alpha = \log_2 \check{\rho}(A_0, A_1)$ and $\beta = \log_2 \hat{\rho}(A_0, A_1)$, where $\check{\rho}$ and $\hat{\rho}$ denote, respectively, the lower spectral radius and the joint spectral radius of the matrices A_0, A_1 (we define these notions in the next section). In Section 2, we estimate these values and we obtain the following improved bounds for α and β:

$$1.2690 < \alpha < 1.2736 \quad \text{and} \quad 1.3322 < \beta < 1.3326. \tag{3}$$

Our estimates are, respectively, within 0.4% and 0.03% of the exact values. In addition, we show in Theorem 3 that the smallest and the largest rates of growth of u_n are effectively attained, and there exist positive constants C_1, C_2 such that $C_1 n^\alpha \le u_n \le C_2 n^\beta$ for all $n \in \mathbb{N}$.

Although the sequence u_n does not exhibit an asymptotic polynomial growth, we then show in Theorem 6 that for "almost all" values of n the rate of growth is actually equal to $\sigma = \log_2 \bar{\rho}(A_0, A_1)$, where $\bar{\rho}$ is the Lyapunov exponent of the matrices. For almost all values of n the number of overlap-free words does not grow as n^α, nor as n^β, but in an intermediary way, as n^σ. This means in particular that the value $\frac{\log u_n}{\log n}$ converges to σ as $n \to \infty$ along a subset of density 1. We obtain the following bounds for the limit σ, which provides an estimation within 0.8% of the exact value:

$$1.3005 < \sigma < 1.3098.$$

These bounds clearly show that $\alpha < \sigma < \beta$.

Our linear algebraic approach not only allows us to improve the estimates of the asymptotics of the number of overlap-free words, but also clarifies some

[2] The number of overlap-free words of length n is referenced in the On-Line Encyclopedia of Integer Sequences under the code A007777; see [18]. The sequence starts 1, 2, 4, 6, 10, 14, 20, 24, 30, 36, 44, 48, 60, 60, 62, 72,...

aspects of the nature of these words. For instance, we show that the "non purely overlap-free words" used in [6] to compute u_n are asymptotically negligible when considering the total number of overlap-free words.

The paper is organized as follows. In the next section we formulate the main theorems (Theorems 2 and 3). The proofs of these theorems are quite technical and are not presented here for sake of conciseness. We refer the reader to [7] for a presentation of the proof and all numerical values. We deduce from these results sharp estimates for α and β. Then in Section 3 we introduce the average growth rate σ and approximate it.

2 The Minimal and Maximal Rates of Growth of the Overlap-Free Words

In the sequel we use the following notation: $\mathbb{R}^d$ is the d-dimensional space, inequalities $x \geq 0$ and $A \geq 0$ mean that all the entries of the vector x (respectively, of the matrix A) are nonnegative. We denote $\mathbb{R}^d_+ = \{x \in \mathbb{R}^d,\ x \geq 0\}$, by $|x|$ we denote a norm of the vector $x \in \mathbb{R}^d$, and by $\|\cdot\|$ any matrix norm. In particular, $|x|_1 = \sum\limits_{i=1}^{d} |x_i|$, $\|A\|_1 = \sup_{|x|_1=1} |Ax| = \max_{j=1,\ldots d} \sum_{i=1}^{d} |A_{ij}|$. We write $\mathbf{1}$ for the vector $(1,\ldots,1)^T \in \mathbb{R}^d$, $\rho(A)$ for the spectral radius of the matrix A, that is, the largest magnitude of its eigenvalues. If $A \geq 0$, then there is a vector $v \geq 0$ such that $Av = \rho(A)v$ (the so-called Perron-Frobenius eigenvector). For two functions f_1, f_2 from a set Y to $\mathbb{R}_+$ the relation $f_1(y) \asymp f_2(y)$ means that there are positive constants C_1, C_2 such that $C_1 f_1(y) \leq f_2(y) \leq C_2 f_1(y)$ for all $y \in Y$.

To compute the number u_n of overlap-free words of length n we use several results from [6] that we summarize in the following theorem:

Theorem 1. *There exist two nonnegative matrices $F_0, F_1 \in \mathbb{R}^{30\times 30}$, and nonnegative vectors $w, y_3, \ldots, y_{15} \in \mathbb{R}^{30}_+$ allowing to compute the number of overlap-free words in the following way: For $n \geq 16$, let y_n be the solution of the recurrence equations:*

$$\begin{aligned} y_{2n} &= F_0 y_n \\ y_{2n+1} &= F_1 y_n. \end{aligned} \tag{4}$$

Then, for any $n \geq 16$, the number of overlap-free words of length n is equal to $w^T y_{n-1}$.

It follows from this result that the number u_n of overlap-free words of length $n \geq 16$ can be obtained by first computing the binary expansion $d_k \cdots d_1$ of $n-1$, i.e., $n-1 = \sum_{j=0}^{k-1} d_{j+1} 2^j$, and then defining

$$u_n = w^T F_{d_1} \cdots F_{d_{k-4}} y_m \tag{5}$$

where $m = d_{k-3} + d_{k-2}2 + d_{k-1}2^2 + d_k 2^3$. To arrive at the results summarized in Theorem 1, Cassaigne builds a system of recurrence equations allowing the computation of a vector U_n whose entries are the number of overlap-free words of certain types (there are 16 different types). These recurrence equations also involve the recursive computation of a vector V_n that counts other words of length n, the so-called "single overlaps". The single overlap words are not overlap-free, but have to be computed, as they generate overlap-free words of larger lengths. We now present the main result of this section which improves the above theorem in two directions. First we reduce the dimension of the matrices from 30 to 20, and second we prove that u_n is given asymptotically by the norm of a matrix product. The reduction of the dimension to 20 has a straightforward interpretation: when computing the asymptotic growth of the number of overlap-free words, one can neglect the number of "single overlaps" V_n defined by Cassaigne. We call the remaining words *purely overlap-free words,* as they can be entirely decomposed in a sequence of overlap-free words via Cassaigne's decomposition (see [6] for more details).

Theorem 2. *There exist two nonnegative matrices $A_0, A_1 \in \mathbb{R}_+^{20\times 20}$ allowing to describe the asymptotics of the number of overlap-free words in the following way: let $\|\cdot\|$ be a matrix norm, and let $A(n) : \mathbb{N} \to \mathbb{R}_+^{20\times 20}$ be defined as $A(n) = A_{d_1} \cdots A_{d_k}$ with $d_k \dots d_1$ the binary expansion of $n-1$. Then,*

$$u_n \asymp ||A(n)||. \tag{6}$$

These matrices A_0, A_1 are submatrices of F_0 and F_1 described in Theorem 1.

The matrices F_0, F_1 in Theorem 1 are both nonnegative and hence possess a common invariant cone $K = \mathbb{R}_+^{30}$. We say that a cone K is invariant for a linear operator B if $BK \subset K$. All cones are assumed to be solid, convex, closed, and pointed. We start with the following simple result proved in [15].

Lemma 1. *For any cone $K \subset \mathbb{R}^d$, for any norm $|\cdot|$ in $\mathbb{R}^d$ and any matrix norm $\|\cdot\|$ there is a homogeneous continuous function $\gamma : K \to \mathbb{R}_+$ positive on* int$K$ *such that for any $x \in \text{int}K$ and for any matrix B that leaves K invariant one has*

$$\gamma(x)\|B\| \cdot |x| \le |Bx| \le \frac{1}{\gamma(x)}\|B\| \cdot |x|.$$

Corollary 1. *Let two matrices A_0, A_1 possess an invariant cone $K \subset \mathbb{R}^d$. Then for any $x \in \text{int}K$ we have $|A_{d_1} \cdots A_{d_k} x| \asymp \|A_{d_1} \cdots A_{d_k}\|$ for all k and for all indices $d_1, \dots, d_k \in \{0, 1\}$.*

In view of Corollary 1 and of Eq. (5), Theorem 2 may seem obvious, at least if we consider the matrices F_i instead of A_i. One can however not directly apply Lemma 1 and Corollary 1 to the matrices A_0, A_1 or to the matrices F_0, F_1 because the vector corresponding to x is not in the interior of the positive orthant, which is an invariant cone of these matrices. To prove Theorem 2 we construct a wider invariant cone of A_0 and A_1 by using special properties of these matrices. Theorem 2 allows us to express the rates of growth of the sequence u_n in terms of norms of products of the matrices A_0, A_1 and then to use joint spectral characteristics of these matrices to estimate the rates of growth. More explicitly, Theorem 2 yields the following corollary:

Corollary 2. *Let $A_0, A_1 \in \mathbb{R}_+^{20\times 20}$ be the matrices defined in Theorem 2 and let $A(n) : \mathbb{N} \to \mathbb{R}_+^{20\times 20}$ be defined as $A(n) = A_{d_1} \cdots A_{d_k}$ with $d_k \dots d_1$ the binary expansion of $n-1$. Then*

$$\frac{\log u_n}{\log n} - \log \|A(n)\|^{1/k} \to 0 \quad \textit{as} \quad n \to \infty. \tag{7}$$

Proof. Since $\frac{\log_2 n}{k} \to 1$ as $n \to \infty$, we have

$$\lim_{n\to\infty} \left(\frac{\log_2 u_n}{\log_2 n} - \frac{\log_2 \|A_{d_1} \cdots A_{d_k}\|}{k} \right) = \lim_{n\to\infty} \frac{\log_2 u_n - \log_2 \|A_{d_1} \cdots A_{d_k}\|}{k} = \lim_{n\to\infty} \frac{\log_2 \left(u_n \cdot \|A_{d_1} \cdots A_{d_k}\|^{-1} \right)}{k}.$$

By Theorem 2 the value $\log_2 \left(u_n \cdot \|A_{d_1} \cdots A_{d_k}\|^{-1} \right)$ is bounded uniformly over $n \in \mathbb{N}$, hence it tends to zero, being divided by k. □

We now analyze the smallest and the largest exponents of growth α and β defined in Eq. (1). For a given set of matrices $\Sigma = \{A_1, \dots, A_m\}$ we denote by $\check{\rho}$ and $\hat{\rho}$ its lower spectral radius and its joint spectral radius:

$$\check{\rho}(\Sigma) = \lim_{k\to\infty} \min_{d_1,\dots,d_k \in \{1,\dots,m\}} \|A_{d_1} \cdots A_{d_k}\|^{1/k}, \tag{8}$$
$$\hat{\rho}(\Sigma) = \lim_{k\to\infty} \max_{d_1,\dots,d_k \in \{1,\dots,m\}} \|A_{d_1} \cdots A_{d_k}\|^{1/k}.$$

Both limits are well-defined and do not depend on the chosen norm. Moreover, for any product $A_{d_1} \cdots A_{d_k}$ we have

$$\check{\rho} \le \rho(A_{d_1} \cdots A_{d_k})^{1/k} \le \hat{\rho} \tag{9}$$

(see [14,3] for surveys on these notions).

Theorem 3. *For $k \ge 1$, let $\alpha_k = \min_{2^{k-1} < n \le 2^k} \frac{\log u_n}{\log n}$ and $\beta_k = \max_{2^{k-1} < n \le 2^k} \frac{\log u_n}{\log n}$. Then, with A_0, A_1 defined in Theorem 2,*

$$\alpha = \lim_{k\to\infty} \alpha_k = \log_2 \check{\rho}(A_0, A_1) \quad \textit{and} \quad \beta = \lim_{k\to\infty} \beta_k = \log_2 \hat{\rho}(A_0, A_1). \tag{10}$$

Moreover, there are positive constants C_1, C_2 such that

$$C_1 \le \min_{2^{k-1} < n \le 2^k} u_n n^{-\alpha} \quad \textit{and} \quad C_1 \le \max_{2^{k-1} < n \le 2^k} u_n n^{-\beta} \le C_2 \tag{11}$$

for all $k \in \mathbb{N}$.

The proof of this theorem can be found in [7].

Corollary 3. *There are positive constants C_1, C_2 such that*

$$C_1 n^{\alpha} \le u_n \le C_2 n^{\beta}, \ n \in \mathbb{N}.$$

We end this section by giving sharp estimates for α and β, obtained thanks to their representations as joint spectral quantities of A_0 and A_1. We also conjecture that the lower bound on β is actually its exact value. We refer the reader to the journal version of the present paper [7] for a complete description of how these bounds have been derived.

Theorem 4

$$\begin{aligned} 1.2690 < \alpha < 1.2736 \\ 1.3322 < \beta < 1.3326 \end{aligned} \tag{12}$$

In [7] we also make (and give arguments for) the following conjecture:

Conjecture 1

$$\beta = \log_2 \sqrt{\rho(A_0 A_1)} = 1.3322\ldots.$$

3 The Average Rate of Growth: The Lyapunov Exponent

We have seen that $\alpha < \beta$. In particular, the sequence u_n does not have a constant rate of growth, and the value $\frac{\log u_n}{\log n}$ does not converge as $n \to \infty$. This was already noted by Cassaigne in [6]. Nevertheless, it appears that the value $\frac{\log u_n}{\log n}$ actually has a limit as $n \to \infty$, not along all the natural numbers $n \in \mathbb{N}$, but along a subsequence of $\mathbb{N}$ of density 1. In other terms, the sequence converges with probability 1. The limit, which differs from both α and β can be expressed by the so-called Lyapunov exponent $\bar{\rho}$ of the matrices A_0, A_1. To show this we apply the following result proved by Oseledets in 1968. For the sake of simplicity we formulate it for two matrices, although it can be easily generalized to any finite set of matrices.

Theorem 5. *[13] Let A_0, A_1 be arbitrary matrices and $d_1, d_2, \ldots$ be a sequence of independent random variables that take values 0 and 1 with equal probabilities $1/2$. Then the value $\|A_{d_1} \cdots A_{d_k}\|^{1/k}$ converges to some number $\bar{\rho}$ with probability 1. This means that for any $\varepsilon > 0$ we have $P\big(\big|\|A_{d_1} \cdots A_{d_k}\|^{1/k} - \bar{\rho}\big| > \varepsilon\big) \to 0$ as $k \to \infty$.*

The limit $\bar{\rho}$ in Theorem 5 is called the *Lyapunov exponent* of the set $\{A_0, A_1\}$. This value is given by the following formula:

$$\bar{\rho}(A_0, A_1) = \lim_{k \to \infty} \Big(\prod_{d_1, \ldots, d_k} \|A_{d_1} \cdots A_{d_k}\|^{1/k} \Big)^{1/2^k} \tag{13}$$

(for the proof see, for instance, [16]). To understand what this gives for the asymptotics of our sequence u_n we introduce some further notation. Let $\mathcal{P}$ be some property of natural numbers. For a given $k \in \mathbb{N}$ we denote

$$P_k(\mathcal{P}) = 2^{-(k-1)} \mathrm{Card}\big\{n \in \{2^{k-1}+1, \ldots, 2^k\}, \quad n \,\text{satisfies}\, \mathcal{P} \big\}.$$

Thus, P_k is the probability that the integer n uniformly distributed on the set $\{2^{k-1}+1, \ldots, 2^k\}$ satisfies $\mathcal{P}$. Combining Proposition 2 and Theorem 5 we obtain

Theorem 6. *There is a unique number σ such that for any $\varepsilon > 0$ we have*

$$P_k\Big(\Big|\frac{\log u_n}{\log n} - \sigma\ \Big| > \varepsilon\Big) \to\ 0 \qquad \text{as } k \to \infty.$$

Moreover, $\sigma = \log_2 \bar{\rho}$, where $\bar{\rho}$ is the Lyapunov exponent of the matrices $\{A_0, A_1\}$ defined in Theorem 2.

Thus, for almost all numbers $n \in \mathbb{N}$ the number of overlap-free words u_n has the same exponent of growth $\sigma = \log_2 \bar{\rho}$. Let us recall that a subset $\mathcal{A} \subset \mathbb{N}$ is said to have density 1 if $\frac{1}{n}\mathrm{Card}\big\{r \le n,\ r \in \mathcal{A}\big\} \to\ 1$ as $n \to \infty$. We say that a sequence f_n converges to a number f along a set of density 1 if there is a set $\mathcal{A} \subset \mathbb{N}$ of density 1 such that $\lim\limits_{n\to\infty, n\in\mathcal{A}} f_n =\ f$. Theorem 6 yields.

Corollary 4. *The value $\frac{\log u_n}{\log n}$ converges to σ along a set of density 1.*

We show in [7] how to derive bounds on σ:

Theorem 7

$$1.3005 < \sigma < 1.3098. \tag{14}$$

4 Conclusions

The goal of this paper is to precisely characterize the asymptotic rate of growth of the number of overlap-free words. Based on Cassaigne's description of these words with products of matrices, we first prove that these matrices can be simplified, by decreasing the state space dimension from 30 to 20. This improvement is not only useful for numerical computations, but allows to characterize the overlap-free words that "count" for the asymptotics: we call these words *purely overlap free,* as they can be expressed iteratively as the image of shorter purely overlap free words.

We have then proved that the lower and upper exponents α and β defined by Cassaigne are effectively reached for an infinite number of lengths, and we have characterized them respectively as the logarithms of the *lower spectral radius* and the *joint spectral radius* of the simplified matrices that we constructed. This characterization allows us to compute them within 0.4% of their exact value. Finally we have shown that for almost all values of n, the number of overlap-free words of length n do not grow as n^α, nor as n^β, but in an intermediary way as n^σ, and we have provided sharp bounds for this value of σ.

The computational results we report in this paper have all been obtained in a few minutes of computation time on a standard PC desktop and can therefore easily be improved.

This work opens obvious questions: Can joint spectral characteristics be used to describe the rate of growth of other languages, such as for instance the more general repetition free languages ? The generalization does not seem to be straightforward for several reasons: first, the somewhat technical proofs of the links between u_n and the norm of a corresponding matrix product take into account the very structure of these particular matrices, and second, it is known that a bifurcation occurs for the growth of repetition-free words: for some members of this class of languages the growth is polynomial, as for overlap-free words, but for some others the growth is exponential [9], and one could wonder how the joint spectral characteristics developed in this paper could represent both kinds of growth.

Acknowledgment. We would like to thank Prof. Stephen Boyd (Stanford University), Yuri Nesterov, and François Glineur (Université catholique de Louvain) for their helpful suggestions on semi-definite programming techniques. This research was carried out during the visit of the second author to the Université catholique de Louvain (Louvain-la-Neuve, Belgium). That author is grateful to the university for its hospitality.

References

1. Berstel, J.: Growth of repetition-free words–a review. Theoretical Computer Science 340(2), 280–290 (2005)
2. Blondel, V.D., Nesterov, Y., Theys, J.: On the accuracy of the ellipsoid norm approximation of the joint spectral radius. Linear Algebra and its Applications 394(1), 91–107 (2005)
3. Blondel, V.D., Tsitsiklis, J.N.: A survey of computational complexity results in systems and control. Automatica 36(9), 1249–1274 (2000)
4. Brlek, S.: Enumeration of factors in the thue-morse word. Discrete Applied Mathematics 24, 83–96 (1989)
5. Carpi, A.: Overlap-free words and finite automata. Theoretical Computer Science 115(2), 243–260 (1993)
6. Cassaigne, J.: Counting overlap-free binary words. In: Enjalbert, P., Wagner, K.W., Finkel, A. (eds.) STACS 1993. LNCS, vol. 665, pp. 216–225. Springer, Heidelberg (1993)
7. Jungers, R.M., Protasov, V., Blondel, V.D.: Overlap-free words and spectra of matrices (submitted, preprint, 2007), http://arxiv.org/abs/0709.1794
8. Jungers, R.M., Blondel, V.D.: On the finiteness conjecture for rational matrices. In: Linear Algebra and its Applications (to appear, 2007), doi:10.1016/j.laa.2007.07.007
9. Karhumäki, J., Shallit, J.: Polynomial versus exponential growth in repetition-free binary words. Journal of Combinatorial Theory Series A 105(2), 335–347 (2004)
10. Kfoury, A.J.: A linear time algorithm to decide whether a binary word contains an overlap. Theoretical Informatics and Applications 22, 135–145 (1988)
11. Kobayashi, Y.: Enumeration of irreducible binary words. Discrete Applied Mathematics 20, 221–232 (1988)
12. Lepistö, A.: A characterization of 2+-free words over a binary alphabet, Master thesis, University of Turku, Finland (1995)
13. Oseledets, V.I.: A multiplicative ergodic theorem. Lyapunov characteristic numbers for dynamical systems. Transactions of the Moscow Mathematical Society 19, 197–231 (1968)

14. Protasov, V.Y.: The joint spectral radius and invariant sets of linear operators. Fundamentalnaya i prikladnaya matematika 2(1), 205–231 (1996)
15. Protasov, V.Y.: On the asymptotics of the partition function. Sbornik Mathematika 191(3-4), 381–414 (2000)
16. Protasov, V.Y.: On the regularity of de rham curves. Izvestiya Mathematika 68(3), 27–68 (2004)
17. Restivo, A., Salemi, S.: Overlap-free words on two symbols. In: Perrin, D., Nivat, M. (eds.) Automata on Infinite Words. LNCS, vol. 192, pp. 198–206. Springer, Heidelberg (1985)
18. Sloane, N.J.A.: On-line encyclopedia of integer sequences, http://www.research.att.com/ njas/sequences
19. Thue, A.: Uber unendliche Zeichenreihen. Kra. Vidensk. Selsk. Skrifter. I. Mat. Nat. Kl. 7, 1–22 (1906)
20. Thue, A.: Uber die gegenseitige Lage gleicher Teile gewisser Zeichenreihen. Kra. Vidensk. Selsk. Skrifter. I. Mat. Nat. Kl. 1, 1–67 (1912)

On Stateless Multihead Automata: Hierarchies and the Emptiness Problem

Oscar H. Ibarra[1,⋆], Juhani Karhumäki[2,⋆⋆], and Alexander Okhotin[2,3,⋆⋆]

[1] Department of Computer Science, University of California,
Santa Barbara, CA 93106, USA
ibarra@cs.ucsb.edu

[2] Department of Mathematics, University of Turku, FIN-20014 Turku, Finland
{karhumak,alexander.okhotin}@utu.fi

[3] Academy of Finland

Abstract. We look at stateless multihead finite automata in their two-way and one-way, deterministic and nondeterministic variations. The transition of a k-head automaton depends solely on the symbols currently scanned by its k heads, and every such transition moves each head one cell left or right, or instructs it to stay. We show that stateless $(k+4)$-head two-way automata are more powerful than stateless k-head two-way automata. In the one-way case, we prove a tighter result: stateless $(k+1)$-head one-way automata are more powerful than stateless k-head one-way automata. Finally, we show that the emptiness problem for stateless 2-head two-way automata is undecidable.

1 Introduction

Inspired by biologically-motivated models of computing [3,4,6], stateless multihead two-way finite automata and stateless multicounter machines were recently introduced by Yang, Dang and Ibarra [7]. These stateless machines are essentially one-state machines. The previous results [7] are mostly concerned with decidability/undecidability of decision problems such as emptiness and reachability. In this paper, we investigate the language accepting power of stateless multihead finite automata.

Denote two-way nondeterministic (deterministic) finite automata by 2NFA (2DFA), similarly denote their one-way variants by 1NFA (1DFA). We consider *stateless k-head 2NFAs* and define them as pairs of an alphabet Σ and a set of transitions δ. Let $¢, \$ \notin \Sigma$, be the left and right end markers. Each transition in δ is of the form $a_1 \ldots a_k \to d_1 \ldots d_k$, where $a_i \in \Sigma \cup \{¢, \$\}$ is the symbol scanned by i-th head, while $d_i \in \{\ell, s, r\}$ tells where each i-th head is to be moved (ℓ, s and r stand for *left*, *stay* and *right*, respectively). If there is at most one transition for every collection of symbols $a_1, \ldots, a_k \in \Sigma^k$, we refer to such an automaton as a *stateless k-head 2DFA*. If none of the transitions move any heads to the left, such an automaton is called a *stateless k-head 1NFA (1DFA)*.

⋆ Supported in part by NSF Grants CCF-0430945 and CCF-0524136, and a Nokia Visiting Fellowship at the University of Turku.

⋆⋆ Supported by the Academy of Finland under grants 44087 and 118540.

For an input string $w \in \Sigma^*$, machines work on a tape containing $¢w\$$ and start with all heads on the left end marker. At every step of the computation, the symbols $a_1, \ldots, a_k$ currently scanned by all k heads are considered, any corresponding transition $a_1 \ldots a_k \to d_1 \ldots d_k \in \delta$ is chosen, and each i-th heads is moved according to d_i. If no such transition exists, the automaton rejects. If any of the heads falls off the tape, the automaton rejects as well. If the transition instructs all heads to stay, the automaton halts and accepts. The string is accepted if there exists a computation resulting in acceptance. As an example, the stateless 2-head 1DFA with instructions $¢¢ \to sr$, $¢a \to sr$, $¢b \to rr$, $ab \to rr$, and $b\$ \to ss$ recognizes the language $L = \{a^n b^{n+1} \mid n \geqslant 0\}$.

We shall also consider the well-known *k-head 2NFAs (2DFAs) with states*, which use a finite set of states Q, and in which transitions are quintuples $(q, a_1, \ldots, a_k, q', d_1, \ldots, d_k)$, with $a_i \in \Sigma \cup \{¢, \$\}$ and $d_i \in \{\ell, s, r\}$, and with $q, q' \in Q$ being the current and the next states of the automaton. The automaton starts on a tape containing $¢w\$$ with all heads over $¢$ and having an internal state $q_0 \in Q$. At every step of the computation such an automaton may apply only transitions labelled by the current state q, and along with moving the heads it enters state q'. The automaton accepts by entering a designated state $q_f \in Q$.

It is known that for both for multihead 2NFAs with states and for 2DFAs with states, $k+1$ heads are better than k heads [2]. For stateless machines, we would like to be able to show a similar result, i.e., that for $k \geqslant 1$, stateless $(k+1)$-head 2NFAs (resp., 2DFAs) are better than those with only k heads. Although the case $k = 1$ is obvious, we are not able to give a proof for the general case at this time. Proving such a result using diagonalization (as in the case of automata with states [2]) seems quite difficult, as this would involve constructing a stateless multihead 2-NFA M that is capable of diagonalizing over all stateless k-head 2NFAs. However, it is not at all clear how M can accomplish this without states. Nevertheless, in Section 2, we show how to reduce the hierarchy problem for stateless multihead 2NFAs (resp., 2DFAs) to the hierarchy for multihead 2NFAs (resp. 2DFAs) with states. But the resulting hierarchy we obtain is not as tight, as we are only able to prove that stateless $(k+4)$-head 2NFAs (resp., 2DFAs) are better than those with k heads.

In Section 3, we consider stateless multihead one-way machines. We show that stateless $(k+1)$-head 1NFAs (resp., 1DFAs) are more powerful than stateless k-head 1NFAs (resp., 1DFAs), matching the known hierarchy for one-way machines with states. In Section 4, we show that the emptiness problem (deciding if the language accepted is empty) for stateless 2-head 2DFAs is undecidable, strengthening a recent result [7]. It remains an interesting open question whether this result can be shown to hold for stateless 2-head 1DFAs (or 1NFAs).

2 Stateless Multihead Two-Way Automata

We shall mainly establish hierarchies of stateless automata by simulating automata with states and using known hierarchy theorems for the latter automata. However, a rough infinite hierarchy of languages recognized by stateless multihead automata (with respect to heads) can be established directly, without using any previous work:

Proposition 1. *There is an infinite head-hierarchy of stateless multihead 1DFAs (resp., 1NFAs, 2NFAs, 2DFAs) over a unary alphabet.*

Proof. It is sufficient to show that for every $k \geqslant 1$, there is a language that cannot be accepted by any stateless k-head 2NFA but can be accepted by a stateless k'-head 1DFA for some $k' > k$.

For $k \geqslant 1$, define the singleton language $L_k = \{a^k\}$. L_k can be accepted by the stateless $(k+1)$-head 1DFA with the following transitions:

$$¢^{k+1} \to s^k r,\ ¢^k a \to s^{k-1} rr,\ ¢^{k-1} a^2 \to s^{k-2} rsr,\ ¢^{k-2} a^3 \to s^{k-3} rssr, \ldots,$$
$$¢ a^k \to r s^{k-1} r,\ a^k \$ \to s^{k+1}.$$

Clearly, for every k, there are at most a finite number of stateless k-head 2NFAs that we can define and, hence, only a finite number of distinct unary languages that can be accepted by such machines, and this number depends only on k. Let this number be $f(k)$. It follows that there is an $1 \leqslant i \leqslant f(k)+1$ such that L_i cannot be accepted by any stateless k-head 2NFA, but L_i can be accepted by a stateless $(i+1)$-head 1DFA and, hence, also by a stateless $(f(k)+1)$-head 1DFA. □

Let us now establish more precise separations. Our first hierarchy relies upon the following simulation:

Lemma 1. *Let M_1 be a k-head 2DFA (2NFA) with states, where $k \geqslant 1$. Let Σ be the input alphabet of M_1. Then there exists a stateless $(k+3)$-head 2DFA (2NFA, respectively) M_2 over $\Gamma \supset \Sigma$ and a string $x \in \Gamma^*$, such that $L(M_2) \cap x\Sigma^* = x \cdot L(M_1)$.*

Proof. Let M_1 have states $q_1, \ldots, q_n$, with initial state q_1 and unique halting/accepting state q_n. We assume that none of q_i's is in Σ. An input to M_1 is of the form $¢a_1 \ldots a_m\$$, with $m \geqslant 0$ and $a_i \in \Sigma$.

We show how to construct from M_1 a stateless $(k+3)$-head 2DFA or 2NFA M_2, which, when given $¢q_1 \ldots q_n a_1 \ldots a_n\$$, accepts if and only if M_1 accepts $¢a_1 \ldots a_n\$$, that is, the string x in the statement of the theorem is $q_1 \ldots q_n$. Given $¢q_1 \ldots q_n a_1 \ldots a_n\$$, M_2 simulates M_1 on the input $¢a_1 \ldots a_n\$$.

In the beginning, heads $k+1$ and $k+2$ stand over q_1, head $k+3$ remains at the left end marker, while heads $1, \ldots, k$ proceed to the beginning of the input. This is done by the following transitions:

$$¢^k ¢¢¢ \to r^k rrs$$
$$(q_i)^k q_1 q_1 ¢ \to r^k sss \quad (1 \leqslant i < n)$$

To simplify the notation, assume that the symbol q_n is the left end marker used by M_1 (instead of ¢). Then heads $1, \ldots, k$ assume their initial position at q_n.

Three extra heads of M_2 are used as follows. Head $k+3$ will stand either at ¢ or at q_1, thus storing a single bit, the number of steps of the simulated computation modulo 2. At the first step (as well as at every odd step), when head $k+3$ sees ¢, head $k+1$ scans the current state of M_1, while head $k+2$ is moving to the next state of M_1. At every even step, when head $k+3$ sees q_1, the roles of heads $k+1$ and $k+2$ are reversed: $k+2$ stands over the current state, while $k+1$ looks for the next state.

The behaviour at odd steps is implemented as follows. For each transition $(q_i, a_1, \ldots, a_k, q_{i'}, d_1, \ldots, d_k)$ of M_1, where $q_i, q_j \in Q$, $a_1, \ldots, a_k \in \Sigma \cup \{q_n, \$\}$ and $d_1, \ldots, d_k \in \{\ell, s, r\}$, define the following transition of M_2:

$$a_1 \ldots a_k q_i q_j ¢ \rightarrow \begin{cases} s^k srs \text{ if } q_j < q_{i'} \\ s^k s\ell s \text{ if } q_j > q_{i'} \\ d_1 \ldots d_k ssr \text{ if } q_j = q_{i'} \end{cases}$$

That is, while head $k+2$ scans a state other than $q_{i'}$, it moves towards $q_{i'}$, while other heads wait and continue scanning their symbols. This allows us to know the exact state of M_1 during the entire movements of head $k+2$. Once head $k+2$ reaches $q_{i'}$, the transition of M_1 is simulated in a single step of M_2, and at the same time head $k+3$ is moved from ¢ to q_1, thus indicating that it is head $k+2$ that currently sees the state of M_1, while head $k+1$ can be anywhere and should move towards the next state of M_1.

The behaviour at even steps of the computation of M_1 is implemented in M_2 symmetrically:

$$a_1 \ldots a_k q_j q_i q_1 \rightarrow \begin{cases} s^k rss \text{ if } q_j < q_{i'} \\ s^k \ell ss \text{ if } q_j > q_{i'} \\ d_1 \ldots d_k ss\ell \text{ if } q_j = q_{i'} \end{cases}$$

Finally, once M_1 enters the accepting state q_n, M_2 should accept as well, that is, and for all $q_j \in Q$ and $a_1, \ldots, a_k \in \Sigma \cup \{q_n, \$\}$,

$$a_1 \ldots a_k q_n q_j ¢ \rightarrow s^k sss$$
$$a_1 \ldots a_k q_j q_n q_1 \rightarrow s^k sss$$

This completes the construction of M_2, which is applicable both to deterministic and nondeterministic cases. □

It is known that $(k+1)$-head 2DFAs are more powerful than k-head 2DFAs [2], and the same result holds for 2NFAs. This gives an infinite hierarchy (with respect to heads) of stateless multihead two-way DFAs.

Theorem 1. *For $k \geqslant 1$, stateless $(k+4)$-head 2DFAs (2NFAs) are more powerful than stateless k-head 2DFAs (2NFAs, respectively).*

Proof. Let $L \subseteq a^*$ be a language defined by Monien [2], which is accepted by a $(k+1)$-head 2DFA M_1 with states (2NFA, respectively), but cannot be accepted by any k-head 2DFA with states (2NFA, respectively). Let M_2 be the corresponding $(k+4)$-head two-way stateless machine defined in Lemma 1, which recognizes $L' \subseteq \Gamma^*$ with $L' \cap x\Sigma^* = xL \subseteq xa^*$.

Suppose L' can be accepted by a stateless k-head 2DFA (2NFA) M_3. We can then construct from M_3 a k-head 2DFA (2NFA) with states M_4 accepting the original language L. The input to M_4 is $¢a^d\$$. M_4 simulates the computation of M_3 on $¢xa^d\$$, but since x is not on its input, M_4 simulates the moves of the k heads on x in its finite-state control. Hence, L can be accepted by a k-head 2DFA (2NFA) with states. This is a contradiction, which shows that L' is a desired example. □

It is an interesting open question whether Theorem 1 can be made tighter. Note that if one can improve the simulation in Lemma 1 so that M_2 needs less than $k+3$ heads, one can do this.

Next, we show that any language accepted by a multihead 2DFA (resp., 2NFA) with states can be accepted by a stateless multihead 2DFA (resp., 2NFA) at the price of more heads. The proof is based upon the following simulation, which is similar to the one by Yang, Dang and Ibarra [7].

Lemma 2. *Every language accepted by a k-head 2DFA (resp., 2NFA) with n states is accepted by a stateless $(k + \lceil \log_2 n \rceil)$-head 2DFA (resp., 2NFA).*

Proof. Consider an arbitrary k-head 2DFA (resp., 2NFA) M with states $q_0, \dots, q_{n-1}$. We construct a stateless DFA (resp., 2NFA) M' to simulate the k-head 2DFA M. The automaton M' has $k + \lceil \log_2 n \rceil$ heads: heads $1, \dots, k$ operate exactly as the corresponding heads of M, while the additional heads $k+1, \dots, k+\lceil \log_2(n+1) \rceil$ are used to keep track of the state. At every moment, the position of heads $k+1, \dots, k+\lceil \log_2(n+1) \rceil$ represents a number between 0 and $n-1$ in binary notation: if head $k+i$, with $1 \leq i \leq \lceil \log_2 n \rceil$, is at the left end marker marker, we consider the i-th bit as 0, and if it is at the next symbol to the left (whether it is the first symbol of the input, or the right end marker if the input is empty), we consider this bit as 1. This number represents the index of the current state of M.

The automaton M' starts with all heads on the left end marker; the position of heads $k+1, \dots, k+\lceil \log_2(n+1) \rceil$ represents the state q_0, that is, the initial state of M. At every step of the computation, M' simulates a single transition of M. It can see the current state of M from the symbols observed by heads $k+1, \dots, k+\lceil \log_2(n+1) \rceil$. Then M' moves its heads $1, \dots, k$ with all its hends on the left end marker according to the transition table of M, and at the same time moves its heads $k+1, \dots, k+\lceil \log_2(n+1) \rceil$ to encode the next state of M. □

Theorem 2. *Stateless multihead 2DFAs (resp., 2NFAs) are equivalent to multihead 2DFAs (resp., 2NFAs) with states, which are, in turn, equivalent to $\log n$ space-bounded deterministic (resp., nondeterministic) Turing machines.*

Since over a unary alphabet, $(k+1)$-head 1DFAs (resp., 1NFAs) with states are better than k-head 1DFAs (resp, 1NFAs) with states [2], we again obtain, as a corollary, that there is an infinite head-hierarchy of stateless multihead 2DFAs (resp., 2NFAs) over a unary alphabet.

3 Stateless Multihead One-way Automata

We now look at stateless multihead 1DFAs (resp., 1NFAs) and show a tight hierarchy. Our starting point is the result of Rosenberg [5] that the language

$$L_k = \{u_{\frac{k(k-1)}{2}} \# u_{\frac{k(k-1)}{2}-1} \# \dots \# u_2 \# u_1 \# v_1 \# v_2 \# \dots \# u_{\frac{k(k-1)}{2}-1} \# u_{\frac{k(k-1)}{2}} \mid u_i, v_i \in \{a,b\}^*, u_i = v_i\}$$

is recognized by a k-head 1DFA with states. Yao and Rivest [8] have further established that this language cannot be recognized by any $(k-1)$-head 1NFA with states. Using a variant of this language, we show a tight hierarchy for stateless multihead one-way automata.

Theorem 3. *There is a language that is accepted by a stateless k-head 1DFA that cannot be accepted by any $(k-1)$-head 1NFA with states.*

Proof. Let $m = \frac{k(k-1)}{2}$. Consider the language

$$L'_k = \{u_m \dagger_{m-1} u_{m-1} \dagger_{m-2} \cdots \dagger_1 u_1 \ddagger_1 v_1 \ddagger_2 v_2 \ddagger_3 \cdots \ddagger_m v_m \mid u_i, v_i \in \{a,b\}^*, u_i = v_i\}$$

over the alphabet $\Sigma_k = \{a, b, \dagger_1, \dots, \dagger_{m-1}, \ddagger_1, \dots, \ddagger_m\}$.

We construct a stateless k-head 1DFA M_1 which accepts all the strings in L'_k plus some extraneous strings not in L'_k. This is because M_1 cannot check the number, locations, and the markings (symbols different from a, b). However, as we shall see, these extraneous strings will not affect the correctness of the proof.

The construction, which is done inductively on k, is an adaptation of the method of Rosenberg [5]. While Rosenberg essentially relies on internal states, in our stateless construction the automaton is guided by the numbers attached to the markers.

Basis $k = 2$: the language $\{w \ddagger_1 w \mid w \in \{a,b\}^*\}$ is recognized by a 2-head 1-DFA with the following transitions: $¢¢ \to rs$, $a¢ \to rs$, $b¢ \to rs$, $\ddagger_1 ¢ \to rr$, $aa \to rr$, $bb \to rr$, $\$\ddagger_1 \to ss$.

Induction step. The computation proceeds as follows. At the first phase, heads are moved to their initial positions: head k goes to $\ddagger_{m-k+2}$, each head i $(2 \leqslant i \leqslant k-1)$ proceeds to $\dagger_{m-i+1}$, while head 1 stays at the start marker. At the second phase, head k moves across the substrings $v_{m-k+2}, \dots, v_m$, and as it starts from each $\ddagger_{m-i+1}$ to read v_{m-i+1}, head i simultaneously starts from $\dagger_{m-i+1}$ and reads u_{m-i+1}. Finally, at the third phase heads $1, \dots, k-1$ are moved to $\dagger_{m-k+1}$, from where the inner part of the string will be tested for membership in L'_{k-1} as claimed in the induction hypothesis. The third phase has a special form for $k = 3$.

The first phase is implemented by moving heads $2, \dots, k$ together, and once the destination of each head is reached, this head is left behind and the rest of the heads continue their movement, until k reaches its final point. This is done using following transitions:

$$\begin{aligned}
¢¢^{k-1} &\to sr^{k-1},\\
¢\dagger_{m-1}\cdots\dagger_{m-i+2}xx^{k-i} &\to s^{i-1}rr^{k-i} \quad \bigl(i \in \{2,\ldots,k-1\},\ x \in \{\dagger_{m-i+2}, a, b\}\bigr),\\
¢\dagger_{m-1}\cdots\dagger_{m-k+2}x &\to s^{k-1}r \quad \bigl(x \in \{\dagger_{m-k+1},\ldots,\dagger_1,\ddagger_1,\ldots,\ddagger_{m-k+1},a,b\}\bigr).
\end{aligned}$$

Note that the sequence $\dagger_{m-1}\cdots\dagger_{m-i+2}$ is empty when $i = 2$.

The movement of heads in phase two is defined in the following way:

$$¢\dagger_{m-1}\cdots\dagger_{m-i+2}x\dagger_{m-i-1}\cdots\dagger_{m-k+1}y \to s^{i-1}rs^{k-i-1}r$$
$$\text{(for all } i \in \{2,\ldots,k-1\} \text{ and } xy \in \{\dagger_{m-i+1}\ddagger_{m-i+1}, aa, bb\}\text{)},$$
$$x\dagger_{m-2}\cdots\dagger_{m-k+1}y \to rs^{k-2}r \quad \text{(for all } xy \in \{¢\ddagger_m, aa, bb\}\text{)},$$

Let us first define the third phase for the case $k = 3$. The tape contains $¢u_3\dagger_2u_2\dagger_1u_1\ddagger_1v_1\ddagger_2v_2\ddagger_3v_3\$$, and after the second phase head 1 is over $\dagger_2$, head 2 is over $\dagger_1$ and head 3 is over \$. Now head 2 is to be moved to $\ddagger_1$, which is done by transitions $\dagger_2x\$ \to srs$ with $x \in \{\dagger_1, a, b\}$, and then head 1 is moved to $\dagger_1$ using transitions $x\ddagger_1\$ \to rss$ with $x \in \{\dagger_2, a, b\}$. It remains to compare u_1 to v_1. Instead of applying the induction hypothesis, for $k = 3$ it is easier to implement this comparison again using transitions $xy\$ \to rrs$, for all $xy \in \{\dagger_1\ddagger_1, aa, bb\}$. Acceptance is done by $\ddagger_1\ddagger_2\$ \to sss$.

Let us now define phase three for $k \geqslant 4$. All heads should catch up with head $k-1$, which is currently over $\dagger_{m-k+1}$. The heads are moved one by one in the following order: first $k-2$, then $k-3$, and so on until head 1. The following transitions implement this:

$$\dagger_{m-1}\cdots\dagger_{m-i+1}x(\dagger_{m-k+1})^{k-i-2}\$ \to s^{i-1}rs^{k-i-1}$$
$$\text{(for all } i \in \{1,\ldots,k-2\} \text{ and } x \in \{\dagger_{m-i+1},\ldots,\dagger_{m-k+2},a,b\}\text{)}$$

Once the first three phases check the conditions $u_i = v_i$ for all $i \in \{m, m-1,\ldots,m-k+2\}$ and put heads $1,\ldots,k-1$ over $\dagger_{m-k+1}$, it remains to check the membership of the string $u_{m-k+1}\dagger_{m-k+2}\cdots\dagger_1u_1\ddagger_1v_1\ldots\ddagger_{m-k+1}v_{m-k+1}$ in L'_{k-1}. By the induction hypothesis, there exists a $(k-1)$-head DFA recognizing this language. Let $T \subseteq (\Sigma_{k-1})^{k-1} \times \{s,r\}^{k-1}$ be its set of transitions. For every transition $c_1\ldots c_{k-1} \to d_1\ldots d_{k-1}$ in this automaton, the constructed automaton contains the transition $c'_1\ldots c'_{k-1}\$ \to d_1\ldots d_{k-1}s$, where

$$c'_i = \begin{cases} \dagger_{m-k+1}, \text{ if } c_i = ¢\\ \ddagger_{m-k+2}, \text{ if } c_i = \$\\ \qquad c_i, \text{ otherwise}\end{cases}$$

The resulting automaton recognizes L'_k.

Now suppose $L(M_1)$ is accepted by a $(k-1)$-head 1NFA M_2 with states. Then, we can construct from M_2 a $(k-1)$-head 1NFA M_3 with states accepting the original language L_k as follows: When M_3 is given $¢w\$$ (note that the $\dagger$ and $\ddagger$ markings are not in w), M_3 simulates the computation of M_2, but uses its finite-state to remember the markings and their order and insert these markings at the appropriate places for the heads to simulate. Note also that M_3 can make sure that it is only simulating the computation of M_2 on strings with valid format. Hence L can be accepted by a $(k-1)$-head 1NFA with states. This is impossible. It follows that there is a language accepted by a stateless k-head 1DFA that cannot be accepted by k-head 1NFA. □

Corollary 1. *Stateless k-head 1DFAs (resp, 1NFAs) are strictly more powerful than stateless $(k-1)$-head 1DFAs (resp., 1NFAs).*

Let us now recall another result by Yao and Rivest [8], who constructed a language recognized by a 2-head 1NFA with states but not recognized by a k-head 1DFA with states for any k. The following stronger statement involving stateless 1NFAs can be established:

Theorem 4. *There exists a language recognized by a stateless 2-head 1NFA, which is not recognized by any k-head 1DFA with states for any k.*

Proof. Yao and Rivest [8] give the following example:

$$L = \{\#w_1x_1 \ldots \#w_nx_n \mid n \geqslant 0,\ w_i \in \{a,b\}^*,\ x_i \in \{0,1\}^*,\ \exists i \exists j : w_i = w_j, x_i \neq x_j\}$$

Let $\Sigma = \{\dagger, \ddagger, a, b, 0, 1, ¢, \$\}$ and consider a variant of the above language:

$$L' = \{\dagger\ddagger w_1x_1 \ldots \dagger\ddagger w_nx_n \mid n \geqslant 0,\ w_i \in \{a,b\}^*,\ x_i \in \{0,1\}^*,\ \exists i \exists j : w_i = w_j, x_i \neq x_j\}$$

Let us prove that this language is also not recognized by any k-head 1DFA with states. Suppose the contrary; then, given a k-head 1DFA with states for this language, one can easily construct a k-head 1DFA for L, which contradicts the result of Yao and Rivest [8, Th.4].

Construct a stateless 2-head 1NFA that recognizes L' modulo intersection with $(\dagger\ddagger\{a,b\}^*\{0,1\}^*)^*$. In general, this automaton operates similarly to the 2-head 1NFA with states sketched by Yao and Rivest, and uses double markers to simulate a few internal states. In the beginning, head 1 nondeterministically chooses an instance of $\ddagger$, using transitions $\sigma¢ \to rs$, for all $\sigma \in \{¢, \dagger, \ddagger, 0, 1, a, b\}$. Next, head 1 waits over $\ddagger$, while head 2 nondeterministically chooses another instance of $\ddagger$ as follows: $\ddagger\sigma \to sr$, for all $\sigma \in \{¢, \dagger, \ddagger, 0, 1, a, b\}$. Once head 1 scans $\ddagger$ in front of w_ix_i, while head 2 scans $\ddagger$ before w_jx_j, both heads synchronously move to the right, ensuring that $w_i = w_j$: $\ddagger\ddagger \to rr$, $aa \to rr$, $bb \to rr$. Once the symbols from x_i and x_j are encountered, the heads proceed further as long as these strings remain identical: $00 \to rr$, $11 \to rr$. If any symbols in x_i and x_j do not match, the string is accepted: $01 \to ss$, $10 \to ss$. If one of these substrings is shorter than the other, then one head arrives to $\dagger$, while the other still reads symbols; in this case the automaton also accepts: $\dagger 0 \to ss$, $\dagger 1 \to ss$, $0\dagger \to ss$, $1\dagger \to ss$. If x_i and x_j are identical, then both heads come to $\dagger$ simultaneously, and since the transition by $\dagger\dagger$ is undefined, the automaton rejects.

Let L'' be the language recognized by this automaton and suppose it is recognized by a k-head 1DFA with states for some $k \geqslant 1$. Then one can construct a k-head 1DFA with states for $L'' \cap (\dagger\ddagger\{a,b\}^*\{0,1\}^*)^* = L'$, which contradicts the claim proved above. □

Next we show, that even for unary inputs, multihead 1DFAs are surprisingly powerful:

Theorem 5. *For every $m \geqslant 1$, the singleton language $L_m = \{ a^{2^m-1} \}$ can be accepted by a stateless $(2m+1)$-head 1DFA.*

Proof. Of $2m+1$ heads used by the automaton, head 1 is the main head, and the rest of the heads form m pairs $(i, i+m)$. At the first step, heads $1, 2, \ldots, m+1$ (that is, the main head and the first head from each pairs) are moved to position 1, while heads $m+2, \ldots, 2m+1$ (second components of all pairs) remain in position 0.

Then heads $(m+1, 2m+1)$ (that is, the last pair), which are only one position apart, are moved towards the end of the string, until $m+1$ sees the end marker. From here, heads $1, 2, \ldots, m$ (the main head and the first components of all unused pairs) move synchronously with head $2m+1$, until head $2m+1$ sees the end marker. For the last pair, this will take only one step, and after that heads $1, 2, \ldots, m$ will be at position 2, heads $m+2, \ldots, 2m$ will be at the start marker, while heads $m+1$ and $2m+1$ will be at the end marker.

Then the next pair $(m, 2m)$ is taken, and the same sequence of steps is repeated. Note that the distance between these heads is now 2. The result is that heads m and $2m$ are moved to the end, while heads $1, 2, \ldots, m-1$ are moved to position 4. This is continued with the rest of the pairs, until the following configuration is reached: heads 1 and 2 are in position 2^{m-1}, head $m+2$ is in position 0, the rest of the heads are at the end marker.

From here, heads 2 and $m+2$ are moved towards the end of the string, until head 2 sees the end marker. At this point, heads 1 and $m+2$ are at the same position if and only if the length of the string is $2^m - 1$. After that head $m+2$ is moved together with head 1, and the input is accepted if and only if these two heads arrive to the end at the same time. This happens if and only if the input has length $2^m - 1$. □

4 The Emptiness Problem

It has been shown by Yang, Dang and Ibarra [7] that the emptiness problem (is the language accepted by a given machine empty?) for stateless 3-head 1DFAs is undecidable. It remains open whether this result holds for stateless 2-head 1DFAs (or 1NFAs). In this section, we show that the emptiness problem for stateless 2-head machines is undecidable if two-way movement is allowed.

Theorem 6. *The emptiness problem for stateless 2-head 2DFAs is undecidable, even when each head makes only one reversal on the input tape.*

The proof is by reduction from the emptiness problem for a restricted class of 2-head 1DFAs with states. Let us define this class.

Definition 1. *A 2-head 1DFA with states, with initial offset and with simultaneous movement of heads is a sextuple $(\Sigma, \#, Q, q_0, \delta, q_f)$, where $\# \in \Sigma$*

is a designated symbol, $q_0, q_f \in Q$ are the initial and the accepting states, $\delta : Q \times \Sigma \times \Sigma \to Q$ is the transition function. Given an input of the form $u\#v$, with $u \in \Sigma^+ \setminus \{\#\}^*$ and $v \in \Sigma^*$, the automaton starts in state q_0 with head 1 over the first symbol of u and head 2 over # in front of v. If the automaton is in state q, the first head scans a and the second head scans b, the automaton goes to state $\delta(q, a, b)$ and both heads are moved to the right by one square. The input is accepted if and only if the state when head 2 reaches the end of the string is q_f.

In a typical case, u will be much shorter than v, and eventually head 1 will reach the marker #. It will process it uniformly with the rest of the symbols, according to the transition function.

Lemma 3. *The emptiness problem for the class of 2-head 1DFAs with states given in Definition 1 is undecidable.*

Proof. Let us define a variant of the language of valid accepting computations of a Turing machine T operating over the input alphabet Γ. The configuration of T on the input $w \in \Gamma^*$ at step i using workspace s is given by a string of length s over some auxiliary alphabet Ω. Denote this string by $C_T(w, s, i)$. Then the language of computation histories is defined as

$$\begin{aligned}\mathrm{VALC}(T) = \{C_T(w,s,0)\#C_T(w,s,1)\natural \ldots \natural C_T(w,s,n) \mid \\ \text{at each } i\text{-th step T uses at most } s \text{ squares} \\ C_T(w,s,n) \text{ is an accepting configuration}\}\end{aligned}$$

The exact form of C_T can be defined so that this language can be recognized by a two-head automaton as is Definition 1. On the other hand, $\mathrm{VALC}(T) = \varnothing$ if and only if $L(T) = \varnothing$. Since the emptiness of a Turing machine is undecidable, so is the given decision problem. □

Proof (Proof of Theorem 6). The proof is a reduction from the emptiness problem for the automata given in Definition 1.

Let $A = (\Sigma, \#, Q, q_{init}, \delta, q_f)$ be such an automaton, let $\Sigma' = \Sigma \times Q \times Q \times \{1, 2\}$. Let $w = a_1 \ldots a_{m-1}\#a_{m+1} \ldots a_n$ be a string given to A, let q_i $(m \leqslant i \leqslant n)$ be the state of A after 2nd head reads a_i. Then $q_m = q_{init}$. For convenience, define $q_0 = q_1 = \ldots = q_{m-1} = q_{init}$ (though head 2 never reads $a_0, \ldots a_{m-1}$). Then the computation of A on w is represented by the following string over Σ':

$$x_n^{(1)} x_n^{(2)} x_{n-1}^{(1)} x_{n-1}^{(2)} \ldots x_1^{(1)} x_1^{(2)}, \quad \text{with } x_i^{(j)} = (a_i, q_{i-1}, q_i, j) \tag{1}$$

Each quadruple (a_i, q_{i-1}, q_i, j) represents A with its heads 1 and 2 in positions $i - m$ and i, respectively, with symbol a_i under head 2, currently being in state q_{i-1} and about to enter state q_i. Note that the order of symbols is reversed, and each symbol of w is represented by an “odd” and an “even” symbol, which differ only in the last component.

Construct a stateless 2-head 2DFA B over Σ' to accept the language of all strings of this form corresponding to the strings accepted by A. At the first stage of the computation of B, its heads go together to the end of the input, with head 2 always being one square ahead of head 1. While travelling like this, the heads check the general form (1) of the computation. This behaviour is implemented by the following transitions:

$$\text{¢¢} \to sr \tag{2}$$
$$\text{¢}(a, q_f, q', 1) \to rr \quad (a \in \Sigma; q, q' \in Q) \tag{3}$$
$$(a, q, q', 1)(a, q, q', 2) \to rr \quad (a \in \Sigma; q, q' \in Q) \tag{4}$$
$$(a, q', q'', 1)(b, q, q', 2) \to rr \quad (a, b \in \Sigma; q, q', q'' \in Q) \tag{5}$$

Transition (3) checks the last state for being accepting. If an odd and an even symbol in some pair have different data, then (4) will not be applicable and the input will be rejected. Similarly, if two consecutive pairs violate the sequence of states, then (5) is not applicable.

Once head 2 reaches the end marker, with head 1 lagging behind by one symbol, the heads exchange their positions using the transition

$$(a, q_{init}, q_{init}, 2)\$ \to r\ell, \tag{6}$$

and then head 1 stays over the end marker, while head 2 proceeds to the left until it encounters #:

$$\$(a, q_{init}, q_{init}, i) \to s\ell \quad (a \in \Sigma \setminus \{\#\}, i \in \{1, 2\})$$
$$\$(\#, q_{init}, q, 2) \to \ell s \quad (q \in Q)$$

At this point, head 1 is over the last symbol before \$, which should be of the form $(a, q_{init}, q_{init}, 2)$, while head 2 scans the leftmost symbol $(\#, q_{init}, q, 2)$. At this time, both heads are reading even symbols, and they start simultaneously moving left, maintaining equal parity of the symbols they scan. This allows the transitions in this phase to be distinct from the previously defined transitions. The following transitions simulate the operation of A:

$$(b, q'', q''', i)(a, q, q', i) \to \ell\ell \quad (a, b \in \Sigma; q, q', q'', q''' \in Q; \delta(q, a, b) = q'; i \in \{1, 2\})$$

If all transitions are correct, head 2 will eventually reach the start marker, where B accepts:

$$(b, q'', q'', 2)\text{¢} \to ss \quad (b \in \Sigma; q'', q''' \in Q)$$

It remains to consider the cases when the input is ill-formed. Suppose the general form (1) is violated, that is, let the string be of the form

$$x_n^{(1)} x_n^{(2)} x_{n-1}^{(1)} x_{n-1}^{(2)} \ldots x_i^{(j)} y \ldots \tag{7}$$

where symbols up to $x_i^{(j)}$ are as in (1), while y is not as required. Then B eventually reaches a configuration with head 1 over $x_i^{(j)}$ and head 2 over y.

Suppose it is the alternation of even and odd symbols that has been violated. Then $x_i^{(j)} = (a_i, q_{i-1}, q, j)$ and $y = (b, q', q'', j)$, and the transition is either undefined, or it is of the form $(a, q, q', i)(b, q'', q''', i) \rightarrow \ell\ell$. In the latter case, both heads move left by one symbol and reach their previous configuration, from which they will again move over $x_i^{(j)}$ and y. Thus the computation goes into an infinite loop.

If the string prematurely ends with an odd symbol, then eventually head 1 will scan $(a, q, q', 1)$, while head 2 will scan \$. The transition (6) will not be applicable, and the input will be rejected. In this way the syntax of the input string will be checked before the simulation of A starts, and hence syntactic garbage will not cause mistakes in the simulation. □

Although we are not able to resolve at this time the question of whether or not the emptiness problem for stateless 2-head 1DFAs (or 1NFAs) is undecidable, we can show an interesting result using the following lemma.

Lemma 4 (Domaratzki [1]). *Let Σ be an alphabet, let $\Sigma' = \{a' \mid a \in \Sigma\}$ be its copy and define a homomorphism $h : \Sigma^* \rightarrow (\Sigma')^*$ by $h(a) = a'$ for all $a \in \Sigma^*$. Then the language $\bigcup_{w \in \Sigma^*} w \sqcup\!\sqcup h(w)$ is recognized by a stateless 2-head 1DFA.*

Theorem 7. *There is a fixed stateless 2-head 1DFA M_1 over a 4-letter alphabet, such that it is undecidable to determine, given a DFA M_2, whether or not $L(M_1) \cap L(M_2) = \varnothing$.*

Proof. Let $\Sigma = \{a, b\}$ and consider the twin shuffle language L_1 as in Lemma 4, defined over the alphabet $\{a, b, a', b'\}$. Let $\{(u_1, v_1), \ldots, (u_m, v_m)\}$ be an instance of PCP over $\{a, b\}$, and consider the regular language $L_2 = \left(\bigcup_{i=1}^{m} u_i h(v_i)\right)^+$. The intersection $L_1 \cap L_2$ is empty if and only if this is a yes-instance. Since PCP is undecidable, this proves undecidability of the emptiness of intersection. □

It remains an interesting open question whether the emptiness problem for stateless 2-head 1DFAs (or 1NFAs) is undecidable.

References

1. Domaratzki, M.: Personal communication (August 2007)
2. Monien, B.: Two-way multihead automata over a one-letter alphabet. RAIRO Informatique theoretique 14(1), 67–82 (1980)
3. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000)
4. Păun, G.: Membrane Computing, An Introduction. Springer, Heidelberg (2002)
5. Rosenberg, A.L.: On multi-head finite automata. IBM Journal of Research and Development 10(5), 388–394 (1966)
6. Yang, L., Dang, Z., Ibarra, O.H.: Bond computing systems: a biologically inspired and high-level dynamics model for pervasive computing. In: Akl, S.G., Calude, C.S., Dinneen, M.J., Rozenberg, G., Wareham, H.T. (eds.) UC 2007. LNCS, vol. 4618, pp. 226–241. Springer, Heidelberg (2007)
7. Yang, L., Dang, Z., Ibarra, O.H.: On stateless automata and P systems. In: Pre-Proceedings of Workshop on Automata for Cellular and Molecular Computing (August 2007)
8. Yao, A.C., Rivest, R.L.: $k + 1$ heads are better than k. Journal of the ACM 25(2), 337–340 (1978)

Myhill-Nerode Theorem for Recognizable Tree Series Revisited[⋆]

Andreas Maletti[⋆⋆]

International Computer Science Institute
1947 Center Street, Suite 600, Berkeley, CA 94704, USA

Abstract. In this contribution the Myhill-Nerode congruence relation on tree series is reviewed and a more detailed analysis of its properties is presented. It is shown that, if a tree series is deterministically recognizable over a zero-divisor free and commutative semiring, then the Myhill-Nerode congruence relation has finite index. By [Borchardt: *Myhill-Nerode Theorem for Recognizable Tree Series*. LNCS 2710. Springer 2003] the converse holds for commutative semifields, but not in general. In the second part, a slightly adapted version of the Myhill-Nerode congruence relation is defined and a characterization is obtained for all-accepting weighted tree automata over multiplicatively cancellative and commutative semirings.

1 Introduction

By the Myhill-Nerode theorem, we know that for every regular string language L, there exists a unique (up to isomorphism) minimal deterministic finite string automaton that recognizes L. This result was extended to several devices including finite tree automata (see the discussion in [1]), to weighted string automata [2] over (multiplicatively) cancellative semirings, and to weighted tree automata [3] over semifields (see [4,5] for an introduction to semirings). For the weighted devices, the minimal deterministic automaton is no longer unique up to isomorphism. The structure of it is still unique but the distribution of the weights on the transitions may vary. In [2] this is called *unique up to pushing*. Weighted tree automata and transducers recently found promising applications (see [6] for a survey) in natural language processing, where the size of the automata is crucial and thus minimization essential.

Let us recall the Myhill-Nerode congruence of [7]. Two trees t and u are equal in the Myhill-Nerode congruence $\equiv_\psi$ for a given tree series ψ over the semifield $(A, +, \cdot, 0, 1)$, if there exist nonzero coefficients $a, b \in A$ such that for all contexts C we observe the equality $a^{-1} \cdot (\psi, C[t]) = b^{-1} \cdot (\psi, C[u])$. In this expression, the coefficients a and b can be understood as the weights of t and u,

[⋆] This is the extended abstract of a talk first presented at AutoMathA 2007, Palermo, Italy.

[⋆⋆] Author is on leave from the Institute for Theoretical Computer Science, Technische Universität Dresden, 01062 Dresden, Germany. The author gratefully acknowledges the support of DAAD and the AutoMathA program of the ESF.

respectively. Both sides of the previous equation can be understood as futures; the futures ψ_t and ψ_u are given for every context C by $(\psi_t, C) = a^{-1} \cdot (\psi, C[t])$ and $(\psi_u, C) = b^{-1} \cdot (\psi, C[u])$. Roughly speaking, in ψ_t a context is assigned the weight of $C[t]$ in ψ with the weight of t cancelled out. In other words, trees t and u are equal if and only if their futures ψ_t and ψ_u coincide.

The Myhill-Nerode congruence $\equiv_\psi$ has two major applications: (i) it exactly characterizes whether ψ is deterministically recognizable; i.e., ψ is deterministically recognizable if and only if $\equiv_\psi$ has finite index; and (ii) it presents a minimal deterministic wta that recognizes the tree series ψ. In this contribution, we consider the Myhill-Nerode relation for semirings which are not necessarily semifields. We will show that, for all commutative and zero-divisor free (i.e., $a \cdot b = 0$ implies that $0 \in \{a, b\}$) semirings, a deterministically recognizable tree series ψ yields a Myhill-Nerode congruence $\equiv_\psi$ with finite index. Thus whenever $\equiv_\psi$ has infinite index, then ψ is not deterministically recognizable. This extends a result of [7] from commutative semifields to commutative and zero-divisor free semirings. Secondly, we also consider the opposite direction with a particular focus on the minimal deterministic wta. We show how *all-accepting* [8] wta over cancellative semirings are related to unweighted tree automata. This connection can be used to minimize deterministic all-accepting wta over commutative and cancellative semirings. A Myhill-Nerode theorem for all-accepting wta over semifields is already presented in [8]. We contribute an explicit minimization and an extension of the result to cancellative semirings. Note that every cancellative semiring can be embedded into a semifield, but solving the problem in the semifield might yield a wta using coefficients that do not exist in the cancellative semiring (e.g., for the natural numbers the resulting wta might use fractions). It then remains open whether a wta using only coefficients of the cancellative semiring exists.

Finally, we also investigate the construction of a minimal wta in the general case (again over a cancellative semiring). To this end, we present a slightly adapted Myhill-Nerode relation. However, one main point remains open: In cancellative semirings (as opposed to semifields) the Myhill-Nerode congruence relation is not always implementable. It remains an open problem to define suitable properties on ψ and the underlying semiring $\mathcal{A}$ such that the refined Myhill-Nerode congruence is implementable. We demonstrate the applicability of the general approach by deriving such properties and thus a Myhill-Nerode theorem for deterministic all-accepting weighted tree automata.

2 Preliminaries

We use $\mathbb{N}$ to represent the nonnegative integers. Further we denote $\{n \in \mathbb{N} \mid 1 \leqslant n \leqslant k\}$ by $[1, k]$. A set Σ that is nonempty and finite is also called an *alphabet*. A *ranked alphabet* is an alphabet Σ with a mapping $\mathrm{rk}_\Sigma \colon \Sigma \to \mathbb{N}$. We write Σ_k for $\{\sigma \in \Sigma \mid \mathrm{rk}_\Sigma(\sigma) = k\}$. Given a ranked alphabet Σ, the set of *Σ-trees*, denoted by T_Σ, is inductively defined to be the smallest set T such that for every $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $t_1, \ldots, t_k \in T$ also $\sigma(t_1, \ldots, t_k) \in T$. We generally write α instead of $\alpha()$ whenever $\alpha \in \Sigma_0$. Let $\square$ be a distinguished nullary symbol. A *context* C is a tree from $T_{\Sigma \cup \{\square\}}$ such that the nullary symbol $\square$ occurs exactly once in C. The set of all contexts over Σ is denoted by C_Σ. Finally, we write $C[t]$ for the tree of T_Σ that is obtained by replacing in the context $C \in C_\Sigma$ the unique occurrence of $\square$ with the tree $t \in T_\Sigma$.

Let $\equiv$ and $\cong$ be equivalence relations on a set S. We write $[s]_\equiv$ for the equivalence class of $s \in S$ and $(S/{\equiv}) = \{[s]_\equiv \mid s \in S\}$ for the set of equivalence classes. We drop the subscript from $[s]_\equiv$ whenever it is clear from the context. We say that $\equiv$ is *coarser* than $\cong$ if $\cong \subseteq \equiv$. Now suppose that $S = T_\Sigma$. We say that $\cong$ is a *congruence* (on the term algebra T_Σ) if for every $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $s_1, \dots, s_k, t_1, \dots, t_k \in T_\Sigma$ with $s_i \cong t_i$ also $\sigma(s_1, \dots, s_k) \cong \sigma(t_1, \dots, t_k)$.

A *(commutative) semiring* is an algebraic structure $(A, +, \cdot, 0, 1)$ consisting of two commutative monoids $(A, +, 0)$ and $(A, \cdot, 1)$ such that $\cdot$ distributes over $+$ and 0 is absorbing with respect to $\cdot$. As usual we use $\sum_{i \in I} a_i$ for sums of families $(a_i)_{i \in I}$ of $a_i \in A$ where for only finitely many $i \in I$ we have $a_i \neq 0$. The semiring $(A, +, \cdot, 0, 1)$ is called *zero-sum free* if for every $a, b \in A$ the condition $a + b = 0$ implies that $a = 0 = b$. We call a semiring $(A, +, \cdot, 0, 1)$ *zero-divisor free* if $a \cdot b = 0$ implies that $a = 0$ or $b = 0$. Moreover, $\mathcal{A}$ is called *cancellative* if $a \cdot b = a \cdot c$ implies $b = c$ for every $a, b, c \in A$ with $a \neq 0$. Generally, we write $a|b$ whenever there exists an element $c \in A$ such that $a \cdot c = b$. Note that in a cancellative semiring such an element c, if any exists, is uniquely determined unless $a = 0 = b$. In cancellative semirings, we thus write b/a for that uniquely determined element c provided that (i) $a|b$ and (ii) $a \neq 0$ or $b \neq 0$. Finally, a *semifield* $\mathcal{A} = (A, +, \cdot, 0, 1)$ is a semiring such that for every $a \in A \setminus \{0\}$ there exists an element $a^{-1} \in A$ such that $a \cdot a^{-1} = 1$.

Let S be a set and $(A, +, \cdot, 0, 1)$ be a semiring. A *(formal) power series* ψ is a mapping $\psi \colon S \to A$; the set of all such mappings is denoted by $\mathcal{A}\langle\langle S \rangle\rangle$. Given $s \in S$, we denote $\psi(s)$ also by (ψ, s) and write the series as $\sum_{s \in S} (\psi, s)\, s$. The *support* of ψ is $\operatorname{supp}(\psi) = \{s \in S \mid (\psi, s) \neq 0\}$. The series with empty support is denoted by $\widetilde{0}$. Power series $\psi, \psi' \in \mathcal{A}\langle\langle S \rangle\rangle$ are added componentwise and multiplied componentwise with a semiring element; i.e., $(\psi + \psi', s) = (\psi, s) + (\psi', s)$ and $(a \cdot \psi, s) = a \cdot (\psi, s)$ for every $s \in S$ and $a \in A$. In this paper, we only consider power series in which the set S is a set of trees. Such power series are also called *tree series*.

There exists an abundance of (conceptionally) equivalent definitions of weighted tree automata [9,10,11] for various restricted semirings. Here we will only consider the general notion of [11,7]. A *weighted tree automaton* [7] (for short: wta) is a tuple $(Q, \Sigma, \mathcal{A}, F, \mu)$ where Q is a nonempty, finite set of *states*; Σ is a ranked alphabet of *input symbols*; $\mathcal{A} = (A, +, \cdot, 0, 1)$ is a semiring; $F \colon Q \to A$ is a *final weight assignment*; and $\mu = (\mu_k)_{k \in \mathbb{N}}$ with $\mu_k \colon \Sigma_k \to A^{Q^k \times Q}$ is a *tree representation*. The wta M is called *(bottom-up) deterministic* (respectively, *(bottom-up) complete*), if for every $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $q_1, \dots, q_k \in Q$ there exists at most (respectively, at least) one $q \in Q$ such that $\mu_k(\sigma)_{q_1 \cdots q_k, q} \neq 0$. The wta induces a mapping $h_\mu \colon T_\Sigma \to A^Q$ that is defined for every $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, $q \in Q$, and $t_1, \dots, t_k \in T_\Sigma$ by

$$h_\mu(\sigma(t_1,\dots,t_k))_q = \sum_{q_1,\dots,q_k \in Q} \mu_k(\sigma)_{q_1\cdots q_k,q} \cdot h_\mu(t_1)_{q_1} \cdot \ldots \cdot h_\mu(t_k)_{q_k} \enspace .$$

The wta M recognizes the tree series $S(M) \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$ given by

$$(S(M),t) = \sum_{q \in Q} F(q) \cdot h_\mu(t)_q$$

for every tree $t \in T_\Sigma$. A tree series $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$ is called *recognizable* (respectively, *deterministically recognizable*), if there exists a wta M (respectively, deterministic wta M) such that $S(M) = \psi$. The sets of all recognizable and deterministically recognizable tree series are denoted by $\mathcal{A}^{\text{rec}}\langle\langle T_\Sigma \rangle\rangle$ and $\mathcal{A}^{\text{rec}}_{\text{det}}\langle\langle T_\Sigma \rangle\rangle$, respectively.

3 Recognizable Yields Finite Index

In this section, we show that the Myhill-Nerode congruence given by [3, Section 5] is necessarily of finite index for every deterministically recognizable series over a zero-divisor free semiring. Thus we derive a necessary criterion for a series $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$ to be recognizable by some deterministic wta. Moreover, we also obtain a lower bound on the number of states of any deterministic wta that recognizes ψ. The development in this section closely follows [7, Chapter 7] where the same statements are proved for semifields.

Let us start with the definition of the Myhill-Nerode relation for a tree series $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$. Intuitively, two trees $t, u \in T_\Sigma$ are related if they behave equal in all contexts $C \in C_\Sigma$ (up to fixed factors). The factors can be imagined to be the weights of the trees t and u.

Definition 1 (see [7, Chapter 7]). *Let $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$. The relation $\equiv_\psi \subseteq T_\Sigma \times T_\Sigma$ is defined for every $t, u \in T_\Sigma$ by $t \equiv_\psi u$ if and only if there exist $a, b \in A \setminus \{0\}$ such that for every $C \in C_\Sigma$ we observe*

$$a \cdot (\psi, C[t]) = b \cdot (\psi, C[u]) \enspace .$$

The relation $\equiv_\psi$ is equivalent to the Myhill-Nerode relation presented in [3, Section 5] provided that the semiring is a semifield. Our first lemma states that $\equiv_\psi$ is indeed an equivalence relation, and moreover, a congruence whenever the underlying semiring is zero-divisor free.

Lemma 2 (cf. [7, Lemma 7.1.2(ii)]). *Let $\mathcal{A}$ be a zero-divisor free semiring and $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$. Then $\equiv_\psi$ is a congruence.*

Proof. The proof follows the proof of [7, Lemma 7.1.2(ii)], where it is proved for semifields. □

We presented a congruence which is uniquely determined by ψ. First we show that every deterministic and complete wta $M = (Q, \Sigma, \mathcal{A}, F, \mu)$ also induces a congruence relation. For the development of this we need some additional notions. The *fta underlying* M (see [12,13] for a detailed introduction to *finite tree automata*; for short: fta) is defined as $\mathcal{B}(M) = (Q, \Sigma, \delta, F')$ where

– $q \in \delta_\sigma(q_1, \dots, q_k)$ iff $\mu_k(\sigma)_{q_1 \cdots q_k, q} \neq 0$ for every $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $q, q_1, \dots, q_k \in Q$; and
– $q \in F'$ iff $F_q \neq 0$ for every $q \in Q$.

We note that the fta underlying a deterministic and complete wta is deterministic and complete. Now let M be a deterministic and complete wta and $\mathcal{B}(M) = (Q, \Sigma, \delta, F')$ be the fta underlying M. We define the mapping $R_M \colon T_\Sigma \to Q$ for every $t \in T_\Sigma$ by $R_M(t) = q$ where $q \in Q$ is the unique state such that $q \in \delta(t)$. Existence and uniqueness are guaranteed by completeness and determinism of $\mathcal{B}(M)$, respectively. We denote $\ker(R_M)$ by $\equiv_M$. The following lemma follows the traditional unweighted approach.

Lemma 3. *Let M be a deterministic and complete wta over Σ. Then $\equiv_M$ is a congruence with finite index.*

Having two congruences, namely $\equiv_M$ and $\equiv_{S(M)}$, let us try to relate them. In fact, it turns out that $\equiv_{S(M)}$ is coarser than $\equiv_M$ for every deterministic and complete wta M over a zero-divisor free semiring. This shows that we need at least as many states as there are equivalence classes in $\equiv_\psi$ to recognize ψ with some deterministic and complete wta.

Theorem 4. *Let $\mathcal{A}$ be a zero-divisor free semiring, and let M be deterministic and complete wta over $\mathcal{A}$. Then $\equiv_{S(M)}$ is coarser than $\equiv_M$.*

Proof. Let $M = (Q, \Sigma, \mathcal{A}, F, \mu)$, and let $t, u \in T_\Sigma$ be such that $t \equiv_M u$; that is $R_M(t) = R_M(u)$. Let $p = R_M(t)$. Thus, also $R_M(C[t]) = R_M(C[u])$. Let $a = h_\mu(u)_{R_M(u)}$ and $b = h_\mu(t)_{R_M(t)}$. We claim that for every context $C \in C_\Sigma$

$$a \cdot (S(M), C[t]) = b \cdot (S(M), C[u]) \enspace .$$

Let us distinguish two cases for $q = R_M(C[t])$. First, let us suppose that $F_q = 0$. Then the displayed equation holds because $(S(M), C[t]) = 0 = (S(M), C[u])$. In the remainder suppose that $F_q \neq 0$. Clearly, since $t \equiv_M u$ also $C[t] \equiv_M C[u]$ because $\equiv_M$ is a congruence by Lemma 3. Thus

$$\begin{aligned} a \cdot (S(M), C[t]) &= h_\mu(u)_p \cdot F_q \cdot h_\mu(C[t])_q = h_\mu(u)_p \cdot F_q \cdot h_\mu(C)^p_q \cdot h_\mu(t)_p \\ &= h_\mu(t)_p \cdot F_q \cdot h_\mu(C[u])_q = b \cdot (S(M), C[u]) \end{aligned}$$

where $h_\mu(C[t])_q = h_\mu(C)^p_q \cdot h_\mu(t)_p$ can be proved in a straightforward manner. Consequently, $\equiv_{S(M)}$ is coarser than $\equiv_M$. □

As already argued this theorem admits an important corollary, which shows a lower bound on the number of states of any deterministic and complete wta that recognizes a certain series.

Corollary 5. *Let $\mathcal{A}$ be a zero-divisor free semiring and $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$. Every deterministic and complete wta M over $\mathcal{A}$ with $S(M) = \psi$ has at least* index($\equiv_\psi$) *states.*

Proof. Let $M = (Q, \Sigma, \mathcal{A}, F, \mu)$. By Theorem 4, we have that $\equiv_\psi$ is coarser than $\equiv_M$. Thus $\operatorname{card}(Q) \geq \operatorname{index}(\equiv_M) \geq \operatorname{index}(\equiv_\psi)$. □

Let us show that the statement does not hold, if we consider arbitrary semirings. Essentially, if the semiring admits zero-divisors, then it can store information in the weight.

Example 6. Let $\mathbb{Z}_4 = (\{0,1,2,3\}, +, \cdot, 0, 1)$ where $+$ and $\cdot$ are the usual addition and multiplication, respectively, modulo 4. Clearly, $2 \cdot 2 = 0$ and thus $\mathbb{Z}_4$ is not zero-divisor free. Let $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$. We consider the series $\psi \in \mathbb{Z}_4\langle\langle T_\Sigma \rangle\rangle$ which is defined for every $t \in T_\Sigma$ by

$$(\psi, t) = \begin{cases} 1 & \text{if } |t|_\alpha = 0 \\ 2 & \text{if } |t|_\alpha = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Let $M = (\{\star\}, \Sigma, \mathbb{Z}_4, F, \mu)$ with $F_\star = 1$ and

$$\mu_0(\alpha)_\star = 2 \qquad \mu_0(\beta)_\star = 1 \qquad \mu_2(\sigma)_{\star\star,\star} = 1 \ .$$

It can easily be checked that $S(M) = \psi$. Let us suppose that still $\equiv_\psi$ is coarser than $\equiv_M$. Since $\operatorname{card}(Q) = 1$ this means that $t \equiv_\psi u$ for all $t, u \in T_\Sigma$. We consider the trees $\sigma(\alpha, \alpha)$ and β. By definition, we should have that there exist $a, b \in [1, 3]$ such that for every $C \in C_\Sigma$

$$a \cdot (\psi, C[\sigma(\alpha, \alpha)]) = b \cdot (\psi, C[\beta]) \ .$$

Now consider the context $C = \Box$. Thus $a \cdot 0 = b \cdot 1$ and thus $b = 0$. However, $b \in [1, 3]$ which is the desired contradiction. Thus $\sigma(\alpha, \alpha) \not\equiv_\psi \beta$ and $\equiv_\psi$ is not coarser than $\equiv_M$.

Finally, let us conclude this section with an application of Corollary 5. We can envision at least two uses of Corollary 5. It can be used to show that some wta is minimal (or almost so), and it can be used to show that some tree series ψ is not recognizable. The standard examples for the latter use concerns the tree series size and height over the natural numbers and the arctic semiring $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$, respectively (see discussion at [7, Examples 7.3.2 and 8.1.8]). We demonstrate the application of Corollary 5 on another example.

Example 7. Let Σ be a ranked alphabet such that $\Sigma = \Sigma_2 \cup \Sigma_0$. We use the tropical semiring $\mathcal{T} = (\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)$. It is easily checked that $\mathcal{T}$ is zero-divisor free (and cancellative), but not a semifield. We define the mapping $\operatorname{zigzag}\colon T_\Sigma \to \mathbb{N}$ for every $\alpha \in \Sigma_0$ and $\sigma, \delta \in \Sigma_2$ and $t_1, t_2, t_3 \in T_\Sigma$ by

$$\operatorname{zigzag}(\alpha) = 1$$
$$\operatorname{zigzag}(\sigma(\alpha, t_1)) = 2$$

$$\mathrm{zigzag}(\sigma(\delta(t_1, t_2), t_3)) = 2 + \mathrm{zigzag}(t_2) \ .$$

It is straightforward to show that zigzag $\in \mathcal{T}^{\mathrm{rec}}\langle\langle T_\Sigma \rangle\rangle$. In fact, zigzag can be recognized by a top-down deterministic wta [7, Section 4.2] with only 2 states. But can zigzag be recognized by a (bottom-up) deterministic wta over Σ and $\mathcal{T}$? We use Corollary 5 to show that no deterministic wta recognizes zigzag. Clearly, this is achieved by proving that $\equiv_{\mathrm{zigzag}}$ has infinite index. Let $t, u \in T_\Sigma$. Suppose that $t \equiv_{\mathrm{zigzag}} u$. Then there exist $a, b \in \mathbb{N}$ such that for every context $C \in C_\Sigma$

$$a + (\mathrm{zigzag}, C[t]) = b + (\mathrm{zigzag}, C[u]) \ .$$

Now consider the contexts $C_1 = \square$ and $C_2 = \sigma(\alpha, \square)$. We obtain the equations

$$a + \mathrm{zigzag}(t) = b + \mathrm{zigzag}(u)$$
$$a + 2 = b + 2$$

From the second equality we can conclude that $a = b$ and $\mathrm{zigzag}(t) = \mathrm{zigzag}(u)$. Hence ker(zigzag) is coarser than $\equiv_{\mathrm{zigzag}}$. However, ker(zigzag) has infinite index, which shows that also $\equiv_{\mathrm{zigzag}}$ has infinite index, and thus, by Corollary 5, no deterministic wta can recognize zigzag.

4 Finite Index Yields Recognizable

In this section we investigate whether the lower bound established in the previous section can be achieved. Certainly, [7, Theorem 7.4.1] shows that for every $\psi \in \mathcal{A}^{\mathrm{rec}}_{\mathrm{det}}\langle\langle T_\Sigma \rangle\rangle$ (with $\mathcal{A}$ a semifield) there exists a deterministic and complete wta over $\mathcal{A}$ with exactly index($\equiv_\psi$) states so that $S(M) = \psi$. In this section we investigate this issue for deterministic all-accepting wta over cancellative semirings. The principal approach can also be extended to deterministic wta over certain cancellative semirings. Let us illustrate the problem in the semiring $(\mathbb{N}, +, \cdot, 0, 1)$ that is not a semifield but cancellative.

Consider the series $\psi \colon T_\Sigma \to \mathbb{N}$ with $\Sigma = \{\gamma^{(1)}, \alpha^{(0)}\}$ and

$$(\psi, \gamma^n(\alpha)) = \begin{cases} 2 & \text{if } n = 0 \\ 3 & \text{if } n = 1 \\ 4 & \text{otherwise.} \end{cases}$$

It is easily checked that $\alpha \not\equiv_\psi \gamma(\alpha) \not\equiv_\psi \gamma^n(\alpha) \not\equiv_\psi \alpha$ for every $n > 1$ as well as $\gamma^m(\alpha) \equiv_\psi \gamma^n(\alpha)$ for every $m > 1$ and $n > 1$. Thus, index($\equiv_\psi$) = 3. However, it can be shown that there exists no deterministic all-accepting [8] wta M such that $S(M) = \psi$. On the other hand, it is surprisingly easy to construct a deterministic wta M such that $S(M) = \psi$. In fact, M can be constructed such that it has 3 states.

4.1 Minimization of Deterministic All-Accepting wta

Let us discuss the problem for deterministic *all-accepting* wta [8, Section 3.2]. It is known that for every deterministic all-accepting wta M over a semifield there exists a unique (up to isomorphism) minimal deterministic and complete all-accepting wta that recognises $S(M)$ [8, Lemma 3.8]. We plan to extend this result to cancellative semirings. Now let us formally introduce the all-accepting property. We say that the wta $M = (Q, \Sigma, \mathcal{A}, F, \mu)$ is *all-accepting* [8] if $F(q) = 1$ for all $q \in Q$. We abbreviate *all-accepting wta* simply to *aa-wta*.

Let M be a deterministic aa-wta M. The tree series $S(M)$ recognised by M is *subtree-closed* [8, Section 3.1]; that is, for every tree t with $(S(M), t) \neq 0$ also $(S(M), u) \neq 0$ for every subtree u of t. We repeat [8, Observation 3.1] for ease of reference.

Proposition 8 (see [8, Observation 3.1]). *Let M be a deterministic aa-wta. Then $S(M)$ is subtree-closed.*

In order to avoid several cases, we assume that $0/0 = 0$ (i.e., we allow to cancel 0 from 0) for the rest of the paper. First we begin with two conditions which are necessary for a series $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$ to be recognizable by a deterministic aa-wta. The first condition checks whether the weight of a tree can be obtained from the weights of the subtrees and the second condition checks whether finitely many coefficients are sufficient. We say that ψ is *implementable* if

- $((\psi, t_1) \cdot \ldots \cdot (\psi, t_k)) | (\psi, \sigma(t_1, \ldots, t_k))$ for every $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $t_1, \ldots, t_k \in T_\Sigma$; and
- for every $k \in \mathbb{N}$ the following set $C_k(\psi)$ is finite.

$$C_k(\psi) = \left\{ \frac{(\psi, \sigma(t_1, \ldots, t_k))}{(\psi, t_1) \cdot \ldots \cdot (\psi, t_k)} \;\middle|\; \sigma \in \Sigma_k, t_1, \ldots, t_k \in T_\Sigma \right\}$$

Proposition 9. *Let M be a deterministic aa-wta over a cancellative semiring. Then $S(M)$ is implementable.*

Proof. The proof is standard and hence omitted. □

This shows that a series that is not implementable cannot be recognized by any deterministic aa-wta. In fact, this is the reason why the series ψ given at the beginning of Section 4 cannot be recognized by any deterministic aa-wta. Now we will show that the notion of recognizability by deterministic aa-wta over cancellative semirings is closely related to classical unweighted recognizability (as induced by fta). In fact, the weights of the deterministic aa-wta are uniquely determined so that they can also be included in the input ranked alphabet.

Definition 10. *Let $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$ be implementable over the cancellative semiring $\mathcal{A}$. We define the ranked alphabet Δ by $\Delta_k = \Sigma_k \times C_k(\psi)$ for every $k \in \mathbb{N}$. Moreover, let $\cdot|_1 \colon T_\Delta \to T_\Sigma$ be the mapping that replaces every node label of the form $\langle \sigma, c \rangle$ in the input tree simply by a node with label σ. Finally, we define the* *tree language $L(\psi) \subseteq T_\Delta$ as the smallest language L such that for every $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $u_1, \ldots, u_k \in L$ with $t_i = u_i|_1$ for every $i \in [1,k]$*

$$\left\langle \sigma, \frac{(\psi, \sigma(t_1, \ldots, t_k))}{(\psi, t_1) \cdot \ldots \cdot (\psi, t_k)} \right\rangle (u_1, \ldots, u_k) \in L \qquad \Longleftrightarrow \qquad \sigma(t_1, \ldots, t_k) \in \operatorname{supp}(\psi) \ .$$

Theorem 11. *Let $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$ with $\mathcal{A}$ a cancellative semiring. Then ψ is recognizable by some deterministic aa-wta if and only if ψ is implementable and $L(\psi)$ is recognizable.*

Proof. Let $M = (Q, \Sigma, \mathcal{A}, F, \mu)$ be a deterministic aa-wta and $M' = (Q, \Delta, \delta, Q)$ be a deterministic fta. We call M and M' *related* if

$$\mu_k(\sigma)_{q_1 \cdots q_k, q} = c \qquad \Longleftrightarrow \qquad \Big(\langle \sigma, c \rangle (q_1, \ldots, q_k) \to q \Big) \in \delta$$
$$\mu_k(\sigma)_{q_1 \cdots q_k, q} = 0 \qquad \Longleftrightarrow \qquad \forall c \in A \colon \Big(\langle \sigma, c \rangle (q_1, \ldots, q_k) \to q \Big) \notin \delta$$

for every $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, $c \in C_k(\psi) \setminus \{0\}$, and $q, q_1, \ldots, q_k \in Q$.

Now suppose that M and M' are related. We claim that $L(M') = L(S(M))$; the proof of this statement is omitted. Finally, let us now turn to the main statement. First let us suppose that there exists a deterministic aa-wta $M = (Q, \Sigma, \mathcal{A}, F, \mu)$ such that $S(M) = \psi$. By Proposition 9 it follows that ψ is implementable. Clearly, we can construct a deterministic fta M' such that M and M' are related. By the claimed property, we then have $L(M') = L(S(M)) = L(\psi)$, which proves that $L(\psi)$ is recognizable.

For the remaining direction, let ψ be implementable and, without loss of generality, let $M' = (Q, \Delta, \delta, F')$ be a deterministic fta such that $L(M') = L(\psi)$ and every state is reachable and co-reachable. It follows from the implementability condition that ψ is subtree-closed. With this in mind, we necessarily have $F' = Q$ because any reachable state in $Q \setminus F'$ would not be co-reachable. Moreover, for every $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $q, q_1, \ldots, q_k \in Q$ there exists at most one $c \in C$ such that $\langle \sigma, c \rangle (q_1, \ldots, q_k) \to q \in \delta$ because every state is final and $L(M') = L(\psi)$. Finally, no tree in $L(\psi)$ can contain a node $\langle \sigma, 0 \rangle$ for some $\sigma \in \Sigma$. This is due to the fact that $0 = (\psi, \sigma(t_1, \ldots, t_k))/((\psi, t_1) \cdot \ldots \cdot (\psi, t_k))$ only if $(\psi, \sigma(t_1, \ldots, t_k)) = 0$ by zero-divisor freeness of $\mathcal{A}$ and subtree-closedness of ψ. Thus, any reachable state that can recognize a tree of which one node is $\langle \sigma, 0 \rangle$ is not co-reachable. Consequently, there exists no such state and hence no transition which processes $\langle \sigma, 0 \rangle$. For the given fta M we can easily construct a related deterministic aa-wta and the previously proved statements guarantee that $L(S(M)) = L(M') = L(\psi)$. One final observation yields that $L \colon \mathcal{A}\langle\langle T_\Sigma \rangle\rangle \to T_\Delta$ is injective. Thus $L(S(M)) = L(\psi)$ yields that $S(M) = \psi$. □

The theorem admits a very important corollary. Namely, it can be observed that every minimal deterministic aa-wta M recognizing a given tree series ψ yields a minimal deterministic fta recognizing $L(\psi)$. In the opposite direction, every minimal deterministic fta recognizing $L(\psi)$ where ψ is implementable, yields a minimal deterministic aa-wta recognizing ψ.

Corollary 12 (of Theorem 11). *Let $\mathcal{A}$ be a cancellative semiring. For every deterministic aa-wta M there exists a unique (up to isomorphism) minimal deterministic aa-wta recognizing $S(M)$.*

Let us shortly describe a minimization procedure. Let M be a deterministic aa-wta. Then $S(M)$ is implementable and by the proof of Theorem 11 we can obtain a deterministic fta N recognizing $L(S(M))$. Then we minimize N to obtain the unique minimal deterministic fta N' recognizing $L(S(M))$. Finally, we can construct a deterministic aa-wta M' recognizing $S(M)$ again using the notion of relatedness from the proof of Theorem 11.

We can imagine that the established relation between aa-wta and fta can also be exploited in the learning task of [8]. There the underlying semiring is a semifield (and hence cancellative).

4.2 A Myhill-Nerode Congruence for Cancellative Semirings

In this section we consider general deterministic wta over certain cancellative semirings. The main problem is the implementability condition; it is crucial to the condition given in the previous section that the series is subtree-closed. In the general setting, subtree-closedness cannot be assumed.

A more careful analysis shows that the implementation of $\equiv_\psi$ [3,7] uses inverses in an essential manner. Here we present a more refined version of the Myhill-Nerode congruence. Let $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$. Let $\cong_\psi \subseteq T_\Sigma \times T_\Sigma$ be defined for every $t, u \in T_\Sigma$ by $t \cong_\psi u$ if and only if there exist $a, b \in A \setminus \{0\}$ such that for every $C \in C_\Sigma$ there exists a $d \in A$ with

$$(\psi, C[t]) = d \cdot a \quad \text{and} \quad (\psi, C[u]) = d \cdot b \enspace .$$

This relation has several drawbacks as we will see next (it is, in general, no equivalence relation), however, we can already see that $\equiv_\psi$ is coarser than $\cong_\psi$.

Lemma 13. *Let $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$. In general, $\equiv_\psi$ is coarser than $\cong_\psi$. If $\mathcal{A}$ is a semifield, then $\equiv_\psi$ and $\cong_\psi$ coincide.*

Proof. Let $\mathcal{A} = (A, +, \cdot, 0, 1)$. Moreover, let $t, u \in T_\Sigma$ such that $t \cong_\psi u$. Thus there exist $a, b \in A \setminus \{0\}$ such that for every context $C \in C_\Sigma$ there exists a $d \in A$ with

$$(\psi, C[t]) = d \cdot a \quad \text{and} \quad (\psi, C[u]) = d \cdot b \enspace .$$

Thus we also have that $b \cdot (\psi, C[t]) = a \cdot (\psi, C[u])$, and consequently, $t \equiv_\psi u$. For the second statement, suppose that $\mathcal{A}$ is a semifield and $t \equiv_\psi u$. Thus there exist $a, b \in A \setminus \{0\}$ such that for every $C \in C_\Sigma$

$$a \cdot (\psi, C[t]) = b \cdot (\psi, C[u]) \enspace .$$

Hence $(\psi, C[t]) = (\psi, C[u]) \cdot (a^{-1} \cdot b)$ and $(\psi, C[u]) = (\psi, C[t]) \cdot (b^{-1} \cdot a)$. Clearly, $a^{-1} \cdot b$ and $b^{-1} \cdot a$ are both nonzero, and consequently, $t \cong_\psi u$. $\square$

Now let us investigate when $\cong_\psi$ is actually a congruence. A similar analysis was already done in [2] for weighted automata over strings. However, we slightly adapted the notions of greedy factorization and minimal residue (cf. [2, Section 4]).

Lemma 14. *The relation $\cong_\psi$ is reflexive for every $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$.*

Proof. Let $t \in T_\Sigma$. We need to prove that there exists an $a \in A \setminus \{0\}$ such that for every $C \in C_\Sigma$ there exists $d \in A$ with $(\psi, C[t]) = d \cdot a$. To this end, we let $a = 1$ and $d = (\psi, C[t])$. $\square$

Clearly, $\cong_\psi$ is symmetric, so it remains to investigate transitivity. For this, we need an additional property. The semiring $\mathcal{A}$ allows *greedy factorization* if for every $a, b \in A$ there exist $a', b' \in A$ such that for every $c, d \in A$ there exists an $e \in A$ such that $a \cdot c = b \cdot d \neq 0$ implies $c = a' \cdot e$ and $d = b' \cdot e$. A similar property was already defined in [2].

Intuitively, the property demands that when a and b are divisors of a common element h, then there should be elements a' and b', that depend only on a and b and not on h, such that when cancelling a and a' from h we obtain the same element as we would obtain by cancelling b and b'. In this sense it represents a confluency property. It does not matter whether we first cancel a or b; we can later find elements a' and b', which depend solely on the cancelled elements a and b, that we can cancel to obtain a common element.

In semifields the property is trivially fulfilled because if we set $a' = b$ and $b' = a$ and $e = c \cdot b^{-1}$ then $a \cdot c = b \cdot d \neq 0$ implies $c = b \cdot c \cdot b^{-1}$ and $d = a \cdot c \cdot b^{-1}$. The first part of the conclusion is trivial and the second part is given by the hypothesis.

Let us try to give another example in order to explain the property. Suppose that $\mathcal{A}$ is a cancellative semiring with the additional property that a *least common multiple* (lcm) is defined for every two elements (e.g., the semiring of natural numbers fulfils these restrictions). We can then set $a' = \operatorname{lcm}(a, b)/a$ and $b' = \operatorname{lcm}(a, b)/b$ and $e = (a \cdot c)/\operatorname{lcm}(a, b)$ provided that $a \cdot c = b \cdot d$ otherwise set $e = 1$. Since the semiring is cancellative and $a | \operatorname{lcm}(a, b)$ and $b | \operatorname{lcm}(a, b)$ and $\operatorname{lcm}(a, b) | a \cdot c$ (because $a | a \cdot c$ and $b | a \cdot c$), the elements a', b', and e are uniquely determined. We thus obtain that $a \cdot c = b \cdot d \neq 0$ implies that $c = (\operatorname{lcm}(a, b)/a) \cdot ((a \cdot c)/\operatorname{lcm}(a, b))$ and $d = (\operatorname{lcm}(a, b)/b) \cdot ((a \cdot c)/\operatorname{lcm}(a, b))$. The first part of the conclusion is again trivial and the second part yields $b \cdot d = a \cdot c$, which holds by the hypothesis.

Lemma 15. *Let $\mathcal{A}$ be a zero-divisor free semiring that allows greedy factorization. Then $\cong_\psi$ is transitive for every $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$.*

Thus we successfully showed that $\cong_\psi$ is an equivalence relation. The only remaining step is to show that $\cong_\psi$ is even a congruence. Fortunately, this is rather easy.

Lemma 16. *Let $\mathcal{A}$ be a zero-divisor free semiring that allows greedy factorization. Then $\cong_\psi$ is a congruence for every $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$.*

Now let us proceed with the implementation of the congruence by some deterministic and complete wta. We prepare this by presenting conditions that imply that we can successfully implement a congruence. We chose to rephrase Conditions (MN1) and (MN2) from [7] in order to improve readability. In essence, we can already see the automaton in that modified definition of Conditions (MN1) and (MN2).

Definition 17. *Let $\cong \subseteq T_\Sigma \times T_\Sigma$ be a congruence and $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$. We say that $\cong$* respects *ψ if there exists a mapping $F\colon (T_\Sigma/\cong) \to A$ and a mapping $c\colon T_\Sigma \to A \setminus \{0\}$ such that*

- *$(\psi, t) = F([t]) \cdot c(t)$ for every $t \in T_\Sigma$; and*
- *for every $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $T_1, \ldots, T_k \in (T_\Sigma/\cong)$ there exists an $a \in A$, denoted by $b_\sigma(T_1, \ldots, T_k)$, such that*

$$c(\sigma(t_1, \ldots, t_k)) = a \cdot c(t_1) \cdot \ldots \cdot c(t_k)$$

for every $t_i \in T_i$ with $i \in [1, k]$.

Next we state that every series $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$ that is respected by some congruence with finite index can be recognized by a deterministic wta. Thus, the previous definition establishes sufficient conditions so that the congruence is implementable.

Lemma 18. *Let $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$. Moreover, let $\cong$ be a congruence with finite index that respects ψ. Then $\psi \in \mathcal{A}^{\mathrm{rec}}_{\mathrm{det}}\langle\langle T_\Sigma \rangle\rangle$.*

Proof. Since $\cong$ respects ψ, there exist $F\colon (T_\Sigma/\cong) \to A$, $c\colon T_\Sigma \to A \setminus \{0\}$, and $b_\sigma\colon (T_\Sigma/\cong)^k \to A$ for every $k \in \mathbb{N}$ and $\sigma \in \Sigma_k$ such that the conditions of Definition 17 hold. We construct the wta $M_\cong = ((T_\Sigma/\cong), \Sigma, \mathcal{A}, F, \mu)$ where

$$\mu_k(\sigma)_{[t_1]\cdots[t_k],[\sigma(t_1,\ldots,t_k)]} = b_\sigma([t_1], \ldots, [t_k])$$

for every $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $t_1, \ldots, t_k \in T_\Sigma$ and the remaining entries in μ are 0. Clearly, $M_\cong$ is deterministic. The proof of $S(M) = \psi$ is straightforward. □

In fact, the "respects" property is necessary and sufficient, which can be seen in the next theorem.

Theorem 19. *Let $\mathcal{A}$ be a zero-divisor free semiring, and let $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$. The following are equivalent:*

1. *There exists a congruence relation with finite index that respects ψ.*
2. *ψ is deterministically recognizable.*

Proof. The implication 1 to 2 is proved in Lemma 18. It remains to show that 2 implies 1. Let $M = (Q, \Sigma, \mathcal{A}, F, \mu)$ be a deterministic and complete wta such that $S(M) = \psi$. Clearly, $\equiv_M$ is a congruence with finite index by Lemma 3.

Finally, it remains to show that $\equiv_M$ respects ψ. Let $G\colon (T_\Sigma/\cong) \to A$ and $c\colon T_\Sigma \to A \setminus \{0\}$ be defined by $G([t]) = F_{R_M(t)}$ and $c(t) = h_\mu(t)_{R_M(t)}$ for every

$t \in T_\Sigma$. It is easily verified that both mappings are well-defined. First we need to prove that $(\psi, t) = G([t]) \cdot c(t)$ for every $t \in T_\Sigma$.

$$G([t]) \cdot c(t) = F_{R_M(t)} \cdot h_\mu(t)_{R_M(t)} = (S(M), t) = (\psi, t)$$

because M is deterministic. We observe that for every $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $t_1, \ldots, t_k \in T_\Sigma$

$$\begin{aligned} & c(\sigma(t_1, \ldots, t_k)) \\ &= h_\mu(\sigma(t_1, \ldots, t_k))_{R_M(\sigma(t_1,\ldots,t_k))} \\ &= \mu_k(\sigma)_{R_M(t_1)\cdots R_M(t_k), R_M(\sigma(t_1,\ldots,t_k))} \cdot h_\mu(t_1)_{R_M(t_1)} \cdot \ldots \cdot h_\mu(t_k)_{R_M(t_k)} \\ &= \mu_k(\sigma)_{R_M(t_1)\cdots R_M(t_k), R_M(\sigma(t_1,\ldots,t_k))} \cdot c(t_1) \cdot \ldots \cdot c(t_k) \end{aligned}$$

which proves that $\equiv_M$ respects ψ. □

In analogy to Theorem 4 we can show that $\cong_{S(M)}$ is coarser than $\equiv_M$ for every deterministic and complete wta over a zero-divisor free semiring. Thus, the only remaining question is whether $\cong_\psi$ respects ψ. If this would be true and $\cong_\psi$ would have finite index, then $\cong_\psi$ would be implementable and thus a minimal deterministic and complete wta would be found.

Open problem: Find suitable conditions on ψ and $\mathcal{A}$ so that $\cong_\psi$ respects ψ!

4.3 A Myhill-Nerode Theorem for All-Accepting wta

In this section we show how we can use the approach of the previous section to derive a Myhill-Nerode theorem for deterministic aa-wta.

Let $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$ be a tree series over the cancellative semiring $\mathcal{A}$. We define $\simeq_\psi \subseteq T_\Sigma \times T_\Sigma$ by $t \simeq_\psi u$ if and only if there exist $a, b \in A \setminus \{0\}$ such that for every $C \in C_\Sigma$ there exists a $d \in A$ with

$$(\psi, C[t]) = d \cdot a \quad \text{and} \quad (\psi, C[u]) = d \cdot b \quad \text{and} \quad d \in \{0, 1\} \text{ if } C = \square \ .$$

Lemma 20. *If ψ is implementable and $\mathcal{A}$ allows greedy factorization, then $\simeq_\psi$ is a congruence.*

Proof. The proof can be obtained by reconsidering the proofs of Lemmata 14, 15, and 16. □

Let us consider the open problem for deterministic aa-wta over cancellative semirings.

Theorem 21. *Let $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$ be implementable with $\mathcal{A}$ a cancellative semiring that allows greedy factorization. Then $\simeq_\psi$ respects ψ.*

Proof. By Lemma 20, $\simeq_\psi$ is a congruence. Thus we need to show that there exist mappings $F\colon (T_\Sigma/\simeq_\psi) \to A$ and $c\colon T_\Sigma \to A \setminus \{0\}$ such that the conditions of Definition 17 are met. For every $t \in T_\Sigma$ let

$$F([t]) = \begin{cases} 1 & \text{if } t \in \operatorname{supp}(\psi) \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad c(t) = \begin{cases} (\psi, t) & \text{if } t \in \operatorname{supp}(\psi) \\ 1 & \text{otherwise.} \end{cases}$$

We first verify that F is well-defined. Let $t \simeq_\psi u$. We need to prove that $t \in \operatorname{supp}(\psi)$ if and only if $u \in \operatorname{supp}(\psi)$. Since $t \simeq_\psi u$ there exist $a, b \in A \setminus \{0\}$ such that for every context $C \in C_\Sigma$ there exists $d \in A$ with

$$(\psi, C[t]) = d \cdot a \quad \text{and} \quad (\psi, C[u]) = d \cdot b \quad \text{and} \quad d \in \{0,1\} \text{ if } C = \square \ .$$

Now consider the context $C = \square$. Thus $(\psi, t) = d \cdot a$ and $(\psi, u) = d \cdot b$ with $d \in \{0,1\}$. Depending on d either (i) $(\psi, t) = 0 = (\psi, u)$ or (ii) $t, u \in \operatorname{supp}(\psi)$, which proves that F is well-defined. It remains to verify the properties of Definition [17]. First, for every $t \in T_\Sigma$

$$F([t]) \cdot c(t) = \begin{cases} 1 \cdot (\psi, t) & \text{if } t \in \operatorname{supp}(\psi) \\ 0 & \text{otherwise} \end{cases} = (\psi, t) \ .$$

Second, let $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $t_1, \ldots, t_k \in T_\Sigma$. We need to show that there exists a $b_\sigma([t_1], \ldots, [t_k])$ such that $c(\sigma(t_1, \ldots, t_k)) = b_\sigma([t_1], \ldots, [t_k]) \cdot c(t_1) \cdot \ldots \cdot c(t_k)$. Since ψ is implementable, we can define $b_\sigma([t_1], \ldots, [t_k]) = (\psi, \sigma(t_1, \ldots, t_k))/((\psi, t_1) \cdot \ldots \cdot (\psi, t_k))$. We should first verify that this is independent of the representatives. Thus, let $u_1, \ldots, u_k \in T_\Sigma$ be such that $t_i \simeq_\psi u_i$ for every $i \in [1, k]$. Then there exist $a_i, b_i \in A \setminus \{0\}$ such that for every context $C \in C_\Sigma$ there exists $d_i \in A$ with

$$(\psi, C[t_i]) = d_i \cdot a_i \quad \text{and} \quad (\psi, C[u_i]) = d_i \cdot b_i \quad \text{and} \quad d_i \in \{0,1\} \text{ if } C = \square$$

for every $i \in [1, k]$. Now if $(\psi, t_i) = 0$ then also $(\psi, C[t]) = 0$ because ψ is implementable. The same argument holds for u_i and $C[u_i]$. Suppose that there exist $i \in [1, k]$ such that $(\psi, t_i) = 0$. Then $(\psi, \sigma(t_1, \ldots, t_k))/((\psi, t_1) \cdot \ldots \cdot (\psi, t_k)) = 0$ and since $(\psi, u_i) = 0$ by $t_i \simeq_\psi u_i$ also $(\psi, \sigma(u_1, \ldots, u_k))/((\psi, u_1) \cdot \ldots \cdot (\psi, u_k)) = 0$. Now suppose that $(\psi, t_i) \neq 0$ for every $i \in [1, k]$. It is immediately clear that $a_i = (\psi, t_i)$ and $b_i = (\psi, u_i)$ by considering the context $\square$. Consequently,

$$\begin{aligned}
&(\psi, \sigma(t_1, \ldots, t_k))/\Big(\prod_{i=1}^{k} (\psi, t_i)\Big) \\
&= (\psi, \sigma(u_1, t_2, \ldots, t_k))/\Big((\psi, u_1) \cdot \prod_{i=2}^{k} (\psi, t_i)\Big) && \text{(via the context } \sigma(\square, t_2, \ldots, t_k)) \\
&= \ldots \\
&= (\psi, \sigma(u_1, \ldots, u_{k-1}, t_k))/\Big(\prod_{i=1}^{k-1} (\psi, u_i) \cdot (\psi, t_k)\Big) \\
&= (\psi, \sigma(u_1, \ldots, u_k))/\Big(\prod_{i=1}^{k} (\psi, u_i)\Big) && \text{(via the context } \sigma(u_1, \ldots, u_{k-1}, \square))
\end{aligned}$$

□

Let us now derive a MYHILL-NERODE theorem for deterministic aa-wta. In [8] such a theorem is shown for the case that the underlying semiring is a semifield. We extend this result to certain cancellative semirings.

Corollary 22. *Let $\psi \in \mathcal{A}\langle\langle T_\Sigma \rangle\rangle$ be implementable with $\mathcal{A}$ a cancellative semiring that allows greedy factorization. The following are equivalent:*

1. *$\simeq_\psi$ has finite index.*
2. *There exists a congruence with finite index that respects ψ.*
3. *ψ is deterministically recognizable.*
4. *ψ is recognized by some deterministic aa-wta M.*

Proof. $1 \to 2$ was shown in Theorem 21. The equivalence of 2 and 3 is due to Theorem 19. Moreover, we already remarked that $\simeq_\psi$ is coarser than $\equiv_M$, which shows $4 \to 1$. It remains to show $3 \to 4$. This can be shown by a straightforward construction that normalizes the final weights to 1. In general, this is only possible in a semifield, but due to the implementability of ψ, it can also be performed in the cancellative semiring $\mathcal{A}$. □

Clearly, the above corollary shows that the tree series that can be recognized by deterministic aa-wta are exactly the implementable tree series that can be recognized by deterministic wta. Moreover, it can be shown that the deterministic aa-wta that can be constructed from the deterministic wta using the final weight normalization mentioned in the proof of Corollary 22 is indeed the minimal deterministic aa-wta recognizing ψ.

References

1. Kozen, D.: On the Myhill-Nerode theorem for trees. Bulletin of the EATCS 47, 170–173 (1992)
2. Eisner, J.: Simpler and more general minimization for weighted finite-state automata. In: HLT-NAACL (2003)
3. Borchardt, B.: The Myhill-Nerode theorem for recognizable tree series. In: Ésik, Z., Fülöp, Z. (eds.) DLT 2003. LNCS, vol. 2710, pp. 146–158. Springer, Heidelberg (2003)
4. Hebisch, U., Weinert, H.J.: Semirings—Algebraic Theory and Applications in Computer Science. World Scientific, Singapore (1998)
5. Golan, J.S.: Semirings and their Applications. Kluwer Academic, Dordrecht (1999)
6. Knight, K., Graehl, J.: An overview of probabilistic tree transducers for natural language processing. In: Gelbukh, A. (ed.) CICLing 2005. LNCS, vol. 3406, Springer, Heidelberg (2005)
7. Borchardt, B.: The Theory of Recognizable Tree Series. PhD thesis, Technische Universität Dresden (2005)
8. Drewes, F., Vogler, H.: Learning deterministically recognizable tree series. J. Autom. Lang. Combin (to appear, 2007)
9. Berstel, J., Reutenauer, C.: Rational Series and Their Languages. In: EATCS Monographs on Theoret. Comput. Sci., vol. 12, Springer, Heidelberg (1988)
10. Bozapalidis, S.: Equational elements in additive algebras. Theory Comput. Systems 32(1), 1–33 (1999)
11. Borchardt, B., Vogler, H.: Determinization of finite state weighted tree automata. J. Autom. Lang. Combin. 8(3), 417–463 (2003)
12. Gécseg, F., Steinby, M.: Tree Automata. Akadémiai Kiadó, Budapest (1984)
13. Gécseg, F., Steinby, M.: Tree languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, pp. 1–68. Springer, Heidelberg (1997)

The View Selection Problem for Regular Path Queries

Sergey Afonin*

Lomonosov Moscow State University, Russia
serg@msu.ru

Abstract. The view selection problem consists of finding a set of views to materialize that can answer the given set of workload queries and is optimal in some sense. In this paper we study the view selection problem for regular path queries over semistructured data and two specific view-based query rewriting formalisms, namely single-word and arbitrary regular rewritings. We present an algorithm that for a given finite set of workload queries, i.e. for a set of regular languages, computes a set of views that can answer every query in the workload and has minimal possible cardinality. If, in addition, a database instance is given then one can construct a viewset such that its size, i.e. amount of space required to store results, is minimal on the database instance.

Keywords: view selection problem, regular path queries, semigroups of regular languages.

1 Introduction

The problem of view based query processing plays an important role in many database applications, including information integration, query optimization, mobile computing and data warehousing. In its general form, the problem is stated as follows. Given a query over database schema and a set of materialized views over the same schema (i.e. a set of queries with precomputed answers – *view extensions*), is it possible to answer the query, completely or partially, using answers to the views? This question has been intensively studied for various data models and different assumptions on views semantic (e.g., [13,11,5,20]). The main approaches to view based query processing are *query rewriting* and *query answering* (see [13] for a survey). In query rewriting approach, given a query Q written in language $\mathcal{Q}$, and a set of views $\mathbf{V}$ that are written in language $\mathcal{V}$ one should construct a rewriting R, in language $\mathcal{R}$, such that $Q(D) = R \circ \mathbf{V}(D)$ for each database instance D. It is worth noticing that query rewriting does not depend on view extensions and can be thought as an algorithm that describes how result of the query can be computed form the views. In contrast, query answering consists of direct computing of all answers to Q from the view extensions. The

* This work has been supported in part by the Russian Federal Agency for Science and Innovations under grant 02.514.11.4078 and by the Russian Foundation for Basic Research under grant 08-01-00882-a.

connection between query rewriting and query answering studied in [20, 4]. In this paper we follow query rewriting approach and we shall say that a viewset $\mathbf{V}$ *answers* a query Q if there exists a rewriting of Q.

The ability to answer a query using *only* the answers to views can significantly decrease query evaluation time. For example, in a mobile computing environment the application does not need to access the network in order to process user's query. If the views can not completely answer a query, they can, nevertheless, speed up query processing, because some computations needed for the query processing may have been done while computing the views. From the other hand, view maintaining can be computationally expensive, and the amount of space that is available to store view results could be bounded. These constraints lead to the natural optimization problem, known as the *view selection problem*, which is, roughly speaking, asking for a set of views that can answer given queries and satisfies specified constraints [6,21,18]. A dual problem is the *view optimization* problem [17], which asks for a viewset that satisfies specified constraints and has the same expressive power as a given viewset (i.e., answer every query that is answered by a given viewset). Possible applications of these problem include view selection and optimization in data management systems, intelligent data placement, minimization of communication costs in distributed systems, optimization of bulk query processing and others (see e.g. [6,7,21,18,17]).

The view selection problem was studied in various settings for relational databases, see e.g. [6, 7], and for XPath queries [21, 18]. In this paper we consider the view selection problems for *regular path queries* over semistructured data. Informally, a database is an edge-labelled directed graph, and a query is a regular language over a finite alphabet Σ (without loss of generality we can assume that Σ is the set of all possible labels of a database graph). The result of a query Q is the set of all pairs (u, v) of graph vertices such that there exists at least one labeled path between u and v in the graph and its labels comprise a word in Q. Although the main topic of current research on semistructured data has shifted toward tree data models and corresponding query languages graph model is important in data integration domain. This model is quite flexible and lays between full text search and XML-like tree data models [9]. Regular path query evaluation has polynomial time complexity with respect to both database and query size (one should check non-emptiness of the intersection of some regular languages), but it can be expensive for large databases because the whole database graph may need to be searched, which is inefficient. Materialized views may decrease query processing time drastically.

For regular path query rewritings the so called *complete* [3] and *partial* [10] rewritings were proposed. In both cases a rewriting of a query Q over Σ with respect to a set of views $\mathbf{V}$ is a regular language R over a fresh alphabet Δ, such that a substitution of languages over Σ instead of corresponding letters of Δ yields the original language Q. The two approaches differ in possible mappings for letters of Δ. In complete rewriting Δ letters are mapped into elements of $\mathbf{V}$ only, while in partial rewriting a Δ-letter can be mapped into a Σ-letter as well. For example, $R_1 = \delta_1^*$ is a complete rewriting of $Q_1 = a^*$, and $R_2 = \delta_1 cb^* + \delta_1 c\delta_2$

is a partial rewriting of $Q_2 = a^*cb^*$ with respect to the views $\mathbf{V} = \{a^*, b^*\}$ and the natural substitution $\delta_1 \mapsto a^*$, $\delta_2 \mapsto b^*$. In general, both rewriting techniques may be considered as special cases of the *language substitution problem* which is stated as follows. Given a regular language Q over a finite alphabet Σ and a regular language substitution φ from finite set Δ into regular languages over Σ one should find a language R over Δ, such that $\varphi(R) = Q$. Additional constraints on the structure of language R may be posed. It is known [14] that for any finite set $\mathbf{V}$ of regular languages over Σ, and the subset $T \subseteq \{\cdot, \cup, ^*\}$ of language operations (concatenation, union, and iteration) it is decidable whether or not the regular language Q may be constructed from elements of $\mathbf{V}$ using a finite number of operations from T. Note, that if all three operations are allowed then we have complete rewriting problem mentioned above. The algorithm for partial rewriting construction [10] computes the exhaustive rewriting that, roughly speaking, uses available views as much as possible. In the above example this algorithm gets the rewriting $R_2' = \delta_1 c \delta_2$ but not $R_2 = \delta_1 cb^* + \delta_1 c \delta_2$.

In this paper we deal with two types of complete rewritings: a *single word rewriting* (the subset T contains only concatenation), and an arbitrary *regular rewriting*. Clearly, that if there exists a single word rewriting of a query, then there exists an arbitrary rewriting of this query with respect to the same set of views. Thus, single word rewritings have weaker expressive power then arbitrary rewritings. Practical meaning of single word rewritings can be justified by some redundancy presented in arbitrary regular rewritings. Known algorithms for regular rewritings of a query Q [3, 14] compute the so-called *maximal* rewriting, which is the set of all words over Δ, such that the substitution of corresponding languages yields a subset of Q. For instance, in the previously considered example the maximal rewriting is $R = \delta_1^*$, while more efficient single word rewriting $R' = \delta_1$ exists. In this case the single word rewriting R' shows that the query can be answered by the views without any computations at all. In order to compute the answer using maximal rewriting R the transitive closure of the view should be constructed. Another attractive property of single word rewritings is that they does not contain recursion (i.e. Kleene star) and admit efficient processing algorithms. Finally, the set of all single word rewritings is an effectively constructable regular language [2].

For now, let us consider the problems under investigation. Assume that a database *workload*, i.e. the set $\mathbf{Q}$ of the most popular queries, is known. What views should be materialized in the database in order to speed up these queries? Trivial solution when every query from the workload correspond to a view is not practical because view maintaining is computationally expensive. In this respect, view selection should be based not only on the ability to answer queries from the workload but on some "efficiency" measure as well. Possible measures are *cost of workload queries evaluation*, *storage constraints*, *cardinality of a viewset*, and *efficiency of rewriting*. We consider the following specific problems (for both single word and regular rewritings).

– *Viewset of minimal cardinality.* What is the minimal number of views that should be materialized in order to answer every query from the workload?

- *Instance based minimal size viewset.* If a particular database instance is given, what is the minimal amount of space required to store the results of a viewset (over this instance) that answer every query from the workload?
- *Oracle based minimal size viewset.* Assume that there is an oracle that for every regular path query estimates the size of the result. What is the minimal size of a viewset that answers queries from the workload?
- *p-Containment of viewsets.* Given two viewsets $\mathbf{V}_1$ and $\mathbf{V}_2$ is it decidable whether or not every query Q that can be rewritten in terms of $\mathbf{V}_1$ can be rewritten in terms of $\mathbf{V}_2$?

It is worth noticing that in some cases (e.g., in data integration systems based on local-as-view approach) there exist a set of views that can not be changed (the views that describe information sources). Nevertheless, database applications are allowed to define "top-level" views that are subject to optimization. Our contribution is the following.

- First, we prove that the search space for minimal cardinality and instance based minimal size viewset construction problems is finite, both for single word and regular rewritings. Thus, both problems are decidable and we propose the algorithms for such viewsets construction.
- We prove that p-containment of viewsets of regular path queries is decidable for single word and regular rewritings.
- We present an algorithm which for a given workload and a database instance computes a viewset of minimal size (on this particular database instance).

The results on single-word rewritings may be considered as a contribution to formal language theory. In particular, we prove that the rank problem for finitely generated semigroups of regular languages is decidable.

The structure of this paper is the following. In Section 2 we introduce basic definition and known results on regular path query rewritings. Algorithms for minimal cardinality viewset construction, for both single word and regular rewritings, are presented in Section 3. This section also contains the the algorithm for checking p-containment of viewsets, for both types of rewritings. Minimal size viewset problem is discussed in Section 4.

2 Regular Path Query Rewritings

In order to fix the notation we start this section with basic notions of formal languages.

An *alphabet* is a finite non-empty set of *symbols*. A finite sequence of symbols from an alphabet Σ is called a *word* in Σ. The *empty* word is denoted by ε. Any set of words is called a *language* over Σ. Σ^* denotes the set of all words (including the empty word) in a given alphabet, Σ^+ denotes the set of all non-empty words in Σ, $\varnothing$ is the empty language (containing no words), and 2^{Σ^*} is the set of all languages over Σ.

Let u be a word in Σ and $A \subseteq \Sigma^*$. The right quotient of A with respect to u is the language $u^{-1}A = \{v \in \Sigma^* \mid uv \in A\}$, and the left quotient is $Au^{-1} = \{v \in \Sigma^* \mid vu \in A\}$. If $A = \{w\}$ we shall write $u^{-1}w$ instead of $u^{-1}\{w\}$.

The *union* of languages L_1 and L_2 is the language $L_1 \cup L_2 = \{w \in \Sigma^* \mid w \in L_1 \vee w \in L_2\}$. The language $L_1L_2 = \{w \in \Sigma^* \mid \exists w_1 \in L_1, w_2 \in L_2 : w = w_1w_2\}$ is called a *concatenation* of L_1 and L_2. $L^k = LL^{k-1}$ is L to the power k. By definition, L to the power zero is the empty word: $L^0 = \{\varepsilon\}$. The Kleene closure (or star) of L is the language $L^* = \cup_{k=0}^{\infty} L^k$. A language is *regular* if it can be obtained from singleton languages (i.e., the letters of the alphabet), the empty language, and $\{\varepsilon\}$, using a finite number of operations of concatenation, union and closure. The set of all regular languages over an alphabet Σ is denoted by $\mathrm{Reg}(\Sigma)$. In the sequel all the languages are assumed regular.

Let Δ be an alphabet and S be a semigroup. A morphism $\varphi : \Delta^+ \to S$ is any function satisfying $\varphi(uv) = \varphi(u)\varphi(v)$ for all $u, v \in \Delta^+$. A morphism $\varphi : \Delta^+ \to 2^{\Sigma^*}$ is called a *regular language substitution* if $\varphi(\delta)$ is a regular language over Σ for all $\delta \in \Delta$. Every regular language substitution $\varphi : \Delta^+ \to \mathrm{Reg}(\Sigma)$ generates the semigroup $\mathcal{S}_\varphi = \langle\{\varphi(\delta) \mid \delta \in \Delta)\}\rangle$. Conversely, with every semigroup $(\mathcal{S}, \cdot) = \langle V_1, \ldots, V_n\rangle$ of regular languages we can associate a language substitution φ defined by the rule $\delta_i \mapsto V_i$.

We turn now to regular path queries over semistructured data and query rewritings. A *semistructured database* is an ordered edge-labeled graph $\mathcal{B} = \langle O, E, \Sigma, \psi\rangle$, where O is a finite set of nodes (objects), Σ is a set of labels (a database alphabet), $E \subseteq O \times O$ is a set of edges, and $\psi : E \to \Sigma$ is the edge-labeling function. With every path in the database graph, i.e. a set of adjusting edges $(e_1, e_2, \ldots, e_m)$, we associate the word $w = a_1a_2 \ldots a_m$ in Σ, defined by the rule $a_i = \psi(e_i)$. Let us denote the regular language of all words associated with all paths between given pair of database nodes, say u and v, as $L_{(u,v)}(\mathcal{B})$. A *query* is a regular language over Σ. The *result* of a query Q on the database $\mathcal{B}$ is the set

$$Q(\mathcal{B}) = \{(u, v) \in O \times O \mid L_{(u,v)}(\mathcal{B}) \cap Q \neq \varnothing\}.$$

Let $\varphi : \Delta^+ \to \mathrm{Reg}(\Sigma)$ be a regular language substitution. The *maximal rewriting* of a regular language $R \subseteq \Sigma^*$ with respect to φ is the set

$$M_\varphi(R) = \{w \in \Delta^+ \mid \varphi(w) \subseteq R\}.$$

The maximal rewriting $M_\varphi(R)$ is called *exact*, if

$$\bigcup_{w \in M_\varphi(R)} \varphi(w) = R.$$

The following theorem is due to D.Calvanese et al. [3], and K.Hashiguchi [14].

Theorem 2.1. *Let $\varphi : \Delta^+ \to \mathrm{Reg}(\Sigma)$ be a regular language substitution. For any regular language $R \subseteq \Sigma^*$ the maximal rewriting $M_\varphi(R)$ is a regular language over Δ.*

As it was mentioned above, the maximal rewriting could be redundant. We say, that a regular language Q admits a *single word rewriting* with respect regular language substitution φ if there exists a word $w \in \Delta^+$, such that $Q = \varphi(w)$. The decidability of single word rewriting problem was proved by K.Hashiguchi [14] and the following theorem was proved in [2].

Theorem 2.2. *Let $\varphi : \Delta^+ \to \mathrm{Reg}(\Sigma)$ be a regular language substitution and w be a word in Δ^+. The membership problem for the semigroup $\mathcal{S}_\varphi$*

$$[w] = \{u \in \Delta^+ \mid \varphi(u) = \varphi(w)\}$$

is a regular language over Δ.

Theorem 2.2 implies that, given a finite set of views and a regular path query Q, one can effectively construct a finite automaton that recognize the set of all possible single word rewritings of Q wrt the viewset.

3 Viewsets of Minimal Cardinality and p-Containment

In this section we study the problem of finding a viewset of minimal cardinality and show decidability of p-containment between viewsets of regular path queries. We focus our attention on single word rewritings and then extend the result to arbitrary regular rewritings. For single word rewritings both problems may be reformulated in terms of finitely generated semigroups of regular languages. Indeed, given a finite set $\mathbf{V}$ of regular languages over Σ one can consider the semigroup $\mathcal{S} = \langle \mathbf{V}, \cdot \rangle$ with $\mathbf{V}$ as the set of generators and language concatenation as a semigroup product. The semigroup $\mathcal{S}$ consists exactly of languages that can be rewritten in terms of $\mathbf{V}$ using single word rewriting. Now, a viewset $\mathbf{V}_1$ is p-contained in a viewset $\mathbf{V}_2$ iff the semigroup $\langle \mathbf{V}_1, \cdot \rangle$ is a subsemigroup of $\langle \mathbf{V}_2, \cdot \rangle$, and minimal cardinality viewset problem is the following. Given a finite set $\mathbf{Q} = \{Q_1, \ldots, Q_n\}$ of regular languages over an alphabet Σ one should find a natural number k and a set $\mathbf{G} = \{G_1, \ldots, G_k\}$ of regular languages over Σ satisfying the following conditions:

- $Q_i \in \langle \mathbf{G} \rangle$ for every $i \in \{1, \ldots, n\}$, and
- for every $k' < k$ and every set $\mathbf{G}' = \{G'_1, \ldots, G'_{k'}\}$ there exists $Q \in \mathbf{Q}$ such that $Q \notin \langle \mathbf{G}' \rangle$.

The above conditions imply that $\langle \mathbf{Q} \rangle \subseteq \langle \mathbf{G} \rangle$. Note, that k is bounded by the number of elements in $\mathbf{Q}$ and set $\mathbf{G}$ always exists, although it can not be constructed by a "trivial" brute force algorithm.

The main idea of the algorithm proposed in this section is to construct factorizations of Q_i (with respect to concatenation) and choose minimal subset of factors that forms a basis of a semigroup. The difficulty related to this approach arise from the fact that a regular language has infinitely many factorizations, in general. For example, the language $L = a^*$ admits infinitely many factorizations of the form $L = FM$, e.g., every pair of languages F and M such that $F \cup M = a^*$ and $\varepsilon \in F \cap M$ forms a factorizations of L. In order to resolve this problem we show that there exist finite set of languages, that can be effectively constructed from $\mathbf{Q}$, and minimal basis consist of elements of this set.

3.1 Maximal Factors of Regular Languages

Let $L \subseteq \Sigma^*$ be a regular language. Consider a family of *language equations* of the form $L = X_1 \dots X_m$. A m-tuple $(L_1, \dots, L_m)$ of languages is a solution to the equation $L = X_1 \dots X_m$ if the equality $L = L_1 \dots L_m$ holds. A m-tuple $(L_1, \dots, L_m)$ is contained in a m-tuple $(L'_1, \dots, L'_m)$ if $L_i \subseteq L'_i$ for all $i = 1, \dots, m$. A solution $(L_1, \dots, L_m)$ is called *maximal* if it is not contained in any other solution to the equation. A language F is called a *maximal factor* of L if there exist a natural number m such that F is a component of some maximal solution to the equation $L = X_1 \dots X_m$. The set of all maximal factors of a language L is denoted by $\mathcal{F}(L)$.

Theorem 3.1 (J.Conway [8]). *A regular language L has finitely many maximal factors and all these factors are regular languages. Moreover, maximal factors are effectively constructable.*

The algorithm for viewset selection depends on maximal factors construction and we briefly describe how this can be done. First, consider the univariate language equation of the form $AY = L$ where $A, L \in \text{Reg}(\Sigma)$. A language $F \subseteq \Sigma^*$ is called a *solution* to the equation $AY = L$ if $AF = L$. Since the union of two solutions is a solution as well, the equation $AY = B$ has at most one maximal solution and this solution is the language $Y_{\max} = \bigcap_{u \in A} u^{-1}L$. Similarly, the language $X_{\max} = \bigcap_{u \in A} Lu^{-1}$ is the maximal solution to the equation $XA = B$. Since every regular language has finitely many different right (left) quotients the equation $XY = L$ has at most finitely many maximal solutions.

Let (P_i, Q_i) $i = 1, \dots, p$ be the maximal solutions to the equation $XY = L$. The set of maximal factors of L is exactly the union of $\{P_i\}$, $\{Q_i\}$, and the set of maximal solutions of equations $P_i X Q_j = L$. Maximal solutions to such equations are to intersection of some left and right quotients of L.

3.2 Algorithm

Let $\mathbf{A}$ be a set of languages over Σ. By *intersection closure* of $\mathbf{A}$, denoted as $\overline{\mathbf{A}}$, we mean the set of all languages that can be obtained by finite intersections of elements of $\mathbf{A}$:

$$\overline{\mathbf{A}} = \{I \subseteq \Sigma^* \mid \exists A_1, \dots, A_m \in \mathbf{A} \; I = A_1 \cap \dots \cap A_m\}.$$

Theorem 3.2. *Let $\mathbf{Q}$ be a finite set of regular languages and $\mathbf{F}$ be the set of all maximal factor of its elements, i.e.*

$$\mathbf{F} = \cup_{Q \in \mathbf{Q}} \mathcal{F}(Q).$$

There exists a minimal cardinality set $\mathbf{G}$ of generators for $\mathbf{Q}$ such that $\mathbf{G} \subseteq \overline{\mathbf{F}}$.

Proof. Let A and B be regular languages. We call a language F a *common factor* of A and B if $A = P_1 F P_2, \;\; B = Q_1 F Q_2$ for some languages P_1, P_2, Q_1, Q_2. We prove now that if there exits a common factor F for A and B when F can be extended to the intersection $F' = F_A \cap F_B$ of some maximal factors $F_A \in \mathcal{F}(A)$ and $F_B \in \mathcal{F}(B)$, that is $F \subseteq F'$ and F' is a common factor of A and B.

Indeed, languages (P_1, F, P_2) form a solution to the equation $A = X_1X_2X_3$. By Theorem 3.1 there exists a maximal solution, say (M_1, M_2, M_3), that contains the solution (P_1, F, P_2). It is follows that every tuple (P_1, L, P_2) is a solution to the equation if L satisfy the inequalities $F \subseteq L \subseteq M_2$. Similarly, the solution (Q_1, F, Q_2) is contained in some maximal solution to the equation $B = X_1X_2X_3$, and the intersection of corresponding components of these maximal solutions is the desired language F'. Clearly, that if F is a common factor of languages $A_1, \dots, A_p$, then there exists a common factor F' that is the intersection of some maximal factors of the languages $A_1, \dots, A_p$.

Now assume that $\mathbf{G} = \{G_1, \dots, G_k\}$ is a minimal cardinality set of generators and this set is not contained in $\overline{\mathbf{F}}$. Without loss of generality assume that $F_1 \notin \overline{\mathbf{F}}$. Every language $Q \in \mathbf{Q}$ has a factorization of the form

$$Q = G_{i_1} \dots G_{i_m},$$

and G_1 is a common factor of some languages from $\mathbf{Q}$. Thus, there exists $G_1' \in \overline{\mathbf{F}}$ such that the set $\mathbf{G}^1 = \{G_1', G_2 \dots, G_k\}$ is the set of generators for $\mathbf{Q}$. Repeatedly applying this procedure to all elements of $\mathbf{G}$ one can construct the set of generators that has the same cardinality as $\mathbf{G}$. □

In the sequel we shall call a viewset that is a subset of $\overline{\mathbf{F}}$ *the maximal viewset.* It is worth noting that in the proof we did not assume that the "initial" minimal cardinality set consists of regular languages, thus if we drop the restriction that minimal set should contains only regular languages we will not decrease the number of generators.

As an immediate corollary from Theorem 3.2 and the finiteness of the set of maximal factors of a regular language we have

Theorem 3.3. *Minimal cardinality viewset problem is decidable for single word rewriting.*

Proof. Given a finite set $\mathbf{Q} = \{Q_1, \dots, Q_n\}$ the algorithm first computes the intersection closure $\overline{\mathbf{F}}$ of the set $\mathcal{F}(\mathbf{Q})$ of all maximal factors, and then, for every subset $\mathbf{G}$ of $\overline{\mathbf{F}}$, checks whether or not $\mathbf{G}$ is the set of generators for $\mathbf{Q}$. □

The algorithm proposed by this theorem relies on maximal factor construction and the algorithm that checks that a language is representable as concatenation of given regular languages. The number of maximal factors of a regular language L is at most 2^{2N},where N is the number of states of the minimal deterministic automaton for L. The best known bound for the number of intersection of maximal factors is 2^{N^2} [19]. Thus, the size of set $\overline{\mathbf{F}}$ is double-exponential in the total size of minimal automata for languages in $\mathbf{Q}$. Known algorithms for single word rewritings [1,14] are based on the *limitedness property of distance* *automata* which is PSPACE-complete [16,15] and for which only exponential time complexity algorithms are known. The number of states of a distance automaton to be checked is no more then 2^M, where M is the number of states of deterministic automaton for Q. Summing up, this algorithm requires PSPACE-complete problem to be called a triple-exponential number of times.

In the rest of this section we show how the minimal cardinality viewset selection problem can be reduced to the *minimal test length* problem. The crucial point in this reduction is Theorem 2.2. Let φ be a substitution defined by the rule $\varphi(\Delta) = \overline{\mathbf{F}}$. We shall abuse the notation and denote by $[Q]$ the set of all words w over Δ such that $\varphi(w) = Q$. The minimal cardinality viewset can be represented as a subset $B \subseteq \Delta$. For every query Q from the workload $\mathbf{Q}$ one can construct an automaton for $[Q]$, and the set B should satisfy the following condition: $[Q] \cap B^* \neq \varnothing$ for every $Q \in \mathbf{Q}$. For every query Q_i one can effectively construct the finite set of subsets of Δ, say $R_1^{(i)}, \ldots, R_{k_i}^{(i)} \subseteq \Delta$, such that Q_i can be represented in terms of $\varphi(B)$ iff $R_j^{(i)} \subseteq B$ for some $j \leqslant k_i$. These sets correspond to (labels on) simple paths in minimal automaton for $[Q_i]$. The minimal cardinality viewset is the minimal cardinality subset $B \subseteq \Delta$ such that for every Q_i there exists $j \leqslant k_i$ satisfying $R_j^{(i)} \subseteq B$. Although the minimal test length problem is NP-complete such reduction allows to avoid checking every possible subset of the set $\overline{\mathbf{F}}$. Moreover, when membership languages $[Q]$ are constructed for all $Q \in \mathbf{Q}$ the remaining computations use transparent structures and the complexity does not depend on number of states of automata for $\overline{\mathbf{F}}$.

3.3 Arbitrary Rewritings and p-Containment of Viewsets

Similarly to maximal solutions to linear language equations one can consider maximal solutions to equations of the form $r(X_1, \ldots, X_n) = L$ where r is an arbitrary regular expression over alphabet $\Sigma \cup \{X_1, \ldots, X_n\}$. It is known that for every regular language L there exist only finitely many different languages that are components of maximal solutions (see e.g. [19]).

Theorem 3.4. *Let $\mathbf{Q}$ be a finite set of regular languages. There exists a finite set of languages $\mathbf{F}$, that can be effectively constructed from $\mathbf{Q}$, such that at least one minimal cardinality viewset (wrt regular rewriting) is contained in $\mathbf{F}$.*

The proof is essentially the same as for Theorem 3.2.

The algorithm described in the previous section may produce several viewsets of minimal cardinality. Different viewsets can be compared by their expressive power. We say that a viewset $\mathbf{V}_1$ is *p-contained* in a viewset $\mathbf{V}_2$ if every query that can be answered using $\mathbf{V}_1$ can be answered using $\mathbf{V}_2$. The following theorem gives an algorithm for p-containment checking.

Theorem 3.5. *A viewset $\mathbf{V}_1$ is p-contained in a viewset $\mathbf{V}_2$ with respect to regular rewritings iff every view $V \in \mathbf{V}_1$ may be rewritten in terms of the viewset $\mathbf{V}_2$.*

4 Viewsets of Minimal Size

For now assume that the database instance $\mathcal{B}$ is fixed. Every regular path query defines a (finite) binary relation on the set of the database nodes and we define the *size* of a query Q, denoted by $|Q|$, as the cardinality of the corresponding relation. This quantity is proportional to the amount of space, in bytes, required to store the results to the query. Minimal size viewset selection problem is stated as follows: given a workload $\mathbf{Q}$ and a database instance $\mathcal{B}$ find a viewset $\mathbf{V}$ such that its total size $\sum_{V \in \mathbf{V}} |V|$ is minimal on the database $\mathcal{B}$ (the minimum is taken among all the viewsets that answers every query in the workload). The following proposition states that every minimal size viewset is contained in some maximal viewset.

Proposition 4.1. *Let $\mathbf{V}$ be a minimal size viewset for $\mathbf{Q}$ and $\mathcal{B}$. There exists a viewset $\mathbf{G}$ such that:*

– *for every $V \in \mathbf{V}$ there exists $G \in \mathbf{G}$ satisfying $V \subseteq G$;*
– *every language $G \in \mathbf{G}$ is the intersection of some maximal factors: $\mathbf{G} \subseteq \mathcal{F}(\mathbf{Q})$.*

The proof is straightforward. The main idea of the algorithm for instance based minimal size viewset construction is to start from a maximal viewsets and then minimize their elements by subtracting some regular languages.

Proposition 4.2. *Let $\mathbf{G} = \{G_1, \dots, G_k\}$ be a maximal viewset for the workload $\mathbf{Q}$, and $\mathcal{B}$ be a database instance. There exists an effectively constructable viewset*

$$\mathbf{G}' = \{G'_1, \dots, G'_k\}$$

such that the following conditions hold:

– *$G'_i \subseteq G_i$ for all $i = 1, \dots, k$;*
– *for any viewset $\mathbf{G}'' = \{G''_1, \dots, G''_k\}$ satisfying the above condition the inequality $|\mathbf{G}'| \leqslant |\mathbf{G}''|$ holds.*

Proof. Let $\mathbf{G} = \{G_1, \dots, G_k\}$ be a maximal viewset that is not a minimal size viewset. There exist a component $G \in \mathbf{G}$ such that $|\mathbf{G}'| < |\mathbf{G}|$, where $\mathbf{G}'$ is obtained from $\mathbf{G}$ by replacing G with some language $G' \subseteq G$. Clearly, that $G'(\mathcal{B}) \subset G(\mathcal{B})$ and the language

$$G'' = G \setminus \bigcup_{(u,v) \in G(\mathcal{B})} L_{(u,v)}(\mathcal{B})$$

has exact the same size as G. Since the database instance is finite, the number of views to be checked is finite and a minimal size viewset (contained in a given viewset) may be constructed by checking all possible combinations. □

As an immediate corollary of Propositions 4.1 and 4.2 we have:

Theorem 4.1. *There exists an algorithm that, for a given workload* $\mathbf{Q}$ *and a database instance* $\mathcal{B}$*, computes a viewset of minimal size.*

Now, suppose that the database instance is not known, but there exists an oracle $E : \mathrm{Reg}(\Sigma) \to \mathbb{N}$ that estimates the size of a query result. Estimation function E is called *exact* if $E(Q)$ equals to actual query result size. In this case we have implication $Q_1 \subseteq Q_2 \Rightarrow E(Q_1) \leqslant E(Q_2)$. We show that the problem of minimal size (with respect to the oracle E) requires construction of minimal solutions to language equations. Indeed, let $\mathbf{Q} = \{Q_1, Q_2\}$ and the equation $Q_1 X = Q_2$ has a solution. A possible minimal size viewset in this case is a pair $\{Q_1, S\}$, where S is a minimal solution to the equation such that $E(S) \leqslant E(S')$ for every other (minimal) solution S'. The problem is that even such simple linear equation admits uncountably many minimal solution. For example, the equation $\{\varepsilon, a\}X = \{a\}^*$ over the unary alphabet $\Sigma = \{a\}$ has infinitely many minimal solutions: every language $K \subseteq \Sigma^*$ that contains the empty word and satisfies the condition that if K contains the word a^n $(n \geqslant 0)$ then it contains only one of the words a^{n+1} or a^{n+2} is a minimal solution. It is not clear whether or not the minimal size viewset can be computed with respect to exact query size estimation function.

5 Conclusion

In this paper we proved that the search space for minimal cardinality and instance based minimal size viewset construction problems is finite, both for single word and regular rewritings. Thus, both problems are decidable and we proposed algorithms for such viewsets construction. We also proved that p-containment of viewsets of regular path queries is decidable for single word and regular rewritings. The main result on single-word rewritings may be stated in terms of semigroups of regular languages: The rank problem for finitely generated semigroups of regular languages is decidable.

It could be interesting to consider similar problems for partial rewritings. Recursionless (i.e., finite) partial rewritings was recently investigated in [12]. It seems that the main problem for view selection under partial rewriting is to choose an appropriate measure for the viewset. Since partial rewriting admits letters of original alphabet in the rewriting language, the empty viewset is the minimal cardinality and minimal size viewset. Thus, a relevant viewset quality measure should take into account (in addition cardinality and storage constraints) how often a query evaluation algorithm will access the database.

References

1. Afonin, S., Khazova, E.: Membership and finiteness problems for rational sets of regular languages. International Journal of Foundations of Computer Science 17(3), 493–506 (2006)
2. Afonin, S., Khazova, E.: A note on finitely generated semigroups of regular languages. In: Andre, J.M., et al. (eds.) Proceedings of the International Conference "Semigroups and Formal Languages", pp. 1–8. World Scientific, Singapore (2007)

3. Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.: Rewriting of regular expressions and regular path queries. Journal of Computer and System Sciences 64, 443–465 (2002)
4. Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.Y.: View-based query processing: On the relationship between rewriting, answering and losslessness. In: Eiter, T., Libkin, L. (eds.) ICDT 2005. LNCS, vol. 3363, pp. 321–336. Springer, Heidelberg (2004)
5. Calvanese, D., Giacomo, G.D., Lenzerini, M., Vardi, M.Y.: Answering regular path queries using views. In: ICDE, pp. 389–398 (2000)
6. Chirkova, R., Halevy, A.Y., Suciu, D.: A formal perspective on the view selection problem. The VLDB Journal 11(3), 216–237 (2002)
7. Chirkova, R., Li, C.: Materializing views with minimal size to answer queries. In: PODS 2003, pp. 38–48. ACM Press, New York (2003)
8. Conway, J.: Regular Algebra and Finite Machines. Chapman and Hall, Boca Raton (1971)
9. Franklin, M., Halevy, A., Maier, D.: From databases to dataspaces: a new abstraction for information management. SIGMOD Rec. 34(4), 27–33 (2005)
10. Grahne, G., Thomo, A.: Algebraic rewritings for optimizing regular path queries. Theoretical Computer Science 296(3), 453–471 (2003)
11. Grahne, G., Thomo, A.: Query containment and rewriting using views for regular path queries under constraints. In: PODS 2003, pp. 111–122. ACM Press, New York (2003)
12. Grahne, G., Thomo, A.: Boundedness of regular path queries in data integration systems. In: Proc. of IDEAS, pp. 85–92. IEEE Computer Society, Los Alamitos (2007)
13. Halevy, A.Y.: Theory of answering queries using views. SIGMOD Record (ACM Special Interest Group on Management of Data) 29(4), 40–47 (2000)
14. Hashiguchi, K.: Representation theorems on regular languages. Journal of computer and system sciences 27, 101–115 (1983)
15. Kirsten, D.: Desert automata and the finite substitution problem. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 305–316. Springer, Heidelberg (2004)
16. Leung, H., Podolskiy, V.: The limitedness problem on distance automata: Hashiguchi's method revisited. Theoretical Computer Science 310(1–3), 147–158 (2004)
17. Li, C., Bawa, M., Ullman, J.D.: Minimizing view sets without losing query-answering power. In: Van den Bussche, J., Vianu, V. (eds.) ICDT 2001. LNCS, vol. 1973, pp. 99–113. Springer, Heidelberg (2000)
18. Mandhani, B., Suciu, D.: Query caching and view selection for XML databases. In: Böhm, K., Jensen, C.S., Haas, L.M., Kersten, M.L., Larson, P.-Å., Ooi, B.C. (eds.) VLDB, pp. 469–480. ACM Press, New York (2005)
19. Polák, L.: Syntactic semiring and language equations. In: Champarnaud, J.-M., Maurel, D. (eds.) CIAA 2002. LNCS, vol. 2608, pp. 182–193. Springer, Heidelberg (2003)
20. Segoufin, L., Vianu, V.: Views and queries: determinacy and rewriting. In: PODS 2005, pp. 49–60. ACM Press, New York (2005)
21. Tajima, K., Fukui, Y.: Answering XPath queries over networks by sending minimal views. In: Nascimento, M.A., Özsu, M.T., Kossmann, D., Miller, R.J., Blakeley, J.A., Schiefer, K.B. (eds.) VLDB, pp. 48–59. Morgan Kaufmann, San Francisco (2004)

Optimal Higher Order Delaunay Triangulations of Polygons*

Rodrigo I. Silveira and Marc van Kreveld

Department of Information and Computing Sciences
Utrecht University, 3508 TB Utrecht, The Netherlands
{rodrigo,marc}@cs.uu.nl

Abstract. This paper presents an algorithm to triangulate polygons optimally using order-k Delaunay triangulations, for a number of quality measures. The algorithm uses properties of higher order Delaunay triangulations to improve the $O(n^3)$ running time required for normal triangulations to $O(k^2 n \log k + kn \log n)$ expected time, where n is the number of vertices of the polygon. An extension to polygons with points inside is also presented, allowing to compute an optimal triangulation of a polygon with $h \geq 1$ components inside in $O(kn \log n) + O(k)^{h+2}n$ expected time. Furthermore, through experimental results we show that, in practice, it can be used to triangulate point sets optimally for small values of k. This represents the first practical result on optimization of higher order Delaunay triangulations for $k > 1$.

1 Introduction

One of the best studied topics in computational geometry is the triangulation. When the input is a point set P, it is defined as a subdivision of the plane whose bounded faces are triangles and whose vertices are the points of P. When the input is a polygon, the goal is to decompose it into triangles by drawing diagonals.

Triangulations have applications in a large number of fields, including computer graphics, multivariate analysis, mesh generation, and terrain modeling. Since for a given point set or polygon, many triangulations exist, it is possible to try to find one that is the best according to some criterion that measures some property of the triangulation.

The properties of interest are application-dependent, but are generally either local properties of the triangles (like area, height or minimum angle) or global properties of the triangulation (such as total edge length). For example, in automatic mesh generation for finite element methods, criteria like minimizing the minimum/maximum angle and height of the triangles are related to the error of the finite element approximation [3,21]. In the context of terrain modeling, terrains are many times represented by *triangulated irregular networks*, which are

* This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) under the project GOGO.

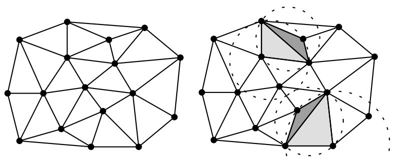


Fig. 1. A Delaunay triangulation ($k = 0$) (left), and an order-2 triangulation (right). Light gray triangles are first order, the medium grey ones are second order.

triangulations where each point has an elevation. When terrain models need to be *realistic*, for example in visualization or terrain analysis for hydrology, criteria like avoiding *ill-shaped* triangles and having few local minima are particularly relevant [6,18].

For a given set of points P, a well-known triangulation is the Delaunay triangulation. It is defined as a triangulation where the circumcircle of the three vertices of any triangle does not contain any other point of P. It is unique when no four points are cocircular, and can be computed in $O(n \log n)$ time for n points. The Delaunay triangulation has a large number of known properties, and it optimizes several measures, like max min angle or min max smallest enclosing circle, among others. This is the reason why its triangles are said to be well-shaped. However, for many applications, the Delaunay triangulation is not flexible enough. For example, when used for terrain modeling, the triangulation does not take the third dimension into account, which may result in artifacts like interrupted valley lines or artificial local minima.

To overcome this limitation, Gudmundsson et al. [11] define *higher order Delaunay triangulations*, a class of well-shaped triangulations where a few points are allowed inside the circumcircles of the triangles. A triangle is said to be *order-k Delaunay* if its circumcircle contains at most k points. A triangulation is *order-k Delaunay* if all its triangles are order-k Delaunay (see Figure 1). For $k = 0$, each non-degenerate point set has only one higher order Delaunay triangulation, equal to the Delaunay one. As the parameter k is increased, more points inside the circumcircles imply a reduction in the shape quality of the triangles, but also an increase in the number of triangulations that are considered, and hence greater flexibility to optimize some other criterion, while limiting the badness of the shape of the triangles. The concept of higher order Delaunay triangulation has been successfully applied to several areas, including terrain modeling [7], minimum interference networks [1] and multivariate splines [20].

In this paper we focus mainly on triangulations of polygons. Optimal polygon triangulation has been a subject of study for a long time, both because it has applications of its own, like in finite element methods, and also because it gives insight into the generally harder problem of optimal point set triangulation. When the goal is to optimize only one criterion, optimal polygon triangulations can be computed in polynomial time for many measures. The constrained Delaunay triangulation [5], which generalizes the standard definition in order to force certain edges into the triangulation, can be used to triangulate polygons,

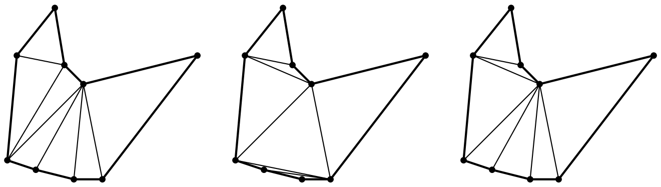


Fig. 2. Three different triangulations of a polygon: the constrained Delaunay triangulation (left), the minimum weight triangulation (center), and the minimum weight triangulation constrained to order-2 Delaunay triangulations (right). The third one combines nicely shaped triangles with the minimization of the weight.

with triangle shape properties similar to the ones of the Delaunay triangulation. Many other measures can be optimized using a dynamic programming algorithm attributed to Klincsek [15], and also, independently, proposed by Gilbert [10]. This approach allows to find in $O(n^3)$ time an optimal triangulation of a simple polygon for any *decomposable* measure. Intuitively, a measure is decomposable if the measure of the whole triangulation can be computed efficiently from the measures of two pieces, together with the information on how the pieces are glued together. See [3] for a formal definition. Decomposable measures include the following: min / max angle, min / max circumcircle, min / max length of an edge, min / max area of triangle, and the sum of the edge lengths. The algorithm by Klincsek can be extended to other measures that are not decomposable, like maximum vertex degree. For convex polygons, the min/max area of triangle measures can be optimized even faster, in $O(n^2 \log n)$ time [14].

Triangulating point sets optimally is in general more difficult than triangulating polygons. For example, the minimum weight triangulation can be computed for polygons in cubic time using Klincsek's algorithm, but is NP-hard for point sets [19]. Only a few methods exist for optimal triangulations of point sets. The edge insertion paradigm [2] can be used to optimize several measures in $O(n^2 \log n)$ or $O(n^3)$ time (depending on the measure). A triangulation of a point set minimizing the maximum edge length can be computed in $O(n^2)$ time [8]. The *greedy triangulation*, which lexicographically minimizes the sorted vector of length edges, can be constructed in $O(n \log n)$ time [17].

Our problem is more involved, since we aim at optimizing a measure over higher order Delaunay triangulations, therefore enforcing well-shaped triangles at the same time as optimizing some other measure. Figure 2 shows an example. There are not many results on optimal higher order Delaunay triangulations. For the case $k = 1$, the triangulations have a special structure that allows a number of measures (for example max/min area triangle, total edge length, number of local minima in a terrain) to be optimized in $O(n \log n)$ time [11]. A few other measures, like minimizing the maximum area ratio of edge-adjacent or vertex-adjacent triangles, can also be optimized efficiently [16]. Other measures like maximizing the number of convex edges or minimizing the maximum vertex degree have been shown to be NP-hard [16]. For $k > 1$, fewer results are known.

Minimizing local minima in a terrain is NP-hard for orders at least n^ε, where ε is any positive constant [7].

In this paper we present an extension of the algorithm by Klincsek [15] that allows to optimize a decomposable measure for a simple polygon over order-k Delaunay triangulations. A straightforward extension of Klincsek's algorithm leads to $O(kn^3 + n^3 \log n)$ running time. Our main contribution is improving this to $O(k^2 n \log k + kn \log n)$, by exploiting properties of this special class of triangulations. This represents an important improvement, given that small values of k are most important [7].

We also explain how to extend our algorithm to triangulate polygons with points inside, and present experimental results on the structure of order-k Delaunay triangulations that suggest that in practice, the same approach can be used to triangulate point sets for small values of k optimally. This constitutes the first practical result on optimization of higher order Delaunay triangulations for $k > 1$.

Note that in this paper we use the standard definition of *order* of a triangle, as in [11,20], which does not take the boundary edges of the polygon into account. This implies that for a given polygon and value k, our algorithm may find that no order-k triangulation of that polygon exists. In such a case, it is always possible to increase k until a triangulation is found. The authors recently studied possible definitions of the order of a triangle that take a set of constraining edges into account [22], which guarantee that a polygon can always be triangulated for any k. However, although seven different definitions for this notion of *constrained order* were proposed in [22], no single definition can be regarded as the *natural* or *right* one.

2 Higher Order Delaunay Triangulations

In this section we present some basic concepts on higher order Delaunay triangulations, together with some results that will be needed later. From now on we assume non-degeneracy of the input set P: no four points are cocircular.

Definition 1. *(from [11]) A triangle $\triangle uvw$ in a point set P is* order-k Delaunay *if its circumcircle $C(u, v, w)$ contains at most k points of P. A triangulation of a set P of points is an* order-k Delaunay triangulation *if every triangle of the triangulation is order-k.*

Definition 2. *(from [11]) Let P be a set of points, and let $\overline{pq}$ be an edge between two points p, $q \in P$. $\overline{pq}$ is an* order-k Delaunay edge *if there exists a circle that passes through p and q that has at most k points of P inside. The* useful order *of an edge is the lowest order of a triangulation that includes that edge.*

For brevity, we will sometimes write *order-k* instead of *order-k Delaunay*, and *k-OD edge* instead of *order-k Delaunay edge*. We also assume that $k \geq 1$ is a given integer, and write *useful edge* instead of *useful k-OD edge*. It is worth mentioning that the order and the useful order of an edge can differ arbitrarily much.

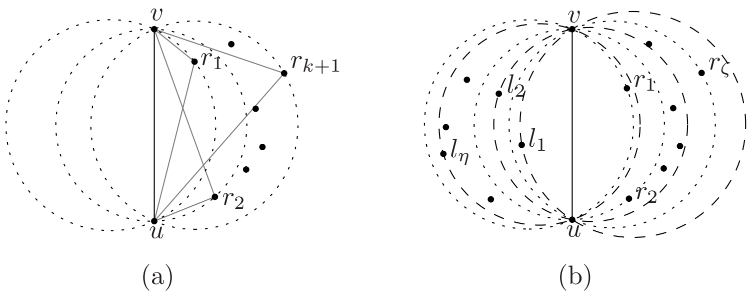


Fig. 3. (a) At most $k+1$ *third points.* (b) Finding the *k-OD* triangles incident to $\overline{uv}$.

Lemma 1. *(from [11]) Let $\overline{uv}$ be a k-OD edge, let s_1 be the point to the left of $\overrightarrow{vu}$, such that the circle $C(u, s_1, v)$ contains no points to the left of $\overrightarrow{vu}$. Let s_2 be defined similarly but to the right of $\overrightarrow{vu}$. Edge $\overline{uv}$ is a useful k-OD edge if and only if $\triangle uvs_1$ and $\triangle uvs_2$ are k-OD triangles.*

We extend the basic definitions and lemma above with one more lemma.

Lemma 2. *Let $\overline{uv}$ be a useful k-OD edge. There are $O(k)$ order-k triangles that have $\overline{uv}$ as one of their edges.*

Proof. We will show that $\overrightarrow{uv}$ can be part of at most $k+1$ triangles on each side. Imagine we slide a circle in contact with u and v until it touches a first point r_1 to the right of the edge (see Figure 3(a)). This could potentially be a *third point* of a triangle that includes $\overrightarrow{uv}$ because it is possible for the circumcircle of triangle $\triangle ur_1v$ to contain less than $k+1$ points. If we slide the circle again until it touches a second point r_2, we now know that the circle contains at least one point (r_1). Continuing in this way it can be seen that the circle defined by u, v and the $(k+1)$-th touching point, r_{k+1}, contains at least k points, hence no further point can be a *third point* because then the circle would contain $k+1$ points. Since an identical argument can be applied to the left side, we conclude that at most $O(k)$ triangles can have $\overrightarrow{uv}$ as one of its edges. ⊠

Next we show that all the order-k triangles formed by useful *k-OD* edges can be computed efficiently.

Lemma 3. *Let P be a set of n points in the plane. In $O(k^2 n \log k + kn \log n)$ expected time one can compute all order-k triangles of P that are incident to at least one useful k-OD edge.*

We provide a sketch of the algorithm. An order-k triangle must meet two conditions: its circumcircle must contain at most k points and it must be empty. For a set of n points there are $O(kn)$ useful *k-OD* edges [11], and by Lemma 2, a given useful edge can be part of $O(k)$ order-k triangles. This makes the total number of order-k triangles $O(k^2 n)$.

All the useful edges can be computed in $O(k^2 n + kn \log n)$ expected time [11]. Moreover, without increasing the previous asymptotic running time, we can store for every useful edge $\overrightarrow{uv}$, the two sets of points that are contained in the two circles that determine its usefulness (see Lemma 1). There are at most k of these points on each side. For each side, we will sort the points according to the order in which they are touched when sliding a circle in contact with u and v (in the same way as in the proof of Lemma 2). This can be done in $O(k \log k)$ time by sorting the centers of the circles.

To find out which *third points* can make an order-k triangle, we need to count the number of points inside each circumcircle. This can be done using the two sorted lists of points as follows. We explain how to do it for the right side, the left side is identical. Let $L = \{l_\eta, ..., l_1\}$ be the sorted points to the left of $\overrightarrow{uv}$, and $R = \{r_1, ..., r_\zeta\}$ the ones to its right. See Figure 3(b). The circumcircle of $\triangle uvr_1$, denoted $C(u, v, r_1)$, contains by definition no points to the right of $\overrightarrow{uv}$ and η points to its left. For the second point, r_2, we know that $C(u, v, r_2)$ contains exactly one point to the right of $\overrightarrow{uv}$ (namely, r_1). To find out how many points it contains to the left of $\overrightarrow{uv}$, we check whether l_η is inside $C(u, v, r_2)$ or not. If it is, then $C(u, v, r_2)$ contains exactly η points to the left of $\overrightarrow{uv}$. If it is outside, we go through L until we find the first l_j that is inside $C(u, v, r_2)$. That implies $C(u, v, r_2)$ contains exactly j points to the left of $\overrightarrow{uv}$. This is then repeated for $r_3, r_4, ..., r_\zeta$. The running time is linear in k because both lists are scanned only once, from left to right. Hence for each useful edge $\overrightarrow{uv}$, we can find part of the triangles incident to $\overrightarrow{uv}$ whose circumcircles contain at most k points in $O(k \log k)$ time.

Notice that this algorithm, when applied to one edge $\overrightarrow{uv}$, does not necessarily find all the order-k triangles adjacent to $\overrightarrow{uv}$. It may happen that some of the order-k triangles adjacent to it have a third point that is not among the points included in the two circles defining the usefulness of $\overrightarrow{uv}$, because these circles contain *at most* k points, but may contain less. However, we show in the full paper that any triangle composed of three useful order-k edges that is missing will be considered when processing one of the other two useful edges that make the triangle, hence no relevant triangle will be missed at the end.

Still, some of these triangles may contain points inside, so we need to discard the ones that are not empty. Let $\triangle uvx$ and $\triangle uvy$ be two triangles, and let α_u (α_v) denote the angle of $\triangle uvx$ at u (at v), and β_u (β_v) the same for $\triangle uvy$. It is easy to see that $\triangle uvx$ contains point y if and only if $\beta_u < \alpha_u$ and $\beta_v < \alpha_v$. Each triangle can be represented by a point in the plane using its angles at u and at v. The empty triangles are the ones lying on the lower-left staircase of the point set, and can be found in $O(k \log k)$ time by a sweep line algorithm.

The total time needed to find the triangles for one useful edge is $O(k \log k)$. Since there are $O(kn)$ useful edges, all the useful edges and order-k triangles can be computed in $O(k^2 n \log k + kn \log n)$ expected time, proving Lemma 3.

3 Triangulating Polygons

As mentioned in the introduction, Klincsek's algorithm allows to triangulate polygons optimally for a large number of measures using dynamic programming.

In this paper we have the additional requirement that the resulting triangulation must be order-k, therefore the classical algorithm must be adapted to include only order-k triangles.

The input of the algorithm is a polygon P, defined by its vertices in clockwise order: $p_0, p_1, \ldots, p_{n-1}$. The output is a k-OD triangulation of optimum cost, if it exists. It may be that the useful order of some of the polygon edges is such that no order-k triangulation of P exists at all.

The dynamic programming algorithm finds an optimal solution by combining solutions of smaller problems in a systematic way. The typical algorithms have $O(n^3)$ running time and use an $n \times n$ matrix L, which in our problem has the following meaning: $L[i, i+j]$ is the cost of the optimal k-OD triangulation of the polygon $P_{i,i+j}$, defined by the edges of P between p_i and p_{i+j}, plus edge $\overline{p_{i+j}p_i}$.

The matrix can be filled in a recursive way. The simplest entries are the ones of the form $L[i, i+1]$, which have cost 0. The recursive formula for $L[i, i+j]$ is:

$$L[i, i+j] = \min_{q=1,2,\ldots,j-1} (Cost(p_i, p_{i+q}, p_{i+j}) \oplus L[i, i+q] \oplus L[i+q, i+j]) \quad (1)$$

The expression $Cost(p_i, p_{i+q}, p_{i+j})$ denotes the cost of triangle $\triangle p_i p_{i+q} p_{i+j}$, and the operator $\oplus$ represents a way to combine the values of the subproblems. Their precise meaning depends on the measure being optimized. Edges $\overline{p_i p_{i+q}}$ and $\overline{p_{i+q} p_{i+j}}$ must be diagonals. Triangles that are not contained entirely inside P or are not k-OD have cost $+\infty$. Checking the latter (verifying that there are no more than k points inside the circumcircle of the triangle) would take $O(\log n + k)$ time [11], but if we precompute all the order-k triangles for each useful edge (see Section 2) and store them in a perfect hashing table [9], we can find out in $O(1)$ time if the triangle is among the order-k triangles. Note that measures of the type $\min\max(\ldots)$ can also be optimized with the same method, by using a recursive formula similar to (1).

We can take advantage of the properties of higher order Delaunay triangulation to reduce the running time significantly. The main steps of the algorithm are the following (details are given below). In the preprocessing phase we compute all the useful edges and filter out the ones that are not fully contained inside the polygon. The order-k triangles adjacent to each useful edge are precomputed. The triangulation algorithm proceeds by applying Equation (1), using a perfect hashing table to store the solutions to the subproblems already computed.

For a given edge $\overline{p_i p_j}$, the number of possible *third points* to form a triangle is not $O(n)$, as in the normal triangulation problem, but $O(k)$ (see Lemma 2). If for every edge we precompute these $O(k)$ points, we can improve the $O(n^3)$ dynamic programming running time to $O(kn^2)$, after spending $O(k^2 n \log k + kn \log n)$ time in the precomputation of the useful edges and the order-k triangles (see Lemma 3).

Every time an edge is considered as a candidate to be in an optimal triangulation, we must also check that it does not intersect the polygon boundary and that it does not lie outside the polygon. This check can be done in $O(\log n)$

time per edge using an algorithm for ray shooting in polygons [13]. This adds an $O(kn \log n)$ term to the preprocessing time, which does not increase the previous asymptotic running time.

Finally, the matrix L has $O(n^2)$ cells, each corresponding to one potential edge. However, we know that only $O(kn)$ edges will be useful, so it is not necessary to keep a data structure of quadratic size. In order to avoid wasting time and space on edges that are not useful, we will not use the standard matrix-based dynamic programming algorithm, but we will use a *memoized* version instead. The idea is to use Equation (1) to compute $L[0, n-1]$, and maintain a perfect hashing table where we will store all the subproblems already solved. Notice that each subproblem $L[i, j]$ is associated with the insertion of an edge $\overline{p_i p_j}$, which must be useful k-OD. Hence, only $O(kn)$ subproblems will be computed and stored. The same table used to store the order-k triangles incident to an edge can be extended to also store the value of the subproblem associated with that edge. To solve one particular problem $O(k)$ time is needed, yielding a total running time of $O(k^2 n)$, plus $O(k^2 n \log k + kn \log n)$ preprocessing time.

Theorem 1. *In $O(k^2 n \log k + kn \log n)$ expected time one can compute an optimal order-k Delaunay triangulation of a simple polygon with n vertices that optimizes a decomposable measure.*

4 Triangulating Polygons with Points Inside

In this section we consider the more general problem of finding an optimal triangulation of a simple polygon that contains h components in its interior. A component can be either a point or a connected component made of several points connected by edges. We will denote the polygon with the components by P. We can reuse the algorithm from the previous section if we connect one vertex of each component to some other vertex in order to remove all the loose parts. To find the optimal triangulation we must try, in principle, all the possible ways to make these connections. The number of them depends on h and on the order k. In principle, there are $O(n)$ ways to connect each component. However, since we need only one edge that connects the component to the outer boundary of the polygon, we don't need to try $O(n)$ but only $O(k)$ edges.

Lemma 4. *Let P be a polygon with h components inside. There is a collection of $O(k)^h$ sets, of h edges each, such that: (i) for every set in the collection, the edges in the set connect all the components in P to the outer boundary; (ii) any order-k Delaunay triangulation includes the edges of some set in the collection.*

Proof. First we show that for a given component in P, any order-k triangulation T of P must connect the component to the rest of the polygon by one of $O(k)$ edges. Let u be the topmost point among the boundary points of the components inside the polygon. Everything above u is part of the outer polygon boundary. Let $\overline{uv}$ be an edge of the Delaunay triangulation of the *point set* induced by P

and its components (ignoring the edges), with v higher than u. Since $\overline{uv}$ is a Delaunay edge, we know from [11] that the useful k-OD edges that cross it have $O(k)$ endpoints on each side of $\overline{uv}$. If $\overline{uv}$ is not part of T, at least one of the useful edges that cross it must be. Let $\overline{xy}$ be the first of these edges (the first one encountered when going from u to v along $\overline{uv}$) in T, then triangle $\triangle uxy$ must be part of T. This implies that edges $\overline{ux}$ and $\overline{uy}$ are part of it as well. Hence, either $\overline{uv}$ or one of the $O(k)$ possible edges of type $\overline{uz}$ (for $z = x$ or $z = y$) must be in T, and connects v to a higher point of the outer boundary of P.

Following the same idea, for each of the h components we can find a set of $O(k)$ useful k-OD edges such that any triangulation T connects each component using one of these $O(k)$ edges. The result follows. $\boxtimes$

Using the previous result, our algorithm will try the $O(k)^h$ different ways to connect the loose components in P. Let $P_1, \cdots, P_\eta$ be the $O(k)^h$ different polygons that are tried, and let H_i be the set of new boundary edges of P_i. For each P_i, besides computing the boundary, we must compute the intersections between the $O(kn)$ useful edges and the new h edges in H_i, which were added to connect the loose components. This is because during the triangulation we need to consider only edges that make the polygon simply-connected.

The computation of these intersections can be done once and maintained between successive polygons without increasing the asymptotic running time. During the preprocessing phase, we will compute all the intersections between useful k-OD edges. A useful k-OD edge can intersect $O(k^2)$ other useful k-OD edges [11]. Therefore the total number of intersections is $O(k^3 n)$, and they can be computed in time $O(kn \log kn + k^3 n)$=$O(kn \log n + k^3 n)$ [4]. We store for each useful edge all the other useful edges that it intersects and in addition we keep a counter. The counter will be used to keep track of how many edges in H_i each useful edge intersects.

The algorithm will iterate through the polygons in such a way that two consecutive polygons P_i and P_{i+1} differ only in the edge chosen for one of the components. Then during the $(i+1)$-th step the counters for H_i are already computed, and one can compute the counters for H_{i+1} very easily, as follows: let e_{out} be the edge that is removed and e_{in} the new edge (that is, $H_{i+1} = H_i \backslash \{e_{out}\} \cup \{e_{in}\}$). Firstly, all the $O(k^2)$ useful edges that intersect e_{out} must have their counters decreased by one. Secondly, all counters of the $O(k^2)$ edges that intersect e_{in} are incremented by one. Both sets of edges were previously computed during the preprocessing phase and can be accessed in constant time. Hence the time needed to update the intersection information from one polygon to the next one is $O(k^2)$.

We conclude that the total time required to compute all the new intersections is $O(kn \log n + k^3 n)$ for the preprocessing and $O(k^2)$ per polygon. Note that the useful edges and order-k triangles do not need to be recomputed, since they only depend on the point set, which has not changed.

Triangulating each generated polygon using the algorithm from the previous section takes $O(k^2 n)$ time, yielding a total time of $O(k^2 n \log k + nk \log n + kn \log n + k^3 n) + O(k)^h (k^2 + k^2 n)$=$O(kn \log n) + O(k)^{h+2} n$ (because $h \geq 1$).

Theorem 2. *An optimal order-k Delaunay triangulation of a simple polygon with n boundary vertices and $h \geq 1$ components inside that optimizes a decomposable measure can be computed in $O(kn \log n) + O(k)^{h+2}n$ expected time.*

5 Some Other Measures

The approach described above can also be used to optimize some other, non-decomposable, measures. The challenging part is adapting the recursive formula to make the subproblems independent. In the full paper we show how this can be done for minimizing the maximum vertex degree, minimizing the number of local minima (if the points have an elevation), and even optimizing functions of quadrilaterals, such as minimizing the maximum area ratio of triangles sharing an edge. In general these variations involve more complicated algorithms and have higher running time.

Interestingly, a similar approach also allows to find the lowest order completion of a polygon and a set of diagonals, that is, finding a higher order Delaunay triangulation, with the lowest possible order, which contains the given diagonals. The more general problem of finding a lowest order completion of a point set with respect to a set of edges is still open; polynomial time algorithms are known only for $k \leq 3$ [12].

6 Application to Point Sets

Any point set can be optimally triangulated using the results from Section 4 if it is seen as a polygon made of its convex hull with points inside. In general, this will lead to a running time exponential in n, so this is of no practical use. For low order Delaunay triangulations, the situation is better. Given a point set and an order k, there are *fixed edges* that are present in any order-k triangulation, and partition the convex hull of the point set into a number of polygons with components inside. For $k = 1$, it is known that these polygons are always empty triangles or quadrilaterals [11], which simplifies the optimization of several measures. As k increases, the number of fixed edges decreases until it is reduced to little more than the convex hull. Moreover, for $k > 1$ the polygons may be larger and may have many components inside. However, our experiments on the structure of higher order Delaunay triangulations suggest that in practice, for small values of k, the appearance of such polygons is rather unlikely.

We summarize part of the results of experiments carried out on randomly generated point sets, which show that for small values of k, the polygons created contain only a few components. The experiments consisted in generating random point sets of between 1000 and 5000 points, and for different values of k, computing the partition into polygons with components inside given by the fixed edges. The size of the polygons and the number of components was registered. These are the two factors, besides k, involved in the running time of the algorithm of Section 4. Tables 1 and 2 show the results for point sets between 1000 and 5000 points, and $k = 1, \ldots, 10$. It is worth mentioning that since the

Table 1. Structure of order-k triangulations: average / maximum number of components per polygon for random point sets of n points, averaged over 200 runs

k	$n = 1000$	$n = 2000$	$n = 3000$	$n = 4000$	$n = 5000$
1	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00
2	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00
3	0.00 / 0.03	0.00 / 0.03	0.00 / 0.04	0.00 / 0.06	0.00 / 0.07
4	0.00 / 0.44	0.00 / 0.68	0.00 / 0.80	0.00 / 0.94	0.00 / 0.90
5	0.01 / 1.15	0.01 / 1.35	0.01 / 1.50	0.01 / 1.53	0.01 / 1.57
6	0.05 / 2.22	0.05 / 2.63	0.04 / 2.77	0.04 / 2.91	0.04 / 3.13
7	0.18 / 5.52	0.17 / 6.77	0.16 / 6.82	0.16 / 7.38	0.16 / 7.87
8	0.55 / 18.95	0.53 / 24.19	0.52 / 29.85	0.52 / 31.28	0.52 / 35.85
9	1.45 / 61.05	1.47 / 108.70	1.47 / 150.83	1.52 / 205.30	1.53 / 242.75
10	3.08 / 115.55	3.33 / 225.36	3.44 / 332.13	3.62 / 443.35	3.72 / 552.41

Table 2. Structure of order-k triangulations: average / maximum size of polygons for random point sets of n points, averaged over 200 runs. Since the dynamic programming algorithm works by considering triangles, the size is measured as the number of Delaunay triangles that the polygon contains. More details in the full version.

k	$n = 1000$	$n = 2000$	$n = 3000$	$n = 4000$	$n = 5000$
1	1.35 / 2.00	1.35 / 2.00	1.35 / 2.00	1.36 / 2.0	1.35 / 2.00
2	1.99 / 6.82	1.99 / 7.41	1.99 / 7.54	2.00 / 7.63	2.00 / 7.77
3	2.84 / 12.77	2.85 / 13.84	2.84 / 14.60	2.85 / 15.14	2.85 / 15.59
4	4.02 / 23.54	4.04 / 25.95	4.04 / 28.02	4.04 / 28.37	4.04 / 29.95
5	5.80 / 43.97	5.82 / 52.90	5.83 / 54.55	5.84 / 56.25	5.85 / 59.72
6	8.63 / 91.66	8.68 / 108.55	8.74 / 115.63	8.74 / 123.63	8.78 / 129.58
7	13.48 / 222.25	13.56 / 263.83	13.72 / 292.36	13.83 / 310.12	13.86 / 337.01
8	21.71 / 569.40	22.40 / 749.15	22.76 / 980.10	23.21 / 1038.78	23.41 / 1223.43
9	34.99 / 1294.76	37.26 / 2425.46	38.32 / 3434.07	39.89 / 4716.74	40.58 / 5661.38
10	49.91 / 1755.54	56.27 / 3566.14	59.04 / 5341.94	62.56 / 7195.49	64.77 / 9034.72

convex hull of the point set limits the growth of the polygons, the results may be influenced slightly by boundary effects.

A few observations are in order. The experiments confirm that for $k \leq 3$, it is very unlikely to find polygons with components inside. Even though for $k \geq 2$ one can build examples where that is the case, they hardly arise in random point sets. Even for orders up to 5 or 6, the size of the polygons and number of components are small enough to be useful for practical purposes. As a result, finding optimal triangulations that in general are NP-hard, like the minimum weight triangulation, can be done in practice if the Delaunay order is low enough. The small values of k are the most useful in practice, for several reasons. On the one hand, as k increases, the shape of the triangles deteriorates. On the other hand, previous experimental results [7], related to realistic terrain modeling, have shown that low values of k are enough to obtain important improvements on several terrain measures (like the number of local minima), making small values of k particularly interesting for these applications.

7 Discussion

We studied algorithms to find higher order Delaunay triangulations of polygons that optimize a decomposable measure. Based on an existing technique for polygon triangulation, we proposed an algorithm to compute an optimal triangulation of a polygon restricted to order-k triangulations. Their specific properties allowed us to reduce an $O(n^2)$ factor to $O(k^2)$, a substantial improvement since k will be, in general, much smaller than n [7]. Our method can also be extended to some non-decomposable measures, like maximum vertex degree. For the more general problem of triangulating optimally a polygon with components inside, we presented an algorithm that is fixed-parameter tractable for $k = O(1)$.

We also gave experimental evidence suggesting that the specific structure of order-k Delaunay triangulations, for small values of k, makes the algorithm presented here applicable to point sets. This constitutes the first practical result on optimal higher order Delaunay triangulations for $k > 1$, allowing to optimize any decomposable function over a class of well-shaped triangulations.

Acknowledgments. We thank Remco Burema for carrying out the experimental work in this paper.

References

1. Benkert, M., Gudmundsson, J., Haverkort, H.J., Wolff, A.: Constructing interference-minimal networks. In: Wiedermann, J., Tel, G., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2006. LNCS, vol. 3831, pp. 166–176. Springer, Heidelberg (2006)
2. Bern, M., Edelsbrunner, H., Eppstein, D., Mitchell, S., Tan, T.S.: Edge insertion for optimal triangulations. Discrete Comput. Geom. 10(1), 47–65 (1993)
3. Bern, M., Eppstein, D.: Mesh generation and optimal triangulation. In: Du, D.-Z., Hwang, F.K. (eds.) Computing in Euclidean Geometry, Lecture Notes Series on Computing, 2nd edn. vol. 4, pp. 47–123. World Scientific, Singapore (1995)
4. Chazelle, B., Edelsbrunner, H.: An optimal algorithm for intersecting line segments in the plane. J. ACM 39(1), 1–54 (1992)
5. Chew, L.P.: Constrained Delaunay triangulations. Algorithmica 4, 97–108 (1989)
6. De Floriani, L., Falcidieno, B., Pienovi, C.: Delaunay-based representation of surfaces defined over arbitrarily shaped domains. Comput. Vision Graph. Image Process. 32, 127–140 (1985)
7. de Kok, T., van Kreveld, M., Löffler, M.: Generating realistic terrains with higher-order Delaunay triangulations. Comp. Geom. Theory Appl. 36, 52–65 (2007)
8. Edelsbrunner, H., Tan, T.S.: A quadratic time algorithm for the minmax length triangulation. SIAM J. Comput. 22, 527–551 (1993)
9. Fredman, M.L., Komlos, J., Szemeredi, E.: Storing a sparse table with $O(1)$ worst case access time. J. ACM 31(3), 538–544 (1984)

10. Gilbert, P.D.: New results in planar triangulations. Report R-850, Coordinated Sci. Lab., Univ. Illinois, Urbana, IL (1979)
11. Gudmundsson, J., Hammar, M., van Kreveld, M.: Higher order Delaunay triangulations. Comput. Geom. Theory Appl. 23, 85–98 (2002)
12. Gudmundsson, J., Haverkort, H., van Kreveld, M.: Constrained higher order Delaunay triangulations. Comput. Geom. Theory Appl. 30, 271–277 (2005)
13. Hershberger, J., Suri, S.: A pedestrian approach to ray shooting: Shoot a ray, take a walk. J. Algorithms 18, 403–431 (1995)
14. Keil, J.M., Vassilev, T.S.: Algorithms for optimal area triangulations of a convex polygon. Comput. Geom. Theory Appl. 35(3), 173–187 (2006)
15. Klincsek, G.T.: Minimal triangulations of polygonal domains. Discrete Math. 9, 121–123 (1980)
16. van Kreveld, M., Löffler, M., Silveira, R.I.: Optimization for first order Delaunay triangulations. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 175–187. Springer, Heidelberg (2007)
17. Levcopoulos, C., Krznaric, D.: The greedy triangulation can be computed from the Delaunay triangulation in linear time. Comput. Geom. Theory Appl. 14, 197–220 (1999)
18. Mark, D.: Network models in geomorphology. In: Anderson, M.G. (ed.) Modelling Geomorphological Systems, ch. 4, pp. 73–97. John Wiley & Sons, Chichester (1988)
19. Mulzer, W., Rote, G.: Minimum weight triangulation is NP-hard. In: Proc. 22nd Annu. ACM Sympos. Comput. Geom., pp. 1–10 (2006)
20. Neamtu, M.: Delaunay configurations and multivariate splines: a generalization of a result of B. N. Delaunay. Trans. Amer. Math. Soc. 359(7), 2993–3004 (2007)
21. Pebay, P.P., Baker, T.J.: A comparison of triangle quality measures. In: Proceedings of the 10th International Meshing Roundtable, pp. 327–340 (2001)
22. Silveira, R.I., van Kreveld, M.: Towards a Definition of Higher Order Constrained Delaunay Triangulations. In: Proceedings of the 19th Annual Canadian Conference on Computational Geometry (CCCG 2007), pp. 161–164 (2007)

Coloring Geometric Range Spaces

Greg Aloupis[1], Jean Cardinal[1], Sébastien Collette[1,⋆], Stefan Langerman[1,⋆⋆], and Shakhar Smorodinsky[2,⋆⋆⋆]

[1] Université Libre de Bruxelles, CP212, Bld. du Triomphe, 1050 Brussels, Belgium.
Partially supported by the Communauté française de Belgique - ARC
{greg.aloupis,jcardin,secollet,slanger}@ulb.ac.be
[2] Institute of Mathematics, Hebrew University, Givat-Ram, Jerusalem 91904, Israel
shakhar@cims.nyu.edu

Abstract. Given a set of points in $\mathbb{R}^2$ or $\mathbb{R}^3$, we aim to color them such that every region of a certain family (for instance disks) containing at least a certain number of points contains points of many different colors. Using k colors, it is not always possible to ensure that every region containing k points contains all k colors. Thus, we introduce two relaxations: either we allow the number of colors to increase to $c(k)$, or we require that the number of points in each region increases to $p(k)$. We give upper bounds on $c(k)$ and $p(k)$ for halfspaces, disks, and pseudo-disks. We also consider the dual question, where we want to color regions instead of points. This is related to previous results of Pach, Tardos and Tóth on decompositions of coverings.

1 Introduction

In this contribution, we are interested in coloring finite sets of points in $\mathbb{R}^2$ or $\mathbb{R}^3$ so that any region (within a specified family) that contains at least some fixed number of points, also contains a significant number of distinctly colored points. For example, we study the following problem: *Does there exist a constant α such that given any set of points in the plane, it is always possible to color the points with k colors so that any halfplane containing at least αk points contains a point of each color?* In Section 2 we answer this question on the affirmative.

We also allow the number of available colors and the number of required distinct colors to be different. We ask, for instance, *Does there exist a constant α such that given a set of points in the plane, it is always possible to color the points with αk colors so that any halfplane containing at least k points also contains points of k distinct colors?* We show this is true as well. We ask similar questions for other types of regions such as disks and pseudo-disks

These types of problems can be seen as coloring range spaces induced by intersections of sets of points with geometric objects. The corresponding dual range spaces are those obtained by considering a finite set of regions in $\mathbb{R}^2$ or $\mathbb{R}^3$,

⋆ Chargé de Recherches du FRS-FNRS.
⋆⋆ Chercheur Qualifié du FRS-FNRS.
⋆⋆⋆ http:// www.cims.nyu.edu/∼shakhar/

and defining the ranges as the subsets of all regions containing a given point, for every possible point. We also consider coloring problems on these kinds of range spaces. The types of problems we ask when dealing with dual range spaces are analogous to the preceding questions. For instance: *Does there exist a constant α such that given any set of disks in the plane, it is always possible to color the disks with αk colors while ensuring that any point contained in at least k disks is contained in disks of k distinct colors?*

Definitions. A *range space* (or *hypergraph*) is a pair (S, R) where S is a set (called the *ground set*) and R is a set of subsets of S. Here, we consider finite restrictions of infinite geometric range spaces of the form $\mathcal{S} = (\mathbb{R}^d, \mathcal{R})$ for $d = 2$ or 3, where $\mathcal{R}$ is an infinite family of regions of $\mathbb{R}^d$. Such a finite restriction is a range space (S, R) where the ground set S is a finite set of points in $\mathbb{R}^d$ and the set of ranges R is the collection of subsets of S defined by the intersection of S with elements of $\mathcal{R} : R = \{S \cap r : r \in \mathcal{R}\}$.

We also consider the corresponding *dual range spaces*, denoted by $\widetilde{\mathcal{S}}$, of the form $\widetilde{\mathcal{S}} = (\mathcal{R}, \{r(p) : p \in \mathbb{R}^d\})$, where $r(p) = \{r \in \mathcal{R} : p \in r\}$ is the set of regions containing the point p. The finite restrictions of these dual range spaces are of the form $(S, \{r(p) \cap S : p \in \mathbb{R}^d\})$, where $S \subset \mathcal{R}$ is finite.

A *coloring* of a range space is an assignment of colors to the elements of the ground set. A c-coloring is a coloring that uses exactly c colors. A range is k-*colorful* if it contains at least k elements of distinct color. We are interested in the following two functions, for a range space $\mathcal{S}$:

1. $c_{\mathcal{S}}(k)$ is the minimum number for which there always exists a $c_{\mathcal{S}}(k)$-coloring of any finite restriction of $\mathcal{S}$, such that every range r is $\min\{|r|, k\}$-colorful.
2. $p_{\mathcal{S}}(k)$ is the minimum number for which there always exists a k-coloring of any finite restriction of $\mathcal{S}$ such that every range of size at least $p_{\mathcal{S}}(k)$ is k-colorful.

Note that $c_{\mathcal{S}}(k)$ and $p_{\mathcal{S}}(k)$ are monotone non-decreasing functions. The goal of this paper is to provide upper bounds on $c_{\mathcal{S}}(k)$, $p_{\mathcal{S}}(k)$, $c_{\widetilde{\mathcal{S}}}(k)$, and $p_{\widetilde{\mathcal{S}}}(k)$ for various families of regions.

Previous results. The functions defined above are related to two previously studied problems. The first one is the decomposition of *f-fold coverings* in the plane: given a covering of the plane by a set of regions such that every point is covered by at least f regions, is it possible to decompose it into two disjoint coverings? This question was first asked by Pach in 1980 [6]. It is similar to deciding whether $p_{\widetilde{\mathcal{S}}}(2) \leq f$ for the dual range space $\widetilde{\mathcal{S}}$ defined by the considered family of regions, the difference being that we do not assume that all points are f-covered. This difference is important in some cases, for instance it is known that all $(d+1) \cdot f$-covers of d-space by halfspaces decompose into f covers but the proof does not directly yield a bound for $p_{\widetilde{\mathcal{S}}}(2)$. For $\mathcal{T}$ the range space defined by translates of a centrally symmetric convex polygon, Pach and Tóth [10] recently proved that $p_{\mathcal{T}}(k) = O(k^2)$ and $p_{\widetilde{\mathcal{T}}}(k) = O(k^2)$. So for these types of regions, a covering can be decomposed into k coverings if each point is covered at least

ck^2 times for some constant c. On the negative side, for the range space induced by arbitrary disks (denoted by $\mathcal{D}$), Pach, Tardos, and Tóth [9] proved that even $p_{\mathcal{D}}(2)$ is unbounded: for any constant k, there exists a set of points that cannot be 2-colored so that all open disks containing at least k points contain one point of each color. In the same paper, a similar result is obtained for $p_{\widetilde{\mathcal{A}}}(2)$ where $\mathcal{A}$ is the range space induced by the family of either strips or axis-aligned rectangles. The fact that $p_{\widetilde{\mathcal{S}}}(2)$ is unbounded implies that for every $k > 2$, $p_{\widetilde{\mathcal{S}}}(k)$ is unbounded as well, since any bound for the latter would imply a bound for the former by merging color classes. The previous impossibilities constitute our main motivation for introducing some slack and defining the problem of $c(k)$-coloring a finite range space such that ranges are k-colorful, with $k \leq c(k)$.

The second previously studied problem is that of computing the chromatic number of geometric hypergraphs, defined as the minimum number of colors needed to make all ranges polychromatic, that is, 2-colorful [12]. One of the main results of that contribution is that any dual range space induced by a finite set of pseudo-disks admits a $O(1)$-coloring that makes all ranges 2-colorful. Hence, for the family of pseudo-disks $\mathcal{P}$, $c_{\widetilde{\mathcal{P}}}(2) = O(1)$. A recent result of Chen, Pach, Szegedy and Tardos ([4], Thm. 3) implies that for any constants c, p, the following holds: there exists a point set such that for any c-coloring of its elements, we can find an axis-aligned rectangle containing at least p points, all of which have the same color. This implies that $c_{\mathcal{A}}(k)$ and $p_{\mathcal{A}}(k)$ are unbounded, where $\mathcal{A}$ is the range space induced on $\mathbb{R}^2$ by the set of all axis-aligned rectangles.

Furthermore, Pach and Tardos [8] proved that for any n, there exists a set of n axis-parallel rectangles in the plane such that one needs $\Omega(\log n)$ colors for coloring the rectangles such that no point is covered by a monochromatic set. Thus, $c_{\widetilde{\mathcal{A}}}(2) = \infty$, implying $c_{\widetilde{\mathcal{A}}}(k) = \infty$.

Our results. In Section 2, we consider the range space $\mathcal{H} = (\mathbb{R}^2, \mathcal{R})$, where $\mathcal{R}$ is the set of all halfplanes. We prove that $c_{\mathcal{H}}(k) \leq 3k - 2$, and $p_{\mathcal{H}}(k) \leq 4k - 1$. In other words, we can ensure that a halfplane contains k points of different colors in two ways: either we k-color the point set but require that the halfplane contains at least $4k - 1$ points, or we allow the point set to be $(3k - 2)$-colored.

In Section 3, we consider the range space $\mathcal{L} = (\mathbb{R}^3, \mathcal{R})$, where $\mathcal{R}$ is the set of all *lower halfspaces*. We prove that $c_{\mathcal{L}}(k) = O(k)$; and that $c_{\widetilde{\mathcal{L}}}(k) = O(k)$.

We provide a number of results on range spaces defined by disks and pseudo-disks in Section 4. For the range space $\mathcal{D}$ defined by disks, we prove that $c_{\mathcal{D}}(k) = O(k)$ by mapping disks in $\mathbb{R}^2$ to lower halfspaces in $\mathbb{R}^3$ and using the result of Section 3. For a dual range space $\widetilde{\mathcal{P}}$ defined by pseudo-disks we prove that $c_{\widetilde{\mathcal{P}}}(k) = O(k)$. Since halfplanes are a special case of pseudo-disks, we directly have $c_{\widetilde{\mathcal{H}}}(k) = O(k)$. We also show that $c_{\mathcal{P}}(k) = O(k)$, with similar arguments.

By lifting a 2D point set to the unit paraboloid $z = x^2 + y^2$ in 3D, every lower halfspace in 3D isolates a set of points which is contained in a disk in the original set of points, and thus $p_{\mathcal{L}}(k) \geq p_{\mathcal{D}}(k)$. We also prove that $p_{\widetilde{\mathcal{L}}}(k) = p_{\mathcal{L}}(k)$: coloring lower halfspaces is equivalent in the projective dual to coloring points with respect to lower halfspaces.

All the proofs are constructive, and polynomial-time algorithms can easily be derived from them. The results are summarized in the following table, where the symbol $\star$ indicates new results; and the symbol ∞ indicates a function unbounded in terms of k.

$\mathcal{S}$	$c_{\mathcal{S}}(k)$	$p_{\mathcal{S}}(k)$	$c_{\widetilde{\mathcal{S}}}(k)$	$p_{\widetilde{\mathcal{S}}}(k)$
halfplanes	$\leq 3k-2$ (Thm. 1)$\star$	$\leq 4k-1$ (Thm. 2)$\star$	$O(k)$ (Thm. 4)$\star$	$\leq 8k-3$ (Cor. 1)$\star$
lower halfspaces in $\mathbb{R}^3$	$O(k)$ (Thm. 3)$\star$	∞ (Implied by disks)	$O(k)$ (Cor. 2)$\star$	∞ (Implied by disks)
translates of a cent. sym. convex polygon	$O(k)$ (Thm. 5)$\star$	$O(k^2)$ [10]	$O(k)$ (Thm. 4)$\star$	$O(k^2)$ [10]
axis-aligned rectangles	∞ [4]	∞ [4]	∞ [8]	∞ [9]
disks	$O(k)$ (Cor. 3, Thm. 3)$\star$	∞ (open disks [9])	$\leq 24k+1$ (Rem. 1)$\star$	
pseudo-disks	$O(k)$ (Thm. 5)$\star$	∞ (open disks [9])	$O(k)$ (Thm. 4)$\star$	

Application to Sensor Networks. Let $\mathcal{R}$ be a collection of sensors, each of which monitors the area within a surrounding disk. Assume further that each sensor has a battery life of one time unit. The goal is to monitor a given planar region A for as long as possible. If we activate all sensors in $\mathcal{R}$ simultaneously, A will be monitored for only one time unit. This can be improved if $\mathcal{R}$ can be partitioned into c pairwise disjoint subsets, each of which covers A. Each subset can be used in turn, allowing us to monitor A for c units of time. Obviously if there is a point in A covered by only c sensors then we cannot partition $\mathcal{R}$ into more than c families. Therefore it makes sense to ask the following question: what is the minimum number $p(k)$ for which we know that if every point in A is covered by $p(k)$ sensors then we can partition $\mathcal{R}$ into k pairwise disjoint covering subsets? This is exactly the type of problem that we described. For more on the relation between these partitioning problems and sensor networks, see the paper of Buchsbaum *et al.* [2].

2 Halfplanes

In this section we study the case where the family $\mathcal{R}$ is the set of all halfplanes in $\mathbb{R}^2$. We denote by $\mathcal{H} = (\mathbb{R}^2, \mathcal{R})$ the corresponding infinite range space.

It is not always possible to color a set of points S with k colors such that every halfplane of size k (containing k points of S) is k-colorful, even for $k = 2$. The simplest example consists of an odd number of points in convex position. This is our main motivation for allowing either the number of colors or the range size to be greater than k.

For the proof of Theorems 1 and 2 the notion of Tukey depth is used.

Definition 1 ([15]). *Given a set S of points in $\mathbb{R}^d$, the* Tukey depth *of a point p (not necessarily in the set) is the maximum integer t with the property that every halfspace containing p contains at least t points of S.*

It is well known that for any set of n points in the plane, there exists a point in $\mathbb{R}^2$ at depth $t \geq n/3$. The *depth-k region* is the set of all points at Tukey depth k or more. It is easily seen that this region is the intersection of all halfplanes containing more than $n - k$ points of S and therefore its boundary is a convex polygon. We now turn to some useful observations regarding depth-k regions.

Lemma 1. *Let S be a finite set of more than $3k$ points in $\mathbb{R}^2$. Then every open halfplane not intersecting the depth-k region of S and the bounding line of which is tangent to the depth-k region of S contains at most $2k - 2$ points of S. The corresponding closed halfplane contains at least k points.*

Proof. Let Π be an open halfplane not intersecting the depth-k region such that its bounding line ℓ is tangent to the depth-k polygon, and let Π' be the corresponding closed halfplane. Π' contains at least k points since the point of tangency belongs to Π' and has depth k. On the other hand, ℓ contains either a side of the polygon or precisely one of its vertices, v. In the former case Π contains less than k points because its complement contains more than $n - k$ points. In the latter case, Π is contained in the union of two open halfplanes, Π_1 and Π_2; their bounding lines pass through v and its two neighbors in the polygon (respectively). Since each of Π_1 and Π_2 contains at most $k - 1$ points, Π contains at most $2k - 2$ points. □

We define the *orientation of a halfplane* as the absolute angle of the inward normal of the line bounding it. Thus, for example, the orientation of the halfplane defined by all points lying above the x-axis is $\frac{\pi}{2}$.

Let p be a point of S lying outside the depth-k region. It is easily seen that the set of orientations of all closed halfplanes that are tangent to the depth-k region and that contain p form a closed (circular) interval of length at most π. Thus, each point may be represented as an arc on the unit circle. Let $\mathcal{A}$ be the set of arcs corresponding to points in S outside or on the boundary of the depth-k region, and let $\mathcal{A}'$ be the same set of arcs but open (in particular, degenerate arcs that consisted of only one point are removed).

Lemma 2. *Every point on the unit circle is covered by at most $2k - 1$ arcs of $\mathcal{A}'$, and every point that is not the endpoint of an arc is covered by at least k arcs. Furthermore, the minimum number of segments covering any point is at most $k - 1$.*

Proof. Every point p on the unit circle represents the orientation of a closed halfplane Π' tangent to the depth-k region. Thus if p is not the endpoint of an arc, then the number of arcs that cover p is at least the number of points in Π', which is at least k by Lemma 1. As in the proof of Lemma 1, if the boundary ℓ of the halfplane contains a vertex v but no edge of the depth k region, then Π' is contained in the union of v and two open halfplanes Π_1 and Π_2 which have their bounding lines passing through v and its two neighboring edges in the polygon. Since each of Π_1 and Π_2 contains at most $k - 1$ points, and there might be a point at v, Π' contains at most $2k - 1$ points. If ℓ contains an edge of the depth k region, then all points on ℓ correspond to either empty arcs or to the endpoint of some arc. Thus the arcs that cover p correspond to points in the open halfplane Π bounded by ℓ and their number is at most $k-1$. □

Theorem 1. $c_{\mathcal{H}}(k) \le 3k-2$. *That is, we can color any set of points in the plane with* $3k-2$ *colors such that any halfplane containing* h *points is* $\min\{h,k\}$*-colorful.*

Proof. A *proper coloring* of a set of arcs on the unit circle is an assignment of colors to the arcs such that no pair of arcs of the same color overlap. In [14] it was proved that every set of arcs on the unit circle has a proper coloring with $m+M$ colors, where m (resp. M) is the minimum (resp. maximum) number of arcs covering each point of the circle. Combining this with Lemma 2, we conclude that the corresponding set $\mathcal{A}'$ can be $(3k-2)$-colored. Accordingly we can color the points (outside the depth-k region) of S that correspond to $\mathcal{A}'$. The remaining points are colored arbitrarily. Thus there exists a $(3k-2)$-coloring of S such that every open halfplane not intersecting – but tangent to – the depth-k region is colorful (the colors of points inside that halfplane are pairwise distinct).

Now it remains to prove that every halfplane of size h is $\min\{h,k\}$-colorful. Given such a halfplane Π, there are two cases: (i) Π does not intersect the depth-k region, meaning that it is strictly contained in an open halfplane Π' which has its boundary line tangent to the depth-k region, and thus no two points in it are colored with the same color. (ii) Π intersects the depth-k-region and thus contains a closed halfplane Π' tangent to it. If the point p on the circle corresponding to Π' is not the endpoint of an arc, then Π' contains at least k points of different colors. If p is the endpoint of an arc then Π' contains at least all points corresponding to arcs that cover a point infinitesimally to the left of p, which also have at least k different colors. □

We now consider the depth-$2k$ region. As described in the preceding, points outside the depth-$2k$ region are associated with a set of closed arcs, $\mathcal{A}$, on the unit circle. Recall that each arc in $\mathcal{A}$ has length at most π and that by Lemma 1 every point on the unit circle is covered by at least $2k$ arcs.

Lemma 3. *Let* $\mathcal{A}$ *be a set of arcs of length at most* π *on the unit circle. If every point on the circle is covered at least* $2k$ *times then* $\mathcal{A}$ *has a* k*-colorful* k*-coloring.*

Proof. As Pach noticed [7], a $2k$-covering of the unit circle with arcs of length at most π is decomposable into k disjoint coverings (by repeatedly removing a minimal covering of the unit circle). Thus we can assign one color to all arcs within each covering, so that each point on the circle is covered by k colors. □

Theorem 2. $p_{\mathcal{H}}(k) \le 4k-1$. *That is, we can color any set of points in the plane with* k *colors such that any halfplane containing at least* $4k-1$ *points is* k*-colorful.*

Proof. Let $\mathcal{A}$ be the set of arcs corresponding to the points that lie outside or on the boundary of the depth-$2k$ region. By Lemma 3, $\mathcal{A}$ can be made k-colorful, as it covers every point of the unit circle at least $2k$ times. This means that there exists a k-coloring of S such that every closed halfplane tangent to the depth-$2k$ region is k-colorful. As we consider large point sets in comparison to k, there always exists a depth-$2k$ region (specifically, as long as $n \geq 6k$).

Let Π be a halfplane containing at least $4k-1$ points. Π must intersect (or touch) the depth-$2k$ region, because every open halfplane tangent to the region contains at most $4k-2$ points, by Lemma 1. Thus Π contains a closed halfplane Π' with its boundary tangent to the depth-$2k$ region. By construction, Π' must be k-colorful and therefore so must Π. □

Corollary 1. $p_{\widetilde{\mathcal{H}}}(k) \leq 8k-3$. *That is, we can color any set of halfplanes with k colors such that any point in the plane covered by $8k-3$ halfplanes is contained in halfplanes of k different colors.*

Proof. If we restrict ourselves to lower halfplanes, then $p_{\widetilde{\mathcal{H}}}(k) = p_{\mathcal{H}}(k)$ by projective duality. So if we are given a set of halfplanes (lower and upper), every point which is covered $8k-3$ times is covered at least $4k-1$ times by either lower halfplanes or upper halfplanes. Thus we can color the lower and the upper halfplanes independently, using theorem 2 and obtain: $p_{\widetilde{\mathcal{H}}}(k) \leq 8k-3$. □

3 Lower Halfspaces in $\mathbb{R}^3$

Here, we deal with the case where $\mathcal{R}$ consists of all *lower halfspaces* in $\mathbb{R}^3$. We call $\mathcal{L} = (\mathbb{R}^3, \mathcal{R})$ the corresponding infinite range space and consider the value of $c_{\mathcal{L}}(k)$. The depth-k region in $\mathbb{R}^3$ is bounded by a convex polyhedron.

Lemma 4. *Given a set of more than $4k$ points in $\mathbb{R}^3$, every open halfspace not intersecting the depth-k polyhedron and which has a bounding plane tangent to the depth-k polyhedron contains at most $3k-3$ points. The corresponding closed halfspace contains at least k points.*

Proof. The proof is similar to that of Lemma 1 in $\mathbb{R}^2$. We consider open and closed halfspaces tangent to the depth-k polyhedron and note that any tangent closed halfspace contains at least k points otherwise a point of the depth-k polyhedron has depth less than k. A halfspace is either tangent at a vertex, an edge, or a face of the polyhedron; if an open halfspace is tangent at a face, it contains at most $k-1$ points; if an open halfspace is tangent at an edge (a vertex resp.) it is contained in the union of two (three resp.) open halfspaces tangent at a face of the polyhedron. □

In what follows, we consider lower halfspaces defined by planes tangent to the depth-k polyhedron. Each normal vector to one of these planes corresponds to precisely one lower halfspace and defines one point on the unit sphere. We map the points from the unit sphere onto the xy plane so that every lower halfspace corresponds to a single point in $\mathbb{R}^2$. This representation is used in the remainder of the section.

Lemma 5. *Let R_x denote the set of points in $\mathbb{R}^2$ corresponding to lower half-spaces tangent to the depth-k polyhedron and containing $x \in S$. Let p and q be two points of S outside the depth-k polyhedron. Then,*

1. *R_x is a connected subset of $\mathbb{R}^2$.*
2. *The boundaries of R_p and R_q intersect at most twice.*

Proof. The first property follows directly from the convexity of the depth-k polyhedron. Given a point x outside the depth-k region, any convex combination of the normal vectors of all planes tangent to the polyhedron and incident to x define a halfspace containing x.

To prove that the boundaries of R_p and R_q intersect at most twice, we look at all planes tangent to the polyhedron, and incident to p and q. These map to points that are on the boundary of both R_p and R_q. As p and q are distinct they define a line. Through this line, there exist at most two planes tangent to the depth-k polyhedron. □

The proof of the next theorem uses the following definition and lemma [5]. We use the standard notion of chromatic number $\chi(G)$ of a graph G, defined as the minimum number of colors needed to color the vertices so that no edge is monochromatic.

Definition 2. *A simple graph $G = (V, E)$ is called k-degenerate for some positive integer k, if every (vertex-induced) subgraph of G has a vertex of degree at most k.*

Lemma 6. *Let $G = (V, E)$ be a k-degenerate graph. Then $\chi(G) \leq k + 1$.*

Proof. Proceed by induction on $n = |V|$. Let $v \in V$ be a vertex of degree at most k. By the induction hypothesis, the graph $G \setminus v$ (obtained by removing v and all of its incident edges from G) is $(k + 1)$-colorable. Since v has at most k neighbors there is always a color that can be assigned to v, and that is distinct from the colors of its neighbors. □

Theorem 3. *$c_{\mathcal{L}}(k) = O(k)$. That is, we can color any set of points in $\mathbb{R}^3$ with $O(k)$ colors such that any lower halfspace containing h points is* $\min\{h, k\}$*-colorful.*

Proof. Let $\mathcal{A} = \{R_x | x \in S$, outside or on the surface of the depth-k polyhedron$\}$. By Lemma 5, we know that $\mathcal{A}$ is a set of pseudo-disks. Let $\mathcal{A}'$ be the corresponding open pseudo-disks. By Lemma 4, we also know that every point in the projection of the sphere on $\mathbb{R}^2$ belongs to at most $3k - 2$ regions of $\mathcal{A}'$.

By a lemma of Sharir [11], the complexity of an arrangement of the set of bounding curves of n pseudo-disks such that any point belongs to the interior of at most i of the pseudo-disks is $O(ni)$. Thus the complexity of the bounding curves in $\mathcal{A}'$ is $O(nk)$. Now consider the intersection graph of $\mathcal{A}'$. This graph is $O(k)$-degenerate. To see this, consider a pair of intersecting regions $r_1, r_2 \in \mathcal{A}'$. Either the boundaries of r_1 and r_2 intersect (at some vertex) in which case we know that there are $O(nk)$ such vertices, or one of the regions, say r_1, is contained in r_2. However, since every point belongs to at most $3k-2$ regions, every region is contained in at most $3k-3$ other regions, hence the total number of such pairs of regions is at most $O(nk)$. Thus the number of edges in the intersection graph is $O(nk)$. This is true for every induced subgraph and hence by Lemma 6, this graph is $O(k)$-colorable. A similar observation was made by Chan [3].

Now it remains to prove that every halfspace of size $h \geq k$ is k-colorful. Given such a halfspace, there are two possibilities. Either the halfspace does not intersect the depth-k polyhedron, meaning that it is strictly contained in an open halfspace tangent to the polyhedron, and thus every point it contains has a unique color; or the halfspace intersects the polyhedron and thus contains a closed halfspace tangent to it, meaning that it contains at least k different colors. □

Corollary 2. $c_{\widetilde{\mathcal{L}}}(k) = O(k)$. *That is, we can color any set of lower halfspaces in* $\mathbb{R}^3$ *with* $O(k)$ *colors so that any point in the intersection of more than* k *of them is covered by* k *different colors.*

Proof. Given a set of halfspaces in $\mathbb{R}^3$, we consider their bounding planes. By projective duality, a set of planes can be mapped to a set of points, such that a point is above k planes if and only if in the projective dual a plane is above k points. In other words, by applying Theorem 3 in the dual, we derive a coloring for the halfspaces in the primal, which is correct as the inclusion relation (above-below) is preserved by projective duality: every lower halfspace containing at least k points in the primal is a point covered by k halfspaces in the dual. □

4 Disks and Pseudo-disks

In this section we consider the case where the ranges in $\mathcal{R}$ are disks or pseudo-disks. We denote by $\mathcal{D} = (\mathbb{R}^2, \mathcal{R})$ the range space for disks, and by $\widetilde{\mathcal{D}}$ its dual, where the ground set is the set of disks and the ranges are the subsets of all disks having a common point. Similarly, we use the notations $\mathcal{P}$ and $\widetilde{\mathcal{P}}$ for the range spaces defined by pseudo-disks.

The proof given above for lower halfspaces in $\mathbb{R}^3$ can be used to prove that $c_{\mathcal{D}}(k) = O(k)$. This is seen by a standard lifting transformation of disks and points in the plane, to points and halfspaces in $\mathbb{R}^3$ that preserves the incidence relations.

Corollary 3. $c_{\mathcal{D}}(k) = O(k)$.

Proof. Given a set S of points in $\mathbb{R}^2$, we proceed by lifting the points onto the parabola of equation $z = x^2 + y^2$ in $\mathbb{R}^3$. It is known that any disk in $\mathbb{R}^2$ is the projection onto the plane xy of the intersection between the parabola and a lower halfspace in $\mathbb{R}^3$. The result follows by applying Theorem 3 to this set. □

In the following, we give a bound for the value of $c_{\widetilde{\mathcal{P}}}(k)$, where $\widetilde{\mathcal{P}}$ is the dual range space defined by pseudo-disks. Similar to the proof of Theorem 3, we analyze the degeneracy of a graph induced by a finite set of regions.

Definition 3. *Let S be a finite family of simple closed Jordan regions in $\mathbb{R}^2$. We denote by $G_k(S)$ the graph on S where the edges are all pairs $r, s \in S$ such that there exists a point p that belongs to $r \cap s$ and at most k other regions of S.*

Lemma 7. *Let S be a family of pseudo-disks. Then $G_k(S)$ is $O(k)$-degenerate and hence the chromatic number of $G_k(S)$ is at most $O(k)$.*

We aim to show that the number of edges in any (vertex-induced) subgraph of G with m vertices is at most $O(km)$, and therefore, the average degree in any induced subgraph is at most $O(k)$. Thus, there must exist a vertex of degree at most $O(k)$ in any induced subgraph. Hence, $G_k(S)$ is $O(k)$-degenerate and by Lemma 6 it is $O(k)$-colorable as asserted. We need the following lemmas.

Lemma 8. *There exists a constant c such that for any set S of n pseudo-disks, $G_0(S)$ has at most cn edges.*

Proof. See for instance the proof of Lemma 5.1 in [12]. □

Lemma 9. *Let S be a family of n pseudo-disks and let $G = (S, E)$ be a subgraph of the intersection graph of S (thus E is a subset of the set of all pairs of regions from S that have a non-empty intersection). For each edge $e = (a, b) \in E$ choose a point $p_e \in a \cap b$ that belongs to the intersection of a and b. Let X be the set of all pairs (e, r) such that $e \in E$ and $r \in S \setminus \{a, b\}$ contains the point p_e chosen for the edge e. Suppose that $|E| > 4cn$ where c is the constant from Lemma 8. Then $|X| \geq \frac{|E|^2}{4cn}$*

Proof. The proof proceeds in two steps. In the first step, we prove the following bootstrapping inequality: $|X| \geq |E| - cn$. In the second step we use a random sampling argument similar to the one used for the Crossing Lemma (see [1]).

The proof of the first step proceeds by induction on $|E| - cn$. For the case $|E| - cn \leq 0$ the claim is trivial. Assume that the claim holds for some positive integer k (namely, for $|E|$ and n satisfying $|E| - cn = k$). Suppose that $|E| - cn = k+1$. Since $|E| > cn$, Lemma 8 implies that there must exist a region $r \in S$, and an edge $e \in E$ which generates at least one configuration $(e, r) \in X$ (namely, that point p_e belongs to r, for otherwise X is empty, meaning that there is no edge of $G_k(S)$ for any $k > 0$; thus the graph is a subgraph of $G_0(S)$ and the number of edges in E by Lemma 8 is at most cn). After removing e from E we are left with $|E| - 1$ edges, n regions, and a set X' of configurations, where $|X| \geq |X'| + 1$. We have $|E| - 1 - cn = k$, so we can apply the induction hypothesis to obtain $|X'| \geq |E| - 1 - cn$. Thus $|X| \geq |X'| + 1 \geq |E| - cn$. This completes the proof of the first step.

Let X denote the set of configurations, as above. We take a random sample S' of the regions in S by choosing each region independently with some fixed probability p (to be determined later on). Let E' denote the subset of edges in E, for which all defining regions are in S'. Let $n' = |S'|$; $m' = |E'|$, and let $X' \subset X$ denote the subset of configurations in X for which all the defining regions a, b and r are in S'. By the above bootstrapping inequality, we have $|X'| \geq m' - cn'$. Note that $|X'|$, m' and n' are random variables, so the above inequality holds for their expectations as well. Hence, using linearity of expectation, $\mathbf{E}[|X'|] \geq \mathbf{E}[m'] - c\,\mathbf{E}[n']$. It is easily seen that $\mathbf{E}[n'] = pn$. We have $\mathbf{E}[m'] = p^2\,|E|$ and $\mathbf{E}[|X'|] = p^3\,|X|$. Indeed, the probability that a given edge $e \in E$ belongs to E' is the probability that the two regions defining e are chosen in S', which is p^2 for any fixed $e \in E$. Similarly, the probability that a configuration of a region $r \in S$ that contains a point p_e is counted in X' is p^3. Substituting these values in the above inequality, we get $p^3\,|X| \geq p^2\,|E| - cpn$, or $|X| \geq \frac{|E|}{p} - \frac{cn}{p^2}$. This inequality holds for any $0 < p \leq 1$, and we choose $p = 2cn/\,|E|$ (by assumption, $p \leq 1$) to obtain $|X| \geq |E|^2/4cn$. This completes the proof of the lemma. □

Proof of Lemma 7: Let X denote the set of configurations as above when E is the set of edges of $G_k(S)$ and for each edge $e \in E$, p_e is the point witnessing that $e \in E$ (i.e., p_e is a point that belongs to the regions defining e and at most k other regions of S). By Lemma 9 we have: $|X| \geq |E|^2/4cn$.

On the other hand, note that by definition of $G_k(S)$ any point p_e can belong to at most k regions of S so obviously $|X| \leq k\,|E|$.

Combining the two bounds we have: $|E| \leq 4ckn$. Thus the sum of degrees of vertices in the graph $G_k(S)$ is at most $8ckn$, so the average degree is at most $8ck$. Thus there always exists a vertex with degree at most $8ck$, hence $G_k(S)$ is $8ck$-degenerate. This completes the proof of the lemma. □

Theorem 4. $c_{\widetilde{\mathcal{P}}}(k) = O(k)$

Proof. We know by Lemma 7 that there exists a constant c such that $G_k(S)$ is ck-degenerate. We show that we can color the pseudo-disks in S with $ck + 1$ color such that for any point p with depth $d(p)$, the set of disks E_p containing p is $\min\{d(p), k\}$-colorful. We use $ck + 1$ colors to color pseudo-disks inductively. The proof is by induction on $|S| = n$. Let $r \in S$ be a region for which the degree in $G_k(S)$ is at most ck. By Lemma 7, there exists such a region. The induction hypothesis is that $S \setminus \{r\}$ admits a valid coloring. To complete the inductive step, we must assign a color to r so that the new coloring is still valid. Note that by the inductive hypothesis, points that belong to r and at least k other regions are already contained in some k regions (in $S \setminus \{r\}$), all colors of which are distinct. Hence, the color of r will not affect the validity for those points. We may only run into trouble for those points $p \in r$ that are contained in at most i (for $i \leq k - 1$) other regions. However, note that any region containing p is a neighbor of r in $G_k(S)$ by definition. Note also that by the induction hypothesis, all regions containing such a point p get distinct colors. Moreover, since the number of neighbors of r in $G_k(S)$ is at most ck we can color r with a color distinct from all its neighbors in $G_k(S)$. Thus for any point in r that belongs to exactly i (for $i \leq k - 1$) other regions, all regions covering this point including r will have distinct color. This completes the inductive step and hence the proof of the theorem. □

Remark 1. For the special case of real disks, it can be shown that the constant in Lemma 8 is $c = 3$ (we omit the details here). Thus by Lemma 7, the graph $G_k(S)$ is $24k$-degenerate. Hence in the special case of real disks, we have that $c_{\widetilde{\mathcal{D}}}(k) \leq 24k + 1$.

For the version in the primal range space in which we color points rather than regions, we can also prove the following:

Theorem 5. $c_{\mathcal{P}}(k) = O(k)$

Proof. The proof is very similar to the proof of Theorem 4 and uses the same ingredients. The analog of Lemma 8 is provided in [13]. □

Acknowledgements. The authors thank Janos Pach, as well as the participants to the algorithmic lunches at the ULB CS Department, for helpful discussions.

References

1. Aigner, M., Ziegler, G.M.: Proofs from the book. Springer, Heidelberg (1998)
2. Buchsbaum, A., Efrat, A., Jain, S., Venkatasubramanian, S., Yi, K.: Restricted strip covering and the sensor cover problem. In: ACM-SIAM Symposium on Discrete Algorithms (SODA 2007) (2007)
3. Chan, T.M.: Low-dimensional linear programming with violations. SIAM Journal on Computing 34(4), 879–893 (2005)
4. Chen, X., Pach, J., Szegedy, M., Tardos, G.: Delaunay graphs of point sets in the plane with respect to axis-parallel rectangles (manuscript, 2006)
5. Lick, D.R., White, A.T.: k-degenerate graphs. Canadian Journal on Mathematics 12, 1082–1096 (1970)
6. Pach, J.: Decomposition of multiple packing and covering. In: 2. Kolloq. über Diskrete Geom., Inst. Math. Univ. Salzburg, pp. 169–178 (1980)
7. Pach, J.: Personal communication (2007)
8. Pach, J., Tardos, G.: Personal communication (2006)
9. Pach, J., Tardos, G., Tóth, G.: Indecomposable coverings. In: Akiyama, J., Chen, W.Y.C., Kano, M., Li, X., Yu, Q. (eds.) CJCDGCGT 2005. LNCS, vol. 4381, pp. 135–148. Springer, Heidelberg (2007)
10. Pach, J., Tóth, G.: Decomposition of multiple coverings into many parts. In: Proc. of the 23rd ACM Symposium on Computational Geometry, pp. 133–137 (2007)
11. Sharir, M.: On k-sets in arrangement of curves and surfaces. Discrete & Computational Geometry 6, 593–613 (1991)
12. Smorodinsky, S.: On the chromatic number of some geometric hypergraphs. SIAM Journal on Discrete Mathematics (to appear)
13. Smorodinsky, S., Sharir, M.: Selecting points that are heavily covered by pseudo-circles, spheres or rectangles. Combinatorics, Probability and Computing 13(3), 389–411 (2004)
14. Tucker, A.: Coloring a family of circular arcs. SIAM Journal of Applied Mathematics 229(3), 493–502 (1975)
15. Tukey, J.: Mathematics and the picturing of data. In: Proceedings of the International Congress of Mathematicians, vol. 2, pp. 523–531 (1975)

Local Algorithms for Dominating and Connected Dominating Sets of Unit Disk Graphs with Location Aware Nodes

J. Czyzowicz[1], S. Dobrev[2], T. Fevens[3], H. González-Aguilar[4], E. Kranakis[5], J. Opatrny[3], and J. Urrutia[6]

[1] Départ. d'informatique, Univ. du Québec en Outaouais, Gatineau, QC, Canada
[2] Slovak Academy of Sciences, Bratislava, Slovakia
[3] Department of CSE, Concordia University, Montréal, QC, Canada
[4] Centro de Investigacion en Matematicas, Guanajuato, Gto., Mexico
[5] School of Computer Science, Carleton University, Ottawa, ON, Canada
[6] Instituto de Matemáticas, Área de la investigación cientifica, D.F. México, México

Abstract. Many protocols in ad-hoc networks use dominating and connected dominating sets, for example for broadcasting and routing. For large ad hoc networks the construction of such sets should be *local* in the sense that each node of the network should make decisions based only on the information obtained from nodes located a constant number of hops from it. In this paper we use the *location awareness* of the network, i.e. the knowledge of position of nodes in the plane to provide local, constant approximation, deterministic algorithms for the construction of dominating and connected dominating sets of a Unit Disk Graph (UDG). The size of the constructed set, in the case of the dominating set, is shown to be 5 times the optimal, while for the connected dominating set $7.453 + \epsilon$ the optimal, for any arbitrarily small $\epsilon > 0$. These are to our knowledge the first *local* algorithms whose time complexities and approximation bounds are independent of the size of the network.

Keywords: Approximation factor, Dominating set, Connected dominating set, Local algorithm, Location awareness, Unit disk graph.

1 Introduction

Many of the existing networks are large and complex. The number of connections (links) between nodes often remains relatively small, each node being capable to communicate, on average, with a bounded number of neighbors. Distributed algorithms implemented over such networks must often achieve some *global* computational tasks, like computing good approximations of dominating and independent sets, vertex and edge colorings, spanners, etc., despite the fact that each node is confined to *local* communication. Consequently, in the last twenty years, local algorithms have been investigated by several researchers in distributed computing. For example, in a k-local algorithm, for a given parameter k, a node is allowed to communicate at most k times with its neighbors. Work on this model includes Luby's randomized independent set algorithm [23], sparse partitions introduced in [3], and particularly the seminal work of Linial [22].

A distributed algorithm is called *local* if each node of the network makes decisions based on the information obtained uniquely from the nodes located no more than a constant (independent of the size of the network) number of hops from it. Thus, during the algorithm, no node is ever aware of the existence of the parts of the network further away than this constant number of hops. There are several reasons why such local algorithms are practical for networks:

1. A solution is consistent regardless of the order in which the nodes or edges are considered in the calculations.
2. Changes in the network outside of a fixed-size neighborhood do not influence the computation of in a node. Moreover, a change in the network requires solely a local recalculation of the solution.
3. It is possible to calculate only a part of the required solution as needed by a subnetwork.
4. Messages do not propagate indefinitely throughout the network and the algorithm terminates in a constant number of steps.

Wireless ad hoc and sensor networks are most often modeled by *Unit Disk Graphs* (abbreviated by UDGs). Nodes of a UDG are located on a plane and two nodes are considered adjacent when their distance is at most equal to some given constant. Hence it is assumed that the wireless nodes have equal transmission range c. Most NP-hard graph theory problems remain NP-hard when restricted to the class of UDGs. However, it turns out that for the class of UDGs it is possible to design algorithms offering better approximative solutions.

Many graph-theoretic problems do not admit local algorithms solving them, even if restricted to the class of UDGs. Conversely, it turns out, that several of these problems become solvable in the local setting when the network is *location aware*, i.e. when each node knows its *geographic position* (e.g. Cartesian coordinates). Furthermore algorithms for location aware networks are sometimes easier to design and they may lead to better time complexities and/or approximation bounds. With the advent of Global Positioning System (GPS) the assumption of location awareness seems relevant.

The recent survey of open problems in [2] lists the problem of local computation of dominating sets for UDGs as one of the important open problems in distributed computing. In this paper we consider the problems of minimum dominating set and minimum connected dominating set for a location aware network, represented by a UDG graph. We design local algorithms providing constant approximation solutions to both problems. To the best of our knowledge this is the first solution when both of these parameters, i.e. time complexity and approximation bounds are independent on the size of the network.

1.1 Related Work

The dominating set and connected dominating set problems are known to be NP-hard, even when restricted to the class of UDGs, see [7]. The importance of these problems have motivated researchers to investigate approximation schemes. For general graphs there exists an $O(\log n)$ approximation algorithm for the minimum dominating set problem [15], and it is known that no polynomial-time approximation of $o(\log n)$ exists unless every problem in NP can be solved deterministically in $O(n^{poly \log n})$ time [24]. For general graphs it is also known that, unless $P = NP$, there is an $\epsilon > 0$ such that the minimum independent dominating set cannot be approximated within a factor of $O(n^{\epsilon})$, [12].

The situation is quite different in the case of UDGs. Despite the fact that the dominating set and connected dominating set problems for the class of unit disk graphs remain NP-hard, constant approximation is possible (e.g. [25], [1]), even polynomial-time approximation schemes (PTAS) are known for this case, e.g., [6]. The first such solution has been proposed by [11], where the geometric representation of the UDG was supposed to be part of the input. Additionally, [27] have given a PTAS for the minimum dominating set problem of a UDG for which the geometric representation was not given.

The approximation bounds, which appear to be better when restricted to the class of UDGs apply mainly to the centralized setting. In the distributed scenario, however, especially if the geometric information of the input UDG is not given, the best approximation of the minimum dominating set problem is often not better than one obtained for the case of general graphs. The first algorithm achieving a nontrivial approximation ratio $o(\Delta)$, for Δ being the maximum node degree, in a nontrivial time $o(diam(G))$ was developed in [14]. Kuhn et al. [16] proposed a distributed approximation based on LP relaxation techniques. In[19] they designed a PTAS for the minimum dominating set for the class of graphs of *polynomially bounded growth*. In such graphs in the k-neighborhood of any node the size of an independent set is bound by a polynomial function of k and they include the class of UDGs.

Since the pioneering work of Linial [22] on locality in distributed computing many papers on local algorithms have been published. However, despite the fact that several lower bounds and impossibility results in distributed computing are now known (e.g. [9]), most of them apply to the computational models which do not involve locality. The only nontrivial lower bound in local distributed computing known to the authors of this paper has been $\Omega(\log^* n)$ time for 3-coloring of the ring by Linial. On the other hand, it was shown in [26] that there are nontrivial *Locally Checkable Labeling* (LCL) problems having local, i.e. constant-time solutions. Peleg discussed several problems in distributed computing in the context of a *locality-sensitive approach* [28]. Wang et al. [29] proposed a local algorithm that constructs a bounded degree and planar spanner for UDGs.

For the class of general graphs Kuhn et al. [17] have given approximation lower bounds for covering problems as a function of the size of the neighborhood through which each message may be propagated. In the case of UDGs Kuhn et al. [20] have given local approximation algorithms for the class of covering and packing linear programs. The recent paper of Kuhn et al. [21] offers the best solution for the dominating set problem for the class of UDGs.

The local algorithms mentioned above are such that, either the approximation bounds proposed, or the worst case time bounds obtained depend on the size n of the network. However, these algorithms do not use the geographic information. In this paper we prove that these bounds are not always valid when the location of nodes is allowed to be part of the input. Since finding a geographic representation of a given UDG graph (cf. [4]) or even its approximation (cf. [18]) is known to be NP-hard, it seems that the locality is a powerful information and using it may result in better algorithmic bounds. For example, for a different problem of broadcasting in the *geometric radio networks* [8] have shown an $O(n)$ time algorithm using the geographic information and $\Omega(n \log n)$ time lower bound when the geographic information was not available. In this paper we present two local, constant time distributed algorithms producing constant approximations of minimum dominating sets and minimum connected dominating sets, respectively.

1.2 Preliminaries and Results of the Paper

Consider a graph $G(V, E)$ with vertex set V and edge set E. A subset S of V is called *dominating set* if every vertex of G is either in S or adjacent to a vertex in S. A dominating set S is called a *connected dominating set* if the subgraph of G induced by S is connected. S is an *independent set* if there is no edge of G between any two elements of S.

We assume that a wireless network consists of nodes that have the same *circular* transmission range of size 1. Thus, it can be represented by a UDG with an edge connecting two nodes when they are at most a unit distance from each other. We assume that the network is *location aware*, i.e. each node of the graph knows its geometric position in the plane.

We suppose that at each time unit a vertex may send a message to each of its neighbors or receive a message from its neighbour. Notice that, due to the local nature of our algorithms, nodes do not need to send their full coordinates, but only last k digits of them for some constant k.

In Sect. 2 we give an algorithm for the construction of a dominating set of a unit disk graph whose competitive ratio is 5 (Theorem 1). Section 3 gives an algorithm for the construction of a connected dominating set of a unit disk graph. This algorithm has a competitive ratio $7.453 + \epsilon$, for any $\epsilon > 0$ (Theorem 2). Both algorithms are of constant time complexity (the constant is a function of the degree of the network) and independent of the size of the network.

Because of the page limit, most proofs have been removed.

2 Local Algorithm for Dominating Set of a UDG

As previously noted, given a UDG a local algorithm decides to include a node into a dominating set using a fixed size neighborhood independently of decisions possibly taken at other nodes. Thus the algorithm could actually construct a very large dominating set due to symmetries that could be present in the graph. We therefore need to make sure that potential symmetries in the graph are broken in some way. Using the coordinates of nodes we associate with each node a class number that depends on the position of the node within a regular tiling of the plane. As depicted in Fig. 1, in the tiling used by our algorithms each tile consists of 12 hexagons of unit diameter and each hexagon of the tile represents a single class. In a hexagon we assume that its right-hand side boundary, starting from the top apex of the hexagon up to the bottom apex of the hexagon belongs to the hexagon; thus, only the top apex and the two right upper apexes are considered to belong to the hexagon (see Fig. 1). Note, that the scattered class numbering of the tile does not improve the worst-case time complexity of the subsequent algorithms, but it may enhance their performance in practice.

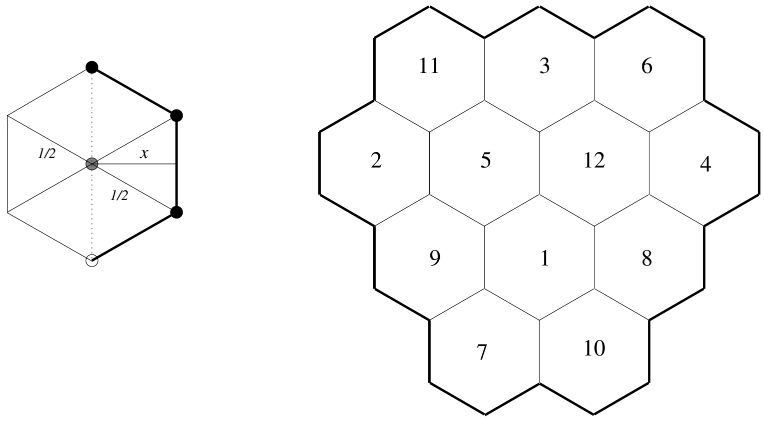


Fig. 1. Left: the dimensions of the hexagon, "bold" lines show the boundary belonging to the hexagon. Right: a tile divided into 12 hexagons of diameter 1 and the class numbering of the hexagons.

We assume that the tiling starts by placing one tile with the center of the hexagon of class number 1 in coordinates $(0, 0)$, while other tiles are placed so that the hexagon of Class 3 is made adjacent to hexagons of classes 7 and 10 or the hexagon of Class 11 is made adjacent to hexagons of Classes 8 and 4, etc. Figure 2 depicts the tiling of the plane that is used in our algorithm.

The following Lemma 1 provides an important separation property of the hexagons of the tiling that is useful in the sequel.

Lemma 1. *In the tiling of the plane given above, any two points of the plane that are of the same class, but belonging to two different hexagons, are at Euclidean distance greater than 2. Moreover, given the coordinates of a point P in the plane, one can determine the class number of P using a constant number of basic arithmetic and* mod *operations.*

As a consequence of Lemma 1, if each node is aware of the tiling being used, then any node can calculate from its own coordinates its class number. In the

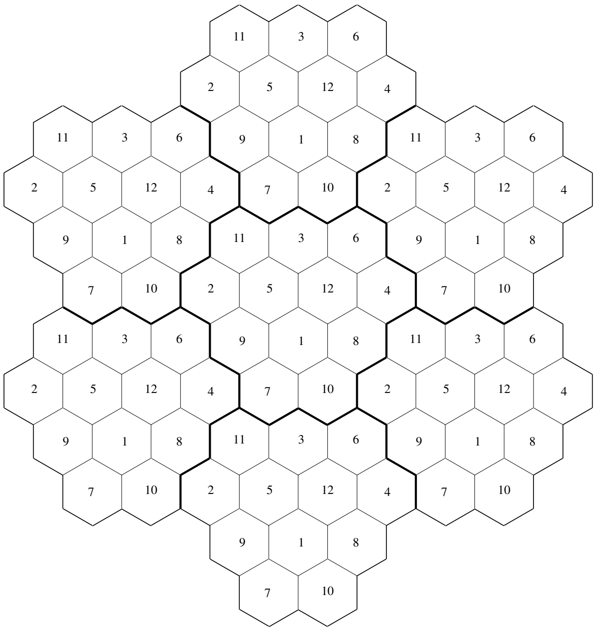


Fig. 2. Tiling of the plane with tiles consisting of 12 hexagons each

following discussions, this knowledge of the tiling as well as class numbering being used is assumed to be available to each node. Also note that this is a constant size information.

2.1 Construction of the Dominating Set

The main idea of our algorithm for computing locally a dominating set is as follows. Nodes determine their class number and acquire the class numbers of all their neighbors. In each hexagon, a dominator, if any, is the node closest to the center of the hexagon not dominated by any node of lower class. More precisely, to calculate a dominating set of a unit disk graph G, each node of G executes Algorithm 1. Let $\mathcal{D}$ be the set of all nodes of G designated by Algorithm 1 as dominators. In the next few lemmas we discuss some properties of Algorithm 1 and of the associated dominating set $\mathcal{D}$. We conclude with Theorem 1.

Lemma 2. *The selection of a dominator by Algorithm 1 in a hexagon of Class i depends only on information received from nodes that are at most $i-1$ hops away from nodes in the given hexagon.*

Lemma 3. *Every vertex of G is either in $\mathcal{D}$ or adjacent to a vertex in $\mathcal{D}$. Thus, set $\mathcal{D}$ is a dominating set of G.*

Lemma 4. *The Euclidean distance between any two nodes of $\mathcal{D}$ is more than one. Thus $\mathcal{D}$ is an independent set of G.*

Algorithm 1. Local Dominating Set Algorithm

```
   // Execution starts by a node either when a node needs to find its dominator,
   // or if it receives a request to find a dominator in its hexagon.
1: Determine your class number using your coordinates and the tiling information.
2: Find all your neighbors and obtain their coordinates and class numbers.
3: If your class number is 1 then the node $N$ in your hexagon closest to the center of
       it is designated as a dominator. Continue to Step 6.
4: Find whether there is a node in your hexagon that has no neighbor of lower class.
       If such nodes exist then the one of them that is closest to the center of the
       hexagon is designated as a dominator. Continue to Step 6.
5: If you have a neighbor $M$ of a lower class number then send to $M$ a request
       to execute the algorithm for finding its dominator. Once the replies from all
       neighbors of lower class number are received, determine if you are already
       dominated. Inform your neighbors in your hexagon of the result. When all
       nodes of your hexagon finish this calculation, node $N$ in your hexagon closest
       to the center and not dominated yet is designated as a dominator, if such a
       node exists.
6: Inform all your neighbors that a dominator selection in your hexagon is completed
       and give them its result.
7: Terminate your execution of the algorithm.
```

We now summarize the properties of the dominating set calculated by Algorithm 1 in the following theorem.

Theorem 1. *Let G be a unit disk graph and $\mathcal{D}$ be the set of dominators calculated by Algorithm 1. $\mathcal{D}$ is a dominating, independent set of G and for any dominating set $\mathcal{D}^*$ of G, we have $|D|/|D^*| \leq 5$. Thus the competitive ratio of Algorithm 1 is 5.*

Proof. According to Lemmas 3 and 4, $\mathcal{D}$ is both a dominating and an independent set of G. To show that $|D|/|D^*| \leq 5$ is done as in [25]. □

Figure 3 gives an example of a unit disk graph and its placement for which the ratio between the minimum dominating set and the dominating set $\mathcal{D}$ calculated by Algorithm 1 is equal to 5. The dominating set $\mathcal{D}$ computed by the algorithm consists of vertices of degree 1 or 2 while the optimum one consists of the vertices of degree 6. Thus the competitive ratio in Theorem 1 cannot be improved.

3 Local Algorithm for Connected Dominating Set of a UDG

Our construction of a connected dominating set starts with the dominating set constructed by the algorithm of Sect. 2 and we make it connected by adding selected vertices, called *bridges*. We first need to introduce some notation and study properties of optimal connected dominating sets.

Let OCDS denote an *optimal connected dominating set*, i.e. the minimal subset of vertices which induces a connected subgraph of a given unit disk graph G

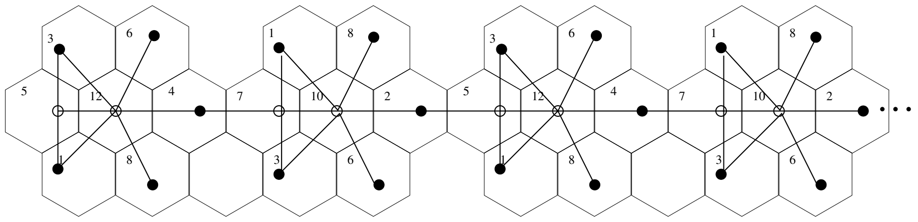


Fig. 3. Graph realizing competitive ratio 5

and such that any vertex of G has a neighbor in OCDS. Consider a Minimum Spanning Tree T of OCDS, i.e. a connected subgraph of G containing the OCDS and such that it has the smallest possible sum of lengths of its edges. Note that the minimal angle between any two incident edges (x, y) and (x, z) of T is $\pi/3$, otherwise edge (y, z) would be used in T instead of (x, y) or (x, z). Suppose that T is rooted at some vertex r, of degree smaller than six. We denote by D_v the disk of radius 1 centered at v. Consider the dominating set $\mathcal{D}$ constructed in Sect. 2. We will find an upper bound on the number of vertices of $\mathcal{D}$. For this purpose we will charge to vertices of T small subsets of $\mathcal{D}$ as follows. Suppose $(u, v) \in T$, i.e. u is the parent of v in T. We charge to v all vertices centered inside D_v except those which are inside D_u. Hence the number d_v of vertices charged to v is $d_v = |\mathcal{D} \cap (D_v \setminus D_u)|$. Note that, since T contains a dominating set of the UDG, each vertex of $\mathcal{D}$ is charged to at least one vertex of T.

Furthermore, let E_v be the set of children of v, such that there is no vertex of $\mathcal{D}$ reachable from both v and a vertex of E_v, i.e., $E_v = |\{w : w \text{ is a child of } v \text{ in } T \text{ and } \mathcal{D} \cap (D_v \cap D_w) = \emptyset\}|$. Denote $e_v = |E_v|$.

Lemma 5. *Let T be a Minimum Spanning Tree with root r of an optimal connected dominating set of UDG G with d_v, e_v defined for any vertex v of T as above.*

1. *If v is different from r then $d_v + e_v \leq 4$.*
2. *$d_r + e_r \leq 5$.*

Corollary 1. *Let f be the number of edges $\{u, v\}$ in T such that $\mathcal{D} \cap D_u \cap D_v \neq \emptyset$. Then $|\mathcal{D}| \leq 3|OCDS| + 2 + f$.*

Proof. Indeed, from the definition of f and e_v we derive that
$|OCDS| - 1 - f = \sum_{v \in OCDS} e_v$. Summing over all $v \in OCDS$ we get
$|\mathcal{D}| = \sum_{v \in OCDS} d_v = d_r + \sum_{v \in OCDS \setminus \{r\}} d_v \leq 1 + \sum_{v \in OCDS} (4 - e_v) =$
$4 \cdot |OCDS| + 1 - \sum_{v \in OCDS} e_v$. Now observe that the term in right-hand side is equal to $4 \cdot |OCDS| + 1 - (|OCDS| - 1 - f) = 3 \cdot |OCDS| + 2 + f$. □

Note that the competitive ratio of $\mathcal{D}$ with respect to the OCDS is better than the one from Theorem 1 since f can be at most $|OCDS| - 1$.

Just like the dominating set previously considered in Sect. 2, we consider a tiling of the plane with tiles. Each tile consists of c hexagons of radius 1 that are being assigned different class numbers and such that hexagons of the same class number are at distance at least k from each other. The tiles we consider in this section achieve distances k larger than 2 and it can be easily proved that in this case the required class number c is in $\Theta(k^2)$.

Intuitively, the algorithm for constructing a connected dominating set is as follows. Find a dominating set using Algorithm 1 and select a coordinator vertex in each non-empty hexagon using some election procedure. The coordinators are responsible for augmenting the existing dominating set by adding bridges, each bridge (one or two vertices) joining at least two connected components. We suppose that each vertex will communicate information at distance less than k hops from it. Each hexagon is assigned a class number between 1 to c so that vertices of two hexagons of the same class number are at least at distance k from each other. The algorithm consists of two phases, each of which consists of c rounds. In the first phase, the algorithm repeatedly inserts single vertex bridges, while in the second phase, the double vertex bridges are added, eventually resulting in the set becoming connected. In round i, $1 \leq i \leq c$, coordinators from hexagons of class i will act, by trying to add connecting bridges within their hexagons. We suppose that each coordinator takes a decision about adding a bridge based on its knowledge of the part of the UDG at distance less than k from it. Hence, when a coordinator decides to add a bridge it is possible that the bridge is not joining two components from global perspective but rather two components as perceived from the local, limited perspective of the coordinator, and the resulting graph thus may contain some cycles. Algorithm 2 which is described below is a more formal outline of this idea. The complexity analysis of this algorithm follows in the sequel. However, note that previous "global algorithms" made use of a "globally constructed spanning tree" in order to find one-node bridges. We can no longer use a global algorithm to ensure that all our bridges consist of a single node. Rather we can use the structure of the dominating sets to show that if we first add all the single node bridges then we can limit the number of the remaining two-node bridges not just by the size of the dominating set, but rather by the size of the connected dominating set.

Lemma 6. *Consider the set $\mathcal{S}' = \bigcup_{H \in \mathcal{H}} S'_H$, where S'_H is the set of selected vertices of hexagon H after the first phase, and $\mathcal{H}$ is the set of all non-empty hexagons. Consider the Minimum Spanning Tree T of the OCDS used in Lemma 5. Let f be the number of edges $\{u, v\}$ of T such that $\mathcal{D} \cap (D_u \cap D_v) \neq \emptyset$. Then the UDG of $\mathcal{S}'$ has at most $|OCDS| - f$ connected components.*

Lemma 7. *$\mathcal{S}$ as constructed by Algorithm 2 is a connected dominating set.*

We will also use the following lemma from [10]:

Lemma 8. *The size of any independent set in a unit disk graph G is at most* $3.453 \cdot |OCDS| + 8.291$

We are now ready to prove the main theorem of this section.

Algorithm 2. Local Algorithm for Connected Dominating Set

```
1: Compute the dominating set $\mathcal{D}$ applying Algorithm 1.
2: Select a coordinator vertex in each non-empty hexagon and determine its class $c$
       based on the coordinates.
       // The rest of the algorithm is specified for a hexagon $H$.
3: Set the local set of selected vertices $S_H$ to be the vertices of $\mathcal{D}$ at distance less than
       $k$ hops from $H$ (i.e. each vertex of $\mathcal{D}$ is broadcasted up to distance $k$).
4: for bridgesize= 1 to 2 do
5:     for round= 1 to $c$ do
6:         if $class(H) = round$ then
7:             Determine all connected components of the UDG induced by the vertices
                   of $S_H$.
8:             repeat
9:                 Find a set $B$ of vertices of $H$, such that $|B| = bridgesize$ and such
                       that adding $B$ to $S_H$ connects at least two different connected
                       components.
10:                Add $B$ to $S_H$ and locally recompute the connected components.
11:            until no such set $B$ can be found
12:            Broadcast the newly added vertices up to distance $k$ hops.
                   // Hexagon $H$ has done its job for this bridgesize, now it only
                   participates in the broadcasts and updates its $S_H$.
13:        else
14:            for each non-empty hexagon $H'$ at distance at most $k$ do
15:                wait for a message containing the vertices $V'_H$ added in $H'$.
16:                $S_H \leftarrow S_H \cup V'_H$
17:            end for
18:        end if
19:    end for
20: end for
21: The union of $S_H$ for all hexagons $H$ is the desired connected dominating set $\mathcal{S}$.
```

Theorem 2. *Let $k > 3$ be an integer and $\mathcal{S}$ be the connected dominating set computed by Algorithm 2 with parameter k. Then*

$$|\mathcal{S}| \leq \left(7.453 + \frac{15}{k-3}\right) \cdot |OCDS| + \frac{16.6}{k-3} \tag{1}$$

Algorithm 2 does not produce an optimal connected dominating set of a given graph. It is rather obvious that a strictly local algorithm cannot produce an optimal connected dominating set even for a long cycle. In some cases we can lower the number of vertices in our connected dominating set by adding one more phase to Algorithm 2: after the addition of bridges is finished we can check for each vertex of D whether it is still needed for domination, following the order of the class numbers, and remove it from the connected dominating set if it is not needed for domination and it does not create a disconnection using the neighborhood of size $k-1$. However, this does not improve the competitive ratio of Theorem 2.

4 Conclusion

This paper gives the first ever local (constant time) algorithms for constructing constant approximations of dominating and connected dominating sets of unit disk graphs with location aware nodes with approximation ratios 5 and $7.453+\epsilon$, respectively. Although it was shown that the approximation ratio of the first algorithm could not be improved further, we do not have a lower bound on the competitive ratio of the second algorithm. It is important to note, that in the case of location aware networks, assumed in this paper, the dominating set problem is simpler than in the case of general UDGs. This poses a rather fundamental question about the "power of location awareness" and its impact on distributed computing, which we believe may have repercussions on future research studies on this subject.

Note that one of the concepts of this paper of tiling the plane and electing a node inside each tile is a version of an idea exploited, for example, in [20] and [13] where a single node was chosen to represent a cluster of related nodes.

References

1. Alzoubi, K.M., Wan, P.-J., Frieder, O.: Message-optimal connected-dominating-set construction for routing in mobile ad hoc networks. In: MOBIHOC 2002, pp. 157–164 (2002)
2. Aspnes, J., Bush, C., Dolev, S., Fatouroum, P., Georgiou, C., Shvartsman, A., Spirakis, P., Wattenhofer, R.: Eight open problems in distributed computing. Bulletin of the European Association for Theoretical Computer Science 90(109) (October 2006) Columns: Distributed Computing.
3. Awerbuch, B., Peleg, D.: Sparse partitions (extended abstract). In: IEEE Symposium on Foundations of Computer Science (FOCS), pp. 503–513 (1990)
4. Breu, H., Kirkpatrick, D.G.: Unit disk graph recognition is NP-hard. Computational Geometry: Theory and Applications 9, 3–24 (1998)
5. Chávez, E., Dobrev, S., Kranakis, E., Opatrny, J., Stacho, L., Urrutia, J.: Local construction of planar spanners in unit disk graphs with irregular transmission ranges. In: Correa, J.R., Hevia, A., Kiwi, M.A. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 286–297. Springer, Heidelberg (2006)
6. Cheng, X., Huang, X., Li, D., Du, D.-Z.: A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. Networks 42, 202–208 (2003)
7. Clark, B.N., Colbourn, C.J., Johnson, D.S.: Unit disk graphs. Discrete Mathematics 86, 165–177 (1990)
8. Dessmark, A., Pelc, A.: Broadcasting in geometric radio networks. Journal of Discrete Algorithms 5, 187–201 (2007)
9. Fich, F., Ruppert, E.: Hundreds of impossibility results for distributed computing. Distributed Computing 16, 121–163 (2003)
10. Funke, S., Kesselman, A., Meyer, U., Segal, M.: A simple improved distributed algorithm for minimum cds in unit disk graphs. ACM Transactions on Sensor Networks 2(3), 444–453 (2006)
11. Hunt III, H.B., Marathe, M.V., Radhakrishnan, V., Ravi, S.S., Rosenkrantz, D.J., Stearns, R.E.: NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. Journal of Algorithms 26(2), 238–274 (1998)

12. Irving, R.W.: On approximating the minimum independent dominating set. Information Processing Letters 37, 197–200 (1991)
13. Gao, J., Guibas, L.J., Hershberger, J., Zhang, L., Zhu, A.: Discrete Mobile Centers. Discrete & Computational Geometry 30(1), 45–63 (2001)
14. Jia, L., Rajaraman, R., Suel, R.: An efficient distributed algorithm for constructing small dominating sets. Distributed Computing 14, 193–205 (2002)
15. Johnson, D.: Approximation algorithms for combinatorial problems. Journal of Computer and System Sciences 9, 256–278 (1974)
16. Kuhn, F., Wattenhofer, R.: Constant-time distributed dominating set approximation. Distributed Computing 17(4), 303–310 (2005)
17. Kuhn, F., Moscibroda, T., Wattenhofer, R.: What cannot be computed locally! In: 23th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, pp. 300–309 (2004)
18. Kuhn, F., Moscibroda, T., Wattenhofer, R.: Unit disk graph approximation. In: DialM: Proceedings of the Discrete Algorithms and Methods for Mobile Computing & Communications; later DIALM-POMC Joint Workshop on Foundations of Mobile Computing, pp. 17–23 (2004)
19. Kuhn, F., Moscibroda, T., Nieberg, T., Wattenhofer, R.: Local approximation schemes for ad hoc and sensor networks. In: DialM: Proceedings of the Discrete Algorithms and Methods for Mobile Computing & Communications; later DIALM-POMC Joint Workshop on Foundations of Mobile Computing, pp. 97–103 (2005)
20. Kuhn, F., Moscibroda, T., Wattenhofer, R.: On the locality of bounded growth. In: 24th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, pp. 60–68 (2005)
21. Kuhn, F., Moscibroda, T., Wattenhofer, R.: The price of being near-sighted. In: SODA, pp. 980–989. ACM Press, New York (2006)
22. Linial, N.: Locality in distributed graph algorithms. SIAM Journal on Computing 21(1), 193–201 (1992)
23. Luby, M.: A simple parallel algorithm for the maximal independent set problem. In: Proc. 17th Annual ACM Symposium on Theory of Computing (STOCS), May 1985, pp. 1–10 (1985)
24. Lund, C., Yannakakis, M.: On the hardness of approximating minimization problems. Journal of the ACM 41, 960–981 (1994)
25. Marathe, M.V., Breu, H., Hunt III, H.B., Ravi, S.S., Rosenkrantz, D.J.: Geometry based heuristics for unit disk graphs. ArXiv Mathematics e-prints (1994)
26. Naor, M., Stockmeyer, L.: What can be computed locally? SIAM J. on Computing 24, 1259–1277 (1995)
27. Nieberg, T., Hurink, J.: A PTAS for the minimum dominating set problem in unit disk graphs. In: Approximation and Online Algorithms, pp. 296–306 (2006)
28. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM Monographs on Discrete Mathematics and Applications (2000)
29. Wang, Y., Li, X.: Localized construction of bounded degree and planar spanner for wireless ad hoc networks. In: Proc. of the 2003 joint workshop on Foundations of mobile computing (DIALM-POMC 2003), pp. 59–68. ACM Press, New York (2003)

Spanners of Complete k-Partite Geometric Graphs[⋆]

Prosenjit Bose, Paz Carmi, Mathieu Couture, Anil Maheshwari, Pat Morin, and Michiel Smid

School of Computer Science, Carleton University
Ottawa, Ontario, Canada

Abstract. We address the following problem: Given a complete k-partite geometric graph K whose vertex set is a set of n points in $\mathbb{R}^d$, compute a spanner of K that has a "small" stretch factor and "few" edges. We present two algorithms for this problem. The first algorithm computes a $(5+\epsilon)$-spanner of K with $O(n)$ edges in $O(n \log n)$ time. The second algorithm computes a $(3+\epsilon)$-spanner of K with $O(n \log n)$ edges in $O(n \log n)$ time. Finally, we show that there exist complete k-partite geometric graphs K such that every subgraph of K with a subquadratic number of edges has stretch factor at least 3.

1 Introduction

Let S be a set of n points in $\mathbb{R}^d$. A *geometric graph* with vertex set S is an undirected graph H whose edges are line segments $\overline{pq}$ that are weighted by the Euclidean distance $|pq|$ between p and q. For any two points p and q in S, we denote by $\delta_H(p,q)$ the length of a shortest path in H between p and q. For a real number $t \geq 1$, a subgraph G of H is said to be a *t-spanner* of H, if $\delta_G(p,q) \leq t \cdot \delta_H(p,q)$ for all points p and q in S. The smallest t for which this property holds is called the *stretch factor* of G. Thus, a subgraph G of H with stretch factor t approximates the $\binom{n}{2}$ pairwise shortest-path lengths in H within a factor of t. If H is the complete geometric graph with vertex set S, then G is also called a t-spanner of the point set S.

Most of the work on constructing spanners has been done for the case when H is the complete graph. It is well known that for any set S of n points in $\mathbb{R}^d$ and for any real constant $\epsilon > 0$, there exists a $(1+\epsilon)$-spanner of S containing $O(n)$ edges. Moreover, such spanners can be computed in $O(n \log n)$ time; see Salowe [7] and Vaidya [8]. For a detailed overview of results on spanners for point sets, see the book by Narasimhan and Smid [6].

For spanners of arbitrary geometric graphs, much less is known. Althöfer *et al.* [1] have shown that for any $t > 1$, every weighted graph H with n vertices contains a subgraph with $O(n^{1+2/(t-1)})$ edges, which is a t-spanner of H. Observe that this result holds for any weighted graph; in particular, it is valid for any

⋆ Research partially supported by NSERC, MRI, CFI, and MITACS.

geometric graph. For geometric graphs, a lower bound was given by Gudmundsson and Smid [5]: They proved that for every real number t with $1 < t < \frac{1}{4}\log n$, there exists a geometric graph H with n vertices, such that every t-spanner of H contains $\Omega(n^{1+1/t})$ edges. Thus, if we are looking for spanners with $O(n)$ edges of arbitrary geometric graphs, then the best stretch factor we can obtain is $\Theta(\log n)$.

In this paper, we consider the case when the input graph is a complete k-partite Euclidean graph. Let S be a set of n points in $\mathbb{R}^d$, and let S be partitioned into subsets $S_1, S_2, \ldots, S_k$. Let $K_{S_1 \ldots S_k}$ denote the *complete k-partite graph on S*. This graph has S as its vertex set and two points p and q are connected by an edge (of length $|pq|$) if and only if p and q are in different subsets of the partition. The problem we address is formally defined as follows:

Problem 1. Let $k > 1$ be an integer, let S be a set of n points in $\mathbb{R}^d$, and let S be partitioned into subsets $S_1, S_2, \ldots, S_k$. Compute a t-spanner of the k-partite complete graph $K_{S_1 \ldots S_k}$ that has a "small" number of edges and whose stretch factor t is "small".

The main contribution of this paper is to present an algorithm that computes such a t-spanner with $O(n)$ edges in $O(n \log n)$ time, where $t = 5 + \epsilon$ for any constant $\epsilon > 0$. We also show that if one is willing to use $O(n \log n)$ edges, then our algorithm adapts easily to reach a stretch factor of $t = 3 + \epsilon$. Finally, we give an example of a complete k-partite geometric graph K such that every subgraph of K with a subquadratic number of edges has stretch factor at least 3.

We remark that in a recent paper, Bose *et al.* [2] considered the problem of constructing spanners of point sets that have $O(n)$ edges and whose chromatic number is a most k. This problem is different from ours: Bose *et al.* compute a spanner of the complete graph and their algorithm can choose a "good" k-partition of the vertices. In our problem, the k-partition is given and we want to compute a spanner of the complete k-partite graph.

The rest of this paper is organized as follows. In Section 2, we recall properties of the Well-Separated Pair Decomposition (WSPD) that we use in our algorithm. In Section 3, we provide an algorithm that solves the problem of constructing a spanner of the complete k-partite graph. In Section 4, we show that the spanner constructed by this algorithm has $O(n)$ edges and that its stretch factor is bounded from above by a constant that depends only on the dimension d. In Section 5, we show how a simple modification to our algorithm improves the stretch factor to $5 + \epsilon$ while still having $O(n)$ edges. In Section 6, we show how to achieve a stretch factor of $3 + \epsilon$ using $O(n \log n)$ edges. We also provide a lower bound of 3 on the stretch factor for the general geometric k-partite spanner problem.

For ease of presentation, we will present all our results for the case when $k = 2$. The generalization to arbitrary values of k is a little bit more involved and will be given in the full version.

2 The Well-Separated Pair Decomposition

In this section, we recall crucial properties of the Well-Separated Pair Decomposition (WSPD) of Callahan and Kosaraju [4] that we use for our construction. Our presentation follows the one in Narasimhan and Smid [6].

Definition 1. *Let S be a set of points in $\mathbb{R}^d$. The* bounding box $\beta(S)$ *of S is the smallest axes-parallel hyperrectangle that contains S.*

Definition 2. *Let X and Y be two sets of points in $\mathbb{R}^d$ and let $s > 0$ be a real number. We say that X and Y are* well-separated *with respect to s if there exists two balls B_1 and B_2 such that (i) B_1 and B_2 have the same radius, say ρ, (ii) $\beta(X) \subseteq B_1$, (iii) $\beta(Y) \subseteq B_2$, and (iv)* $\min\{|xy| : x \in B_1 \cap \mathbb{R}^d, y \in B_2 \cap \mathbb{R}^d\} \geq s\rho$.

Definition 3. *Let S be a set of points in $\mathbb{R}^d$ and let $s > 0$ be a real number. A* well-separated pair decomposition (WSPD) of S with separation constant s *is a set of unordered pairs of subsets of S that are well-separated with respect to s, such that for any two distinct points $p, q \in S$ there is a unique pair $\{X, Y\}$ in the WSPD such that $p \in X$ and $q \in Y$.*

Lemma 1 (Lemma 9.1.2 in [6]). *Let $s > 0$ be a real number and let X and Y be two point sets that are well-separated with respect to s.*

1. *If $p, p', p'' \in X$ and $q \in Y$, then $|p'p''| \leq (2/s)|pq|$.*
2. *If $p, p' \in X$ and $q, q' \in Y$, then $|p'q'| \leq (1 + 4/s)|pq|$.*

Callahan and Kosaraju [3] have shown how to construct a t-spanner of S from a WSPD: All one has to do is pick from each pair $\{X, Y\}$ an arbitrary edge (p, q) with $p \in X$ and $q \in Y$. In order to compute a spanner of S that has a linear number of edges, one needs a WSPD that has a linear number of pairs. Callahan and Kosaraju [4] showed that a WSPD with a linear number of pairs always exists and can be computed in time $O(n \log n)$. Their algorithm uses a split-tree.

Definition 4. *Let S be a non-empty set of points in $\mathbb{R}^d$. The* split-tree *of S is defined as follows: if S contains only one point, then the split-tree is a single node that stores that point. Otherwise, the split-tree has a root that stores the bounding box $\beta(S)$ of S, as well as an arbitrary point of S called the* representative *of S and denoted by rep(S). Split $\beta(S)$ into two hyperrectangles by cutting its longest interval into two equal parts, and let S_1 and S_2 be the subsets of S contained in the two hyperrectangles. The root of the split-tree of S has two sub-trees, which are recursively defined split-trees of S_1 and S_2.*

The precise way Callahan and Kosaraju used the split-tree to compute a WSPD with a linear number of pairs is of no importance to us. The only important aspect we need to retain is that each pair is uniquely determined by a pair of nodes in the tree. More precisely, for each pair $\{X, Y\}$ in the WSPD that is output by their algorithm, there are unique internal nodes u and v in the split-tree such that the sets S_u and S_v of points stored at the leaves of the subtrees rooted at u and v are precisely X and Y. Since there is such a unique correspondence, we will denote pairs in the WSPD by $\{S_u, S_v\}$, meaning that u and v are the nodes corresponding to the sets $X = S_u$ and $Y = S_v$.

If R is an axes-parallel hyperrectangle in $\mathbb{R}^d$, then we use $L_{\max}(R)$ to denote the length of a longest side of R.

Lemma 2 (Lemma 9.5.3 in [6]). *Let u be a node in the split-tree and let u' be a node in the subtree of u such that the path between them contains at least d edges. Then $L_{\max}(\beta(S_{u'})) \leq \frac{1}{2} \cdot L_{\max}(\beta(S_u))$.*

Lemma 3 (Lemma 11.3.1 in [6]). *Let $\{S_u, S_v\}$ be a pair in the WSPD, let ℓ be the distance between the centers of $\beta(S_u)$ and $\beta(S_v)$, and let $\pi(u)$ be the parent of u in the split-tree. Then $L_{\max}(\beta(S_{\pi(u)})) \geq \frac{2\ell}{\sqrt{d}(s+4)}$.*

3 A First Algorithm

We now show how the WSPD can be used to address the problem of computing a spanner of a complete bipartite graph. In this section, we introduce an algorithm that outputs a graph with constant stretch factor and $O(n)$ edges. The analysis of this algorithm is presented in Section 4. In Section 5, we show how this algorithm can be improved to achieve a stretch factor of $5 + \epsilon$.

The input set $S \subseteq \mathbb{R}^d$ is the disjoint union of two sets R and B containing red and blue points, respectively. The graph K_{RB} is the complete bipartite geometric graph. We first need a definition.

Definition 5. *Let T be the split-tree of S that is used to compute the WSPD.*

1. *For a node u in T, we denote by S_u the set of all points in the subtree rooted at u.*
2. *We define BWSPD to be the subset of the WSPD obtained by removing all pairs $\{S_u, S_v\}$ such that $S_u \cup S_v \subseteq R$ or $S_u \cup S_v \subseteq B$.*
3. *A node u in T is* bichromatic *if there exist points r and b in S_u and a node v in T such that $r \in R$, $b \in B$, and $\{S_u, S_v\}$ is in the BWSPD.*
4. *A node u in T is a* red-node *if $S_u \subseteq R$ and there exists a node v in T such that $\{S_u, S_v\}$ is in the BWSPD.*
5. *A red-node u in T is a* red-root *if it does not have a proper ancestor that is a red-node in T.*
6. *A red-node u in T is a* red-leaf *if it does not have another red-node in its subtree.*
7. *A red-node u' in T is a* red-child *of a red-node u in T if u' is in the subtree rooted at u and there is no red-node on the path strictly between u and u'.*
8. *The notions of* blue-node, blue-root, blue-leaf, *and* blue-child *are defined as above, by replacing red by blue.*
9. *For each set S_u that contains at least one red point, $rep_R(S_u)$ denotes a fixed arbitrary red point in S_u. For each set S_u that contains at least one blue point, $rep_B(S_u)$ denotes a fixed arbitrary blue point in S_u.*

10. *The* distance *between two sets S_v and S_w, denoted by dist(S_v, S_w), is defined to be the distance between the centers of their bounding boxes.*
11. *Let u be a red-node or a blue-node in T. Consider all pairs $\{S_v, S_w\}$ in the BWSPD, where v is a red-node on the path in T from u to the root (this path includes u). Let $\{S_v, S_w\}$ be such a pair for which dist(S_v, S_w) is minimum. We define cl(S_u) to be the set S_w.*

Algorithm 1 computes a spanner of a complete bipartite geometric graph. It considers each pair $\{S_u, S_v\}$ of the WSPD, and decides whether or not it adds a red-blue and/or a blue-red edge between S_u and S_v. The outcome of this decision is based on the following three cases.

Case 1: All points of $S_u \cup S_v$ are of the same color. In this case, there is no edge of K_{RB} to approximate, so the algorithm just ignores this pair.
Case 2: Both S_u and S_v are bichromatic. In this case, the algorithm adds the two edges $(\text{rep}_R(S_u), \text{rep}_B(S_v))$ and $(\text{rep}_B(S_u), \text{rep}_R(S_v))$; see line 21. These two edges will allow us to approximate each edge of K_{RB} having one vertex in S_u and the other vertex in S_v.
Case 3: All points in S_u are of the same color, say red. In this case, only the edge $(\text{rep}_R(S_u), \text{rep}_B(S_v))$ is added; see line 11. In order to approximate each edge of K_{RB} having one (red) vertex in S_u and the other (blue) vertex in S_v, other edges have to be added. This is done in such a way that our final graph contains a "short" path between every red point r of S_u and the red representative $\text{rep}_R(S_u)$ of S_u. Observe that this path must contain blue points that are not in S_u. One way to achieve this is to add an edge between each point of S_u and $\text{rep}_B(\text{cl}(S_u))$; we call this construction a *star*. However, since the subtree rooted at u may contain other red-nodes, many edges may be added for each point in S_u, which could possibly lead to a quadratic number of edges in the final graph. To guarantee that the algorithm does not add too many edges, it introduces a star only if u is a red-leaf; see line 7. If u is a red-node, the algorithm only adds the edge $(\text{rep}_R(S_u), \text{rep}_B(\text{cl}(S_u)))$; see line 10. Then, the algorithm links each red-node u'' that is not a red-root to its red-parent u'. This is done through the edge $(\text{rep}_R(S_{u''}), \text{rep}_B(\text{cl}(u')))$; see line 13.

4 Analysis of Algorithm 1

Lemma 4. *The graph G computed by Algorithm 1 has $O(|R \cup B|)$ edges.*

Lemma 5. *Let r be a point of R, let b be a point of B, and let $\{S_u, S_v\}$ be the pair in the BWSPD for which $r \in S_u$ and $b \in S_v$. Assume that u is a red-node. Then there is a path in G between r and rep$_R(S_u)$ whose length is at most $c|rb|$, where*

$$c = 4\sqrt{d}(\mu d + 1)(1 + 4/s)^3, \qquad \mu = \left\lceil \log\left(\sqrt{d}(1 + 4/s)\right)\right\rceil + 1,$$

and s is the separation constant of the WSPD.

Algorithm 1.

```
Input: $S = R \cup B$, where $R$ and $B$ are two disjoint sets of red and blue points in $\mathbb{R}^d$.
Output: A spanner $G = (S, E)$ of the complete bipartite graph $K_{RB}$.
 1: compute the split-tree $T$ of $R \cup B$
 2: using $T$, compute the WSPD with respect to a separation constant $s > 0$
 3: using the WSPD, compute the BWSPD
 4: $E \leftarrow \emptyset$
 5: for each red-root $u$ in $T$ do
 6:    for each red-leaf $u'$ in the subtree of $u$ do
 7:       for each $r \in S_{u'}$, add to $E$ the edge $(r, \text{rep}_B(\text{cl}(S_{u'})))$
 8:    end for
 9:    for each red-node $u'$ that is in the subtree of $u$ (including $u$) do
10:       add to $E$ the edge $(\text{rep}_R(S_{u'}), \text{rep}_B(\text{cl}(S_{u'})))$
11:       for each pair $\{S_{u'}, S_{v'}\}$ in BWSPD, add to $E$ the edge $(\text{rep}_R(S_{u'}), \text{rep}_B(S_{v'}))$
12:       for each red-child $u''$ of $u'$, add to $E$ the edge $(\text{rep}_R(S_{u''}), \text{rep}_B(\text{cl}(S_{u'})))$
13:    end for
14: end for
15: for each blue-root $u$ in $T$ do
16:    //do the same as on lines 5–16, with red and blue interchanged
17: end for
18: for each $\{S_u, S_v\}$ in the BWSPD for which both $u$ and $v$ are bichromatic do
19:    add to $E$ the edges $(\text{rep}_R(S_u), \text{rep}_B(S_v))$ and $(\text{rep}_B(S_u), \text{rep}_R(S_v))$
20: end for
21: return the graph $G = (R \cup B, E)$
```

Proof. Let w be the red-leaf such that $r \in S_w$, and let $w = w_0, \ldots, w_k = u$ be the sequence of red-nodes that are on the path in T from w to u. Let Π be the path

$$\begin{array}{lllll} r \rightarrow \text{rep}_B(\text{cl}(S_{w_0})) \rightarrow & \text{rep}_R(S_{w_0}) & \rightarrow \text{rep}_B(\text{cl}(S_{w_1})) \rightarrow \text{rep}_R(S_{w_1}) \\ \quad\rightarrow \qquad \ldots \qquad \rightarrow \text{rep}_B(\text{cl}(S_{w_k})) \rightarrow & \text{rep}_R(S_{w_k}) & = \text{rep}_R(S_u). \end{array}$$

The first edge on this path, i.e., $(r, \text{rep}_B(\text{cl}(S_{w_0})))$, is added to the graph G in line 7 of the algorithm. The edges $(\text{rep}_B(\text{cl}(S_{w_i})), \text{rep}_R(S_{w_i}))$, $0 \leq i \leq k$, are added to G in line 10. Finally, the edges $(\text{rep}_R(S_{w_{i-1}}), \text{rep}_B(\text{cl}(S_{w_i})))$, $1 \leq i \leq k$, are added to G in line 13. It follows that Π is a path in G. We will show that the length of Π is at most $c|rb|$.

Let $0 \leq i \leq k$. Recall the definition of $\text{cl}(S_{w_i})$; see Definition 5: We consider all pairs $\{S_x, S_y\}$ in the BWSPD, where x is a red-node on the path in T from w_i to the root, and pick the pair for which $\text{dist}(S_x, S_y)$ is minimum. We denote the pair picked by (S_{x_i}, S_{y_i}). Thus, x_i is a red-node on the path in T from w_i to the root, $\{S_{x_i}, S_{y_i}\}$ is a pair in the BWSPD, and $\text{cl}(S_{w_i}) = S_{y_i}$. We define

$$\ell_i = \text{dist}(S_{x_i}, S_{y_i}).$$

Consider the first edge $(r, \mathrm{rep}_B(\mathrm{cl}(S_{w_0})))$ on the path Π. Since $r \in S_{w_0} \subseteq S_{x_0}$ and $\mathrm{rep}_B(\mathrm{cl}(S_{w_0})) \in S_{y_0}$, it follows from Lemma 1 that

$$|r, \mathrm{rep}_B(\mathrm{cl}(S_{w_0}))| \leq (1 + 4/s)\mathrm{dist}(S_{x_0}, S_{y_0}) = (1 + 4/s)\ell_0.$$

Let $0 \leq i \leq k$ and consider the edge $(\mathrm{rep}_B(\mathrm{cl}(S_{w_i})), \mathrm{rep}_R(S_{w_i}))$ on Π. Since $\mathrm{rep}_R(S_{w_i}) \in S_{w_i} \subseteq S_{x_i}$ and $\mathrm{rep}_B(\mathrm{cl}(S_{w_i})) \in S_{y_i}$, it follows from Lemma 1 that

$$(1)\ \ |\mathrm{rep}_B(\mathrm{cl}(S_{w_i})), \mathrm{rep}_R(S_{w_i})| \leq (1 + 4/s)\mathrm{dist}(S_{x_i}, S_{y_i}) = (1 + 4/s)\ell_i.$$

Let $1 \leq i \leq k$ and consider the edge $(\mathrm{rep}_R(S_{w_{i-1}}), \mathrm{rep}_B(\mathrm{cl}(S_{w_i})))$ on Π. Since $\mathrm{rep}_R(S_{w_{i-1}}) \in S_{w_{i-1}} \subseteq S_{x_i}$ and $\mathrm{rep}_B(\mathrm{cl}(S_{w_i})) \in S_{y_i}$, it follows from Lemma 1 that

$$|\mathrm{rep}_R(S_{w_{i-1}}), \mathrm{rep}_B(\mathrm{cl}(S_{w_i}))| \leq (1 + 4/s)\mathrm{dist}(S_{x_i}, S_{y_i}) = (1 + 4/s)\ell_i.$$

Thus, the length of the path Π is at most $\sum_{i=0}^{k} 2(1 + 4/s)\ell_i$. Therefore, it is sufficient to prove that $\sum_{i=0}^{k} \ell_i \leq 2\sqrt{d}(\mu d + 1)(1 + 4/s)^2|rb|$. It follows from the definition of $\mathrm{cl}(S_u) = \mathrm{cl}(S_{w_k})$ that $\ell_k = \mathrm{dist}(S_{x_k}, S_{y_k}) \leq \mathrm{dist}(S_u, S_v)$. Since, by Lemma 1, $\mathrm{dist}(S_u, S_v) \leq (1 + 4/s)|rb|$, it follows that

$$(2)\ \ \ell_k \leq (1 + 4/s)|rb|.$$

Thus, it is sufficient to prove that

$$(3)\ \ \sum_{i=0}^{k} \ell_i \leq 2\sqrt{d}(\mu d + 1)(1 + 4/s)\ell_k.$$

If $k = 0$, then (3) obviously holds. Assume from now on that $k \geq 1$. For each i with $0 \leq i \leq k$, we define

$$a_i = L_{\max}(\beta(S_{w_i})),$$

i.e., a_i is the length of a longest side of the bounding box of S_{w_i}.

Let $0 \leq i \leq k$. It follows from Lemma 1 that $L_{\max}(\beta(S_{x_i})) \leq \frac{2}{s}\ell_i$. Since w_i is in the subtree of x_i, we have $L_{\max}(\beta(S_{w_i})) \leq L_{\max}(\beta(S_{x_i}))$. Thus, we have

$$(4)\ \ a_i \leq \frac{2}{s}\ell_i \text{ for } 0 \leq i \leq k.$$

Lemma 2 states that

$$(5)\ \ a_i \leq \frac{1}{2}a_{i+d} \text{ for } 0 \leq i \leq k - d.$$

Let $0 \leq i \leq k-1$. Since w_i is a red-node, there is a node w_i' such that $\{S_{w_i}, S_{w_i'}\}$ is a pair in the BWSPD. We have $\ell_i = \mathrm{dist}(S_{x_i}, S_{y_i}) \leq \mathrm{dist}(S_{w_i}, S_{w_i'})$. By applying Lemma 3, we obtain

$$\mathrm{dist}(S_{w_i}, S_{w_i'}) \leq \frac{\sqrt{d}(s+4)}{2} L_{\max}(\beta(S_{\pi(w_i)})) \leq \frac{\sqrt{d}(s+4)}{2} L_{\max}(\beta(S_{w_{i+1}})) = \frac{\sqrt{d}(s+4)}{2} a_{i+1}.$$

Thus, we have

(6) $\ell_i \leq \frac{\sqrt{d}(s+4)}{2} a_{i+1}$ for $0 \leq i \leq k-1$.

First assume that $1 \leq k \leq \mu d$. Let $0 \leq i \leq k-1$. By using (6), the fact that the sequence $a_0, a_1, \ldots, a_k$ is non-decreasing, and (4), we obtain

$$\ell_i \leq \frac{\sqrt{d}(s+4)}{2} a_{i+1} \leq \frac{\sqrt{d}(s+4)}{2} a_k \leq \sqrt{d}(1+4/s)\ell_k.$$

Therefore,

$$\sum_{i=0}^{k} \ell_i \leq k\sqrt{d}(1+4/s)\ell_k + \ell_k \leq (k+1)\sqrt{d}(1+4/s)\ell_k \leq (\mu d+1)\sqrt{d}(1+4/s)\ell_k,$$

which is less than the right-hand side in (3).

It remains to consider the case when $k > \mu d$. Let $i \geq 0$ and $j \geq 0$ be integers such that $i + 1 + jd \leq k$. By applying (6) once, (5) j times, and (4) once, we obtain

$$\ell_i \leq \frac{\sqrt{d}(s+4)}{2} a_{i+1} \leq \frac{\sqrt{d}(s+4)}{2} \left(\frac{1}{2}\right)^j a_{i+1+jd} \leq \sqrt{d}(1+4/s) \left(\frac{1}{2}\right)^j \ell_{i+1+jd}.$$

For $j = \mu = \lceil \log(\frac{\sqrt{d}(s+4)}{s}) \rceil + 1$, this implies that, for $0 \leq i \leq k-1-\mu d$,

(7) $\ell_i \leq \frac{1}{2} \ell_{i+1+\mu d}$.

By re-arranging the terms in the summation in (3), we obtain

$$\sum_{i=0}^{k} \ell_i = \sum_{h=0}^{\mu d} \sum_{j=0}^{\lfloor (k-h)/(\mu d+1) \rfloor} \ell_{k-h-j(\mu d+1)}.$$

Let j be such that $0 \leq j \leq \lfloor (k-h)/(\mu d+1) \rfloor$. By applying (7) j times, we obtain

$$\ell_{k-h-j(\mu d+1)} \leq \left(\frac{1}{2}\right)^j \ell_{k-h}.$$

It follows that

$$\sum_{j=0}^{\lfloor (k-h)/(\mu d+1) \rfloor} \ell_{k-h-j(\mu d+1)} \leq \sum_{j=0}^{\infty} \left(\frac{1}{2}\right)^j \ell_{k-h} = 2\ell_{k-h}.$$

Thus, we have

$$\sum_{i=0}^{k} \ell_i \leq 2 \sum_{h=0}^{\mu d} \ell_{k-h}.$$

By applying (6), the fact that the sequence $a_0, a_1, \dots, a_k$ is non-decreasing, followed by (4), we obtain, for $0 \le i \le k-1$ and $1 \le j \le k-i$,

$$\ell_i \le \frac{\sqrt{d}(s+4)}{2} a_{i+1} \le \frac{\sqrt{d}(s+4)}{2} a_{i+j} \le \sqrt{d}(1+4/s)\ell_{i+j}.$$

Obviously, the inequality $\ell_i \le \sqrt{d}(1+4/s)\ell_{i+j}$ also holds for $j=0$. Thus, for $i = k-h$ and $j = h$, we get

$$\ell_{k-h} \le \sqrt{d}(1+4/s)\ell_k \text{ for } 0 \le h \le \mu d.$$

It follows that

$$\sum_{i=0}^{k} \ell_i \le 2 \sum_{h=0}^{\mu d} \sqrt{d}(1+4/s)\ell_k = 2\sqrt{d}(\mu d+1)(1+4/s)\ell_k,$$

completing the proof that (3) holds. □

Lemma 6. *Assuming that the separation constant s of the WSPD satisfies $s \ge 8 + 6/c$, the graph G computed by Algorithm 1 is a t-spanner of the complete bipartite graph K_{RB}, where $t = 2c + 1 + 4/s$ and c is as in Lemma 5.*

Proof. It suffices to show that for each edge (r, b) of K_{RB}, the graph G contains a path between r and b of length at most $t|rb|$. We will prove this by induction on the lengths of the edges in K_{RB}.

Let r be a point in R, let b be a point in B, and let $\{S_u, S_v\}$ be the pair in the BWSPD for which $r \in S_u$ and $b \in S_v$.

The base case is when (r, b) is a shortest edge in K_{RB}. Since $s > 2$, it follows from Lemma 1 that u is a red-node and v is a blue-node. In line 11 of Algorithm 1, the edge $(\text{rep}_R(S_u), \text{rep}_B(S_v))$ is added to G. By Lemma 1, the length of this edge is at most $(1+4/s)|rb|$. The claim follows from two applications of Lemma 5 to get from r to $\text{rep}_R(S_u)$ and from $\text{rep}_B(S_v)$ to b.

In the induction step, we distinguish four cases.

Case 1: u is a red-node and v is a blue-node. This case is identical to the base case.

Case 2: u is a bichromatic node and v is a blue-node. In the equivalent of line 11 for the blue-nodes, Algorithm 1 adds the edge $(\text{rep}_R(S_u), \text{rep}_B(S_v))$ to G. By Lemma 1, the length of this edge is at most $(1+4/s)|rb|$. Let b^* be a blue node in S_u. Since $s > 2$, it follows from Lemma 1 that $|rb^*| < |rb|$. Thus, by induction, there is a path in G between r and b^* whose length is at most $t|rb^*|$. By a similar argument, there is a path in G between b^* and $\text{rep}_R(S_u)$ whose length is at most $t|b^*, \text{rep}_R(S_u)|$. Applying Lemma 1, it follows that there is a path in G between r and $\text{rep}_R(S_u)$ whose length is at most $t(|rb^*| + |b^*, \text{rep}_R(S_u)|) \le t(2|rb|/s + 2|rb|/s) = 4t|rb|/s$. By Lemma 5, there is a path in G between b and $\text{rep}_B(S_v)$ whose length is at most $c|rb|$. We have shown that there is a path in G between r and b whose length is at most $(1+4/s)|rb| + 4t|rb|/s + c|rb|$. Since $s \ge 8 + 6/c$, this quantity is at most $t|rb|$.

Case 3: Both u and v are bichromatic nodes. In line 21, Algorithm 1 adds the edge $(\text{rep}_B(S_u), \text{rep}_R(S_v))$ to G. By Lemma 1, the length of this edge is at most $(1 + 4/s)|rb|$. By induction, there is a path in G between r and $\text{rep}_B(S_u)$ whose length is at most $t|r, \text{rep}_B(S_u)|$, which, by Lemma 1, is at most $2t|rb|/s$. By a symmetric argument, there is a path in G between b and $\text{rep}_R(S_v)$, whose length is at most $2t|rb|/s$. We have shown that there is a path in G between r and b whose length is at most $(1 + 4/s)|rb| + 4t|rb|/s$, which is at most $t|rb|$. □

Lemma 7. *The running time of Algorithm 1 is $O(n \log n)$, where $n = |R \cup B|$.*

To summarize, we have shown the following: Algorithm 1 computes a t-spanner of the complete bipartite graph K_{RB} having $O(n)$ edges, where t is given in Lemma 6. The running time of this algorithm is $O(n \log n)$. By choosing the separation constant s sufficiently large, the stretch factor t converges to

$$8\sqrt{d}\left(d\left\lceil\frac{1}{2}\log d\right\rceil + d + 1\right) + 1.$$

5 An Improved Algorithm

As before, we are given two disjoint sets R and B of red and blue points in $\mathbb{R}^d$. Intuitively, the way to improve the bound of Lemma 5 is by adding shortcuts along the path from each red-leaf to the red-root above it. More precisely, from (7) in the proof of Lemma 5, we know that if we go $1 + \mu d$ levels up in the split-tree, then the length of the edge along the path doubles. Thus, for each red-node in T, we will add edges to all $2\delta(1 + \mu d)$ red-nodes above it in T. Here, δ is an integer constant that is chosen such that the best result is obtained in the improved bound.

Definition 6. *Let u and u' be red-nodes in the split-tree such that u' is in the subtree rooted at u. For an integer $\zeta \geq 1$, we say that u is* ζ-levels above *u', if there are $\zeta - 1$ red-nodes on the path strictly between u and u'. We say that u' is a* ζ-red-child *of u if u is at most ζ-levels above u'. These notions are defined similarly for the blue-nodes.*

Algorithm 2.

Input: $S = R \cup B$, where R and B are two disjoint sets of red and blue points in $\mathbb{R}^d$, respectively, and a real constant $0 < \epsilon < 1$.
Output: A $(5 + \epsilon)$-spanner $G = (S, E)$ of the complete bipartite graph K_{RB}.

```
1: Choose a separation constant $s$ such that $s \geq 12/\epsilon$ and $(1 + 4/s)^2 \leq 1 + \epsilon/36$ and
   choose an integer constant $\delta$ such that $\frac{2^\delta}{2^\delta - 1} \leq 1 + \epsilon/36$.
2: The rest of the algorithm is the same as Algorithm 1, except for lines 12–14, which
   are replaced by the following:
   let $\zeta = 2\delta(\mu d + 1)$
   for each $\zeta$-red-child $u''$ of $u'$ do
       add to $E$ the edges $(\text{rep}_R(S_{u''}), \text{rep}_B(\text{cl}(S_{u'})))$ and $(\text{rep}_B(\text{cl}(S_{u''})), \text{rep}_R(S_{u'}))$
   end for
```

Lemma 8. *Let r be a point of R, let b be a point of B, and let $\{S_u, S_v\}$ be the pair in the BWSPD for which $r \in S_u$ and $b \in S_v$. Assume that u is a red-node. Let G be the graph computed by Algorithm 2. There is a path in G between r and $\mathit{rep}_R(S_u)$ whose length is at most $(2 + \epsilon/3)|rb|$.*

Proof. Due to space constraints, the complete proof is omitted. Here, we only state how the path can be obtained. Let w be the red-leaf such that $r \in S_w$, and let $w = w_0, w_1, \dots, w_k = u$ be the sequence of red-nodes that are on the path in T from w to u. Throughout the proof, we will use the variables x_i, y_i, ℓ_i, and a_i, for $0 \le i \le k$, that were introduced in the proof of Lemma 5.

If $0 \le k \le 2\delta(\mu d + 1)$, then the path

$$r \rightarrow \mathrm{rep}_B(\mathrm{cl}(S_w)) \rightarrow \mathrm{rep}_R(S_u)$$

satisfies the condition in the lemma. Assume that $k > 2\delta(\mu d + 1)$. We define $m = k \bmod (\delta(\mu d + 1))$ and $m' = \frac{k-m}{\delta(\mu d+1)}$. We consider the sequence of red-nodes $w = w_0, w_{\delta(\mu d+1)+m}, w_{2\delta(\mu d+1)+m}, w_{3\delta(\mu d+1)+m}, \dots, w_k = u$. The path

$$\begin{array}{llll} r \rightarrow & \mathrm{rep}_B(\mathrm{cl}(S_{w_0})) & \rightarrow \mathrm{rep}_R(S_{w_{\delta(\mu d+1)+m}}) & \\ & \rightarrow \mathrm{rep}_B(\mathrm{cl}(S_{w_{2\delta(\mu d+1)+m}})) & \rightarrow \mathrm{rep}_R(S_{w_{2\delta(\mu d+1)+m}}) & \\ & \rightarrow \mathrm{rep}_B(\mathrm{cl}(S_{w_{3\delta(\mu d+1)+m}})) & \rightarrow \mathrm{rep}_R(S_{w_{3\delta(\mu d+1)+m}}) & \\ & \vdots & \vdots & \\ & \rightarrow \quad \mathrm{rep}_B(\mathrm{cl}(S_{w_k})) & \rightarrow \quad \mathrm{rep}_R(S_{w_k}) & = \mathrm{rep}_R(S_u) \end{array}$$

satisfies the condition in the lemma. □

Lemma 9. *Let $n = |R \cup B|$. The graph G computed by Algorithm 2 is a $(5+\epsilon)$-spanner of the complete bipartite graph K_{RB} and the number of edges of this graph is $O(n)$. The running time of Algorithm 2 is $O(n \log n)$.*

We have proved the following result.

Theorem 1. *Let S be a set of n points in $\mathbb{R}^d$ which is partitioned into two subsets R and B, and let $0 < \epsilon < 1$ be a real constant. In $O(n \log n)$ time, we can compute a $(5 + \epsilon)$-spanner of the complete bipartite graph K_{RB} having $O(n)$ edges.*

6 Improving the Stretch Factor

We have shown how to compute a $(5+\epsilon)$-spanner with $O(n)$ edges of any complete bipartite graph. In this section, we show that if we are willing to use $O(n \log n)$ edges, the stretch factor can be reduced to $3 + \epsilon$. We start by showing that a stretch factor less than 3 using a subquadratic number of edges is not possible.

Theorem 2. *For every real number $t < 3$, there is no algorithm that, when given as input two arbitrary disjoint sets R and B of points in $\mathbb{R}^d$, computes a t-spanner of the complete bipartite graph K_{RB} having less than $|R| \cdot |B|$ edges.*

Proof. Let us assume by contradiction that there exists such an algorithm $\mathcal{A}$ for some real number $t < 3$. Let $\epsilon = 3 - t$, let B_1 and B_2 be two balls of diameter $\epsilon/6$ such that the distance between their centers is $1 + \epsilon/6$. Let R be a set of points that are contained in B_1 and let B be a set of points that are contained in B_2. Let G be the graph obtained by running algorithm $\mathcal{A}$ on R and B. By our hypothesis, G has less than $|R| \cdot |B|$ edges. Thus, there exist a point r in R and a point b in B, such (r, b) is not an edge in G. Since any path in G between r and b contains at least three edges, the length of the shortest path in G between r and b in G is at least three. Since $|rb| \leq 1 + \epsilon/3$, it follows that the stretch factor of G is at least $\frac{3}{1+\epsilon/3}$, which is greater than $t = 3 - \epsilon$, contradicting the existence of $\mathcal{A}$. $\square$

Theorem 3. *Let S be a set of n points in $\mathbb{R}^d$ which is partitioned into two subsets R and B, and let $0 < \epsilon < 1$ be a real constant. In $O(n \log n)$ time, we can compute a $(3 + \epsilon)$-spanner of the complete bipartite graph K_{RB} having $O(n \log n)$ edges.*

Proof. Consider the following variant of the WSPD. For every pair $\{X, Y\}$ in the standard WSPD, where $|X| \leq |Y|$, we replace this pair by the $|X|$ pairs $\{\{x\}, B\}$, where x ranges over all points of X. Thus, in this new WSPD, each pair contains at least one singleton set. Callahan and Kosaraju [4] showed that this new WSPD consists of $O(n \log n)$ pairs.

We run Algorithm 2 on R and B, using this new WSPD. Let G be the graph that is computed by this algorithm. Observe that Lemma 8 still holds for G. In the proof of Lemma 9 of the upper bound on the stretch factor of G, we apply Lemma 8 only once. Therefore, the stretch factor of G is at most $3 + \epsilon$. $\square$

References

1. Althöfer, I., Das, G., Dobkin, D.P., Joseph, D., Soares, J.: On sparse spanners of weighted graphs. Discrete & Computational Geometry 9, 81–100 (1993)
2. Bose, P., Carmi, P., Couture, M., Maheshwari, A., Smid, M., Zeh, N.: Geometric spanners with small chromatic number. In: Proceedings of the 5th Workshop on Approximation and Online Algorithms. LNCS, Springer, Berlin (2007)
3. Callahan, P.B., Kosaraju, S.R.: Faster algorithms for some geometric graph problems in higher dimensions. In: Proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms, pp. 291–300 (1993)
4. Callahan, P.B., Kosaraju, S.R.: A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. J. ACM 42(1), 67–90 (1995)
5. Gudmundsson, J., Smid, M.: On spanners of geometric graphs. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 388–399. Springer, Heidelberg (2006)
6. Narasimhan, G., Smid, M.: Geometric Spanner Networks. Cambridge University Press, New York (2007)
7. Salowe, J.S.: Constructing multidimensional spanner graphs. International Journal of Computational Geometry & Applications 1, 99–107 (1991)
8. Vaidya, P.M.: A sparse graph almost as good as the complete graph on points in K dimensions. Discrete & Computational Geometry 6, 369–381 (1991)

Minimum Cost Homomorphisms to Reflexive Digraphs

Arvind Gupta[⋆], Pavol Hell[∗], Mehdi Karimi, and Arash Rafiey

School of Computing Science
Simon Fraser University
Burnaby, B.C., Canada, V5A 1S6
{arvind,pavol,mmkarimi,arashr}@cs.sfu.ca
http://cs.sfu.ca

Abstract. For a fixed digraph H, the *minimum cost homomorphism problem*, MinHOM(H), asks whether an input digraph G, with given costs $c_i(u)$, $u \in V(G)$, $i \in V(H)$, and an integer k, admits a homomorphism to H of total cost not exceeding k.

Minimum cost homomorphism problems encompass many well studied optimization problems such as list homomorphism problems, retraction and precolouring extension problems, chromatic partition optimization, and applied problems in repair analysis.

For undirected graphs the complexity of the problem, as a function of the parameter H, is well understood; for digraphs, the situation appears to be more complex, and only partial results are known. We focus on the minimum cost homomorphism problem for *reflexive* digraphs H. It is known that MinHOM(H) is polynomial if H has a *Min-Max ordering*. We prove that for any other reflexive digraph H, the problem MinHOM(H) is NP-complete. (This was earlier conjectured by Gutin and Kim.) Apart from undirected graphs, this is the first general class of digraphs for which such a dichotomy has been proved. Our proof involves a forbidden induced subgraph characterization of reflexive digraphs with a Min-Max ordering, and implies a polynomial test for the existence of a Min-Max ordering in a reflexive digraph H.

Keywords: homomorphism, minimum cost homomorphism, reflexive digraph, polynomial time algorithm, NP-completeness, dichotomy.

1 Introduction and Terminology

For digraphs G and H, a mapping $f: V(G) \to V(H)$ is a *homomorphism of G to H* if uv is an arc of G implies $f(u)f(v)$ is an arc of H. Let H be a fixed digraph: the *homomorphism problem* for H, denoted HOM(H), asks whether or not an input digraph G admits a homomorphism to H. The *list homomorphism problem* for H, denoted ListHOM(H), asks whether or not an input digraph G, with lists $L_u \subseteq V(H), u \in V(G)$, admits a homomorphism f to H in which all $f(u) \in L_u, u \in V(G)$.

[⋆] Supported by an NSERC Discovery Grant.

Suppose G and H are digraphs, and $c_i(u)$, $u \in V(G)$, $i \in V(H)$, are real *costs*. The *cost of a homomorphism* f of G to H is $\sum_{u \in V(G)} c_{f(u)}(u)$. If H is fixed, the *minimum cost homomorphism problem* for H, denoted MinHOM(H), is the following problem. Given an input digraph G, together with costs $c_i(u)$, $u \in V(G)$, $i \in V(H)$, and an integer k, decide if G admits a homomorphism to H of cost not exceeding k.

If the graph H is *symmetric* (each $uv \in A(H)$ implies $vu \in A(H)$), we may view H as an undirected graph. In this way, we may view the problem MinHOM(H) as applying also to undirected graphs.

The minimum cost homomorphism problem was introduced, in the context of undirected graphs, in [16]. There, it was motivated by a real-world problem in defense logistics; in general, the problem seems to offer a natural and practical way to model many optimization problems. Special cases include for instance the list homomorphism problem [19,21] and the optimum cost chromatic partition problem [18,24,25] (which itself has a number of well-studied special cases and applications [27,29]).

Our interest is in proving dichotomies: given a class of problems such as HOM(H), we would like to prove that for each digraph H the problem is polynomial-time solvable, or NP-complete. This is, for instance, the case for HOM(H) with undirected graphs H [20]; in that case it is known that HOM(H) is polynomial time solvable when H is bipartite or has a loop, and NP-complete otherwise [20]. This is a dichotomy *classification*, since we specifically classify the complexity of the problems HOM(H), depending on H.

For undirected graphs H, a dichotomy classification for the problem

MinHOM(H) has been provided in [17]. (For ListHOM(H), consult [6].) Thus, the minimum cost homomorphism problem for graphs has been handled, and interest shifted to directed graphs. The first studies [13,14,15] focused on irreflexive digraphs (no vertex has a loop), where dichotomies has been obtained for digraphs H such that $U(H)$ is a complete or complete multipartite graph. More recently, [11] promoted the study of digraphs with loops allowed; and, in particular, of reflexive digraphs. Dichotomy has been proved for reflexive digraphs H such that $U(H)$ is a complete graph, or a complete multipartite graph without digons [10,12]. In this paper, we give a full dichotomy classification of the complexity of MinHOM(H) for reflexive digraphs; this is the first dichotomy result for a general class of digraphs - our only restriction is that the digraphs are reflexive. The dichotomy classification we prove verifies a conjecture of Gutin and Kim [10]. (Partial results on ListHOM(H) for digraphs can be found in [3,5,7,8,9,23,32].

Let H be any digraph. An arc $xy \in A(H)$ is *symmetric* if $yx \in A(H)$; the digraph H is *symmetric* if each arc of H is symmetric. Otherwise, we denote by $S(H)$ the *symmetric subgraph* of H, i.e., the undirected graph with $V(S(H)) = V(H)$ and $E(S(H)) = \{uv : uv \in A(H) \text{ and } vu \in A(H)\}$. We also denote by $U(H)$ the *underlying graph* of H, i.e., the undirected graph with $V(U(H)) = V(H)$ and $E(U(H)) = \{uv : uv \in A(H) \text{ or } vu \in A(H)\}$. If H is a reflexive digraph, then both $S(H)$ and $U(H)$ are reflexive graphs. Finally, we denote by $B(H)$ the bipartite graph obtained from H as follows. Each vertex v of H gives rise to two vertices of $B(H)$ - a *white* vertex v' and a *black* vertex v''; each arc vw of H gives rise to an edge $v'w''$ of $B(H)$. Note that if H is a reflexive digraph, then all edges $v'v''$ are present in $B(H)$. The *converse* of G is the digraph obtained from G by reversing the directions of all arcs.

We say that an undirected graph H is a *proper interval graph* if there is an inclusion-free family of intervals $I_v, v \in V(H)$, such that $vw \in E(H)$ if and only if I_v intersects I_w. Note that by this definition proper interval graphs are reflexive. Wegner proved [30] that a reflexive graph H is a proper interval graph if and only if it does not contain an induced cycle C_k, with $k \geq 4$, or an induced claw, net, or tent, as given in Figure 1.

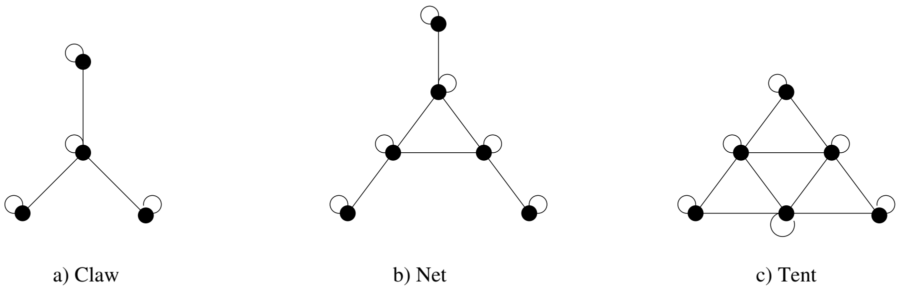


Fig. 1. The claw, the net, and the tent

We say that a bipartite graph H (with a fixed bipartition into white and black vertices) is a *proper interval bigraph* if there are two inclusion-free families of intervals I_v, for all white vertices v, and J_w for all black vertices w, such that $vw \in E(H)$ if and only if I_v intersects J_w. By this definition proper interval bigraphs are irreflexive and bipartite. A Wegner-like characterization (in terms of forbidden induced subgraphs) of proper interval bigraphs is given in [22]: H is a proper interval bigraph if and only if it does not contain an induced cycle C_{2k}, with $k \geq 3$, or an induced biclaw, binet, or bitent, as given in Figure 2.

A linear ordering $<$ of $V(H)$ is a *Min-Max ordering* if $i < j, s < r$ and $ir, js \in A(H)$ imply that $is \in A(H)$ and $jr \in A(H)$. For a reflexive digraph H, it is easy to see that $<$ is a Min-Max ordering if and only if for any j between i and k, we have $ik \in A(H)$ imply $ij, jk \in A(H)$. (Clearly, a Min-Max ordering has the property, by the definition applied to ik and jj. Conversely, the property implies that $is \in A(H)$ and $jr \in A(H)$ if j and s are between i and r or conversely - by considering the arcs ir respectively js; in the remaining cases $i < s < r < j$ or $s < i < j < r$ we apply the property to the two arcs ir and js.) For a bipartite graph H (with a fixed bipartition into white and black vertices), it is easy to see that $<$ is a Min-Max ordering if and only if $<$ restricted to the white vertices, and $<$ restricted to the black vertices satisfy the condition of Min-Max orderings, i.e., $i < j$ for white vertices, and $s < r$ for black vertices, and $ir, js \in A(H)$, imply that $is \in A(H)$ and $jr \in A(H)$). A *bipartite Min-Max ordering* is an ordering $<$ specified just for white and for black vertices.

It is known that if H admits a Min-Max ordering, then the problem MinHOM(H) is polynomial time solvable [13], see also [4,26]; however, there are digraphs with polynomial MinHOM(H) which do not have Min-Max ordering [14]. For undirected graphs, all H without a Min-Max ordering yield an NP-complete MinHOM(H) [17]; moreoever, having a Min-Max ordering can be characterized by simple forbidden induced subgraphs, and recognized in polynomial time [17]. In particular, a *reflexive* graph admits a Min-Max ordering if and only if it is a proper interval graph, and a *bipartite* graph admits a Min-Max ordering if and only if it is a proper interval bigraph [17].

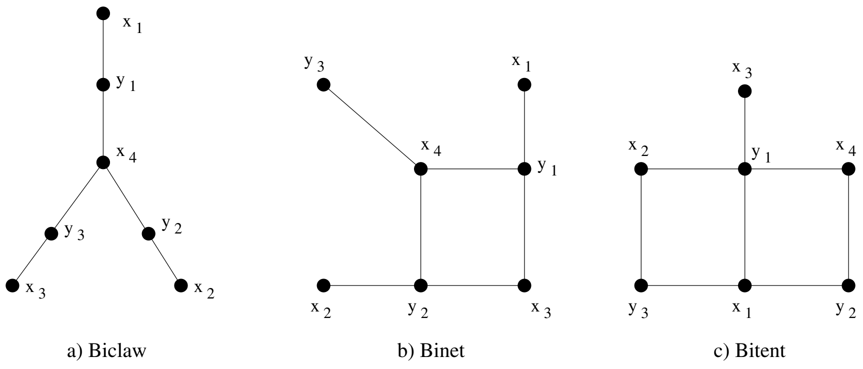


Fig. 2. The biclaw, the binet, and the bitent

We shall give a combinatorial description of reflexive digraphs with Min-Max ordering, in terms of forbidden induced subgraphs. Our characterization yields a polynomial time algorithm for the existence of a Min-Max ordering in a reflexive digraph. It also allows us to complete a dichotomy classification of MinHOM(H) for reflexive digraphs H, by showing that all problems MinHOM(H) where H does not admit a Min-Max ordering are NP-complete. This verifies a conjecture of Gutin and Kim in [10].

2 Structure and Forbidden Subgraphs

Since both reflexive and bipartite graphs admit a characterization of existence of Min-Max orderings by forbidden induced subgraphs, our goal will be accomplished by proving the following theorem. It also implies a polynomial time algorithm to test if a reflexive digraph has a Min-Max ordering.

Theorem 1. *A reflexive digraph H has a Min-Max ordering if and only if*

- *$S(H)$ is a proper interval graph, and*
- *$B(H)$ is a proper interval bigraph, and*
- *H does not contain an induced subgraph isomorphic to H_i with $i = 1, 2, 3, 4, 5, 6$.*

The digraphs H_i are depicted in Figure 3. The resulting forbidden subgraph characterization is summarized in the following corollary. Note that forbidden subgraphs in $S(H)$ directly describe forbidden subgraphs in H, and it is easy to see that each forbidden induced subgraph in $B(H)$ can also be translated to a small family of forbidden induced subgraphs in H.

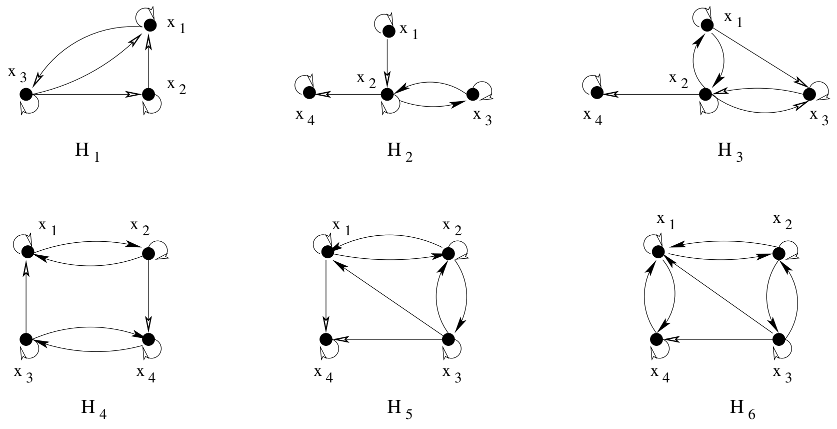


Fig. 3. The obstructions H_i with $i = 1, 2, 3, 4, 5, 6$

Corollary 1. *A reflexive digraph H has a Min-Max ordering if and only if $S(H)$ does not contain an induced $C_k, k \geq 4$, or claw, net, or tent, $B(H)$ does not contain an induced $C_{2k}, k \geq 3$, or biclaw, binet, or bitent, and H does not contain an induced H_i with $i = 1, 2, 3, 4, 5, 6$.*

We proceed to prove the Theorem.

Proof: Suppose first that $<$ is a Min-Max ordering $<$ of H. It is easily seen that $<$ is also a Min-Max ordering of $S(H)$, and that $<$ applied separately to the corresponding white and black vertices of $B(H)$ is a bipartite Min-Max ordering of $B(H)$. To complete the proof of necessity, we now claim that none of the digraphs $H_i, i = 1, 2, 3, 4, 5, 6$ admits a Min-Max ordering. We only show this for H_3, the proofs of the other cases being similar. Suppose that $<$ is a Min-Max ordering of H_3. For the triple x_1, x_2, x_3, we note that x_2 must be between x_1 and x_3 in the ordering $<$, as otherwise the arcs between x_2 and x_1, x_3 would imply that $x_1x_3 \in E(S(H))$. Without loss of generality assume that $x_1 < x_2 < x_3$.

Since x_1 and x_4 are independent and $x_1x_2 \in E(S(H))$, we must have $x_4 > x_1$. A similar argument yields $x_4 < x_3$; however, $x_1 < x_4 < x_3$ is impossible, as $x_1x_3 \in A(H)$ but $x_1x_4 \notin A(H)$.

To prove the sufficiency of the three conditions, we shall prove the following claim.

Lemma 1. *If $S(H)$ has a Min-Max ordering and $B(H)$ has a bipartite Min-Max ordering, then either H has a Min-Max ordering, or H contains an induced H_i (or its converse) for some $i = 1, 2, 3, 4, 5, 6$.*

Proof: Suppose $<$ is a bipartite Min-Max ordering of $B(H)$. A pair u, v of vertices of H is *proper for* $<$ if $u' < v'$ if and only if $u'' < v''$ in $B(H)$. We say a bipartite Min-Max ordering $<$ is *proper* if all pairs u, v of H are proper for $<$. If $<$ is a proper bipartite Min-Max ordering, then we can define a corresponding ordering $\prec$ on the vertices of H, where $u \prec v$ if and only if $u' < v'$ (which happens if and only if $u'' < v''$). It is easy to check that $\prec$ is now a Min-Max ordering of H.

Suppose, on the other hand, that the bipartite Min-Max ordering $<$ on $B(H)$ is not proper. Thus there are vertices v', u' such that $v' < u'$ and $u'' < v''$. Suppose there is no vertex s' such that $s'v'' \in E(B(H))$, $s'u'' \notin E(B(H))$: then we can exchange the position of v'' and u'' in $<$ and still have a bipartite Min-Max ordering. Furthermore, this exchange strictly increases the number of proper pairs in H: any w with $u'' < w'' < v''$ and $u' < w'$ creates a new improper pair u, w but also creates a new proper pair v, w (and the pair u, v is also a new proper pair). Analogously, if there is no vertex t'' such that $u't'' \in E(B(H))$, $v't'' \notin E(B(H))$, we can exchange u', v' and increase the number of proper pairs in H. Suppose we have performed all exchanges until we reached a bipartite Min-Max ordering $<$ which admits no more exchanges. Then there are two possibilities: either $<$ is now proper, and H admits a Min-Max ordering as above, or $<$ is still not proper, and one of the following two cases must occur (up to symmetry):

Case 1: $s'v'', v't'' \in E(B(H))$ and $s'u'', u't'' \notin E(B(H))$.

It is easy to see that since $<$ is a bipartite Min-Max ordering, we must have $u' < s'$ and $t'' < u''$. (Note that means that $s'' \neq t''$.) Since $u'u'', v'v'' \in E(B(H))$, by the same argument we must have $u'v'', v'u'' \in E(B(H))$; and similarly we obtain $s't'' \notin E(B(H))$. If both $v's''$ and $t'v''$ are edges of $B(H)$ then u, v, s, t induce a claw in $S(H)$: indeed in $B(H)$, we have the edges $v't'', t'v'', v'u''$, $u'v'', v's'', s'v''$ and the non-edges $u't'', s'u'', s't''$. This is a contradiction, as $S(H)$ is assumed to have a Min-Max ordering, i.e., be a proper interval graph.

If neither $v's''$ nor $t'v''$ is an edge of $B(H)$, then if $u's''$ is an edge of $B(H)$, then s, v, u induce a copy of H_1 in H, and if $, t'u''$ is an edge of $B(H)$, then t, v, u induce a copy of H_1. Thus consider the case when $u's'', t'u'' \notin E(B(H))$. If $t's'' \in E(B(H))$, then $s', s'', t', t'', v', v''$ would induce a copy of C_6 in $B(H)$, contrary to our assumption that $B(H)$ has a bipartite Min-Max ordering, i.e., is a proper interval bigraph. Thus $t's'' \notin E(B(H))$ and t, s, v, u induce a copy of H_2 in H.

If only one of $v's''$ or $t'v''$ is an edge of $B(H)$, assume first that $v's'' \in E(B(H))$ and $t'v'' \notin E(B(H))$. If $t'u''$ is an edge of $B(H)$, then t, v, u induce a copy of H_1 in H, and if $t's''$ is an edge of $B(H)$, then t, v, s similarly induce a copy of H_1; thus asume that $t'u'', t's'' \notin E(B(H))$. Note that $u's'' \in E(B(H))$, else the vertices $u', u'', v', t'', t', s'', s'$ would induce a biclaw in $B(H)$, contrary to $B(H)$ being a proper interval bigraph. It now follows that s, t, u, v induce a copy of H_3 in H. If $v's'' \notin E(B(H))$ and $t'v'' \in E(B(H))$, the proof is similar, except we obtain copies of H_1 and the converse of H_3.

Case 2: $s'v'', u't'' \in E(B(H))$ and $s'u'', v't'' \notin E(B(H))$.

We again easily observe that we must have $u' < s'', v'' < t''$, and $u'v'', v'u'' \in E(B(H))$. If $s'' = t''$ we obtain a copy of H_1 induced by u, v, s in H; hence we assume that $s'' \neq t''$. Suppose first that $u's'', t'v'' \notin E(B(H))$. We have $s' < t'$ and $t'' < s''$, and so $t's'', s't'' \in A(H)$, implying that u, v, s, t induce a copy of H_4 in H. Suppose next that both $t'v'', u's'' \in E(B(H))$. If $v's''$ is not an edge of $B(H)$, vertices u, v, s induce a copy of H_1 in H, and if $t'u''$ is not an edge of $B(H)$, vertices u, v, t induce a copy of H_1 in H. Thus we have $v's'', t'u'' \in E(B(H))$. Now we have $t' < s'$ and $s'' < t''$, and hence $t's'', s't'' \in E(B(H))$. This is impossible, since u, v, s, t would induce a copy of C_4 in $S(H)$. Finally, if only one of $t'v'', u's''$ is an edge of $B(H)$, say $u's'' \in E(B(H))$ and $t'v'' \notin E(B(H))$ (the other case is symmetric), then with the same argument as above, $v's'' \in E(B(H))$, $s't'' \in E(B(H))$, and s, t, u, v induce (depending on which of the pairs $t'u'', t's''$ are edges of $B(H)$) one of H_1, H_5 (or its converse), or H_6 (or its converse). □

3 Complexity

If H has a Min-Max ordering, then MinHOM(H) is polynomial time solvable [13] see also [4,26]. Now using our forbidden induced subgraph characterization we can prove that reflexive digraphs H without a Min-Max ordering yield NP-complete MinHOM(H) problems. Note that we already know that MinHOM($S(H)$) is NP-complete if $S(H)$ is not a proper interval graph, and MinHOM($B(H)$) is NP-complete if $B(H)$ is not a proper interval bigraph [13]. We begin with a few simple observations. The first one is easily proved by setting up a natural polynomial time reduction from MinHOM($B(H)$) to $MinHOM(H)$ [11].

Proposition 1. *[11] If MinHOM($B(H)$) is NP-complete, then $MinHOM(H)$ is also NP-complete.* □

The next two observations are folklore, and proved by obvious reductions, cf. [10].

Proposition 2. *If MinHOM($S(H)$) is NP-complete, then MinHOM(H) is also NP-complete.* □

Proposition 3. *Let H' be an induced subgraph of the digraph H. If MinHOM(H') is NP-complete then MinHOM(H) is NP-complete.* □

We now continue to prove that MinHOM(H) is NP-complete for digraphs $H = H_1, \ldots, H_6$. Let $\mathcal{I}$ denote the following decision problem: given a graph X and an integer k, decide whether or not X contains an independent set of k vertices. This problem has been useful for proving NP-completeness of minimum cost homomorphism problems for undirected graphs [17], and we use it again for digraphs.

Proposition 4. *[17] The problem $\mathcal{I}$ is NP-complete, even when restricted to three-colourable graphs (with a given three-colouring).* □

We denote by $\mathcal{I}_3$ the restriction of $\mathcal{I}$ to graphs with a given three-colouring. In the following Lemmas, we give polynomial time reductions from $\mathcal{I}_3$. Note that all problems MinHOM(H) are in NP. The NP-completeness of MinHOM(H_1) follows from [10], Lemma 2-4.

Lemma 2. *The problem $MinHOM(H_2)$ is NP-complete.*

Proof: We now construct a polynomial time reduction from $\mathcal{I}_3$ to MinHOM-(H_2). Let X be a graph whose vertices are partitioned into independent sets U, V, W, and let k be a given integer. We construct an instance of MinHOM(H_2) as follows: the digraph G is obtained from X by replacing each edge uv of X with $u \in U, v \in V$ by an arc uv, replacing each edge uw of X with $u \in U, w \in W$ by an arc uw, and replacing each edge vw of X with $v \in V, w \in W$ by an arc wv. The costs are defined by (writing for simplicity $c_i(y)$ for $c_{x_i}(y)$) $c_1(u) = 0, c_2(u) = 1$ for $u \in U$, $c_4(v) = 0, c_2(v) = 1$ for $v \in V$, and $c_3(w) = 0, c_2(w) = 1$, for $w \in W$. All other $c_i(y) = |V(X)|$.

We now claim that X has an independent set of size k if and only if G admits a homomorphism to H_2 of cost $|V(X)| - k$. Let I be an independent set in G. We can define a mapping $f : V(G) \to V(H_2)$ as follows:

- $f(u) = x_1$ for $u \in U \cap I$ and $f(u) = x_2$ for $u \in U - I$
- $f(v) = x_4$ for $v \in V \cap I$ and $f(v) = x_2$ for $v \in V - I$
- $f(w) = x_3$ for $w \in W \cap I$ and $f(w) = x_2$ for $w \in W - I$

This is a homomorphism of G to H_2 of cost $|V(X)| - k$.

Let f be a homomorphism of G to H_2 of cost $|V(X)| - k$. If $k \leq 0$ then we are trivially done, so assume that $k > 0$, which implies that all individual costs are either zero or one. Let $I = \{y \in V(X) \mid c_{f(y)}(y) = 0\}$ and note that $|I| \geq k$. It can be seen that I is an independent set in G: for instance when $uv \in E(G)$ with $u \in I \cap U$ and $v \in I \cap V$, then $f(u) = x_1$ and $f(v) = x_4$, contrary to f being a homomorphism. (The other possibilities are similar, or easier.) □

Lemma 3. *$MinHOM(H_3)$ is NP-complete.*

Proof: The reduction from the proof of Lemma 2 also applies here. □

Lemma 4. *$MinHOM(H_4)$ is NP-complete.*

Proof: We now construct a polynomial time reduction from $\mathcal{I}_3$ to MinHOM-(H_4). Let X be a graph whose vertices are partitioned into independent sets U, V, W, and let k be a given integer. An instance of MinHOM(H_4) is formed as follows: the digraph G is obtained from X by replacing each edge uv of X with $u \in U, v \in V$ by an arc vu, replacing each edge uw of X with $u \in U, w \in W$ by a directed path $um_{uw}w$, and replacing each edge vw of X with $v \in V, w \in W$ by a directed path $vm_{vw}w$. The costs are defined by $c_1(u) = 1, c_3(u) = 0$ for $u \in U$; $c_2(v) = 0, c_3(v) = 1$ for $v \in V$; $c_4(w) = 0, c_1(w) = 1$ for $w \in W$; $c_3(m_{uw}) = c_4(m_{uw}) = |V(X)|$ for each edge uw of X with $u \in U, w \in W$; $c_2(m_{vw}) = c_4(m_{vw}) = |V(X)|$ for each edge vw of X with $v \in V, w \in W$; and $c_i(m) = 0$ for any other vertex $m \in V(G) - V(X)$, and $c_i(y) = |V(X)|$ for any other vertex $y \in V(X)$.

We now claim that X has an independent set of size k if and only if G admits a homomorphism to H_4 of cost $|V(X)| - k$. Let I be an independent set in G. We can define a mapping $f : V(G) \to V(H_2)$ as follows:

- $f(u) = x_3$ for $u \in U \cap I$ and $f(u) = x_1$ for $u \in U - I$
- $f(v) = x_2$ for $v \in V \cap I$ and $f(v) = x_3$ for $v \in V - I$
- $f(w) = x_4$ for $w \in W \cap I$ and $f(w) = x_1$ for $w \in W - I$
- $f(m_{uw}) = x_2$ when $f(u) = x_1$, and $f(m_{uw}) = x_1$ when $f(u) = x_3$ for each edge uw of X with $u \in U, w \in W$
- $f(m_{vw}) = x_3$ when $f(w) = x_4$ and $f(m_{vw}) = x_1$ when $f(w) = x_1$ for each edge vw of X with $v \in V, w \in W$

This is a homomorphism of G to H_4 of cost $|V(X)| - k$.

Let f be a homomorphism of G to H_4 of cost $|V(X)| - k$. We may again assume that all individual costs are either zero or one. Let $I = \{y \in V(X) \mid c_{f(y)}(y) = 0\}$ and note that $|I| \geq k$. It can be again seen that I is an independent set in G, as if $uw \in E(G)$, where $u \in I \cap U$ and $w \in I \cap V$ then $f(u) = x_3$ and $f(w) = x_4$, thus, $f(m_{uw}) = x_3$ or $f(m_{uw}) = x_4$. However, the cost of homomorphism is greater than $|V(X)|$, a contradiction. The other cases can also be treated similarly. □

Lemma 5. *MinHOM(H_5) is NP-complete.*

Proof: We similarly construct a polynomial time reduction from $\mathcal{I}_3$ to Min-HOM(H_5): this time the digraph G is obtained from X by replacing each edge uv of X with $u \in U, v \in V$ by an arc uv; replacing each edge uw of X with $u \in U, w \in W$ by arcs um_{uw}, wm_{uw}; and replacing each edge wv of X with $w \in W, v \in V$ by a directed path $wm_{wv}v$. The costs are $c_1(u) = 1, c_2(u) = 0$ for $u \in U$; $c_2(v) = 1, c_4(v) = 0$ for $v \in V$; $c_3(w) = 1, c_1(w) = 0$ for $w \in W$; $c_1(m_{uw}) = c_2(m_{uw}) = |V(X)|$ for each edge uw of X with $u \in U, w \in W$; $c_1(m_{wv}) = c_4(m_{wv}) = |V(X)|$ for each edge wv of X with $w \in W, v \in V$; $c_i(m) = 0$ for any other vertex $m \in V(G) - V(X)$, and $c_i(y) = |V(X)|$ for any other vertex $y \in V(X)$.

We again claim that X has an independent set of size k if and only if G admits a homomorphism to H_5 of cost $|V(X)| - k$. Let I be an independent set in G. We can define a mapping $f : V(G) \rightarrow V(H_2)$ by $f(u) = x_2$ for $u \in U \cap I$ and $f(u) = x_1$ for $u \in U - I$; $f(v) = x_4$ for $v \in V \cap I$ and $f(v) = x_2$ for $v \in V - I$; $f(w) = x_1$ for $w \in W \cap I$ and $f(w) = x_3$ for $w \in W - I$; $f(m_{uw}) = x_3$ when $f(u) = x_2$, and $f(m_{uw}) = x_4$ when $f(u) = x_1$, for each edge uw of X with $u \in U, w \in W$; $f(m_{wv}) = x_3$ when $f(w) = x_3$ and $f(m_{wv}) = x_2$ when $f(w) = x_1$, for each edge wv of X with $w \in W, v \in V$. This is a homomorphism of G to H_5 of cost $|V(X)| - k$.

Let f be a homomorphism of G to H_5 of cost $|V(X)| - k$. Assuming again that all individual costs are either zero or one, let $I = \{y \in V(X) \mid c_{f(y)}(y) = 0\}$ and note that $|I| \geq k$. It can be seen that I is an independent set in G, as if $uw \in E(G)$, where $u \in I \cap U$ and $w \in I \cap V$ then $f(u) = x_2$ and $f(w) = x_1$, thus, $f(m_{uw}) = x_1$ or $f(m_{uw}) = x_2$. However, the cost of homomorphism is greater than $|V(X)|$, a contradiction. The other cases can also be treated similarly. □

Lemma 6. *$MinHOM(H_6)$ is NP-complete.*

Proof: The proof is again similar, letting the digraph G be obtained from X by replacing each edge uv of X with $u \in U, v \in V$ by an arc uv; replacing each edge uw of X with $u \in U, w \in W$ by a directed path $um_{uw}w$; and replacing each edge vw of X with $v \in V, w \in W$ by an arc wv. The costs are defined by $c_1(u) = 0, c_2(u) = 1$ for $u \in U$; $c_3(v) = 0, c_1(v) = 1$ for $v \in V$; $c_4(w) = 0, c_3(w) = 1$; $c_1(m_{uw}) = c_4(m_{uw}) = |V(X)|$ for each edge uw of X with $u \in U, w \in W$; and letting $c_i(m) = 0$ for any other vertex $m \in V(G) - V(X)$, and $c_i(y) = |V(X)|$ for any other vertex $y \in V(X)$.

It can again be seen that X has an independent set of size k if and only if G admits a homomorphism to H_6 of cost $|V(X)| - k$: letting I be an independent set in G, we define a mapping $f : V(G) \rightarrow V(H_2)$ by $f(u) = x_1$ for $u \in U \cap I$ and $f(u) = x_2$ for $u \in U - I$; $f(v) = x_3$ for $v \in V \cap I$ and $f(v) = x_1$ for $v \in V - I$; $f(w) = x_4$ for $w \in W \cap I$ and $f(w) = x_3$ for $w \in W - I$; $f(m_{uw}) = x_3$ when $f(u) = x_2$ and $f(m_{uw}) = x_2$ when $f(u) = x_1$ for each edge uw, $u \in U, w \in W$. This is a homomorphism of G to H_6 of cost $|V(X)| - k$.

Let f be a homomorphism of G to H_6 of cost $|V(X)| - k$ and assume again that all individual costs are either zero or one. Let $I = \{y \in V(X) \mid c_{f(y)}(y) = 0\}$ and note that $|I| \geq k$. It can again be seen that I is an independent set in G. □

We have proved the following result, conjectured in [10].

Theorem 2. *Let H be a reflexive digraph. If H has a Min-Max ordering, then MinHOM(H) is polynomial time solvable; otherwise, it is NP-complete.*

References

1. Alekseev, V.E., Lozin, V.V.: Independent sets of maximum weight in (p, q)-colorable graphs. Discrete Mathematics 265, 351–356 (2003)
2. Bang-Jensen, J., Gutin, G.: Digraphs: Theory, Algorithms and Applications. Springer, London (2000)
3. Brewster, R.C., Hell, P.: Homomorphisms to powers of digraphs. Discrete Mathematics 244, 31–41 (2002)

4. Cohen, D., Cooper, M., Jeavons, P., Krokhin, A.: A maximal tractable class of soft constraints. J. Artif. Intell. Res. 22, 1–22 (2004)
5. Feder, T.: Homomorphisms to oriented cycles and k-partite satisfiability. SIAM J. Discrete Math 14, 471–480 (2001)
6. Feder, T., Hell, P., Huang, J.: Bi-arc graphs and the complexity of list homomorphisms, J. Graph Theory 42, 61–80 (2003)
7. Feder, T., Hell, P., Tucker-Nally, K.: Digraph matrix partitions and trigraph homomorphisms. Discrete Applied Mathematics 154, 2458–2469 (2006)
8. Feder, T., Hell, P., Huang, J.: List homomorphisms to reflexive digraphs (manuscript, 2005)
9. Feder, T.: Classification of Homomorphisms to Oriented Cycles and k-Partite Satisfiability. SIAM Journal on Discrete Mathematics 14, 471–480 (2001)
10. Gutin, G., Kim, E.J.: Complexity of the minimum cost homomorphism problem for semicomplete digraphs with possible loops (submitted)
11. Gutin, G., Kim, E.J.: Introduction to the minimum cost homomorphism problem for directed and undirected graphs. Lecture Notes of the Ramanujan Math. Society (to appear)
12. Gutin, G., Kim, E.J.: On the complexity of the minimum cost homomorphism problem for reflexive multipartite tournaments (submitted)
13. Gutin, G., Rafiey, A., Yeo, A.: Minimum Cost and List Homomorphisms to Semicomplete Digraphs. Discrete Appl. Math. 154, 890–897 (2006)
14. Gutin, G., Rafiey, A., Yeo, A.: Minimum Cost Homomorphisms to Semicomplete Multipartite Digraphs. Discrete Applied Math. (submitted)
15. Gutin, G., Rafiey, A., Yeo, A.: Minimum Cost Homomorphisms to Semicomplete Bipartite Digraphs (submitted)
16. Gutin, G., Rafiey, A., Yeo, A., Tso, M.: Level of repair analysis and minimum cost homomorphisms of graphs. Discrete Appl. Math. 154, 881–889 (2006)
17. Gutin, G., Hell, P., Rafiey, A., Yeo, A.: A dichotomy for minimum cost graph homomorphisms. European J. Combin. (to appear)
18. Halldórsson, M.M., Kortsarz, G., Shachnai, H.: Minimizing average completion of dedicated tasks and interval graphs. In: Goemans, M.X., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) RANDOM 2001 and APPROX 2001. LNCS, vol. 2129, pp. 114–126. Springer, Heidelberg (2001)
19. Hell, P.: Algorithmic aspects of graph homomorphisms. In: Survey in Combinatorics 2003, London Math. Soc. Lecture Note Series, vol. 307, pp. 239–276. Cambridge University Press, Cambridge (2003)
20. Hell, P., Nešetřil, J.: On the complexity of H-colouring. J. Combin. Theory B 48, 92–110 (1990)
21. Hell, P., Nešetřil, J.: Graphs and Homomorphisms. Oxford University Press, Oxford (2004)
22. Hell, P., Huang, J.: Interval bigraphs and circular arc graphs. J. Graph Theory 46, 313–327 (2004)
23. Hell, P., Nešetřil, J., Zhu, X.: Complexity of Tree Homomorphisms. Discrete Applied Mathematics 70, 23–36 (1996)
24. Jansen, K.: Approximation results for the optimum cost chromatic partition problem. J. Algorithms 34, 54–89 (2000)
25. Jiang, T., West, D.B.: Coloring of trees with minimum sum of colors. J. Graph Theory 32, 354–358 (1999)
26. Khanna, S., Sudan, M., Trevisan, L., Williamson, D.: The approximability of constraint satisfaction problems. SIAM Journal on Computing 30, 1863–1920 (2000)

27. Kroon, L.G., Sen, A., Deng, H., Roy, A.: The optimal cost chromatic partition problem for trees and interval graphs. In: D'Amore, F., Marchetti-Spaccamela, A., Franciosa, P.G. (eds.) WG 1996. LNCS, vol. 1197, pp. 279–292. Springer, Heidelberg (1997)
28. Lovász, L.: Three short proofs in graph theory. J. Combin. Theory, Ser. B 19, 269–271 (1975)
29. Supowit, K.: Finding a maximum planar subset of a set of nets in a channel. IEEE Trans. Computer-Aided Design 6, 93–94 (1987)
30. Wegner, G.: Eigenschaften der nerven homologische-einfactor familien in R^n, Ph.D. Thesis, Universität Gottigen, Gottigen, Germany (1967)
31. West, D.: Introduction to Graph Theory. Prentice Hall, Upper Saddle River (1996)
32. Zhou, H.: Characterization of the homomorphic preimages of certain oriented cycles. SIAM Journal on Discrete Mathematics 6, 87–99 (1993)

On the Complexity of Reconstructing H-free Graphs from Their Star Systems

Fedor V. Fomin[1], Jan Kratochvíl[2,⋆], Daniel Lokshtanov[1], Federico Mancini[1], and Jan Arne Telle[1]

[1] Department of Informatics, University of Bergen, N-5020 Bergen, Norway
{fomin, daniello, federico, telle}@ii.uib.no.

[2] Dept. of Applied Mathematics and Institute for Theoretical Computer Science, Charles University, Praha, Czech Republic

Abstract. In the Star System problem we are given a set system and asked whether it is realizable by the multi-set of closed neighborhoods of some graph, i.e., given subsets $S_1, S_2, \cdots, S_n$ of an n-element set V does there exist a graph $G = (V, E)$ with $\{N[v] : v \in V\} = \{S_1, S_2, \cdots, S_n\}$? For a fixed graph H the H-free Star System problem is a variant of the Star System problem where it is asked whether a given set system is realizable by closed neighborhoods of a graph containing no H as an induced subgraph. We study the computational complexity of the H-free Star System problem. We prove that when H is a path or a cycle on at most 4 vertices the problem is polynomial time solvable. In complement to this result, we show that if H belongs to a certain large class of graphs the H-free Star System problem is NP-complete. In particular, the problem is NP-complete when H is either a cycle or a path on at least 5 vertices. This yields a complete dichotomy for paths and cycles.

1 Introduction

The closed neighborhood of a vertex in a graph is sometimes called the "star" of the vertex. The "star system" of a graph is then the multi-set of closed neighborhoods of all the vertices of the graph and the Star System problem is the problem of deciding whether a given system of sets is a star system of some graph. The Star System problem is a natural combinatorial problem that fits into a broader class of *realizability* problems. In a realizability problem we are given a list P of invariants or properties (like a sequence of vertex degrees, set of cliques, number of colorings, etc) and the question is whether the given list is *graphical*, i.e., corresponds to the list of parameters of some graph. One of the well studied problems of realizability is the case when P is a degree sequence. This can be seen as a modification of the Star System problem where, instead of stars, the list P contains only the sizes of the stars. In this case, graphic sequences can be characterized by the Erdős-Gallai Theorem [7].

The Star System problem (also known as the Closed Neighborhood Realization problem) is equivalent to a number of other interesting problems. For

⋆ Supported by Czech research grant 1M0545.

example, it is equivalent to the question of whether a given $0-1$ matrix A is symmetrizable, i.e., whether by permuting rows (or columns) A can be turned into a symmetric matrix with all diagonal entries equal to 1. We refer to the recent survey of Boros et al. [6] for further equivalent problems related to the Matrix Symmetrization and Star System problems.

The question of the computational complexity of the Star System problem was first posed by Gert Sabidussi and Vera Sós at a conference in the mid-70s [8] (and since this appears to be the oldest reference to the problem, we choose to use the Star System terminology). At the same conference Babai observed that the Star System problem was at least as hard as the Graph Isomorphism problem. There are strong similarities with Graph Isomorphism, e.g., the Star System problem is equivalent to deciding if a given bipartite graph allows an automorphism of order 2 such that each vertex is adjacent to its image. In view of these connections to Graph Isomorphism the NP-hardness of the Star System problem came as a surprise. The proof of this fact was achieved in two steps. First, a related effort of Lubiw [11] showed that deciding whether an arbitrary graph has an automorphism of order 2 is NP-complete. Then Lalonde [10] showed that the Star System problem was NP-complete by a reduction from Lubiw's problem. This reduction came as a small surprise considering that, after Lubiw's proof, Babai had written that between Lubiw's problem and the Star System problem he "did not believe there was a deeper relationship" [3].

The result of Lalonde was rediscovered by Aigner and Triesch [1,2] who proved it in a stronger form, and indeed discovered a subproblem which is equivalent to Graph Isomorphism. It is easy to see that the problems of reconstructing graphs from their closed neighborhood hypergraphs and from their open neighborhood hypergraphs are polynomially equivalent. It is more convenient however, to describe the results of Aigner and Triesch in the language of open neighborhoods. They proved that deciding if a set system is the open neighborhood hypergraph of a *bipartite* graph is Graph Isomorphism-complete, while deciding if the open neighborhood hypergraph of a bipartite graph can be realized by a nonisomorphic (and non-bipartite) graph becomes again NP-complete.

Since bipartite graphs (and their complements) are hereditary classes of graphs, it is natural to pay closer attention to restriction of the Star System problem to classes of graphs defined by forbidden induced subgraphs. The problem we investigate in this paper is the following variation of the Star System problem, for a fixed graph H:

H-free Star System Problem
Input: A set system $\mathcal{S}$ over a ground set V
Question: Does there exist an H-free graph $G = (V, E)$ such that $\mathcal{S}$ is the star system of G?

Our main result is a complete dichotomy in the case when H is either a cycle C_k, or a path P_k on k vertices. We prove that the H-free Star System problem for $H \in \{C_k, P_k\}$ is polynomial time solvable when $k \leq 4$ (Section 3) and NP-complete when $k > 4$ (Section 4). Our NP-completeness result for paths and cycles follows from a more general result, which shows that there exists a much larger family of graphs for which if H belongs to it then the H-free Star System problem is NP-complete.

2 Preliminaries

We use standard graph notation with $G = (V, E)$ being a simple loopless undirected graph with vertex set V and edge set E. We denote by $N[v]$ and $N(v)$ the closed and open neighborhoods of a vertex v, respectively, and by $\overline{G}$ the complement of a graph G having an edge uv iff $u \neq v$ and $uv \notin E(G)$. We also call $N[v]$ the star of v and say that v is the center of $N[v]$. An *automorphism* of a graph $G = (V, E)$ is an isomorphism $f : V \rightarrow V$ of the graph to itself, and it has order 2 if for every vertex x we have $f(f(x)) = x$, i.e., the image of its image is itself. For a graph $G = (V, E)$ we define the $|V|$-element multi-set $Stars(G) = \{N[v] : v \in V\}$. For a fixed graph H we say that a graph G is *H-free* if G does not contain an induced subgraph isomorphic to H.

3 Forbidding Short Paths and Cycles

3.1 Forbidding Short Paths

In this section we show that the P_k-free Star System Problem is solvable in polynomial time for $k \leq 4$. For $k \leq 2$ the P_k-free System Problem is trivially polynomial time solvable. For $k = 3$ the realizable graph is a disjoint union of cliques, and in this case the problem is again trivial. The proof that the P_4-free Star System can be solved in polynomial time occupies the remaining part of this subsection.

The graphs without induced P_4 are called *cographs*. We exploit the following characterization of cographs.

Proposition 1 ([5]). *A graph G is a cograph if and only if every non-trivial induced subgraph of G contains at least one pair of vertices x and y, such that either $N[x] = N[y]$ or $N(x) = N(y)$.*

For a set system $\mathcal{S}$ over a ground set V, we say that $(x, y) \in V^2$ is a *closed pair* (of $\mathcal{S}$) if for every $S \in \mathcal{S}$ we have that $x \in S$ if and only if $y \in S$. Also, we define (x, y) to be an *open pair* (of $\mathcal{S}$) if there is exactly one set $S_x \in \mathcal{S}$ containing x and not y, exactly one set $S_y \in \mathcal{S}$ containing y and not x, and $S_x \setminus \{x\} = S_y \setminus \{y\}$.

We will show that a given set system $\mathcal{S}$ is the star system of a cograph G if and only if it can be reduced to one set on a single element by sequentially contracting closed and open pairs. We start by showing that, if $\mathcal{S}$ is the star system of a cograph, then $\mathcal{S}$ contains a closed or an open pair.

Lemma 1. *If $\mathcal{S} = Stars(G)$ for a nontrivial cograph $G = (V, E)$ then $\mathcal{S}$ has either a closed or an open pair.*

Proof. First of all recall that, since $\mathcal{S} = Stars(G)$, for every $v \in V$, we have that $N[v] = S_v \in \mathcal{S}$. By Proposition 1, G has a pair of vertices x and y such that either

$N[x] = N[y]$ or $N(x) = N(y)$. In the first case (x, y) is a closed pair in $\mathcal{S}$, because for every $z \in V$ we have that $x \in N[z]$ if and only if $y \in N[z]$. Hence every set of $\mathcal{S}$ contains either both x and y, or none of them. In the latter case, $N[x]$ contains x and not y, $N[y]$ contains y and not x, and $N[x]\backslash\{x\} = N(x) = N(y) = N[y]\backslash\{y\}$. Thus for every $z \in V \setminus \{x, y\}$ it follows that $x \in N[z]$ if and only if $y \in N[z]$. This means that (x, y) is an open pair because every set in S contains either both x and y, or none of them, and there are exactly two sets, S_x and S_y, such that $S_x \setminus \{x\} = S_y \setminus \{y\}$.

In the rest of the subsection, we will show how to contract closed and open pairs, such that the resulting star system represents a cograph if and only if it did before the contraction. We start with the closed pairs.

Lemma 2. *Given a set system $\mathcal{S}$ on V, let R be an inclusion maximal subset of V containing no closed pairs, and let $\mathcal{S}_R$ be the set $\{Z \mid \exists S \in \mathcal{S}$ such that $S \cap R = Z\}$. Then there is a cograph $G = (V, E)$ with $Stars(G) = \mathcal{S}$ if and only if there is a cograph G' with $Stars(G') = \mathcal{S}_R$.*

Proof. Let us assume that there is a cograph G with $Stars(G) = \mathcal{S}$ and let $G[R] = G'$. Then we claim that $Stars(G') = \mathcal{S}_R$. First, note that for every $v \in R$, $N_{G'}[v]$ is in $\mathcal{S}_R$. Additionally, consider $S \in \mathcal{S}_R$. Let x be a vertex of G such that $S = N[x] \cap R$. By maximality of R there is $x' \in R$ such that $N[x'] = N[x]$, thus $N_{G'}[x'] = N[x] \cap R = S$. Hence, for every $S \in \mathcal{S}_R$ there is a v in R such that $S = N_{G'}[v]$. Finally, observe that by construction $\mathcal{S}_R$ has no duplicate sets, and that $Stars(G')$ also has no duplicate sets because a duplicate set would imply that R contains a closed pair, contradicting that R has none. Together, this implies that $Stars(G') = \mathcal{S}_R$.

On the other hand, suppose that there is a cograph G' with $Stars(G') = \mathcal{S}_R$. For every element v in V there is a unique x in R such that (v, x) is a closed pair. Let $f : V \to R$ be the mapping such that for every vertex $v \in V$, the pair $(v, f(v))$ is a closed pair. Also, let f^{-1} be the inverse image of f. That is, for a vertex $x \in R$, we have that $f^{-1}(x) = \{v \in V \mid f(v) = x\}$. Finally, let $G = (V, \{(u, v) : f(u) = f(v) \ \vee \ (f(u), f(v)) \in E(G')\})$. We claim that $Stars(G) = \mathcal{S}$ and that G is a cograph.

For each $v \in V$, $N[v] = \bigcup_{x \in N_{G'}[f(v)]} f^{-1}(x)$ by the definition of G and $N_{G'}[f(v)] \in \mathcal{S}_R$. Furthermore, for every set $S \in \mathcal{S}_R$, we have that $\bigcup_{x \in S} f^{-1}(x) \in \mathcal{S}$ by definition of $\mathcal{S}_R$. Thus, since for every $v \in V$ there exists $S \in \mathcal{S}_R$ such that $S = N_{G'}[f(v)]$, we can conclude that $N[v]$ is in $\mathcal{S}$. On the other hand, for every $S \in \mathcal{S}$, there exists S' in $\mathcal{S}_R$ such that $S = \bigcup_{x \in S'} f^{-1}(x)$. Let u be the element of R such that $N_{G'}[u] = S'$. Then $S = \bigcup_{x \in N_{G'}[u]} f^{-1}(x) = N[u]$. This means that $S \in Stars(G)$. Finally, we know that for each vertex $u \in R$, there are $|f^{-1}(u)|$ copies of the star $\bigcup_{x \in N_{G'}[u]} f^{-1}(x) = N[u]$ in S. Also, we know that in V there are $|f^{-1}(u)|$ vertices with the same closed neighborhood $N[u]$ for each $u \in R$. This proves that $Stars(G) = \mathcal{S}$.

We will now prove that G is also a cograph, by showing that it does not contain an induced P_4. Observe first that $G[R] = G'$ is a cograph by definition. For the sake of contradiction, let us assume that there is a set $P \subseteq V$ of 4 vertices that induces a P_4 in G. If there is a pair u and v of distinct vertices in P such that $f(u) = f(v)$ then $N[u] = N[v]$, and specifically u and v have the same neighbourhood in P which is impossible because no pair of distinct vertices of a P_4 have the same neighbourhood. If no such pair exists, then $P' = \{f(x) : x \in P\}$ must induce a P_4 in G' as well. This leads to a contradiction, concluding the proof of the lemma.

Before we show how to contract the open pairs, we give a lemma to resolve some ambiguity about open pairs and cographs. Notice, in fact, that given a cograph G and the corresponding star system, there might be an open pair (x, y) such that $N_G(x) \neq N_G(y)$ (see Fig. 1). However, we will prove that given any open pair, we can always find a cograph G' isomorphic to G, such that $N_{G'}(x) = N_{G'}(y)$.

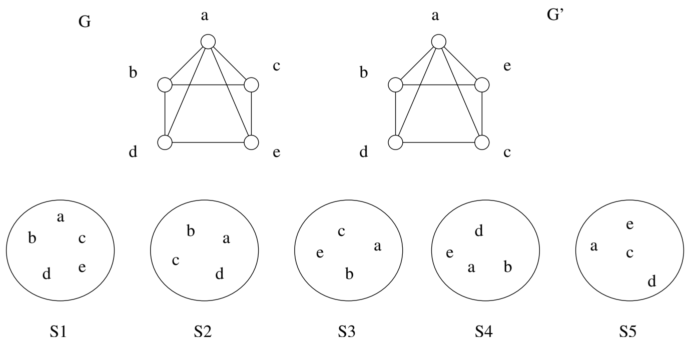


Fig. 1. If we consider the cograph G on the left, we can see that the vertices b and c form an open pair in the corresponding star system (the stars S_4 and S_5 satisfy the conditions), even though their open neighborhoods in the graph are different. However it is possible to find a cograph G' relabelling the vertices of G, such that: the two graphs have the same star system; b and c form an open pair; and $N_{G'}(b) = N_{G'}(c)$.

Lemma 3. *If $Stars(G) = \mathcal{S}$ for a cograph $G = (V, E)$ and (x, y) is an open pair of $\mathcal{S}$, there is a cograph $G' = (V', E')$ such that $Stars(G') = \mathcal{S}$, $xy \notin E'$ and $N_{G'}(x) = N_{G'}(y)$.*

Proof. If $xy \notin E$, then $N[x]$ contains x and not y, $N[y]$ contains y and not x, so by uniqueness of S_x and S_y we have that $N(x) = N(y)$. By letting $G' = G$ we are done. Now, let us assume $xy \in E$. Then there are vertices x' and y' such that $N[x']$ contains x and not y and $N[y']$ contains y and not x. Clearly, x, y, x' and y' must be distinct vertices. Following this, if $x'y' \notin E$, then $\{x', x, y, y'\}$ induces a P_4 in G. Thus, as G is a cograph, $x'y' \in E$. This means that $C = \{x', x, y, y'\}$ induces a C_4 in G.

We now proceed to show that C is a module of G, that is, for any $z \in V \setminus C$, either $N[z] \cap C = C$ or $N[z] \cap C = \emptyset$. Observe that as x and y are an open pair, $N[x'] \setminus \{x\} = N[y'] \setminus \{y\}$ so $x' \in N[z]$ if and only if $y' \in N[z]$. As x' is the only vertex such that $N[x']$ contains x but not y and y' is the only vertex such that $N[y']$ contains y and not x it follows that $x \in N[z]$ if and only if $y \in N[z]$. For the sake of contradiction, let us suppose that $x \in N[z]$ and $y' \notin N[z]$. Then, by the discussion above $x' \notin N[z]$ so $\{z, x, x', y'\}$ induces P_4 in G, contradicting that G is a cograph. Let us assume now that $x \notin N[z]$ and $y' \in N[z]$. Similarly to the previous case, $y \notin N[z]$ so $\{z, y', y, x\}$ induces a P_4 in G, again giving a contradiction. From this it follows that $x \in N[z]$ if and only if $y' \in N[z]$. Together with the equivalences above this proves that each of the vertices z, y', x' and x, is in $N[z]$ for a given z, if and only if the other three are as well. This means that C is a module of G.

We build G' from G by simply switching the labels of y and y'. We prove that G' meets the requirements of the statement of the Lemma. Clearly $(x, y) \notin E(G')$ and $N_{G'}(x) = N_{G'}(y)$. Furthermore $N_{G'}[z] = N[z]$ for any $z \in V \setminus C$. It remains to show that $\{N[x], N[y], N[x'], N[y']\} = \{N_{G'}[x], N_{G'}[y], N_{G'}[x'], N_{G'}[y']\}$. But $N[x] = N_{G'}[x']$, $N[x'] = N_{G'}[x]$, $N[y] = N_{G'}[y']$ and $N[y'] = N_{G'}[y]$. Thus $\mathit{Stars}(G') = \mathit{Stars}(G) = \mathcal{S}$ which concludes the proof.

We are now ready to give the contraction rule for open pairs.

Lemma 4. *Let (x, y) be an open pair of $\mathcal{S}$, and let $\mathcal{S}'$ be the set system obtained by deleting the unique star of S containing x but not y, and removing x from all the other stars of $\mathcal{S}$. There is a cograph G with $\mathit{Stars}(G) = \mathcal{S}$ if and only if there is a cograph F with $\mathit{Stars}(F) = \mathcal{S}'$.*

Proof. Suppose there is a cograph G with $\mathit{Stars}(G) = \mathcal{S}$. By Lemma 3, there exists a cograph G' for which $\mathit{Stars}(G') = \mathcal{S}$ and the only star in $\mathcal{S}$ containg x but not y, is exactly $N_{G'}[x]$. This means that removing the star representing $N_{G'}[x]$ from $\mathcal{S}$ and x from all the other sets of $\mathcal{S}$, we get exactly the star system of $F = G' \setminus \{x\}$ which clearly is a cograph.

Suppose now that there is a cograph F with $\mathcal{S}' = \mathit{Stars}(F)$. We build G from F by adding the vertex x and making x adjacent to the open neighbourhood of y. Clearly $\mathit{Stars}(G) = \mathcal{S}$. We prove that G is a cograph by obtaining a contradiction. Observe that both $G \setminus \{x\}$ and $G \setminus \{y\}$ are isomorphic to F, meaning that they cannot contain an induced P_4. Thus, if there is a P_4 in G, it contains both x and y. However x and y have the same open neighbourhood, which leads to a contradiction because no pair of distinct vertices of a P_4 has the same open neighbourhood.

We are now in the position to prove the main result of this subsection.

Theorem 1. *The P_4-free Star System Problem is solvable in $O(n^4)$ time.*

Proof. To decide whether a given set system $\mathcal{S}$ is a star system of a P_4-free graph, we use the following algorithm. If $\mathcal{S}$ has an open pair, apply Lemma 4 to create a new and smaller set system $\mathcal{S}'$ that is a star system of a cograph if and only if

$\mathcal{S}$ is. Apply the algorithm recursively on S'. If $\mathcal{S}$ has a closed pair, apply Lemma 2 to create a new and smaller set system $\mathcal{S}_R$ that is a star system of a cograph if and only if $\mathcal{S}$ is. Apply the algorithm recursively on $\mathcal{S}_R$. If $\mathcal{S}$ contains a single set on a single element, answer "yes". If neither of the above cases apply, answer "no". Correctness follows directly from Lemma 1. Let us argue for the runtime. We store our set system so that we can insert, delete and check membership in a set in constant time. At every step of the algorithm, if we do not answer "no", we reduce the set system by at least one element by applying Lemma 4 or Lemma 2. Hence the algorithm can have at most n main steps. To check whether a given pair is an open pair or a closed pair takes $O(n)$ time, therefore finding all closed, or all open pairs takes $O(n^3)$ time. When we apply Lemma 4 to reduce the graph we need to remove a set and an element from all other sets. This can be done in $O(n)$ time. When Lemma 2 is applied, we need to delete at most n elements from all sets, and then remove all duplicate sets. Deleting the elements takes $O(n^2)$ time, while finding and deleting all duplicates takes $O(n^3)$ time. Thus we can conclude that the algorithm terminates in $O(n^4)$ time.

3.2 Forbidding C_3 and C_4

In this subsection we show that the C_3-free and C_4-free Star System Problems are solvable in polynomial time.

Theorem 2. *The C_3-free Star System problem is solvable in $O(n^3)$ time.*

Proof. Let $\mathcal{S}$ be a set system on a ground set V. The crucial observation is that if $\mathcal{S}$ is a star system of a C_3-free graph $G = (V, E)$, then for every edge $uv \in E$ there are exactly two sets containing u and v. In fact, since $uv \in E$, we have that u and v should be in at least two stars, one of which is centered in u and one centered in v. Let S_u and S_v be these stars. If there is a third star S containing u and v, then the center of this star, $x \neq u, v$ is adjacent to u and v, and thus xuv forms a C_3 in G, which is a contradiction.

Let us assume that the system $\mathcal{S}$ is connected, i.e., for every two elements u and v there is a sequence of elements $u = u_1, u_2, \ldots, u_k = v$ such that for every $i \in \{1, \ldots, k-1\}$ there is a set $S \in \mathcal{S}$ containing u_i and u_{i+1}. (If $\mathcal{S}$ is not connected, then we apply our arguments for each connected component of $\mathcal{S}$.)

Assume that we have correctly guessed the star $S_v \in \mathcal{S}$ of a vertex v in some C_3-free graph G with $Stars(G) = \mathcal{S}$. Then each $x \in S_v$, $x \neq v$, is adjacent to v in G. Thus there is a unique star $S_x \neq S_v$ containing both v and x, and vertex x should be the center of S_x. Now every vertex y from S_x should have a unique star containing x and y, and so on. Since $\mathcal{S}$ is connected, we thus have that after guessing the star for the first vertex v we can uniquely assign stars to the remaining vertices. There are at most n guesses to be made for the first vertex and we can in $O(n^2)$ time check the correctness of the guess, i.e., check if the star system of the constructed graph corresponds to $\mathcal{S}$, to prove the theorem.

Theorem 3. *The C_4-free Star System Problem is solvable in $O(n^4)$ time.*

Proof. The proof is based on the following observation. Let $G = (V, E)$ be a C_4-free graph and let $x, y \in V$. Let $S_1, S_2, \ldots, S_t$ be the set of stars of G containing both x and y. Then

$$2 \leq |\bigcap_{i=1}^{t} S_i| \leq t \text{ if } xy \in E \tag{1}$$

$$t = 0 \text{ or } |\bigcap_{i=1}^{t} S_i| \geq t + 2 \text{ if } xy \notin E \tag{2}$$

In fact, if $xy \in E$, then x and y have $t - 2$ common neighbors. Every vertex $v \in \cap_{i=1}^{t} S_i \setminus \{x, y\}$ is adjacent to x and y, thus v is the center of the star S_i for some $i \in \{1, \ldots, t\}$ and (1) follows.

If $xy \notin E$ and $t > 0$, then x and y have t neighbors in common. Moreover, because G is C_4-free, these neighbors form a clique in G. Thus $\cap_{i=1}^{t} S_i$ contains all these t vertices plus the vertices x and y which yields (2).

Given a set system $\mathcal{S}$ on ground set V, the algorithm checking if $\mathcal{S}$ is a star system of some C_4-free graph is simple. We construct a graph $G = (V, E)$ with $xy \in E$ if and only if the sets of $\mathcal{S}$ containing both x and y satisfy (1). Finally, we check that $Stars(G) = \mathcal{S}$ and that G is C_4-free. If this is the case then the answer is yes, otherwise the answer is no.

4 Forbidding Long Paths and Cycles

In this section we show that there exists an infinite family of graphs H for which the H-free Star System problem is NP-complete. In particular both P_k and C_k, with $k > 4$, belong to it.

Definition 1. *For an arbitrary graph H, we define $B(H)$ to be its bipartite neighborhood graph, i.e., the bipartite graph with both color classes having $|V(H)|$ vertices labelled by $V(H)$ and having an edge between a vertex labelled u in one color class and a vertex labelled v in the other color class iff $uv \in E(H)$.*

For example, for the cycle on 5 vertices C_5, we have $\overline{C_5} = C_5$ and $B(\overline{C_5}) = C_{10}$. Our main NP-completeness result is that the H-free Star System problem is NP-complete whenever $B(\overline{H})$ has a cycle or two vertices of degree larger than two in the same connected component. For a bipartite graph $G = (V, E)$ with color classes V_1, V_2 we say that an automorphism $f : V \to V$ is side-switching if $f(V_1) = V_2$ and $f(V_2) = V_1$. Consider the following two problems.

AUT-BIP-2SS
Input: A bipartite graph G
Question: Does G have an automorphism of order 2 that is side-switching?

AUT-BIP-2SS-NA
Input: A bipartite graph G

Question: Does G have an automorphism of order 2 that is side-switching where every vertex and its image are non-adjacent?

Lalonde [10] has shown that the AUT-BIP-2SS problem is NP-complete. Together with Sabidussi he also reduced AUT-BIP-2SS to AUT-BIP-2SS-NA. The proof of our main NP-completeness result is a (nontrivial) refinement of the reduction of Lalonde-Sabidussi, which will ensure that AUT-BIP-2SS-NA remains NP-complete for various restricted classes of bipartite graphs.

To relate NP completeness of AUT-BIP-2SS-NA to the Star System Problem, we use the following lemma.

Lemma 5. *If AUT-BIP-2SS-NA is NP-complete on bipartite $B(\overline{H})$-free graphs, then the H-free Star System Problem is NP-complete.*

Proof. We reduce the first problem, which takes as input a bipartite $B(\overline{H})$-free graph F, to the second, which takes as input a set system S. We may assume the two partition sides of F are of equal size, since otherwise an automorphism switching the two sides cannot exist. Let the vertices of one color class of F be $\{v_1, v_2, \cdots, v_n\}$ and of the other $\{w_1, w_2, \cdots, w_n\}$. The set system we construct will be $S = \{S_1, S_2, \cdots, S_n\}$ where $S_i = \{w_j : v_i w_j \notin E(F)\}$, i.e., the non-neighbors of v_i on the other side.

As already noted by Babai [3], it is not hard to see that F is a Yes-instance of AUT-BIP-2SS-NA iff there exists a graph G with $Stars(G) = S$. Let us give the argument. The equivalence of those two problems is most naturally proved by noting that they are both equivalent to the question if the bipartite complement C_F of F, with $V(C_F) = V(F)$ and $E(C_F) = \{v_i w_j : v_i w_j \notin E(F)\}$, has an automorphism of order 2 such that every vertex *is adjacent* to its image, and thus also side-switching.

It remains to show that if $Stars(G) = S$ then G must be H-free. First note that if $Stars(G) = S$, then its bipartite closed neighborhood graph $C(G)$ - constructed by adding to its bipartite neighborhood graph $B(G)$ all $|V(H)|$ edges between pairs of vertices having the same label - is isomorphic to C_F. We therefore have that $B(\overline{G}) = F$, in other words, the bipartite neighborhood graph of the complement of G is isomorphic to F. Moreover, if H is an induced subgraph of G then clearly $B(\overline{H})$ is an induced subgraph of $B(\overline{G}) = F$ and thus since F is $B(\overline{H})$-free we must have G being H-free.

Definition 2. *Let D_p be the class of bipartite graphs of girth larger than p where any two vertices of degree three or more have distance at least p.*

Theorem 4. *For any integer p the problem AUT-BIP-2SS-NA is NP-complete even when restricted to graphs in D_p.*

Proof. We reduce from the NP-complete AUT-BIP-2SS problem and adapt the construction given by Lalonde and Sabidussi [10] for our purposes.

Given a bipartite graph $G = (V, E)$ with color classes A and B we describe how to construct $H \in D_p$ with the property that G is a yes-instance of AUT-BIP-2SS iff H is a yes-instance of AUT-BIP-2SS-NA. Note firstly that we can assume G

has no vertex v of degree 1 since if we remove each such v (simultaneously) and add a cycle of length $2k$, where k is greater than the maximum cycle length in G, attached to the unique neighbor of v, then G has a side-switching automorphism of order 2 if and only if the new graph has one.

Let p' be the smallest even integer at least as large as p. Let H be the graph obtained by replacing each edge of G by two paths of length $p'+1$. Note that the inner vertices of these paths are then the only vertices of degree 2 in H. Moreover, we have $H \in D_p$ and the two color classes of H respect A and B.

If $f : V(G) \to V(G)$ is an order-two side-switching automorphism of G, then define $g : V(H) \to V(H)$ as follows:

- $g(v) = f(v)$ for every $v \in A \cup B$,
- for the newly added vertices of degree 2, let $u, uv_1^1, uv_2^1, \ldots, uv_{p'}^1, v$ and $u, uv_1^2, uv_2^2, \ldots, uv_{p'}^2, v$ be the two paths joining u and v, and let $x, xy_1^1, xy_2^1, \ldots, xy_{p'}^1, y$ and $x, xy_1^2, xy_2^2, \ldots, xy_{p'}^2, y$ be the two paths joining $x = f(v)$ and $y = f(u)$. Then set $g(uv_j^i) = xy_{p'+1-j}^{3-i}$ for $i = 1, 2$ and $j = 1, 2, \ldots, p'$.

It is straightforward to see that g is an order-two side-switching automorphism. The only place where $ug(u)$ might be an edge would be in the middle of a path $u, uv_1^1, uv_2^1, \ldots, uv_{p'}^1, v$ when $f(u) = v$, but note that the vertices of one path are mapped onto vertices of the other one and $xg(x) \notin E(H)$ is fulfilled.

On the other hand, suppose $g : V(H) \to V(H)$ is an order-two side-switching automorphism of H. Since the original vertices of G have degrees greater than 2 in H, the restriction of g to $V(G)$ is a correctly defined mapping $g : V(G) \to V(G)$. Since the paths of length $p'+1$ uniquely correspond to edges of G, this restriction of g is an automorphism of G. It is obviously of order 2, and since the sides of H respect the sides of G, it is side-switching. (Note that we even did not need to assume that $ug(u) \notin E(H)$ for this implication.)

Definition 3. *Let H be a graph. We define a function $f(H)$ from graphs to integers and infinity. If $B(\overline{H})$ is acyclic with no connected component having two vertices of degree larger than two then let $f(H) = \infty$. Otherwise, let $f(H)$ be the smallest of i) the length of the smallest induced cycle of $B(\overline{H})$, and ii) the length of the shortest path between any two vertices of degree larger than two in $B(\overline{H})$.*

For example, for the cycle on 5 vertices C_5, we have $B(\overline{C_5}) = C_{10}$ and thus $f(C_5) = 10$. Note that if $f(H) \neq \infty$ then $D_{f(H)}$ is contained in the class of bipartite $B(\overline{H})$-free graphs. We therefore have the following Corollary of Lemma 5 and Theorem 4.

Corollary 1. *The H-free Star System Problem is NP-complete whenever $f(H) \neq \infty$. Moreover, if $\mathcal{F}$ is a set of graphs for which there exists an integer p such that for any $H \in \mathcal{F}$ we have $f(H) \leq p$, then the $\mathcal{F}$-free Star System Problem (i.e., deciding on an input S if there is a graph having no induced subgraph isomorphic to* any *graph in $\mathcal{F}$) is NP-complete.*

Since $B(\overline{C_k})$ contains a cycle for any $k \geq 5$ we have the corollary.

Corollary 2. *For any $k \geq 5$, the C_k-free Star System problem is NP-complete.*

Similarly, $B(\overline{P_k})$ is connected and contains at least 2 vertices of degree greater or equal to 3 for any $k \geq 5$. Hence we also have the following corollary.

Corollary 3. *For any $k \geq 5$, the P_k-free Star System problem is NP-complete.*

5 Closing Remarks

In this paper we obtained a complete dichotomy for the H-free Star System problem when the forbidden graph H is either a path or a cycle. Moreover, our NP-completeness result holds for H taken from a much larger family of graphs, so that the remaining cases in which the problem might not be NP-complete are very restricted. It is tempting to ask if the H-free Star System problem has a P vs NP-completeness dichotomy in general, i.e., whether for any graph H the H-free Star System problem is either polynomial-time solvable or NP-complete (and thus presumably not Graph Isomorphism-complete). See [4,9] for a discussion of such dichotomy results.

A closely related question is on the complexity of the Star System problem restricted to graph classes defined by several forbidden induced subgraphs as in Corollary 1. By the result of Aigner and Triesch [1,2] (see also [6]) we do then not have dichotomy in general, as there are classes of graphs defined by an infinite set of forbidden induced subgraphs (like forbidding the complements of odd cycles) such that the Star System problem is Graph Isomorphism complete on these classes. However, we do not know whether there is a graph class characterized by a *finite* set of forbidden induced subgraphs such that the Star System problem on this class is Graph Isomorphism complete, or if instead dichotomy may hold in this case.

References

1. Aigner, M., Triesch, E.: Reconstructing a graph from its neighborhood lists. Combin. Probab. Comput. 2, 103–113 (1993)
2. Aigner, M., Triesch, E.: Realizability and uniqueness in graphs. Discrete Math. 136, 3–20 (1994)
3. Babai, L.: Isomorphism testing and symmetry of graphs. Ann. Discrete Math. 8, 101–109 (1980)
4. Bulatov, A., Dalmau, V.: Towards a Dichotomy Theorem for the Counting Constraint Satisfaction Problem. In: Proceedings 44th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2003, pp. 562–570 (2003)
5. Corneil, D.G., Lerchs, H., Burlingham, L.S.: Complement reducible graphs. Discrete Appl. Math. 3, 163–174 (1981)
6. Boros, E., Gurvich, V., Zverovich, I.: Neighborhood hypergraphs of bipartite graphs, tech. rep., RUTCOR (2006)
7. Erdős, P., Gallai, T.: Graphs with prescribed degrees of vertices (in hungarian). Matematikai Lapok 11, 264–274 (1960)

8. Hajnal, A., Sós, V.: Combinatorics. vol. II, vol. 18 of Colloquia Mathematica Societatis János Bolyai. North-Holland, Amsterdam (1978)
9. Hell, P., Nesetril, J.: Graphs and homomorphisms. Oxford Lecture Series in Mathematics and its Applications, vol. 28 (2004)
10. Lalonde, F.: Le problème d'étoiles pour graphes est NP-complet. Discrete Math. 33, 271–280 (1981)
11. Lubiw, A.: Some NP-complete problems similar to graph isomorphism. SIAM J. Comput. 10, 11–21 (1981)

Optimization and Recognition for K_5-minor Free Graphs in Linear Time

Bruce Reed[1,2] and Zhentao Li[3]

[1] Canada Research Chair in Graph Theory
McGill University, Montreal, Canada
[2] Project Mascotte, INRIA
Laboratoire I3S, CNRS, Sophia-Antipolis, France
breed@cs.mcgill.ca
[3] School of Computer Science
McGill University, Montreal, Canada
zhentao.li@mail.mcgill.ca

Abstract. We present a linear time algorithm which determines whether an input graph contains K_5 as a minor and outputs a K_5-model if the input graph contains one. If the input graph has no K_5-minor then the algorithm constructs a tree decomposition such that each node of the tree corresponds to a planar graph or a graph with eight vertices. Such a decomposition can be used to obtain algorithms to solve various optimization problems in linear time. For example, we present a linear time algorithm for finding an $O(\sqrt{n})$ seperator and a linear time algorithm for solving k-realisation on graphs without a K_5-minor. Our algorithm will also be used, in a separate paper, as a key subroutine in a nearly linear time algorithm to test for the existence of an H-minor for any fixed H.

1 Results and Related Work

We say that H is a minor of G if H can be obtained from a subgraph of G via a sequence of edge contractions (see Figure 1). Many natural classes of graphs, such as planar graphs, are characterized by forbidding members of some obstruction set as a minor. In seminal work, Robertson and Seymour [22,23] gave, for any H, a decomposition theorem which yields structural properties of every H-minor free graph. This theorem allows us to obtain fast algorithms for optimization problems on such a class of graphs.

The complexity of the algorithm depends on the exact structure theorem obtained. For example, if H is planar then the H-minor free graphs have bounded tree width. Bodlaender [6] showed we can obtain the decomposition for such graphs in linear time. Using this decomposition, many optimization problems can be solved in linear time, including any problem which can be expressed in second order monadic logic [7] (see also: [24,2,4,5]).

For arbitrary H, a polynomial time algorithm to construct the decomposition was recently developed by Demaine et al. [10]. In the same paper these authors, building on earlier work of Baker [3], Grohe, and others, develop polynomial time algorithms for a number of optimization problems on this class of

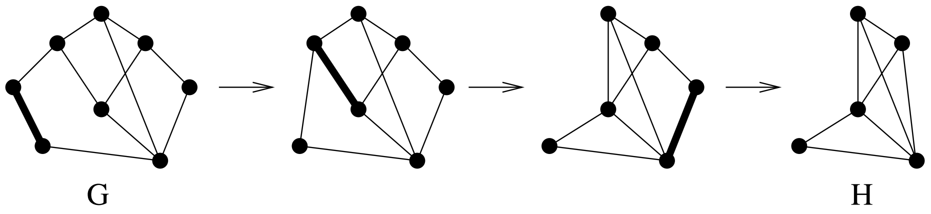


Fig. 1. H is minor of G. The sequence of edge contraction is shown. At each step, we contract the highlighted edge to obtain the next graph.

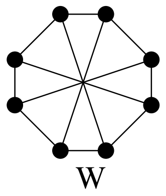


Fig. 2. The special graph W with 8 vertices and 12 edges

graphs. The exponent in the running time of the algorithm depends on $|V(H)|$. It exceeds $\binom{|V(H)|}{2}$. Kawarabayashi, Li, and Reed [14] have developed a near linear time algorithm (running time $n\alpha(n)\log(n)$ where α is the inverse Ackermann function) for constructing this decomposition. This allows them to solve many optimization problems in near-linear time. Their algorithm follows the Robertson-Seymour proof, but needs ideas from this paper and other tools.

In this paper, we restrict our attention to graphs which do not have a clique of order 5 (which we denote by K_5) as a minor. Planar graphs contain no such minors. Indeed Kuratowski [16] showed that a graph is planar precisely if it has neither K_5 nor $K_{3,3}$ as a minor. Wagner [26] proved that a 4-connected graph is planar precisely if it contains no K_5-minor. More strongly, he proved that a graph is K_5-minor free precisely if it is a subgraph of a graph which arises from planar graphs and the graph W of Figure 2 by repeatedly pasting together graphs on cliques. As discussed in detail below, this yields a tree decomposition of G whose nodes correspond to planar graphs.

We present a linear time algorithm which given a graph G either finds a K_5-minor in G or constructs such a tree decomposition. This improves on an $O(n^2)$ algorithm due to Kézdy and McGuiness [15]. This allows us to solve many optimization problems in linear time on K_5-minor free graphs. Some of these problems are discussed by Demaine et al [9,8] who develop $O(n^3)$ algorithms for the same problems. Another application is to find separators. We can find an $O(\sqrt{n})$ separator in linear time in K_5-minor free graphs extending results of Lipton, Tarjan and others for the planar case [17]. This improves an $O(n^{3/2})$ time algorithm due to Alon et. al [1], and also a linear time algorithm due to

Reed and Wood [20] which finds an $O(n^{2/3})$-separator. We give further details of these applications in the final section of the paper.

Our algorithm is recursive. Given an input graph G we construct an auxiliary graph G', solve our problem on G' and use the solution when obtaining a solution for G. We use three different reductions. Two are easy to handle. The third involves constructing G' by contracting the edges of a carefully chosen induced matching with at least $\epsilon|V(G)|$ edges, for some $\epsilon > 0$. If we find a K_5-minor in G', after uncontracting the edges we will have a K_5-minor in G. We only need to do any work if we return a tree decomposition of G'. It turns out that, because of the care we took when choosing the matching we contract, the 3-cuts of G are either subsets of the uncontraction of the 3-cuts of G' or correspond to a P_3 in one of the planar pieces of our decomposition of G'. This allows us to find them all quickly. It is then a simple matter to construct the desired decomposition of G (or find a K_5-minor).

The paper is organized as follows. We first discuss some structural results on K_5-minor free graphs. We then examine how the cutsets of size three in a graph can interact, developing some results which to us are of independent interest. We then describe the algorithm. Finally we turn to applications.

In what follows, an (i, j)-cut X has i vertices and $G - X$ has at least j components.

2 The Structure of K_5-minor Free Graphs

A graph has K_5 as a minor precisely if it has 5 disjoint connected subgraphs every two of which are joined by an edge. From this, it follows easily that G has K_5 as a minor precisely if one of its 2-connected components has such a minor. Furthermore, if $X = \{x, y\}$ is a cutset in G then G has a K_5-minor precisely if there is a component U of $G-X$ such that the graph obtained from the subgraph of G induced by $X \cup U$ by adding the edge xy has K_5 as a minor. Finally, if X is $(3, 3)$-cut then G has a K_5-minor precisely if there is a component U of $G - X$ such that the graph obtained from the subgraph of G induced by $X \cup U$ by adding edges so that X is a clique has K_5 as a minor.

On the other hand, Wagner [26] proved that if a 3-connected graph has no K_5-minor then either it is planar or it has a cutset X of size three such that $G-X$ has at least three components. Thus, by recursing on the subproblems discussed in the last paragraph, we can eventually reduce our problem to testing the planarity of some auxiliary graphs. To be more precise we need some definitions.

A 2-block tree is only defined for 2-connected graphs.

Definition 1. *A* 2-block tree *of a 2-connected graph G, written $[T, \mathcal{G}]$, is a tree T with a set $\mathcal{G} = \{G_t\}_{t \in T}$ with the following properties.*

- *G_t is a graph for each $t \in T$*
- *If G is 3-connected then T has a single node r which is coloured 1 and $G_r = G$.*
- *If G is not 3-connected then there exists a colour 2 node $t \in T$ such that*

1. *G_t is a graph with two vertices u and v and no edges for some 2-cut $\{u,v\}$ in G.*
2. *Let $T_1,\ldots,T_k$ be the connected components (subtrees) of $T-t$. Then $G-\{u,v\}$ has k components $U_1,\ldots,U_k$ and there is a labelling of these components such that T_i is a 2-block tree of $G_i = G[U_i \cup \{u,v\}] \cup \{uv\}$.*
3. *For each i, there exists exactly one colour 1 node $t_i \in T_i$ such that $\{u,v\} \subseteq G_{t_i}$.*
4. *For each i, $tt_i \in E(T)$.*

Note that as discussed below, this tree is not unique. It is essentially a refinement of the SPQR tree (see [11]).

A $(3,3)$-block tree is only defined for 3-connected graphs.

Definition 2. *A* (3,3)-block tree *of a 3-connected graph G, written $[T,\mathcal{G}]$, is a tree T with a set $\mathcal{G} = \{G_t\}_{t\in T}$ with the following properties.*

- *G_t is a graph for each $t \in T$*
- *If G has no $(3,3)$-cut then T has a single node r which is coloured 1 and $G_r = G$.*
- *If G has a $(3,3)$-cut then there exists a colour 2 node $t \in T$ such that*
  1. *G_t is a graph with vertices u,v and w and no edges for some $(3,3)$-cut $\{u,v,w\}$ in G.*
  2. *Let $T_1,\ldots,T_k$ be the connected components (subtrees) of $T-t$. Then $G-\{u,v,w\}$ has k components $U_1,\ldots,U_k$ and there is a labelling of these components such that T_i is a 3-block tree of $G_i = G[U_i \cup \{u,v,w\}] \cup \{uv,vw,uw\}$.*
  3. *For each i, there exists exactly one colour 1 node $t_i \in T_i$ such that $\{u,v,w\} \subseteq G_{t_i}$.*
  4. *For each i, $tt_i \in E(T)$.*

We call a colour 2 node a *cutting node*. We call a colour 1 node a *graph node*.

Given an input graph G, we first find the blocks of G using depth first search (see [25]), then construct the 2-block trees for the blocks, and finally construct the $(3,3)$-block tree for each graph node of the 2-block tree. In doing the latter, we use the fact, proven below, that unless G is $K_{3,3}$, there is a unique $(3,3)$-block tree for G ($K_{3,3}$ has two decompositions, as we can use either side of the bipartition as the unique cutset in a decomposition). We then test if all the graph nodes of each $(3,3)$-block tree are planar and if so find planar embeddings of them in linear time. If one of these graphs is non-planar then G has a K_5-minor.

We also use the fact that if G has more than $64|V(G)|$ edges then there is a simple linear time algorithm to find a K_5-minor in it, as discussed in [21].

3 Cutset Structure

In order to prove that the $(3,3)$-block tree for a 3-connected graph G other than $K_{3,3}$ is unique, it is enough to prove that every $(3,3)$-cut of G appears in a cutting node of every $(3,3)$-block tree. To do this, we need only show that if we start to construct a $(3,3)$-block tree using some $(3,3)$-cut X, then every $(3,3)$-cut Y of G is a $(3,3)$-cut of one of the auxiliary graphs corresponding to the components of $G-Y$. We say that Y separates X if there are vertices of X in different components of $G-Y$. It is not hard to see that we need only show that Y does not separate X. I.e., the uniqueness of the $(3,3)$-block trees follows from:

Lemma 3. *Let U be a 3-connected graph that is not $K_{3,3}$. Let X be a $(3,3)$-cut in U or a $(3,2)$-cut such that $U[X]$ is connected. There does not exist a $(3,3)$-cut separating the vertices of X.*

Proof. Suppose the lemma is false. Let $X=\{a,b,c\}$. Let $U_1,\dots,U_k$ be the connected components of $U-X$. Let Y be a $(3,3)$-cut separating X. WLOG, Y separates $\{a,b\}$.

Y must contain a vertex of U_1 and a vertex of U_2. Hence, every component of $U-Y$ which contains a neighbour of every vertex of Y must intersect X. Since there are three such components, we see that the vertices of X are in different components of $U-Y$. This implies that $G[X]$ is not connected and hence $k\geq 3$. Since $U[U_3\cup X]$ is connected, Y contains a vertex of U_3. Thus, X and Y are disjoint and the vertices of Y are in different components of $G-X$. Thus, each component of $G-X-Y$ has edges to at most one vertex of X and at most one vertex of Y. Since G is 3-connected, there are no such components and G is a $K_{3,3}$.

It follows easily from this lemma that the $(3,3)$-block tree is unique. We will also need the following similar results whose proofs we omit:

Lemma 4. *Let U be a 3-connected graph that is not $K_{3,3}$. Let X be a 3-cut in U which contains an edge. There does not exist a $(3,3)$-cut separating the vertices of X.*

Lemma 5. *Let U be a 3-connected graph that is not $K_{3,3}$. Let X be a $(3,3)$-cut in U. There does not exist a 3-cut Y separating X such that $G[Y]$ is connected.*

On the other hand, it is well known that 2-block trees are not unique. For example, every triangulation of a cycle corresponds to a 2-block tree for it, and these are all distinct. In fact, this is essentially the only way in which different 2-block trees of a graph can arise. This fact is captured in the two following lemmas (see [13,11]).

Lemma 6. *A 2-cut X in a 2-connected graph G whose vertices are joined by 3 internally vertex disjoint paths does not seperate the vertices of any other 2-cut.*

Lemma 7. *A 2-cut $\{x,y\}$ of a 2-connected graph G is in every 2-block tree of G precisely if either xy is an edge or x and y are joined by three internally vertex disjoint paths. Furthermore, reducing using these 2-cuts breaks G up into pieces each of which is either 3-connected or an induced cycle.*

This lemma implies a technical result which we will need:

Corollary 8. *The number of vertices in the 2-cuts of a 2-block tree is at most four times the number of cutting nodes in the tree.*

3.1 Cutset Structure Relative to a Reduction

In this section, we study the correspondence between the cuts in a 3-connected graph G and those in the graph H obtained by contracting an induced matching M in G.

To ease notation, we use f to refer to the function from $V(G)$ to $V(H)$ corresponding to the contraction of a fixed matching M. If v is in no edge of the matching then $f(v) = v$, so $H = f(G)$ and $G = f^{-1}(H)$.

Note that if Y is a cut in H then $f^{-1}(Y)$ is a cut in G. Therefore since M is matching and G is 3-connected, H is 2-connected. The converse is almost true for cutsets of size three in G. For any such 3-cut X, $|f(X)| = 3$ or $|f(X)| = 2$. If $f(X)$ is not a cut in H, then for some component U of $G - X$ we must have that $f(U) \subseteq f(X)$. If U has more than one vertex then there is an edge uv of U. This is not an edge of M as otherwise $f(U) \not\subseteq f(X)$. Since M is induced, we can therefore assume WLOG that $f(u) = u$. But, again, we contradict that $f(U) \subseteq f(X)$.

So U is a vertex u. Since $f(U) \subseteq f(X)$, we know that xu is an edge of the matching M for some vertex x of X. Since M is induced it follows that $f(v) = v$ for every other neighbour v of v and hence $N(u) \subseteq X$. Since u has at least three neighbours, it follows that v has exactly three neighbours x, y, z in G and $X = \{x, y, z\}$. Thus the only cutsets of size three in G whose image in H is not a cutset are $N(u)$ for vertices u of degree three which are in the matching and for which $G - u - N(u)$ is connected.

We are actually more interested in which $(3, 3)$-cuts of G do not correspond to $(2, 3)$-cuts or $(3, 3)$-cuts of H. If this is to happen then for some component U of $G - X$ $f(U) \subseteq f(X)$.

Mimicing the arguments above, we see that this occurs precisely if $U = \{u\}$ and $X = N(u)$ for some vertex u which has degree three, is in the matching and for which $G - u - N(u)$ has exactly two components. Note that in this case $f(X)$ is a $(3, 2)$-cut of H which induces a P_3 or a triangle.

In summary then, the $(3, 3)$-cuts of G are of one of the following types:

1. $f(X)$ is a $(2, 3)$-cut of H
2. $f(X)$ is a $(3, 3)$-cut of H,
3. $f(X)$ is $(3, 2)$ cut of H which induces a connected subgraph of H, and $X = N(v)$ for some vertex v of degree 3 in G which is in the matching we contract.

4 The Algorithm

We are now ready to describe our algorithm. We consider only the 3-connected case, as it is easy to construct the block tree and 2-block tree of a graph in linear time(see [25,13,12]). We need the following:

Theorem 9 *Let G be a graph of minimum degree three with at most $64|V(G)|$ edges. Let $d = 100000$ and let $\epsilon = d^{-6}$. In linear time we can find one of the following:*

1. *A set S of at least $5\epsilon|V(G)|$ vertices of degree 3 which have the same neighbourhood as at least $d+1$ other such vertices.*
2. *A matching M in the subgraph of G induced by the vertices of degree at most d which has size $\geq d^4\epsilon|V(G)|$.*
3. *A minor G' of G such that $|E(G')| \geq 64|V(G')|$, or G' is a subdivision of K_5, in which case we find and return a K_5-model in G.*

Our algorithm starts by obtaining one of the structures 1.,2., or 3. of this theorem in linear time. To do so, this algorithm considers a maximal matching M in the subgraph induced by the set X of vertices of degree less than d. If this matching is large enough, we can return with output 2. If we do not find such a matching, we use a greedy assignment algorithm and bucketsort to attempt to obtain output 1. If we fail we will find a minor of H which has so many edges that we can use the linear time algorithm of [21] to find a K_5-minor.

If output 3 is returned then we are essentially done. If it returns output 1., we recurse on $G - S$. For each vertex v in S, the neighbourhood of v will be a cutting node of $G - S$, and we can add a graph node which is a leaf incident to this node and contains the clique on $v + N(v)$. Adding all these nodes at once yields the $(3,3)$-block tree for G.

If we return with output 2., then we contract M and look at the 2-block tree for the resultant graph. If there are many 2-cuts then looking at these 2-cuts allows us to find many 3-cuts in G. If there are only a few 2-cuts, then our technical corollary tells us that we can find a small set of matching edges such that by uncontracting these edges we end up with a graph with no 2-cuts. Thus, from M we construct either an induced matching N with at least ϵn edges such that contracting the edges of N in G yields a 3-connected graph G_N, or a set of 3-cuts of G each of which contains an edge of M and none of which seperates another, which decompose G into at least ϵn pieces. In the latter case, we recurse by decomposing on this set of 3-cuts. In the former case we recurse on G_N. We then uncontract N and construct the $(3,3)$-block tree of G from the $(3,3)$-block tree of G_N, using our results on the cutset structure relative to a reduction. These tell us that every $(3,3)$-cut of G corresponds either to a $(3,3)$-cut of G_N or to a P_3 in one of the planar graph nodes. Those interested in further details may consult `http://cgm.cs.mcgill.ca/~breed/newk5free.ps` which contains the full details and will eventually contain a pointer to the journal version of this paper.

5 Running Time

We now briefly sketch some of the techniques used to analyse the algorithm's running time. For full details, we refer the readers to `http://cgm.cs.mcgill.ca/~breed/newk5free.ps`. First, we show that if we use linear time to process each graph before and after the recursion and the total size of the graphs we recurse on is a fraction of the size of the original graph, the total running time of the algorithm is linear. Then, we use greedy algorithms and bucksort to obtain graphs to recurse on in linear time. It is easy to return a K_5-model given a K_5-model on a graph we recursed on. To obtain a $(3,3)$-block tree or a K_5-model from a $(3,3)$-block tree obtained recursively in linear time, we examine the pre-images of the cutting nodes and the graph nodes of the $(3,3)$-block tree. The graph nodes are planar as a non-planar graph node contains a K_5-minor which we can find. In each such graph node, we find a special set of 3-cuts which are the $(3,3)$-cuts we missed as discussed in Section 3.1. We omit further details.

6 Applications

6.1 Finding a Separator

We now explain how our algorithm can be combined with a linear time algorithm for finding a $O(\sqrt{n})$ separator in a planar graph to find a $O(\sqrt{n})$ separator in a graph without a K_5-minor in linear time. In this abstract, we restrict our attention to the 3-connected case.

We build the $(3,3)$-block tree for G. Using dynamic programming, for each node t of the tree (cutting or graph) corresponding to a planar graph H_t with vertex set V_t, we can compute the number of vertices in each component of $G - V_t$. It is easy to see that there is some t such that every such component contains at most half the vertices of G. We consider such a node t. If t is a cutting node then X_t is a cutset of size three. Otherwise, for every triangular face f of H_t, we compute the number of vertices n_f in components of $G - V_t$ attached to this face, and add a vertex v_f in f and a set X_f of $n_f - 1$ vertices of degree 1 incident only to v_f. We then find a $O(\sqrt{n})$ separator S in the resultant planar graph. We obtain a separator in G of size at most $3s$ by taking $S \cap V(G)$ along with the vertices of any triangular face f for which S intersects $v_f \cup X_f$. We can then use this separator to solve various optimization problems on G. See [18] for further details.

6.2 *k*-Realizations

The k-Realization problem (k fixed) has the following form. Given a graph G, and a set X of k vertices of G, find all partitions $\Delta = (D_1, \ldots, D_l)$ of X such that there is a set of disjoint trees $\{T_1, \ldots, T_l\}$ with $D_i \subseteq V(T_i)$. This problem can be solved in $O(n^3)$ time in general using Robertson and Seymour's seminal results. In [19], Reed et al. showed that this problem could be solved in linear time on planar graphs. Our decomposition allows us to extend this result to graphs with no K_5-minor. We first consider the 3-connected case.

We construct the $(3,3)$-block tree for G. If there is a graph node which is a leaf of the $(3,3)$-block tree which contains none of X, then we can simply contract it to a vertex without changing the answer to the problem. Indeed, a similar linear time reduction allows us to reduce to a 3-connected minor H of G such that the $(3, 3)$-block tree $[T, \mathcal{H}]$ for H has at most k leaves. We now actually solve the Realization problem for $X \cup Y$ where Y is the set of (at most $3k$) vertices which appear in the cutting nodes of our tree decomposition of H. We do so by restricting our attention to the planar instance $(H'_t, (X \cup Y) \cap H_t)$ of k-Realizations for each graph node t where H' is the graph obtained from H by deleting the extra edges we added for the decomposition. H'_t is planar. We can then combine these solutions because G is obtained by pasting together these subgraphs on cutsets contained in $X \cup Y$.

If G is 2-connected there are further technical complications, but no major additional difficulties.

6.3 Further Applications

Many other optimization problems can be solved in linear time using the decomposition we have developed. These include most of the problems discussed in [9] and [8]. For example, in [8], many problems are solved in $O(n^2)$ time on K_5-minor free graphs by applying Theorem 5 which says that we can find a tree decomposition of a certain subgraph in quadratic time for K_5-minor free graphs. But as discussed in that paper, we can actually find the desired tree decompositions in linear time if the graph is planar (and hence has no $K_{3,3}$ minor). So in linear time, we can find such tree decompositions of all the pieces in our $(3, 3)$-block tree for these subgraphs. We can then paste these together to obtain a tree decomposition of the whole subgraph. Since we are pasting on cliques it is an easy matter to carry out this second step in linear time. Further details will be given in the full version of our paper.

7 More Details

A longer version of this paper can be found at `http://cgm.cs.mcgill.ca/~breed/newk5free.ps`. The version found there is a draft but will allow referees to determine correctness.

References

1. Alon, N., Seymour, P., Thomas, R.: A separator theorem for nonplanar graphs. Journal of the American Mathematical Society 3(4), 801–808 (1990)
2. Arnborg, S., Proskurowski, A.: Linear time algorithms for np-hard problems restricted to partial k-trees. Discrete Applied Mathematics 23, 11–24 (1989)
3. Baker, B.S.: Approximation algorithms for NP-complete problems on planar graphs. J. Assoc. Comput. Mach. 41, 153–180 (1994)
4. Bodlaender, H.L.: Dynamic programming algorithms on graphs of bounded treewidth. In: Lepisto, T., Salomaa, A. (eds.) 15th International Colloquium on Automata, Languages and Programming, vol. 317, pp. 105–118. Springer, Heidelberg (1988)
5. Bodlaender, H.L.: Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. Journal of Algorithms 11, 631–643 (1990)

6. Bodlaender, H.L.: A linear time algorithm for finding tree-decompositions of small treewidth. SIAM Journal on Computing 25, 1305–1317 (1996)
7. Courcelle, B.: The monadic second order logic of graphs. I. recognizable sets of finite graphs. Information and Computation 85, 12–75 (1990)
8. Demaine, E., Hajiaghayi, M., Nishimura, N., Ragde, P., Thilikos, D.: Approximation algorithms for classes of graphs excluding single-crossing graphs as minors. Journal of Computer and System Sciences 69(2), 166–195 (2004)
9. Demaine, E., Hajiaghayi, M., Thilikos, D.: Exponential speedup of fixed parameter algorithms on k. Algorithmica 41(4), 245–267 (2005)
10. Demaine, E.D., Hajiaghayi, M., Kawarabayashi, K.: Algorithmic graph minor theory: Decomposition, approximation and coloring. In: Proc. 46th Ann. IEEE Symp. Found. Comp. Sci., pp. 637–646 (2005)
11. Di Battista, G., Tamassia, R.: On-line maintenance of triconnected components with SPQR-trees. Algorithmica 15, 302–318 (1996)
12. Gutwenger, C., Mutzel, P.: A linear time implementation of spqr-trees. In: Marks, J. (ed.) Graph Drawing, Colonial Williamsburg, 2000, pp. 77–90. Springer, Heidelberg (2001)
13. Hopcroft, J.E., Tarjan, R.E.: Dividing a graph into triconnected components. SIAM J. Comput. 3, 135–158 (1973)
14. Kawarabayashi, K., Li, Z., Reed, B.: Near linear time algorithms for optimization and recognition for minor closed families (in preparation)
15. Kézdy, A., McGuinness, P.: Sequential and parallel algorithms to find a k5 minor. In: Third Annual Symposium on Discrete Algorithms, pp. 345–356. Springer, Heidelberg (1992)
16. Kuratowski, C.: Sur le problème des courbes gauches en topologie. Fundamenta Mathematica 16, 271–283 (1930)
17. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM Journal on Applied Mathematics 36(2), 177–189 (1979)
18. Lipton, R.J., Tarjan, R.E.: Applications of a planar separator theorem. SIAM J. Comput. 9, 615–627 (1980)
19. Reed, B., Robertson, N., Schrijver, L., Seymour, P.: Finding disjoint trees in planar graphs in linear time. In: Graph Structure Theory, pp. 295–302. AMS (1993)
20. Reed, B., Wood, D.: Fast separation in a graph with an excluded minor. In: Euro-Conference on Combinatorics, Graph Theory and Applications, pp. 45–50 (2005)
21. Robertson, N., Seymour, P.D.: Graph minors. XIII: the disjoint paths problem. J. Comb. Theory Ser. B 63(1), 65–110 (1995)
22. Robertson, N., Seymour, P.D.: Graph minors. XVI. excluding a non-planar graph. Journal of Combinatorial Theory, Series B 89, 43–76 (2003)
23. Robertson, N., Seymour, P.D.: Graph minors. XX. wagner's conjecture. J. Comb. Theory Ser. B 92(2), 325–357 (2004)
24. Lagergren, J., Arnborg, S., Seese, D.: Easy problems for tree-decomposable graphs. Journal of Algorithms 12, 308–340 (1991)
25. Tarjan, R.: Depth-first search and linear graph algorithms. SIAM J. Comput. 1, 146–160 (1972)
26. Wagner, K.: Über eine eigenschaft der ebenen komplexe. Math. Ann. 114, 570–590 (1937)

Bandwidth of Bipartite Permutation Graphs in Polynomial Time*

Pinar Heggernes[1], Dieter Kratsch[2], and Daniel Meister[1]

[1] Department of Informatics, University of Bergen, PO Box 7803, N-5020 Bergen, Norway
`Pinar.Heggernes@ii.uib.no, Daniel.Meister@ii.uib.no`

[2] Laboratoire d'Informatique Théorique et Appliquée, Université Paul Verlaine - Metz, 57045 Metz Cedex 01, France
`kratsch@univ-metz.fr`

Abstract. We give the first polynomial-time algorithm that computes the bandwidth of bipartite permutation graphs. Prior to our work, polynomial-time algorithms for exact computation of bandwidth were known only for caterpillars of hair length 2, chain graphs, cographs, and interval graphs.

1 Introduction

The bandwidth problem asks, given a graph G and an integer k, whether there exists a linear layout of the vertices of G such that no edge of G has its endpoints mapped to positions with difference more than k. The problem is motivated from Sparse Matrix Computations, where given an $n \times n$ matrix A and an integer k, the goal is to decide whether there is a permutation matrix P such that PAP^T is a matrix with all nonzero entries on the main diagonal or on the k diagonals on either side of the main diagonal [9]. The graph and the matrix version of the bandwidth problem are equivalent, and both have been studied extensively in the last 40 years.

The bandwidth problem is NP-complete [18], and it remains NP-complete even on very restricted subclasses of trees, like caterpillars of hair length at most 3 [17]. Bandwidth is a benchmark problem known for its difficulty among the often studied NP-hard graph problems. With respect to parameterized complexity [5], the bandwidth problem (with parameter k) is $W[k]$-hard [4]. Thus, not only is it unlikely that an $\mathcal{O}(f(k) \cdot p(n))$-time algorithm exists for its solution with an arbitrary function f and a polynomial p, but it is also much harder than most other well-studied graph problems with respect to parameterized complexity.

Due to the difficulty of the bandwidth problem, approximation algorithms for it attracted much attention. For any constant c, it is NP-hard to compute a c-approximation of the bandwidth of general graphs [20] and even of trees [2].

* This work is supported by the Research Council of Norway through grant 166429/V30. In this extended abstract most proofs are omitted; they can be found in a technical report [12].

Consequently approximation algorithms on restricted graph classes have been received with great interest by the research community, and approximation algorithms have been given for the bandwidth of trees and even caterpillars [7,10,11]. Constant factor approximation algorithms for the bandwidth of AT-free graphs and subclasses of them, including permutation graphs, exist [15]. There are approximation algorithms for the bandwidth of general graphs using advanced techniques [3,6].

Polynomial-time algorithms to compute the bandwidth exactly are known for only a few and very restricted graph classes: caterpillars with hair length at most two [1], chain graphs [14], and cographs [21]. The outstanding result is the polynomial-time algorithm of Kleitman and Vohra that computes the bandwidth of interval graphs [13]. The knowledge on the algorithmic complexity of bandwidth on particular graph classes did not advance much during the last decade. The only progress was made when the NP-completeness of bandwidth of cocomparability graphs was observed, and simple 2-approximation algorithms for the bandwidth of permutation graphs were given [15,16]. Permutation graphs are precisely those graphs for which the graph and its complement are cocomparability, and thus a subclass of cocomparability graphs. However, the algorithmic complexity of the bandwidth problem on permutation graphs remained open for a long time, and is still open. Despite various attempts since the late 1980's, not even the computational complexity of bandwidth of bipartite permutation graphs was resolved prior to our work.

In this paper, we give the first polynomial time algorithm to compute the bandwidth of bipartite permutation graphs, with running time $O(n^4 \log n)$. Our algorithm is based on structural properties of bipartite permutation graphs, in particular the use of strong orderings. Moreover we rely heavily on a deep theorem concerning linear extensions and linear labelings of posets, that for cocomparability graphs guarantees the existence of an optimal bandwidth layout which is a cocomparability ordering [8]. Finally, a novel local exchange algorithm to find so called normalized (partial) k-layouts is the key algorithmic idea of our work. No tools from previous bandwidth algorithms for special graph classes have been used; rather, our algorithm is especially tailored for bipartite permutation graphs through non-standard techniques.

2 Preliminaries

A graph is denoted by $G = (V, E)$, where V is the set of vertices with $n = |V|$ and E is the set of edges with $m = |E|$. The set of neighbors of a vertex v is denoted by $N(v)$, and $N[v] = N(v) \cup \{v\}$. Similarly, for $S \subseteq V$, $N[S] = \bigcup_{v \in S} N[v]$, and $N(S) = N[S] \setminus S$. The subgraph of G induced by the vertices in S is denoted by $G[S]$. For $G' = G[S]$ and $v \in V \setminus S$, $G'+v$ denotes $G[S \cup \{v\}]$, and for any $v \in V$, $G-v$ denotes $G[V \setminus \{v\}]$.

For a given graph $G = (V, E)$ with $V = \{v_1, v_2, ..., v_n\}$, a *layout* $\beta : \{1, \ldots, n\} \to V$ of G is an ordering $(v_{\pi(1)}, \ldots, v_{\pi(n)})$ where π is a permutation of $\{1, \ldots, n\}$. The *distance* between two vertices u, v in a layout β is $d_\beta(u, v) = |\beta^{-1}(u) -$

$\beta^{-1}(v)|$. For a given layout (or ordering) β, we write $u \prec_\beta v$ when $\beta^{-1}(u) < \beta^{-1}(v)$. For a vertex v in G, every vertex u with $u \prec_\beta v$ is *to the left* of v, and every vertex w with $v \prec_\beta w$ is *to the right* of v in β. We will also informally write *leftmost* and *rightmost* vertices accordingly.

For an integer $k \geq 0$, we call β a *k-layout* for G if, for every edge uv of G, $d_\beta(u,v) \leq k$. The *bandwidth* of G, $bw(G)$, is the smallest k such that G has a k-layout. In this paper a layout will be called *optimal* if it is a $bw(G)$-layout for G.

Bipartite permutation graphs are permutation graphs that are bipartite. Let $G = (A, B, E)$ be a bipartite graph. Sets A and B are called *color classes.* A *strong ordering* for G is a pair of orderings (σ_A, σ_B) on respectively A and B such that for every pair of edges ab and $a'b'$ in E with $a, a' \in A$ and $b, b' \in B$, $a \prec_{\sigma_A} a'$ and $b' \prec_{\sigma_B} b$ imply that ab' and $a'b$ are in E. The following characterization of bipartite permutation graphs is the only property that we will need in this paper.

Theorem 1 ([19]). *A bipartite graph is a bipartite permutation graph if and only if it has a strong ordering.*

Spinrad et al. give a linear time recognition algorithm for bipartite permutation graphs that produces a strong ordering if the input graph is bipartite permutation [19]. It follows from the definition of a strong ordering that if $G = (A, B, E)$ is a connected bipartite permutation graph then any strong ordering (σ_A, σ_B) satisfies the following. For every vertex a of A, the neighbors of a appear consecutively in σ_B. Furthermore, if $N(a) \subseteq N(a')$ for two vertices $a, a' \in A$ then a is adjacent to the leftmost or the rightmost neighbor of a' with respect to σ_B.

An ordering σ of a graph G is called a *cocomparability ordering* if for all u, v, w with $u \prec_\sigma v \prec_\sigma w$, $uw \in E$ implies that $uv \in E$ or $vw \in E$. A graph that has a cocomparability ordering is called a *cocomparability graph.* (Bipartite) permutation graphs are cocomparability graphs.

Let V be a set, and let $\prec_P$ be a binary reflexive, antisymmetric and transitive relation over V. Then $P = (V, \prec_P)$ is called a *partially ordered set.* A *linear extension* β of P is a layout of V satisfying $a \prec_P b \Rightarrow a \prec_\beta b$. Hence for all pairs of elements of V, a linear extension preserves their order relation of P. For an integer $k \geq 0$, a *k-linear labeling* for P is a linear extension β of P such that for every pair a, b of elements of V, $a \neq b$: $a \prec_P b \Rightarrow d_\beta(a, b) \leq k$. Fishburn et al. showed an interesting connection between linear labelings of partially ordered sets and the bandwidth of graphs [8]. The *incomparability graph* $G = G(P)$ of a partially ordered set P has vertex set V, and two vertices are adjacent if and only if the corresponding elements $a \neq b$ of V are not in relation in P (neither $a \prec_P b$ nor $b \prec_P a$). It is well-known that if β is a linear extension of P then β is a cocomparability ordering of the incomparability graph $G = G(P)$, and vice versa.

Theorem 2 ([8]). *Let $P = (V, \prec_P)$ be a partially ordered set, where V is finite. Let $k \geq 0$. Then, P has a k-linear labeling if and only if the incomparability graph of P has bandwidth at most k.*

For each cocomparability graph G, there is a partially ordered set P such that G is the incomparability graph of P. Therefore, Theorem 2 implies that every cocomparability graph G has an optimal layout β such that β is a linear extension of P, and thus a cocomparability ordering of G. We shall heavily rely on the following consequence of this for connected bipartite permutation graphs.

Corollary 1. *Let $G = (A, B, E)$ be a connected bipartite permutation graph, and let $k \geq 0$ be an integer. Let (σ_A, σ_B) be a strong ordering for G. If G has a k-layout then G has a k-layout β (a cocomparability ordering) satisfying the following two conditions:*

(C1) *for every pair a, a' of vertices from A, $a \prec_{\sigma_A} a'$ implies $a \prec_\beta a'$, and for every pair b, b' of vertices from B, $b \prec_{\sigma_B} b'$ implies $b \prec_\beta b'$*

(C2) *for every triple a, b, b' of vertices of G where $a \in A$ and $b, b' \in B$ and $ab \in E$ and $ab' \notin E$, neither $a \prec_\beta b' \prec_\beta b$ nor $b \prec_\beta b' \prec_\beta a$.*

3 Bandwidth of Bipartite Permutation Graphs

We call a layout of a connected bipartite permutation graph *normalized* if it satisfies conditions (C1) and (C2) with respect to some given strong ordering. Hence, to decide whether a given connected bipartite permutation graph has a k-layout for some $k \geq 0$, it suffices to check normalized k-layouts. In this section we give the main idea behind our algorithm, and these arguments will be used to prove the correctness of the final algorithm presented in Section 5.

For the rest of the paper, we let $G_i = G[N[\{a_1, a_2, ..., a_i\}]]$ for $1 \leq i \leq s$. If there is a normalized k-layout α for G_i, α satisfies one of the following two conditions, which we will analyze separately.

1. $\alpha^{-1}(a_i) > |V(G_{i-1})|$: To check whether a normalized k-layout α exists for G_i that satisfies this condition, we place all vertices of $N[a_i] \setminus V(G_{i-1})$ to the right of the rightmost vertex of G_{i-1} according to β. The order of the newly added B-vertices is according to σ_B, and we place a_i as far to the left as its rightmost neighbor allows (at most k positions away from the end), but not further left than position $|V(G_{i-1})| + 1$. Let us call this new layout for G_i, β'. Let b be the leftmost neighbor of a_i in β'. If $d_{\beta'}(a_i, b) \leq k$ then β' is the desired k-layout and we are done. In the opposite case, we need to move $d_{\beta'}(a_i, b) - k$ A-vertices between b and a_i to the left of b. If there are fewer A-vertices between b and a_i, it means that there are more than $d_{\beta'}(a_i, b) - k$ B-vertices between them. In this case, the distance between b and a_i cannot be reduced, and since a_i is to the right of all B-vertices of G_{i-1} by our assumption on α, we conclude that a desired k-layout does not exist. Otherwise, let a be the $(d_{\beta'}(a_i, b) - k)$th closest A-vertex to b to the right of b. *A normalized k-layout α as assumed exists if and only if there is a normalized k-layout β_* of G_{i-1} where b appears to the right of a.* To see this, observe that none of the vertices in $N(a_i) \setminus V(G_{i-1})$ has neighbors in G_{i-1}, and b is the leftmost neighbor of a_i in every normalized k-layout of G_i, since b cannot exchange places with other B-vertices in such a layout. Thus appending the layout of $N[a_i] \setminus V(G_{i-1})$ as described above to the end of β_* gives a k-layout for G_i. Checking whether β_* exists for G_{i-1} and how to compute it from β is one of the two key points of our algorithm, and the next section is devoted to this task. The case that we have explained in this paragraph will be resolved by Theorem 4.

2. $\alpha^{-1}(a_i) \leq |V(G_{i-1})|$: To check whether a normalized k-layout α exists for G_i that satisfies this condition, we place all vertices of $N(a_i) \setminus V(G_{i-1})$ to the right of the rightmost vertex of G_{i-1} according to β. The order of the newly added B-vertices is according to σ_B. *A normalized k-layout α for G_i as assumed in this case exists if and only if there is a normalized k-layout β_* for $G_{i-1}+a_i$ where a_i is placed between b_{p-1} and b_p* (there might be other A-vertices between b_{p-1} and b_p as well) *for some B-vertex b_p in G_{i-1} with $p \geq 2$, and $\beta_*^{-1}(a_i) \geq |V(G_i)| - k$.* To see this, observe that a_i is the only vertex in $G_{i-1}+a_i$ that has neighbors in $N(a_i) \setminus V(G_{i-1})$. Hence if β_* is a k-layout for $G_{i-1}+a_i$, the ordering obtained by appending the vertices of $N(a_i) \setminus V(G_{i-1})$ to the end of β_* gives a k-layout for G_i. The condition $\beta_*^{-1}(a_i) \geq |V(G_i)| - k$ is necessary since a_i is adjacent to the rightmost vertex in every normalized k-layout of G_i satisfying the condition of this case. To see that we do not need to consider the case where a_i is moved to the left of b_1, notice that if there is a normalized k-layout for $G_{i-1}+a_i$ where a_i appears to the left of b_1, then all A-vertices appear before all B-vertices, and exchanging the places of a_i and b_1 results also in a normalized k-layout since a_i has no neighbors to its left, and b_1 has no neighbors to its right. Thus we need to check for each B-vertex b_p of G_{i-1} whether there is a normalized layout β_* for $G_{i-1}+a_i$ where a_i is placed between b_{p-1} and b_p. We check this for at most k of the rightmost B-vertices in β, since otherwise the distance between a_i and its rightmost neighbor will be too large. Placing a_i between b_{p-1} and b_p might of course require moving other A-vertices. How to check whether such a layout exists for $G_{i-1}+a_i$ is the second of the two key points of our algorithm, and it will be handled in the next section. The case that we have explained in this paragraph will be resolved by Theorem 5.

4 Deciding the Existence of Desired Layouts

We need to decide, given a normalized k-layout for G_{i-1}, whether there exists a normalized k-layout for G_{i-1} where a given B-vertex b is required to appear to the left of a given A-vertex a, and whether there exists a normalized k-layout for $G_{i-1}+a_i$ where a_i appears between two given B-vertices b_{p-1} and b_p. To check this, our approach is to indeed place the vertices as desired, and then check if the modified layout can be repaired to become a normalized k-layout. For the first question, we want to place b immediately to the right of a, forbid b to move left, and check whether this layout can be turned into a k-layout with this restriction. For the second question, we want to place a_i between b_{p-1} and b_p, forbid it to move in any direction, and check whether this layout can be turned into a k-layout with this restriction. Hence, we need an algorithm that takes as input a given layout with restrictions on how the vertices are allowed to move, and checks whether this can be turned into a normalized k-layout.

In this section we present exactly such an algorithm. Let u and v be adjacent vertices of G. We want to obtain another normalized layout by moving u one position closer to v. This is possible if and only if there is a vertex of the color class of v between u and v in β. Let w be such a vertex that is closest to u. We define *the layout obtained from β by moving u one position closer to v* to be layout β' which we obtain by exchanging the position of w with the vertex next to it in the direction towards u repeatedly until w is next to u, and then exchanging the positions of u and w. It is important to note that whenever two consecutive vertices exchange positions, they are neighbors in G. The orderings defined by β and β' restricted to A or B are equal, v has the same position in β and β', and $d_{\beta'}(u, v) = d_\beta(u, v) - 1$.

Lemma 1. *Let $G = (A, B, E)$ be a connected bipartite permutation graph, and let (σ_A, σ_B) be a strong ordering for G. Let β be a normalized layout for G. Let u and v be adjacent vertices, and let there be a vertex of the color class of v between u and v in β. The layout obtained from β by moving u one position closer to v is normalized.*

Now we formalize how to restrict the positions of certain vertices when modifying a given layout. A *direction assignment* h on the vertices of a graph is a function that assigns one of the following four symbols to each vertex: $\cdot, \leftarrow, \rightarrow, \leftrightarrows$. These symbols stand for directions in which a vertex can be moved relative to a given initial layout. A vertex v with $h(v) = \leftarrow$ can be moved only to the left, and a vertex with $h(v) = \rightarrow$ can only be moved to the right. If $h(v) = \leftrightarrows$ then v can be moved in any direction, whereas v cannot at all change position if $h(v) = \cdot$. Let $k \geq 0$, let β be a normalized layout for G, and let h be a direction assignment for G. We define $\Delta(\beta, h)$ to be the set of normalized k-layouts γ for G satisfying the following three properties for every vertex x of G: if $h(x) = \cdot$ then $\gamma^{-1}(x) = \beta^{-1}(x)$; if $h(x) = \leftarrow$ then $\gamma^{-1}(x) \leq \beta^{-1}(x)$; if $h(x) = \rightarrow$ then $\gamma^{-1}(x) \geq \beta^{-1}(x)$.

(Os1)	$\left.\begin{matrix}\leftarrow\\ \cdot\end{matrix}\right\} \overset{>k}{\overline{\qquad}} \left\{\begin{matrix}\rightarrow\\ \cdot\end{matrix}\right.$	`reject`
(Os2)	$\leftrightarrows \overset{>k}{\overline{\qquad}} \left\{\begin{matrix}\rightarrow\\ \cdot\end{matrix}\right.$	replace by $\rightarrow \overset{>k}{\overline{\qquad}} \left\{\begin{matrix}\rightarrow\\ \cdot\end{matrix}\right.$
(Os3)	$\left.\begin{matrix}\leftarrow\\ \cdot\end{matrix}\right\} \overset{>k}{\overline{\qquad}} \leftrightarrows$	replace by $\left.\begin{matrix}\leftarrow\\ \cdot\end{matrix}\right\} \overset{>k}{\overline{\qquad}} \leftarrow$
(Os4)	$\rightarrow \overset{>k}{\overline{\qquad}} \left\{\begin{matrix}\rightarrow\\ \cdot\end{matrix}\right.$	`MoveAttempt` left vertex to right
(Os5)	$\left.\begin{matrix}\leftarrow\\ \cdot\end{matrix}\right\} \overset{>k}{\overline{\qquad}} \leftarrow$	`MoveAttempt` right vertex to left

Fig. 1. Patterns and rules for deciding the existence of a desired layout

Next we describe the algorithm which we call `MoveRepair`. Input is a graph G, an integer $k \geq 0$, a layout β of G, and a direction assignment h. Algorithm `MoveRepair` generates a sequence of layout and direction assignment pairs (β, h)=$(\beta_0, h_0), (\beta_1, h_1), \ldots, (\beta_l, h_l)$ such that $\Delta(\beta_i, h_i) = \Delta(\beta_{i+1}, h_{i+1})$ for $0 \leq i < l$. In particular, the algorithm detects patterns in the current layout and works according to a set of rules, presented in Figure 1. Let β and h be the layout and assignment before a rule is applied, and let β_* and h_* be the modified layout and assignment as a result of the applied rule. The interpretation is as follows: let u and v be adjacent vertices with $u \prec_\beta v$ and $d_\beta(u,v) > k$. If $h(u) \in \{\cdot, \leftarrow\}$ and $h(v) \in \{\cdot, \rightarrow\}$ then the first rule is applied and the algorithm rejects β. If $h(u) = \leftrightarrows$ and $h(v) \in \{\cdot, \rightarrow\}$ then the second rule is applied, and the symbol of u is changed to $\rightarrow$, hence β_* is the same as β, whereas h_* is the same as h except that $h_*(u) = \rightarrow$.

Algorithm `MoveRepair`
Input: A graph G, a layout β of G, a direction assignment h to the vertices of G.
Output: A layout β_*, and a reply `accept` or `reject`.

```
while there is an edge $uv$ in $G$ satisfying one of the given patterns in Figure 1 do
    execute the corresponding operation on edge $uv$ with input $(\beta, h)$ and output
    $(\beta_*, h_*)$; $\beta =_{\text{def}} \beta_*$;  $h =_{\text{def}} h_*$;
end-while
accept;
```

We should mention that whenever a `reject` is executed, the algorithm terminates with output `reject`, and the consecutive instructions are not executed. Assume that $u \prec_\beta v$. We describe the operation `MoveAttempt` *left vertex to right*. (`MoveAttempt` *right vertex to left* is symmetric and defined analogously.)

Operation `MoveAttempt` left vertex to right
Input: A graph G, a layout β and a direction assignment h on G, two adjacent vertices u, v with distance more than k in β that satisfy the condition of (Os4) in Figure 1.
Output: A layout β_* and a direction assignment h_* on G, or a reply `reject`.

```
if there is no vertex of the color class of $v$ between $u$ and $v$ in $\beta$ then
    reject
else
    let $w$ be the closest vertex to the right of $u$ belonging to the color class of $v$;
    if $h(w) \notin \{\leftarrow, \leftrightarrows\}$ then
        reject
    else
        if all vertices between $u$ and $w$ in $\beta$ have symbol $\rightarrow$ or $\leftrightarrows$ in $h$ then
            $h_* =_{\text{def}} h$;
            for every vertex $x$ between $u$ and $w$ in $\beta$ do $h_*(x) =_{\text{def}} \rightarrow$ end-for
            $h_*(w) =_{\text{def}} \leftarrow$;
            $\beta_* =_{\text{def}}$ the layout obtained from $\beta$ by moving $u$ one position closer to $v$;
        else
            reject
        end-if
    end-if
end-if
```

Since this operation is invoked, u has symbol $\rightarrow$ and v has $\cdot$ or $\rightarrow$. If there is a vertex between u and v belonging to the color class of v then u can be moved one position closer to v if the symbols of vertices between u and w allow this, otherwise not. Let w be a vertex of the color class of v that is to the right of u and closest to u of all such vertices. Moving u one position closer to v is only possible if w has symbol $\leftarrow$ or $\leftrightarrows$, and all vertices between u and w have symbol $\rightarrow$ or $\leftrightarrows$.

By Lemma 1 and the description of the algorithm, it follows that if the input layout to Algorithm `MoveRepair` is normalized then the layout produced after each single operation is also normalized. It is important to note that whenever a vertex is moved in one direction, its direction symbol is fixed to indicate this direction, and it is not allowed to move in the other direction during the same execution of Algorithm `MoveRepair`.

A bad situation would occur if an edge of distance more than k between its endpoints had on both its endpoints "inward" arrows or $\leftrightarrows$, which would give several possibilities to repair this edge and too many possibilities in total. We will ensure that this situation never occurs. We say that (β, h) has the *outward arrows* property if the following is true for every edge uv of G with $u \prec_\beta v$: if $d_\beta(u, v) > k$ then $h(u) \in \{\cdot, \leftarrow\}$ or $h(v) \in \{\cdot, \rightarrow\}$. Observe that the rules of Algorithm `MoveRepair` apply precisely to those edges. Algorithm `MoveRepair` will always be called with input that has the outward arrows property. With the following two lemmas we show that if (β, h) has the outward arrows property then Algorithm `MoveRepair` correctly decides whether $\Delta(\beta, h)$ is empty.

Lemma 2. *Let $G = (A, B, E)$ be a connected bipartite permutation graph, and let (σ_A, σ_B) be a strong ordering for G. Let $k \geq 0$. Let β be a normalized layout for G, and let h be a direction assignment for G that has the outward arrows property. Then each of the layout-assignment pairs generated throughout Algorithm* `MoveRepair` *has the outward arrows property, and if Algorithm* `MoveRepair` *accepts input (β, h) and outputs β_* then $\beta_* \in \Delta(\beta, h)$.*

Lemma 3. *Let $G = (A, B, E)$ be a connected bipartite permutation graph, and let (σ_A, σ_B) be a strong ordering for G. Let β be a normalized layout for G and let h be a direction assignment for G. If Algorithm* `MoveRepair` *rejects input (β, h) then $\Delta(\beta, h)$ is empty.*

Proof. To prove the lemma, we assume that the algorithm rejects but $\Delta(\beta, h)$ is non-empty. Let $(\beta, h)=(\beta_0, h_0), \ldots, (\beta_l, h_l)$ be the layout-assignment pairs generated by the algorithm, where (β_i, h_i) is the result after algorithm step i. In step $l + 1$, the algorithm decides rejection. We first show properties relating assigned direction symbols and vertex positions in layouts in $\Delta(\beta, h)$.

Claim. *Let $\Delta(\beta, h)$ be non-empty. For every $i \in \{0, \ldots, l\}$ and every vertex x of G, the following holds:*

- *if $h_i(x) = \cdot$ then $\gamma^{-1}(x) = \beta_i^{-1}(x)$ for every $\gamma \in \Delta(\beta, h)$*
- *if $h_i(x) = \leftarrow$ then $\gamma^{-1}(x) \leq \beta_i^{-1}(x)$ for every $\gamma \in \Delta(\beta, h)$*
- *if $h_i(x) = \rightarrow$ then $\gamma^{-1}(x) \geq \beta_i^{-1}(x)$ for every $\gamma \in \Delta(\beta, h)$.*

Proof of the claim. Let γ be a layout in $\Delta(\beta, h)$. We show by induction on i that the claim holds for γ. The claim holds for $i = 0$ by the definition of

$\Delta(\beta, h)$. Let the claim be true for (β_{i-1}, h_{i-1}) for some $i > 0$. Let (β_i, h_i) be obtained from (β_{i-1}, h_{i-1}) by application of operation o. For all vertices x such that $\beta_i^{-1}(x) = \beta_{i-1}^{-1}(x)$ and $h_i(x) = h_{i-1}(x)$, the claim holds for (β_i, h_i) by the induction hypothesis. For the other vertices, we distinguish between cases according to o. Let uv be the edge to which o is applied, $u \prec_{\beta_{i-1}} v$. Let o be one of the two operations of (Os2). Then, $\beta_i = \beta_{i-1}$ and h_i differs from h_{i-1} only for u. By definition, $h_{i-1}(v) \in \{\cdot, \rightarrow\}$, and $\gamma^{-1}(v) \geq \beta_{i-1}^{-1}(v)$ by the induction hypothesis. Since $d_{\beta_{i-1}}(u, v) > k$, u must be further to the right in γ, i.e., $\gamma^{-1}(u) \geq \beta_{i-1}^{-1}(u)$. Since $h_i(u) = \rightarrow$, the claim holds for this case. Analogously, it is proved that the claim holds in case o is an operation from (Os3). Let o be from (Os4) or (Os5). Since both cases are symmetric, we consider an operation from (Os4). Since (β_i, h_i) is defined, u is moved one position closer to v. Since $\gamma^{-1}(v) \geq \beta_{i-1}^{-1}(v)$ by induction hypothesis and $d_{\beta_{i-1}}(u, v) > k$ by the assumption about application of o, $\gamma^{-1}(u) > \beta_{i-1}^{-1}(u)$. And since $\beta_i^{-1}(u) = \beta_{i-1}^{-1}(u) + 1$ and $h_i(u) = \rightarrow$, we conclude that the claim holds for u. Let w be the closest vertex to the right of u in β_{i-1} from the color class of v. Remember that $w \prec_{\beta_{i-1}} v$. The only further vertices that change position or direction symbol in (β_i, h_i) with respect to (β_{i-1}, h_{i-1}) are w and the vertices between u and w in β_{i-1}. Since γ is a normalized layout, we conclude that w cannot be to the right of u in γ; otherwise, $\gamma^{-1}(u) \leq \beta_{i-1}^{-1}(u)$, which contradicts the conclusion above. So, $\gamma^{-1}(w) \leq \beta_{i-1}^{-1}(u) = \beta_i^{-1}(w)$, and the claim holds for w with $h_i(w) = \leftarrow$. Correctness for the vertices between u and w in β_{i-1} then immediately follows from the restriction of γ to a normalized layout and the correctness for u, since all these vertices are assigned symbol $\rightarrow$ by h_i. □

Now we continue the proof of Lemma 3. Let uv be the edge which is considered by the algorithm in step $l+1$. Let $u \prec_{\beta_l} v$. Since the algorithm rejects in step $l+1$, the executed operation is one of the set (Os1) or (Os4–5). We first consider set (Os1). According to the definition of (Os1) and the claim, $\gamma^{-1}(u) \leq \beta_l^{-1}(u)$ and $\beta_l^{-1}(v) \leq \gamma^{-1}(v)$ for all $\gamma \in \Delta(\beta, h)$. Since $d_{\beta_l}(u, v) > k$, γ cannot be a k-layout for G. Let the executed operation now be from set (Os4). The case (Os5) is analogous. According to the definition of `MoveAttempt` left vertex to right, we have to distinguish between three cases which can imply rejection. Let w be the closest vertex to u from the color class of v to the right of u in β_l. Note that w exists. If $w = v$ then all vertices between u and v in β_l are from the color class of u, and u can be closer to v only by moving v closer to u. This, however, is not possible for a layout in $\Delta(\beta, h)$ and $h_l(v) = \rightarrow$. So, let $w \prec_{\beta_l} v$. Let $h_l(w) \in \{\cdot, \rightarrow\}$. Then, $\beta_l^{-1}(w) \leq \gamma^{-1}(w)$ for all $\gamma \in \Delta(\beta, h)$. By definition of normalized layouts and since there are only vertices from the color class of u between u and w in β_l, it follows that $\gamma^{-1}(u) \leq \beta_l^{-1}(u)$, which means $\gamma^{-1}(u) = \beta_l^{-1}(u)$ according to the claim and with $h_l(u) = \rightarrow$. This, however, is not possible for layouts in $\Delta(\beta, h)$. Finally, let x be a vertex between u and w in β_l and let $h_l(x) \in \{\cdot, \leftarrow\}$. This particularly means that $x \prec_\gamma w$ for all $\gamma \in \Delta(\beta, h)$. We conclude like in the previous case that $\gamma^{-1}(u) = \beta_l^{-1}(u)$ for all $\gamma \in \Delta(\beta, h)$, which is a contradiction to γ being a k-layout for G. Since we showed contradictions for every possible case, $\Delta(\beta, h)$ cannot be non-empty. This completes the proof of Lemma 3.

Theorem 3. *There is an algorithm that can be implemented to run in time $\mathcal{O}(kn)$ on normalized layouts of connected bipartite permutation graphs for every $k \geq 1$ and simulates a possible computation of algorithm* `MoveRepair`.

Theorem 4. *Let $G = (A, B, E)$ be a connected bipartite permutation graph, and let (σ_A, σ_B) be a strong ordering for G. Let $k \geq 0$. Assume that the following holds:*

(A1) *$A = \{a_1, \ldots, a_s\}$ and $B = \{b_1, \ldots, b_t\}$ where $a_1 \prec_{\sigma_A} \cdots \prec_{\sigma_A} a_s$ and $b_1 \prec_{\sigma_B} \cdots \prec_{\sigma_B} b_t$.*
(A2) *b_q is a neighbor of a_s.*
(A3) *$r \in \{1, \ldots, s\}$.*

Then, given a normalized k-layout for G, there is a polynomial-time algorithm that decides whether there is a normalized k-layout for G such that b_q is to the right of a_r. In the positive case, the algorithm outputs such a layout.

Theorem 5. *Let $G = (A, B, E)$ be a connected bipartite permutation graph, and let (σ_A, σ_B) be a strong ordering for G. Let $k \geq 0$. Assume that the following holds:*

(A1) *$A = \{a_1, \ldots, a_s, a_{s+1}\}$ and $B = \{b_1, \ldots, b_t\}$ where $a_1 \prec_{\sigma_A} \cdots \prec_{\sigma_A} a_{s+1}$ and $b_1 \prec_{\sigma_B} \cdots \prec_{\sigma_B} b_t$.*
(A2) *$N(a_{s+1}) \subseteq N(a_s)$.*
(A3) *b_p is a neighbor of a_{s+1}, where $p \geq 2$.*

Then, given a normalized k-layout for $G - a_{s+1}$, there is a polynomial-time algorithm that decides whether there is a normalized k-layout for G such that a_{s+1} is between b_{p-1} and b_p. In the positive case, the algorithm outputs such a layout.

5 A Polynomial-Time Algorithm for Computing the Bandwidth of Bipartite Permutation Graphs

Our main algorithm, called `BPG-Bandwidth`, is presented on the next page.

Lemma 4. *Let $G = (A, B, E)$ be a connected bipartite permutation graph, and let (σ_A, σ_B) be a strong ordering for G. Let $k \geq 0$. Then,* `BPG-Bandwidth` *on input G, (σ_A, σ_B) and k accepts if and only if* $\mathrm{bw}(G) \leq k$*. In the accepting case, the output layout is a k-layout for G.*

Theorem 6. *There is an algorithm with running time $\mathcal{O}(n^4 \log n)$ that computes the bandwidth of a bipartite permutation graph and outputs a corresponding optimal layout.*

Algorithm `BPG-Bandwidth` (Bipartite Permutation Graphs Bandwidth)
Input: A connected bipartite permutation graph $G = (A, B, E)$, a strong ordering (σ_A, σ_B) for G and an integer k.
Output: A reply `accept` and a k-layout β for G if $\mathrm{bw}(G) \leq k$, or a reply `reject` if $\mathrm{bw}(G) > k$.

```
let $A = \{a_1, \ldots, a_s\}$ and $B = \{b_1, \ldots, b_t\}$ where $a_1 \prec_{\sigma_A} \cdots \prec_{\sigma_A} a_s$ and $b_1 \prec_{\sigma_B} \cdots \prec_{\sigma_B} b_t$;
if $|N(a_1)| > 2k$ then
    reject;   stop;
end-if
let $\beta$ be a normalized $k$-layout for $G_1$;
for $i = 2$ to $s$ do
    let $\delta$ be a normalized $k$-layout for $G[N[a_i] \setminus V(G_{i-1})]$ with $a_i$ leftmost possible;
    let $\delta'$ be a normalized $k$-layout for $G[N(a_i) \setminus V(G_{i-1})]$;
    $\beta' =_{\mathrm{def}} \beta \circ \delta$;
    if $\beta'$ is a $k$-layout for $G_i$ then
        $\beta =_{\mathrm{def}} \beta'$;
        goto the next iteration of main for-loop;
    else
        let $b$ be the leftmost neighbor of $a_i$ in $\beta'$;
        if there are less than $d_{\beta'}(a_i, b) - k$ $A$-vertices between $a_i$ and $b$ then
            reject;   stop;
        else
            let $a$ be the $(d_{\beta'}(a_i, b) - k)$th $A$-vertex closest to $b$ between $b$ and $a_i$ in $\beta'$;
(1)         if there is a normalized $k$-layout $\beta_*$ for $G_{i-1}$ in which $b$ appears to the right of $a$ then
                $\beta =_{\mathrm{def}} \beta_* \circ \delta$;   goto the next iteration of main for-loop (*);
            end-if
        end-if
    end-if
    for each of the $k$ last $B$-vertices $b_p$ in $\beta$ do
(2)     if there is a normalized $k$-layout $\beta_*$ for $G_{i-1}+a_i$ where $a_i$ appears between $b_{p-1}$ and $b_p$ then
            $\beta' =_{\mathrm{def}} \beta_* \circ \delta'$;
            if the distance between $a_i$ and its rightmost neighbor in $\beta'$ is at most $k$ then
                $\beta =_{\mathrm{def}} \beta'$;
                goto the next iteration of main for-loop;
            end-if
        end-if
    end-for
    reject;   stop;
end-for
accept;
```

We use Theorem 4 to decide condition (1). Note that a_i is adjacent to b in this case. We use Theorem 5 to decide condition (2).

Acknowledgments

We would like to thank Fedor V. Fomin and Saket Saurabh for useful discussions on the importance of the bandwidth problem.

References

1. Assmann, S.F., Peck, G.W., Sysło, M.M., Zak, J.: The bandwidth of caterpillars with hairs of length 1 and 2. SIAM J. Algebraic and Discrete Methods 2, 387–393 (1981)
2. Blache, G., Karpinski, M., Wirtgen, J.: On approximation intractability of the bandwidth problem. Technical report, University of Bonn (1997)

3. Blum, A., Konjevod, G., Ravi, R., Vempala, S.: Semi-Definite Relaxations for Minimum Bandwidth and other Vertex-Ordering Problems. In: Proceedings of STOC 1998, pp. 100–105. ACM, New York (1998)
4. Bodlaender, H.L., Fellows, M.R., Hallet, M.T.: Beyond NP-completeness for problems of bounded width (extended abstract): hardness for the W hierarchy. In: Proceedings of STOC 1994, pp. 449–458. ACM, New York (1994)
5. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, New York (1999)
6. Feige, U.: Approximating the Bandwidth via Volume Respecting Embeddings. In: Proceedings of STOC 1998, pp. 90–99. ACM, New York (1998)
7. Feige, U., Talwar, K.: Approximating the Bandwidth of Caterpillars. In: Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX 2005 and RANDOM 2005. LNCS, vol. 3624, pp. 62–73. Springer, Heidelberg (2005)
8. Fishburn, P., Tanenbaum, P., Trenk, A.: Linear discrepancy and bandwidth. Order 18, 237–245 (2001)
9. George, J.A., Liu, J.W.H.: Computer Solution of Large Sparse Positive Definite Systems. Prentice-Hall Inc., Englewood Cliffs, New Jersey (1981)
10. Gupta, A.: Improved bandwidth approximation for trees. In: Proceedings of SODA 2000, pp. 788–793. ACM, SIAM (2000)
11. Haralambides, J., Makedon, F., Monien, B.: Bandwidth Minimization: An Approximation Algorithm for Caterpillars. Mathematical Systems Theory 24(3), 169–177 (1991)
12. Heggernes, P., Kratsch, D., Meister, D.: Bandwidth of bipartite permutation graphs in polynomial time. In: Reports in Informatics, University of Bergen, Norway, vol. 356 (2007)
13. Kleitman, D.J., Vohra, R.V.: Computing the bandwidth of interval graphs. SIAM J. Disc. Math. 3, 373–375 (1990)
14. Kloks, T., Kratsch, D., Müller, H.: Bandwidth of chain graphs. Information Processing Letters 68, 313–315 (1998)
15. Kloks, T., Kratsch, D., Müller, H.: Approximating the bandwidth for AT-free graphs. Journal of Algorithms 32, 41–57 (1999)
16. Meister, D.: Recognition and computation of minimal triangulations for AT-free claw-free and co-comparability graphs. Disc. Appl. Math. 146, 193–218 (2005)
17. Monien, B.: The bandwidth minimization problem with hair length 3 is NP-complete. SIAM J. Alg. Disc. Meth. 7, 505–512 (1986)
18. Papadimitriou, C.: The NP-completeness of the bandwidth minimization problem. Computing 16, 263–270 (1976)
19. Spinrad, J., Brandstädt, A., Stewart, L.: Bipartite permutation graphs. Disc. Appl. Math. 18, 279–292 (1987)
20. Unger, W.: The Complexity of the Approximation of the Bandwidth Problem. In: Proceedings of FOCS 1998, pp. 82–91. IEEE, Los Alamitos (1998)
21. Yan, J.H.: The bandwidth problem in cographs. Tamsui Oxf. J. Math. Sci. 13, 31–36 (1997)

The Online Transportation Problem: On the Exponential Boost of One Extra Server[⋆]

Christine Chung, Kirk Pruhs, and Patchrawat Uthaisombut

University of Pittsburgh
{chung,kirk,utp}@cs.pitt.edu

Abstract. We present a poly-log-competitive deterministic online algorithm for the online transportation problem on hierarchically separated trees when the online algorithm has one extra server per site. Using metric embedding results in the literature, one can then obtain a poly-log-competitive randomized online algorithm for the online transportation on an arbitrary metric space when the online algorithm has one extra server per site.

1 Introduction

The setting for the online transportation problem is a collection of k server sites located in some metric space. Points in the metric space, which represent requests for service, arrive online over time. Server site j has a fixed capacity B_j specifying the number of requests that can be handled by site j. After each request arrives, the online algorithm must irrevocably match that request to a single server site that has not yet reached its capacity. Conceptually it is convenient to think of B_j as the number of servers at site j, and one of these servers traveling to a request that is assigned to site j. The objective is to minimize the total (or equivalently average) distance between the requests and their assigned server sites.

If the locations of the requests are known a priori, then this is the standard offline transportation problem [9,11]. Many of the applications of the transportation problem are naturally online problems. We give two examples here. In the first example, the server sites could be fire stations, the capacity of a fire station could be the number of fire trucks stationed there, and the requests could be the location of a fire. The objective would then be to minimize the average distance that fire trucks travel. In the second example, the server sites could be schools, the capacity of a school could be the number of students that the school can handle, and the requests could be the locations of new students that move to the school district. The objective would then be to minimize the average distance between students and their assigned schools.

1.1 Previous Results

The online weighted matching problem is a special case of the online transportation problem in which each server site has unit capacity. The competitive

⋆ Supported in part by NSF grants CNS-0325353, CCF-0514058 and IIS-0534531.

ratio of every deterministic algorithm for online matching, and hence for online transportation, is at least $2k - 1$ [6,10]. The metric space in the lower bound instances that establish this bound is a star. In a star there is a unique root point that is a unit distance from all other points, and all pairs of non-center points, called leaves, are distance two from each other. [6,10] give a $(2k - 1)$-competitive algorithm, called Permutation in [6], for online matching. However, the competitive ratio of Permutation for the more general online transportation problem is $\Theta(B)$, where $B = \sum_{i=1}^{k} B_i$ is the aggregate capacities of the server sites [7]. In contrast, [6] shows that the competitive ratio of the natural greedy algorithm, that always matches each request to an arbitrary nearest server site with residual capacity, is $2^k - 1$ for online transportation.

The fact that the optimal deterministic competitive ratios are so high prompted [8] to consider resource augmentation analysis for online transportation. Resource augmentation analysis compares the performance of an online algorithm to the optimal solution with less resources. In the context of online transportation, the resource is the number of servers per site. So an s-server c-competitive algorithm A guarantees that if A has $s \cdot B_j$ servers at each site j then the cost of A's assignment is at most c times the cost of the optimal assignment assuming B_j servers at each site j. [6] show that the greedy algorithm is 2-server $\Theta(\min(k, \log B))$-competitive for online transportation. [8] then considers a slightly modified greedy algorithm, that in the cases of approximate ties for the nearest server site, picks a server site that has been assigned the least requests so far. [8] show that this modified greedy algorithm is 2-server $O(1)$-competitive for online transportation. One can also see from the analyses in [8] that the greedy algorithm is 3-server $O(1)$-competitive for online transportation.

[3,12] consider randomized algorithms for the online matching problem against an oblivious adversary that must specify the input a priori. In particular, [3,12] consider the simple randomized greedy algorithm that services each request with an unused server site picked uniformly at random from the closest server sites. It is easy to see that the randomized greedy algorithm is $O(\log k)$-competitive for online matching in a star. [3,12] extend this analysis to show that randomized greedy is $O(\log k)$-competitive in $(\log k)$-Hierarchically Separated Trees ($\log k$-HST's). [1,4] show that for every metric space, there exists a probability distribution over $\log k$-HST's, for which the expected distance between any pair of points in a randomly drawn $\log k$-HST is at most $O(\log^2 k)$ times the distance between this pair of points in the metric space. Combining these two results gives an $O(\log^3 k)$-competitive randomized algorithm (note that this algorithm is not randomized greedy) against an oblivious adversary for online matching on an arbitrary metric space.

For a summary of results for online matching problems, and related online network optimization problems, see [7].

1.2 Our Results

One motivation for considering resource augmentation analysis is that it allows one to show that an algorithm is competitive, without additional resources, for instances where the optimal solution is not so sensitive to changes in the number of available servers per site. More precisely, an s-server c-competitive algorithm would be $c \cdot d$-competitive on instances where a factor of s change in the number of servers per site does not change the optimal cost by more than a factor of d. The smallest resource augmentation considered in [8] was doubling the number of servers per site. Our main aim here is consider the effect of more modest resource augmentation. More precisely, we consider the effect of adding just one additional server per site. We will say that an online algorithm A is $+1$-server c-competitive if A guarantees that if A has $B_j + 1$ servers at each site j then the cost of A's assignment is at most c times the cost of the optimal assignment assuming B_j servers at each site j. Then a $+1$-server c-competitive algorithm would be $c \cdot d$-competitive on instances where adding one server per site does not change the optimal cost by more than a factor of d.

In this paper we consider a natural deterministic online algorithm that we call Balancing of Displaced Servers (BODS). Intuitively, BODS prefers using server sites that have serviced less requests (of positive cost), as did the modified greedy algorithm in [8], but for BODS this preference is only relevant in the case of exact ties.

BODS Description: BODS always assigns the request to a nearest server with residual capacity. BODS breaks ties among closest servers by assigning the request to a server site that has been to date assigned the least number of requests with positive cost. (Note that this is not the same as assigning the request to the site that has serviced the least requests so far, since a request on site j costs nothing if assigned to site j.) If a tie remains, BODS uses the first server site with residual capacity in some arbitrary ordering of the server sites.

From what is known to date about online transportation and matching, the hardest metric space for the online algorithm is always the star. The results in [3,12] can be viewed as a reduction from a general metric space to the star via HST's. So we begin by first considering the star metric space. We show BODS is $+1$-server $O(\log k)$-competitive for online transportation in a star. We found it surprising that such modest resource augmentation drops the competitive ratio so dramatically, from linear in k to logarithmic in k. We then show that BODS is essentially optimally competitive. That is, we show that there is no $+1$-server $o(\log k)$-competitive deterministic online algorithm for online transportation on a star. We then generalize our analysis to show that BODS is $+1$-server $O(\log k)$-competitive for online transportation on a $\log k$-HST. Our proof proceeds by induction on the height of the HST, following the same general structure as the proof in [12]. However the introduction of resource augmentation adds some technical difficulties. For example, when the analysis in [12] considers the subinstances on the subtrees of the root, we now have to be concerned about the possibility that in some of these subinstances there are more requests than servers available to the adversary, and thus there is no feasible optimal solution without resource augmentation. Using the results in [1,4], we obtain a $+1$-server $O(\log^3 k)$-competitive randomized algorithm for online transportation for an arbitrary metric space against an oblivious adversary.

Our results extend those in [3,12] in two ways: (1) For arbitrary metric spaces, our analysis holds for the more general online transportation, not just for the special case of online matching, and (2) for stars and HST's, we obtain a deterministic bound on the competitive ratio instead of a randomized bound against an oblivious adversary. Of course these extensions come at the cost of requiring modest resource augmentation.

2 Online Transportation on a Star

In this section, we assume that the k server sites are at the leaves of a star. We show that BODS is $+1$-server $O(\log k)$-competitive on a star, and that this is essentially the best possible result that one can obtain. We start with the lower bound.

When a request arrives at a leaf, and a server from the leaf site is assigned to the request, we refer to it as a *local service* or *local assignment*. If a request arrives at a leaf site and a server from a different leaf site is assigned to the request, we call it a *remote service* or *remote assignment*. Thus, all root requests cost 1 to service, a leaf request serviced *locally* costs 0, and a leaf request serviced *remotely* costs 2. A request is called an *excess request* if it arrives at a server site s after B_s requests have already arrived at site s. Hence, the only requests that optimal will pay for are those that arrive at the root and those that are excess requests. We define $C_A(I)$ to be the cost of algorithm A on input I.

2.1 The General Lower Bound

Theorem 1. *There is no deterministic $+1$-server $o(\log k)$-competitive algorithm for online transportation on a uniform star with k leaves.*

Proof. Consider the input instance where for all sites j, for $1 \leq j \leq k$, we have $B_j = b$. Assume that the online algorithm has e extra servers per site (so we will eventually use $e = 1$ to prove the statement of the theorem). Let x be an integer value to be set later. First, xb requests arrive at the root node, then continue to arrive b at a time at the next leaf with the fewest remaining available servers until a total of $B = bk$ requests have been made. Call each set of b requests made from a leaf node a *hit*.

First note that the cost of the optimal offline solution C_{OPT} is always xb since it knows the request sequence in advance and for the requests from each hit it will reserve local servers from that site. So it services the xb root requests using the remaining servers that are not already reserved. Then it will be able to service requests from each hit locally. The adversary's strategy finds a value for x that maximizes the competitive ratio.

Since xb requests are initially made from the root node, and there are bk total requests, there are $k-x$ total hits. Let $f(i)$ for all $1 \leq i \leq k-x$ be the maximum (over all un-hit server sites) after hit i of the number of servers that have been used at any site by the online algorithm. Define $f(0)$ to be the maximum number of servers used at any site by the online algorithm for the initial xb root requests. Define $c(i)$ as the number of requests in hit i that are serviced remotely (at a cost of 2) by the online algorithm. By definition of $f(i)$ and the adversarial strategy outlined above, we know that $c(i) = f(i-1) - e$ for $1 \leq i \leq k - x$, where we subtract e from $f(i-1)$ because there are $b+e$ servers at each of the server sites allowing us to service e extra requests locally.

After hit i, for $1 \leq i \leq k - x$, $xb + ib$ total requests have been made, whereas at most $i(b+e)$ have been used at the i sites that have been hit so far. That leaves at least $(x+i)b - i(b+e)$ servers missing at the remaining $k - i$ unhit sites. So, by the pigeon-hole principle, for $1 \leq i \leq k - x$,

$$f(i) \geq \frac{(x+i)b - i(b+e)}{k-i} = \frac{xb - ie}{k-i} \ .$$

By the definition of $c(i)$, the total cost C_{ON} of the solution returned by any online algorithm ON is therefore:

$$\begin{aligned} C_{ON} = xb + 2\sum_{i=1}^{k-x} c(i) \ &= \ xb + 2\sum_{i=1}^{k-x} (f(i-1) - e) \\ \geq xb + 2\sum_{i=0}^{k-x-1} \left(\frac{xb - ie}{k-i} - e \right) \ &= \ xb + 2(xb - ek)(H_k - H_x) \ . \end{aligned}$$

Thus, $\dfrac{C_{ON}}{C_{OPT}} \geq \dfrac{xb + 2(xb - ek)(H_k - H_x)}{xb} \ \geq \ 1 + 2\left(1 - \dfrac{ek}{xb}\right)\left(\ln \dfrac{k+1}{x} - 1\right)$.

By choosing $x = 1$, we get $\frac{C_{ON}}{C_{OPT}} \geq \ln k$. □

2.2 BODS on the Star

A *live* site is a site with at least one server still unassigned. A *dead* site is a site whose servers have all been assigned, so there are no more available at that site to service any future requests. A server is called *displaced* if it has been assigned to a remote request. We define a *restricted instance* to be one where: (A) no more than B_j requests arrive at server site j, (B) no request is serviced locally by BODS, and (C) B requests arrive. We now show, without loss of generality, that we may restrict our analysis of BODS to restricted instances.

Lemma 1. *If there is an instance I for which the $+1$-server competitive ratio of BODS is at least c, then there is a* restricted instance *I' for which the $+1$-server competitive ratio of BODS is at least c.*

Proof. We consider the restrictions in order. (A) Consider request number $B_j + i$ to site j. If $i > 1$ then this request in I' will appear at the root instead of at j. If $i = 1$ and $B_j > 0$ then we decrement the value of B_j in I' by 1, and there is no corresponding request in I'. If $i = 1$ and $B_j = 0$ then remove this server site and the request in I'. (B) Let q be a leaf request in I that arrives at a site j and is serviced locally by BODS. We create I' by removing q from I, and decrementing B_j by 1. (C) For any instance with fewer than B requests, for each unassigned server in the optimal solution another request can arrive at its site. □

Theorem 2. *BODS is* $+1$*-server* $O(\log k)$*-competitive for online transportation on a star with* k *leaves.*

Proof. By Lemma 1, we need only consider restricted inputs. For notational convenience, we relabel the sites based on the number of servers at each site as well as tie-breaking order. In particular, we label the sites so that $B_1 \leq B_2 \leq \ldots \leq B_k$. Furthermore, we label them such that if $B_j = B_{j+1}$, then site j comes earlier than site $j+1$ in the tie-breaking order. Let e_i be the number of servers that site i is augmented by in our online setting for BODS. Eventually we will set each e_i to one, but we believe that it is instructive to leave the e_i's as variables in the proof. The only properties we need of the e_i's is that they are non-decreasing, that is, $e_i \leq e_{i+1}$, and that if $B_i = B_{i+1}$ then $e_i = e_{i+1}$.

We claim that the server sites die in order of increasing site number. To see why, remember that each request in a restricted input is remotely serviced by BODS. Thus the first requests must be root requests, and BODS will choose to assign servers from server sites in a round robin fashion. Root requests will continue until site 1 is dead. We know that site 1 will be the first to die since we have numbered the sites in non-decreasing order of the number of servers at each site, and in the case of sites with the same number of servers, we have taken care to assign lower numbers to sites that are earlier in BODS's tie-breaking order. After site 1 is dead, again since BODS services all requests remotely, all successive requests must arrive at either site 1 or the root until site 2 is dead, and so on.

Define *round* 0 to be the sequence of requests from the first request up until site 1 is dead. For $i \geq 1$, define round i to be the sequence of requests after round $i-1$ up until site $i+1$ is dead, or until a total of B requests have been made, whichever comes first. For $i \geq 0$, let r_i be the number of requests in round i. Note that round 0 consists only of root requests, whereas for $i \geq 1$, round i may consist of both root requests and leaf requests at dead sites.

Let m be the round in which the Bth request appears. Note that $1 \leq m \leq k-1$ since site $m+1$ dies in round m and there are only k server sites. For $j = 1 \ldots m$, let x_j be the number of root requests in round j. Note that $x_0 = r_0$ and $0 \leq x_i \leq r_i$ for $1 \leq i \leq m$. Recall that until the end of round i, requests are made only at the root and the first i dead sites. The number of root requests through round i is $\sum_{j=1}^{i} x_j$. Since the input is restricted, there are at most B_j requests on site j. Thus, the number of leaf requests through round i is at most $\sum_{j=1}^{i} B_j$. So for $1 \leq i \leq m$,

$$\sum_{j=1}^{i} r_j \leq \sum_{j=1}^{i} (B_j + x_j) \ . \tag{1}$$

Note that $r_0 + \sum_{j=1}^{m} x_j$ is the number of total root requests in a restricted input. In a restricted input the optimal solution only pays for root requests.

Hence,

$$C_{OPT}(I) = r_0 + \sum_{j=1}^{m} x_j \ . \tag{2}$$

Since BODS pays at most 2 to service a request, then

$$C_{BODS}(I) \leq 2B \leq 2\left(r_0 + \sum_{j=1}^{m} x_j + \sum_{j=1}^{m} B_j\right) . \tag{3}$$

We now formulate the number of requests in each round. Let α_1 be the position of site 1 in BODS's tie-breaking order on the sites 1 through k. Since no site has fewer than $B_1 + e_1$ servers, and there are k sites, there must be $r_0 = e_1 k + (B_1 - 1)k + \alpha_1$ requests before site 1 is dead. To see this, note that a total of e_1 servers at each site are assigned after the first $e_1 k$ requests, and $B_1 - 1$ more servers are assigned from each site after the next $(B_1 - 1)k$ requests, and the $B_1 + e_1$th server of site 1 is assigned after another α_1 requests.

Similarly, since at the beginning of round i, only $k - i$ sites are alive, let $\alpha_{i+1} \leq k - i$ be the position of site $i + 1$ in BODS's tie-breaking order on the remaining sites $i + 1$ through k. The number of requests that are in round i is then $r_i = (k - i + 1 - \alpha_i) + (B_{i+1} + e_{i+1} - (B_i + e_i) - 1)(k - i) + \alpha_{i+1}$, for $1 \leq i \leq m$. To see why, consider two cases.

Case 1 ($B_{i+1} > B_i$). We need $k - i + 1 - \alpha_i$ requests to finish clearing the $B_i + e_i$th server at each of the remaining $k - i + 1 - \alpha_i$ sites. We then need another $(B_{i+1} + e_{i+1} - (B_i + e_i) - 1)(k - i)$ requests to use up all but one server at the site(s) with $B_{i+1} + e_{i+1}$ servers. Finally, we need α_{i+1} more requests to reach the $B_{i+1} + e_{i+1}$th server of site $i + 1$ in the tie-breaking order.

Case 2 ($B_{i+1} = B_i$). The $k - i + 1 - \alpha_i$ requests will finish clearing off the $B_i + e_i = B_{i+1} + e_{i+1}$th server at the remaining $k - i + 1 - \alpha_i$ sites. Then reverting $k - i$ requests brings us to the point where only the first $B_i + e_i = B_{i+1} + e_{i+1}$th server at any site was used (this site may or may not be site 1). So we need α_{i+1} more requests which, by definition of α_{i+1}, bring us to the point where the $B_{i+1} + e_{i+1}$th server of site $i + 1$ is used.

Thus, for $1 \leq i \leq m$, $\sum_{j=1}^{i} r_j$ can be written

$\sum_{j=1}^{i} ((k - j + 1 - \alpha_j) + (B_{j+1} + e_{j+1} - (B_j + e_j) - 1)(k - j) + \alpha_{j+1})$

$= k - \alpha_1 + (B_{i+1} + e_{i+1})(k - i) - (B_1 + e_1)k + \sum_{j=1}^{i}(B_j + e_j) - (k - i) + \alpha_{i+1}$.

Substituting into (1) then solving for B_{i+1} gives us for all i where $1 \leq i \leq m$:

$$\begin{aligned}
B_{i+1} &\leq \frac{B_1 k + \sum_{j=1}^{i} x_j + \alpha_1 - \alpha_{i+1} - i}{k - i} + \frac{e_1 k - \sum_{j=1}^{i} e_j - e_{i+1}(k - i)}{k - i} \\
&\leq \frac{B_1 k + \sum_{j=1}^{i} x_j}{k - i} + 1 + \frac{e_1 k - \sum_{j=1}^{i} e_j - e_{i+1}(k - i)}{k - i} \quad \text{because } \alpha_{i+1} \geq 1 \\
&\leq \frac{B_1 k + \sum_{j=1}^{i} x_j}{k - i} + 1 + e_1 - e_{i+1} \quad \text{because } e_1 \leq e_2 \leq \ldots \leq e_k \\
&\leq \frac{B_1 k + \sum_{j=1}^{i} x_j}{k - i} + 1 \ . \qquad (4)
\end{aligned}$$

By (3) and (4) we have

$$\begin{aligned}
C_{BODS}(I) &\leq 2\left(r_0 + \sum_{j=1}^{m} x_j + \sum_{j=0}^{m-1} \frac{B_1 k + \sum_{i=1}^{j} x_i}{k-j} + \sum_{j=0}^{m-1} 1\right) \\
&\leq 2\left(r_0 + \sum_{j=1}^{m} x_j + \left(B_1 k + \sum_{j=1}^{m-1} x_j\right) \sum_{j=0}^{m-1} \frac{1}{k-j} + m\right) \\
&\leq 2\left(C_{OPT}(I) + C_{OPT}(I) \ln \frac{k}{k-m} + C_{OPT}(I)\right) \quad \text{by (2).}
\end{aligned}$$

Finally, since $m \leq k-1$, we have $C_{BODS}(I) \leq (2 \ln k + 4) C_{OPT}(I)$. □

The following lemma will be useful in our analysis of BODS on HST's. The proof mimics the proof of Theorem 2.

Lemma 2. *In a star, the number of requests serviced remotely by BODS is at most $2 \ln k + 4$ times the number of requests serviced remotely by any optimal assignment with B_j servers per site.*

3 Generalization to HSTs

We now generalize this result to one on hierarchically separated trees. The theorem and proof presented in this section are based on that of section 3 in [12].

Definition 1. *An α-hierarchically separated tree (α-HST) is a rooted tree $T = (V, E)$ with a distance function on the edges such that (1) If two nodes are siblings in the tree, they are both the same distance from their parent; (2) The distance from a node to its parent is α times the distance of the node to its child; and (3) All leaves are at the same level of the tree.*

To enable us to analyze BODS when the metric space is an HST, we define a variation of our original problem. The results we obtain on this variation will translate back to the original problem. Let our original transportation problem be referred to as TRN. Recall that in TRN the input request sequence was made up of at most B requests, one request for each server that OPT has. We now define the problem TRN2 to be the same as our original problem except for the following modifications. The input request sequence may now have up to $B + k$ requests (the number of servers available to BODS). At any time, to service a request, an algorithm may choose to pay a *service fee* instead of assigning a server to the request. The cost of the service fee is defined to be the length of the path from the request, to the root of the tree, to a leaf of the tree. For example, if a request appears at the root of the tree, the service fee is equal to the root to leaf distance in the HST. As another example, if a request appears at a leaf of the tree, the service fee is equal to twice the root to leaf distance in the HST.

We must now describe the algorithm BODS for this new problem. BODS will always assign a server to a request, never choosing to pay the service fee. To choose a server, BODS behaves exactly as described above.

Note that the service fee is always an upper bound on the cost of servicing a request using a server. The service fee is set intentionally high to deter the optimal offline algorithm from choosing to pay a service fee over assigning a server to the request. Thus, we may assume that the optimal offline algorithm chooses to pay the service fee for a request only when it runs out of servers.

Lemma 3. *If BODS is* $+1$*-server* c*-competitive for TRN2, then BODS is* $+1$*-server* c*-competitive for TRN.*

Lemma 4. *There exists a worst-case input instance* I *in TRN2 against BODS in which OPT does not pay any service fees.*

Theorem 3. *BODS is +1-server* $(8\ln k + 18)$*-competitive for TRN2 when the metric space is an* α*-HST* T *where* $\alpha \geq 4\ln k + 9$*, the server sites are at the leaves of* T*, and the requests arrive at the leaves or at the root of* T*.*

Proof. By Lemma 4, we need only consider input instances where there are at most B requests.

The proof is by induction on the number of levels in T. The base case is the uniform star metric, which we already proved in Theorem 2. For the inductive step, we show that if the theorem is true for each subtree S_i of T that is rooted at child i of the root of T, $1 \leq i \leq z$, then it must be true for T itself.

To be more precise, let H be the height of T, let S be any α-HST of height $h \leq H - 1$ with server sites at the leaves, and let $C_{ALG}(S)$ be the cost of running the algorithm ALG on any input sequence of requests at the leaves or root of S. We assume that $C_{BODS}(S)/C_{OPT}(S) \leq 8\ln k + 18$ holds for any S, and show that this assumption implies that for any α-HST T of height $h+1$, $C_{BODS}(T)/C_{OPT}(T) \leq 8\ln k + 18$.

Let δ be the distance from r, the root of T, to each of its children. Let $\beta = \sum_{i=1}^{H-1} 1/\alpha^i$, where H is the number of levels in T. Note that, $(\beta+1)\delta$ is the distance from r to a leaf of T and $\beta\delta$ is the distance from one of r's children to one of the leaves of T descendant from that child. Let m_i^* and m_i be the number of times that OPT and BODS (respectively) assigned servers in subtree S_i to requests that are *not* in S_i. Let $m^* = \sum_{i=1}^{z} m_i^*$ and $m = \sum_{i=1}^{z} m_i$. So m^* and m are the number of servers that OPT and BODS, respectively, assign to requests outside their subtrees. Let S_i^+ be the instance on subtree S_i defined by the servers of T in S_i, and a subsequence of the requests of the input sequence in T that consists of requests that are serviced by BODS using servers in S_i, where requests outside of S_i are replaced by requests at the root of S_i. Note that there are m_i replacements. Note that S_i is an α-HST with depth $H-1$. Let S_i^B be the instance on subtree S_i obtained from S_i^+ by removing the m_i requests at the root of S_i. It is the case that

$$C_{BODS}(T) \leq \sum_{i=1}^{z} C_{BODS}(S_i^+) + \sum_{i=1}^{z}((\beta+1)\delta + \delta)m_i \ . \tag{5}$$

To justify this, first note that the second term of the right hand side reflects (for each subtree) m_i times the distance from the requests outside S_i (at the leaves of T) to the root r of T, then down to the root of S_i. Also, remember that by definition of BODS there are no requests for which BODS pays a service fee. Thus, any request x is assigned by BODS to a server y. If server y is in subtree S_i, then x belongs to S_i^+. If x appears in S_i, the cost of servicing x is accounted for in the i'th summand in the first summation. If x appears in another subtree or at the root of T, then the cost of servicing x is accounted for in the i'th summand in both the first and the second summations.

Let $R = 8 \ln k + 18$. By the inductive hypothesis, we know for all $1 \leq i \leq z$,

$$C_{BODS}(S_i^+) \leq R \cdot C_{OPT}(S_i^+) \ . \tag{6}$$

We can also observe that

$$C_{OPT}(S_i^+) = C_{OPT}(S_i^B) + \beta\delta m_i \ . \tag{7}$$

By (5), (6), (7), and the definition of m,

$$C_{BODS}(T) \leq R \sum_{i=1}^{z} C_{OPT}(S_i^B) + (R\beta\delta + 2\delta + \beta\delta)m \ . \tag{8}$$

Now all we have left to show is that the right hand side of (8) is less than or equal to $R \cdot C_{OPT}(T)$.

We start by observing that

$$C_{OPT}(T) \geq \sum_{i=1}^{z} C_{OPT}(S_i^B) + \delta m^* \ . \tag{9}$$

Let $R_m = 2 \ln k + 4$. By our assumption that $\alpha \geq 2R_m + 1$, we have

$$\beta = \sum_{i=1}^{H-1} \frac{1}{\alpha^i} \leq \frac{1}{1 - 1/\alpha} - 1 = \frac{1}{\alpha - 1} \leq \frac{1}{2R_m} \ . \tag{10}$$

Note that $R = 4R_m + 2$. Using this fact along with (10), we have

$$R = \frac{1}{2}R + 2R_m + 1 \geq \frac{1}{2}R(2R_m\beta) + 2R_m + 2R_m\beta \geq R_m(R\beta + 2 + \beta) \ . \tag{11}$$

We now need to relate the values m and m^*. Recall that BODS always services requests as locally as possible. If we think of a request that is matched to a server within its own subtree as a *local* assignment, we can reduce this input instance on T to one on the uniform star, where the subtrees of T are the leaves of the star. Since all inputs under consideration have at most B servers, this reduction will be an instance of the problem from section 2. Therefore by Lemma 2, we know that

$$m \leq R_m \cdot m^* \ . \tag{12}$$

By combining (8), (9), (11), and (12), we have $C_{BODS}(T) \leq R \cdot C_{OPT}(T)$. □

Combining Theorem 3 with Lemma 3 gives us the following result.

Theorem 4. *BODS is +1-server* $(8 \ln k + 18)$*-competitive for TRN when the metric space is an* α*-HST* T *where* $\alpha \geq 4 \ln k + 9$*, the server sites are at the leaves of the tree, and the requests arrive at the leaves of* T*.*

4 Generalization to Any Metric Space

We are now ready to extend our results to any metric space that has k server sites plus one server per site for the online algorithm. In this setting requests may arrive at any point in the metric space, whether or not they are designated server sites.

The new algorithm (GBODS) is as follows: first use the procedure given in [4] to generate a random $(4 \ln k + 9)$-HST, call it T, from the metric induced on the k server sites. Then, whenever a request q arrives in the original metric space, we find its nearest server site, and create a new request there, calling it q'. Let $Q = \{q_1, q_2, \ldots, q_B\}$ be the sequence of requests that arrive, and let $Q' = \{q'_2, q'_3, \ldots, q'_B\}$ be the corresponding set of requests created at the server sites nearest to the requests in Q. Let $T(Q')$ be the input instance on HST T with request sequence Q'. We use BODS on $T(Q')$ to find an available server s'_i for each q'_i in Q'. We then assign each server s'_i to each original request q_i in Q.

Fakcharoenphol et al [4] showed that any metric space of n points can be approximated by a randomly generated α-HST where the points in the metric will be at the leaves of the tree, and the expected distance between points in the tree will be no more than $\alpha \log n$ times their original distance. Applying this fact along with our Theorem 4, and losing a constant factor due to applications of the triangle inequality when mapping the solution of input $T(Q')$ back to the points of Q in the original metric space, we have the following theorem.

Theorem 5. *In expectation, for the online transportation problem, the algorithm GBODS is +1-server* $O(\log^3 k)$*-competitive.*

5 Conclusions

The interesting question that naturally arises from the results here is:

> *Is there a* $+1$*-server poly-log-competitive deterministic algorithm for online transportation on an arbitrary metric space?*

Intuitively the star is the hardest metric space, and the deterministic +1-server $O(\log k)$-competitiveness result for a star should extend to an arbitrary metric space. But current metric embedding techniques do not seem sufficient to address this question. One can obtain a poly-log-competitive deterministic offline algorithm by deterministically generating a collection of HST's, and then using the tree that gives the best results [2]. But it is not clear how an online algorithm should learn or construct the right metric embedding online as it sees requests.

So it seems like the above question could well be a vehicle to extend the current understanding of metric embeddings. We are aware of the result in [5] that gives a deterministic online algorithm that uses a collection of HST's and is poly-log-competitive. In the algorithm in [5], $\log k$ different HST's are generated a priori, and then the online algorithm always uses an HST that is guaranteed to approximate the distance between points that arrive online. This does not work for online matching and online transportation because it essentially simulates the greedy algorithm, which is not competitive in a general metric space.

Alternatively, if it somehow turned out that there is no +1-server poly-log-competitive deterministic algorithm for online transportation on an arbitrary metric space, then this would be interesting because it would be the first example of an online matching/transportation problem where the hardest metric space was not a star.

Acknowledgments. We thank Anupam Gupta for helpful discussions.

References

1. Bartal, Y.: On approximating arbitrary metrics by tree metrics. In: ACM Symposium on Theory of Computing, pp. 161–168 (1998)
2. Charikar, M., Chekuri, C., Goel, A., Guha, S., Plotkin, S.: Approximating a finite metric by a small number of tree metrics. In: Symposium on Foundations of Computer Science, p. 379 (1998)
3. Csaba, B., Pluhar, A.: A randomized algorithm for the on-line weighted bipartite matching problem. Journal of Scheduling (to appear)
4. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. Journal of Computer and System Sciences Special Issue on STOC 2003 69(3), 485–497 (2004)
5. Gupta, A., Hajiaghayi, M.T., Räcke, H.: Oblivious network design. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 970–979 (2006)
6. Kalyanasundaram, B., Pruhs, K.: Online weighted matching. Journal of Algorithms 14(3), 478–488 (1993)
7. Kalyanasundaram, B., Pruhs, K.: On-line network optimization problems. In: Developments from a June 1996 seminar on Online Algorithms, pp. 268–280. Springer, Heidelberg (1998)
8. Kalyanasundaram, B., Pruhs, K.R.: The online transportation problem. SIAM Journal of Discrete Mathematics 13(3), 370–383 (2000)
9. Kennington, J.L., Helgason, R.V.: Algorithms for Network Programming. John Wiley & Sons, Inc., New York, NY, USA (1980)
10. Khuller, S., Mitchell, S.G., Vazirani, V.V.: On-line algorithms for weighted bipartite matching and stable marriages. Theoretical Computer Science 127(2), 255–267 (1994)
11. Lawler, E.: Combinatorial Optimization: Networks and Matroids. Holt, Rinehart & Winston, New York (1976)
12. Meyerson, A., Nanavati, A., Poplawski, L.: Randomized online algorithms for minimum metric bipartite matching. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 954–959 (2006)

Average Rate Speed Scaling*

Nikhil Bansal[1], David P. Bunde[2], Ho-Leung Chan[3], and Kirk Pruhs[3]

[1] IBM T. J. Watson Research Center
nikhil@us.ibm.com
[2] Computer Science Department, Knox College
dbunde@knox.edu
[3] Computer Science Department, University of Pittsburgh
{hlchan,kirk}@cs.pitt.edu

Abstract. Speed scaling is a power management technique that involves dynamically changing the speed of a processor. This gives rise to dual-objective scheduling problems, where the operating system both wants to conserve energy and optimize some Quality of Service (QoS) measure of the resulting schedule. Yao, Demers, and Shenker [8] considered the problem where the QoS constraint is deadline feasibility and the objective is to minimize the energy used. They proposed an online speed scaling algorithm Average Rate (AVR) that runs each job at a constant speed between its release and its deadline. They showed that the competitive ratio of AVR is at most $(2\alpha)^\alpha/2$ if a processor running at speed s uses power s^α. We show the competitive ratio of AVR is at least $((2-\delta)\alpha)^\alpha/2$, where δ is a function of α that approaches zero as α approaches infinity. This shows that the competitive analysis of AVR by Yao, Demers, and Shenker is essentially tight, at least for large α. We also give an alternative proof that the competitive ratio of AVR is at most $(2\alpha)^\alpha/2$ using a potential function argument. We believe that this analysis is significantly simpler and more elementary than the original analysis of AVR in [8].

1 Introduction

Current processors produced by Intel and AMD allow the speed of the processor to be changed dynamically. Intel's SpeedStep and AMD's PowerNOW technologies allow the Windows XP operating system to dynamically change the speed of such a processor to conserve energy. In this setting, the operating system must not only have a *job selection policy* to determine which job to run, but also a *speed scaling* policy to determine the speed at which the job will be run. In current CMOS based processors, the speed satisfies the well-known cube-root-rule, that the speed is approximately the cube root of the power. Energy consumption is power integrated over time. The operating system is faced with a dual objective optimization problem as it both wants to conserve energy, and optimize some Quality of Service (QoS) measure of the resulting schedule.

* D.P. Bunde was supported in part by Howard Hughes Medical Institute grant 52005130. Ho-Leung Chan and Kirk Pruhs were supported in part by NSF grants CNS-0325353, CCF-0514058 and IIS-0534531.

The first theoretical worst-case study of speed scaling algorithms was in the seminal paper [8] by Yao, Demers, and Shenker. Their QoS objective was deadline feasibility and the objective was to minimize the energy used. More precisely, each job i has a release time r_i when it arrives in the system, a work requirement w_i, and a deadline d_i by which the job must be finished. If job i runs at constant speed s, then it completes in w_i/s units of time. In this setting, an optimal job selection policy is Earliest Deadline First (EDF). They assumed a speed to power function $P(s) = s^\alpha$, where $\alpha > 1$ is some constant. If the cube-root rule holds, then $\alpha = 3$. Yao, Demers, and Shenker [8] showed that the optimal energy feasible schedule is found by a simple greedy algorithm that we call YDS.

Yao, Demers, and Shenker [8] also proposed an online speed scaling algorithm, Average Rate (AVR). Conceptually, AVR runs each job i at speed $w_i/(d_i - r_i)$ throughout interval $[r_i, d_i]$, independent of other jobs. This spreads the work of each job as evenly over time as possible. By the convexity of the speed to power function, this even spreading is energy optimal if the instance consists of only one job. The speed of the processor at any time t is then just the sum of the speeds of the jobs active at that time, that is $\sum_{i:t\in[r_i,d_i]} \frac{w_i}{d_i-r_i}$. AVR is an appealing speed scaling algorithm because in some sense it is perfectly fair to all jobs, and each job runs as if it were the only job in the instance.

Yao, Demers, and Shenker [8] showed that the competitive ratio, with respect to energy, of AVR is at least α^α. They also showed that the competitive ratio of AVR, with respect to energy, is at most $(2\alpha)^\alpha/2$. We now outline this upper bound competitive analysis of AVR. A job is defined to be of *type A* if the optimal schedule is always ahead of AVR on this job. A job is defined to be of *type B* if AVR is always ahead of the optimal schedule on this job. A schedule is *bitonic* if every job is of type A or type B. [8] observes that there is a worst-case instance that is bitonic, and that the competitive ratio of AVR is at most $2^{\alpha-1}$ times the competitive ratio of AVR on instances of jobs of just one type (A or B). [8] then considers instances consisting only of type-A jobs. [8] then introduces an auxiliary objective function that is related to, but is not exactly, the energy used. In a somewhat involved reduction, [8] shows that with respect to this auxiliary objective, there is a worst-case instance where the optimal schedule is non-preemptive, each job starts when it is released, and the spans of the jobs are nested (where the *span* of job i is the interval $[r_i, d_i]$). When $\alpha = 2$, [8] then shows that for such instances, optimizing the auxiliary objective function can be represented in terms of the eigenvalues of a particular tree-induced matrix, and shows how to bound the largest eigenvalue for such tree-induced matrices. [8] states that this argument can be readily generalized to an arbitrary α, and using Hölder's inequality, give a bound on the ℓ_p norm of a certain tree-induced matrix that would replace the eigenvalue argument used in the $\alpha = 2$ case.

So the natural question left open is, "What is the exact competitive ratio of AVR?" Based on simulation results, [8] conjectured that the competitive ratio of AVR is exactly α^α. That is, that the lower bound in [8] is correct, and intuitively, that AVR can not simultaneously be losing badly on both type-A and type-B jobs. In the case that the cube-root rule holds, $\alpha^\alpha = 3^3 = 27$ is the best known competitive ratio for any online algorithm. If the conjecture from [8] was true, this would be evidence in favor of adopting the AVR speed scaling policy. Not only would AVR have the best known competitive ratio in the case that the cube-root rule holds, but AVR is appealingly fair to all jobs.

Unfortunately, in section 4, we show that the upper bound on the competitive ratio from [8] is essentially tight, at least for larger α. More precisely, we show that AVR has competitive ratio at least $((2-\delta)\alpha)^\alpha/2$, where δ is a function of α that approaches zero as α approaches infinity. In the case obeying the cube-root rule, we get a lower bound of approximately 48 on the competitive ratio of AVR.

In section 5, we give an alternative proof that the competitive ratio of AVR is at most $(2\alpha)^\alpha/2$. Our analysis uses a potential function argument. We believe that this analysis is significantly simpler and more elementary than the original analysis of AVR in [8]. Our competitive analysis of AVR branches off from the analysis in [8] outlined above after the observation that the competitive ratio of AVR is at most $2^{\alpha-1}$ times the competitive ratio of AVR on jobs of just one type. We give a potential function argument that AVR is α^α-competitive on type-A jobs. We include a complete analysis of AVR in this paper, including the elements of the analysis from [8] that we use. In principle, verifying this analysis requires only basic algebra, except that some basic calculus is used to verify the positivity/negativity of certain polynomials over particular ranges.

2 Other Related Results

There are now enough speed scaling papers in the literature that it is not practical to survey all such papers here. We limit ourselves to those papers most related to the results presented here.

Yao, Demers, and Shenker [8] also proposed another online speed scaling algorithm, Optimal Available (OA). The algorithm OA runs at the optimal speed (which can be computed using the YDS algorithm) assuming the current state and that no more jobs will be released in the future. [8] showed that the competitive ratio of OA is at least α^α. Using a potential function analysis, Bansal, Kimbrel, and Pruhs [2] showed that OA is actually α^α-competitive.

Bansal, Kimbrel, and Pruhs [2] also introduced an online speed scaling algorithm that we call BKP. Intuitively, BKP tries to mimic the offline YDS schedule in some way. Formally, at time t BKP runs at speed $e\ v(t)$ where $v(t) = \max_{t'>t} \frac{w(t,et-(e-1)t',t')}{e(t'-t)}$ and $w(t,t_1,t_2)$ is the amount of work that has release time at least t_1, deadline at most t_2, and that has already arrived by time t. [2] showed that BKP is simultaneously $O(1)$-competitive for total energy, maximum temperature (assuming cooling obeys Fourier's law), maximum power, and maximum speed. Specifically, [2] showed that the competitive ratio of BKP with respect to energy is at most $2(\alpha/(\alpha-1))^\alpha e^\alpha$. With respect to maximum speed, [2] showed that BKP is e-competitive and that this competitive ratio is optimal among randomized algorithms.

A naive implementation of YDS runs in time $O(n^3)$. This can be improved to $O(n^2)$ if the intervals have a tree structure [4]. Li, Yao and Yao [5] gave an implementation that runs in $O(n^2 \log n)$ time for the general case. For hard real-time jobs with fixed priorities, Yun and Kim [9] showed that it is NP-hard to compute a minimum-energy schedule. They also gave a fully polynomial time approximation scheme for the problem. Kwon and Kim [3] gave a polynomial time algorithm to schedule a processor with discrete speeds. Li and Yao [6] gave an algorithm with running time $O(d \cdot n \log n)$ where d is the number of speeds.

Albers, Müller, and Schmelzer [1] consider the problem of finding energy-efficient deadline-feasible schedules on multiprocessors. [1] showed that the offline problem is NP-hard, and gave $O(1)$-approximation algorithms. [1] also gave online algorithms that are $O(1)$-competitive when job deadlines occur in the same order as their release times.

3 Formal Problem Statement

A problem instance consists of n jobs. Job i has a release time r_i, a deadline $d_i > r_i$, and work $w_i > 0$. In the online version of the problem, the scheduler learns about a job only at its release time; at this time, the scheduler also learns the exact work requirement and the deadline of the job. We assume that time is continuous. A schedule specifies for each time a job to be run and a speed at which to run the job. The speed is the amount of work performed on the job per unit time. A job with work w run at a constant speed s thus takes $\frac{w}{s}$ time to complete. More generally, the work done on a job during a time period is the integral over that time period of the speed at which the job is run. A schedule is *feasible* if for each job i, work at least w_i is done on job i during $[r_i, d_i]$. Note that the times at which work is performed on job i do not have to be contiguous. If a job is run at speed s, then the power is $P(s) = s^\alpha$ for some constant $\alpha > 1$.

The energy used during a time period is the integral of the power over that time period. Our objective is to minimize the total energy used by the schedule.

If A is a scheduling algorithm, then $A(I)$ denotes the schedule output by A on input I. A schedule is R-competitive for a particular objective function if the value of that objective function on the schedule is at most R times the value of the objective function on an optimal schedule. An online scheduling algorithm A is R-competitive, or has competitive ratio R, if $A(I)$ is R-competitive for all instances.

For a schedule T, let $s_{T,j}(t)$ denote the speed job j runs at time t in the schedule T, and let $s_T(t) = \sum_j s_{T,j}(t)$ denote the speed of the processor at time t in schedule T. If U is a subcollection of jobs, let $s_{T,U}(t)$ denote the sum of the speeds of the jobs in U at time t in the schedule T. We will also substitute an algorithm for a schedule in this notation. So for example, $s_{AVR}(t)$ is the speed of the algorithm AVR at time t. We use OPT to denote a particular optimal schedule. We say that job i is *active* between its release time and its deadline. We call $w_i/(d_i - r_i)$ the *density* of job i since this is the job's work divided by the length of the interval in which it is active.

Algorithm AVR: At all times t, run the earliest-deadline job at speed $s_{AVR}(t) = \sum_i \frac{w_i}{d_i - r_i}$, where the sum is over jobs i active at time t.

Consider a fixed optimum schedule OPT. A job is said to be of *type A* if

$$\int_{r_j}^{t} s_{OPT,j}(t)dt \geq \int_{r_j}^{t} \frac{w_j}{d_i - r_i} dt \qquad \text{for all } r_j \leq t \leq d_j$$

Intuitively, these are the jobs that OPT runs consistently ahead of their density. Similarly, the jobs of *type B* are those that OPT runs consistently behind their density, meaning they satisfy

$$\int_{r_j}^{t} s_{OPT,j}(t)dt \leq \int_{r_j}^{t} \frac{w_j}{d_i - r_i} dt \qquad \text{for all } r_j \leq t \leq d_j.$$

In general, a job need not be of either type (or it can also be of both types, in which case OPT executes exactly as in AVR). We say an instance is *bitonic* if every job is of type A, type B, or both (in which case it is arbitrarily assigned one of the types). A simple observation (Lemma 5) shows that if AVR is c-competitive for bitonic instances, then it is also c-competitive in general.

4 The Lower Bound

We give an instance on which AVR uses up at least $((2-\delta)\alpha)^\alpha/2$ times the energy used by an energy optimum solution, where δ is a function of α that tends to zero as α increases.

Instance Description: For convenience we will work with a continuous version of the job instance. We say that work arrives at rate $a(t)$ at time t to mean that $a(t)dt$ units of work arrive during the infinitesimally small interval $[t, t+dt]$.

The instance consists of two sets of jobs A and B. The work in A arrives during the time interval $[0, 1-\epsilon]$, at rate

$$a(t) = \frac{1}{(1-t)^{1/\alpha}}$$

and all the work in A has deadline 1. Here $\epsilon > 0$ is an arbitrarily small but fixed constant. The work in B arrives during the interval $[1-1/c, 1-\epsilon/c]$ (where c is a constant that will be set to $\alpha - 1$ later) at rate

$$b(t) = \frac{c}{c^{1/\alpha}(1-t)^{1/\alpha}}$$

and the work in B arriving at time t has deadline $1 + c(1-t)$.

Lemma 1. *On the instance above, the optimal algorithm uses total energy at most* $2\ln(1/\epsilon)$.

Proof: It suffices to give some feasible schedule that uses energy $2\ln(1/\epsilon)$. Consider the schedule that completes all jobs in A by running at speed $a(t)$ during $[0, 1-\epsilon]$. The energy usage is

$$\int_0^{1-\epsilon} (a(t))^\alpha dt = [-\ln(1-t)]_0^{1-\epsilon} = \ln(1/\epsilon)$$

For jobs in B, note that they are released before time 1 and have deadlines in $[1+\epsilon, 2]$. Consider any time $x \in [1+\epsilon, 2]$. The jobs with deadline in $[1+\epsilon, x]$ are released during $[1-\frac{x-1}{c}, 1-\frac{\epsilon}{c}]$. Their total amount of work is

$$\int_{1-(x-1)/c}^{1-\epsilon/c} b(t)dt = \int_{1-(x-1)/c}^{1-\epsilon/c} \frac{c}{c^{1/\alpha}(1-t)^{1/\alpha}} dt$$

Let $y = 1 + c(1-t)$. Then $dy = -c \cdot dt$, and the amount of work equals

$$\int_{1-(x-1)/c}^{1-\epsilon/c} \frac{c}{c^{1/\alpha}(1-t)^{1/\alpha}} dt = \int_{x}^{1+\epsilon} \frac{-1}{(y-1)^{1/\alpha}} dy = \int_{1+\epsilon}^{x} \frac{1}{(y-1)^{1/\alpha}} dy$$

Therefore, consider the schedule that processes jobs in B at speed $\hat{b}(y) = \frac{1}{(y-1)^{1/\alpha}}$ continuously during $[1+\epsilon, 2]$. For any $x \in [1+\epsilon, 2]$, the amount of work done by time x equals the amount work with deadline by x. So the schedule completes each job in B by its deadline. The energy usage to complete all jobs in B is

$$\int_{1+\epsilon}^{2} (\hat{b}(t))^{\alpha} dt = [\ln(y-1)]_{1+\epsilon}^{2} = \ln(1/\epsilon)$$

Since the intervals of execution of work in A and B do not overlap, the total energy used is $2\ln(1/\epsilon)$ and the lemma follows. □

Lemma 2. *On the instance above, AVR uses total energy at least* $\alpha^{\alpha}(1+\frac{c}{c^{1/\alpha}(c+1)})^{\alpha}\ln(1/\epsilon) + K$*, where K is a constant independent of ϵ.*

Proof: Consider the work in A. The work released at time t is scheduled by AVR uniformly during the interval $[t, 1]$. Thus, at any time $x \in [0, 1]$, the density due to work in A is

$$\mathrm{den}_a(x) = \int_0^x a(t) \cdot \frac{1}{1-t} dt = \int_0^x \frac{1}{(1-t)^{1/\alpha}} \cdot \frac{1}{1-t} dt = \alpha \left(\frac{1}{(1-x)^{1/\alpha}} - 1 \right)$$

Now consider the work in B. Note that for work released at time t, the duration between its release time and deadline is $1 + c(1-t) - t = (c+1)(1-t)$. Thus, at any time $x \in [1-\frac{1}{c}, 1-\frac{\epsilon}{c}]$, the density due to work in B is

$$\begin{aligned}\mathrm{den}_b(x) &= \int_{1-1/c}^{x} \frac{c}{c^{1/\alpha}(1-t)^{1/\alpha}} \cdot \frac{1}{(c+1)(1-t)} dt \\ &= \frac{c}{c^{1/\alpha}(c+1)} \cdot \alpha \left(\frac{1}{(1-x)^{1/\alpha}} - c^{1/\alpha} \right)\end{aligned}$$

During the interval $[1-\frac{1}{c}, 1-\epsilon]$, AVR runs at speed equal to the total density due to work in A and B. Therefore, the energy usage of AVR is at least

$$\begin{aligned}&\int_{1-1/c}^{1-\epsilon} (\mathrm{den}_a(t) + \mathrm{den}_b(t))^{\alpha} dt \\ &\quad = \int_{1-1/c}^{1-\epsilon} \left(\alpha \left(1 + \frac{c}{c^{1/\alpha}(c+1)} \right) \cdot \frac{1}{(1-t)^{1/\alpha}} - \alpha \frac{2c+1}{c+1} \right)^{\alpha} dt \qquad (1)\end{aligned}$$

Let $Y = 1 + \frac{c}{c^{1/\alpha}(c+1)}$. Note that for all $t \in [1-\frac{1}{c}, 1-\epsilon]$, we have that $1-t \leq 1/c$ and hence

$$\frac{2c+1}{c+1} \cdot \frac{(1-t)^{1/\alpha}}{Y} \leq \frac{2c+1}{c+1} \cdot \frac{1}{c^{1/\alpha}} \cdot \frac{c^{1/\alpha}(c+1)}{c^{1/\alpha}(c+1)+c} \leq \frac{2c+1}{(c+1)+c} = 1$$

Then, by factoring $\alpha Y \frac{1}{(1-t)^{1/\alpha}}$, the right side of (1) can be written as

$$\begin{aligned}
&\int_{1-1/c}^{1-\epsilon} \alpha^\alpha Y^\alpha \frac{1}{1-t}\left(1 - \frac{2c+1}{c+1} \cdot \frac{(1-t)^{1/\alpha}}{Y}\right)^\alpha dt \\
&\geq \int_{1-1/c}^{1-\epsilon} \frac{\alpha^\alpha Y^\alpha}{1-t}\left(1 - \alpha\frac{2c+1}{c+1} \cdot \frac{(1-t)^{1/\alpha}}{Y}\right) dt \quad \text{as } 1-\alpha x \leq (1-x)^\alpha \text{ for } x \leq 1 \\
&= \int_{1-1/c}^{1-\epsilon} \alpha^\alpha Y^\alpha \left(\frac{1}{1-t} - Z(1-t)^{(1/\alpha)-1}\right) dt \qquad \text{where } Z = \tfrac{\alpha(2c+1)}{Y(c+1)} \\
&= \alpha^\alpha Y^\alpha \left[-\ln(1-t) + \alpha Z(1-t)^{1/\alpha}\right]_{1-1/c}^{1-\epsilon} \\
&= \alpha^\alpha Y^\alpha \left(-\ln\epsilon + \alpha Z \epsilon^{1/\alpha} + \ln\frac{1}{c} - \alpha Z(\frac{1}{c})^{1/\alpha}\right) \\
&\geq \alpha^\alpha Y^\alpha \ln(1/\epsilon) + \alpha^\alpha Y^\alpha \left(\ln\frac{1}{c} - \alpha Z(\frac{1}{c})^{1/\alpha}\right) \qquad \text{since } \epsilon > 0
\end{aligned}$$

Since α, c, Y and Z are independent of ϵ the lemma follows. □

Theorem 3. *The competitive ratio of AVR is at least $((2-\delta)\alpha)^\alpha/2$, where δ is a function of α that tends to zero as α increases.*

Proof: By Lemma 1 and 2, when ϵ tends to zero, the competitive ratio of AVR is at least $((1+\frac{c^{1-1/\alpha}}{c+1})\alpha)^\alpha/2$. Putting $c = \alpha - 1$, the competitive ratio is at least $((1+\frac{(\alpha-1)^{1-1/\alpha}}{\alpha})\alpha)^\alpha/2$, which equals $((2-\delta)\alpha)^\alpha/2$ where $\delta = 1 - \frac{(\alpha-1)^{1-1/\alpha}}{\alpha}$.

Note that for large α (in particular for $\alpha \geq 2$, we have that

$$\begin{aligned}
\delta &= 1 - (\alpha-1)^{-1/\alpha}\frac{\alpha-1}{\alpha} \\
&= 1 - e^{(-1/\alpha)\ln(\alpha-1)}(1-\frac{1}{\alpha}) \\
&\leq 1 - \left(1 - \frac{1}{\alpha}\ln(\alpha-1)\right)(1-\frac{1}{\alpha}) \qquad \text{using } e^x \geq 1+x \text{ for } x<0 \\
&= \frac{\ln(\alpha-1)}{\alpha} + \frac{1}{\alpha} - \frac{\ln(\alpha-1)}{\alpha^2} \qquad (2)
\end{aligned}$$

Hence δ approaches zero as α approaches infinity. □

We remark that our bound $((2-\delta)\alpha)^\alpha/2$ is asymptotically $2^{\alpha-1}\alpha^{\alpha-1/2-o(1)}$ for large α, and hence within $\alpha^{1/2+o(1)}$ of the best known upper bound. To see this, by (2), we obtain that

$$\lim_{\alpha\to\infty}\left(\frac{\alpha}{\ln\alpha}\right)\delta \leq \lim_{\alpha\to\infty}\left(\frac{\ln(\alpha-1)}{\ln\alpha} + \frac{1}{\ln\alpha} - \frac{\ln(\alpha-1)}{\alpha\ln\alpha}\right) = 1.$$

Similarly,

$$\delta \geq 1 - \frac{\alpha^{1-1/\alpha}}{\alpha} = 1 - \frac{1}{e^{(\ln \alpha/\alpha)}} \geq 1 - \frac{1}{1 + \frac{1}{\alpha}\ln \alpha} = \frac{\ln \alpha}{\alpha + \ln \alpha},$$

and hence

$$\lim_{\alpha\to\infty} \left(\frac{\alpha}{\ln \alpha}\right) \delta \geq \lim_{\alpha\to\infty} \frac{\alpha}{\alpha + \ln \alpha} = 1.$$

Thus the expression $(2-\delta)^\alpha \alpha^\alpha/2 = 2^{\alpha-1}\alpha^\alpha(1-\delta/2)^\alpha \approx 2^{\alpha-1}\alpha^\alpha\alpha^{-\delta\alpha/(2\ln\alpha)} = 2^{\alpha-1}\alpha^\alpha\alpha^{-1/2-o(1)}$.

5 An Elementary Proof that AVR is $2^{\alpha-1}\alpha^\alpha$-competitive

This section gives a complete elementary proof that AVR is $2^{\alpha-1}\alpha^\alpha$-competitive. This proof uses some elements of the analysis of AVR in [8] and some variations on elements of the analysis of OA in [2]. We start with the analysis of AVR on instances consisting of only type-A jobs. The analysis for general instances then follows along the same lines as in [8], and is included here for completeness.

Lemma 4. *For instances consisting of only type-A jobs,* AVR *is α^α-competitive with respect to energy.*

Proof: We use an amortized local competitiveness argument (for more information on such arguments in scheduling problems, see [7]). At any time t, either a task arrives or finishes, or else an infinitesimal interval of time dt elapses and AVR consumes $s_{AVR}(t)^\alpha dt$ units of energy. We will define a potential function $\phi(t)$ that satisfies the following properties:

– The potential function $\phi(t)$ has value 0 before any jobs arrive and after the last deadline.
– The potential function $\phi(t)$ does not increase as a result of AVR completing a job, OPT completing a job, or the release of a job.
– At any time t,

$$s_{AVR}(t)^\alpha + \frac{d\phi(t)}{dt} \leq \alpha^\alpha s_{OPT}(t)^\alpha. \tag{3}$$

Integrating equation 3 over time and using the other two stated properties, we can conclude the desired result.

Before we can define the potential function we need to introduce some notation. Let t_0 denote the current time and t_i denote the time of the i^{th} deadline occurring after t_0. Then let I_i denote the interval of time $[t_i, t_{i+1})$. Let $\tau_i = t_{i+1} - t_i$ be the length of interval I_i. Let s_i denote the speed at which AVR will work during interval I_i if no new jobs arrive. This can be computed by summing the densities of active jobs whose deadline is at or after time t_{i+1}. Let $w_{AVR,i} = s_i\tau_i$ denote the amount of work that AVR plans to complete during interval I_i. Let $w_{OPT,i}$ be the portion of the work AVR allocates to interval I_i that OPT has not yet completed. Because all jobs are of type A, all work that is unfinished by OPT is also unfinished by AVR. Without loss of generality, we assume that when OPT is working on a job j, work is removed from the term $w_{OPT,i}$ that contains work from job j with the smallest index i. That is, OPT removes work from the earlier intervals first.

We define the potential function $\phi(t)$ as follows:

$$\phi(t) = \alpha \sum_{i \geq 0} s_i^{\alpha-1}(w_{AVR,i} - \alpha w_{OPT,i}) \tag{4}$$

This potential function is a slight modification of the potential function used in [2] to analyze the algorithm OA. The difference is that their potential function uses $w_{OPT,i}$ to denote the work of jobs unfinished for OPT with deadline in I_i.

Now we show that ϕ has the claimed properties. This function is clearly 0 when there are no active jobs. The completion of a job by OPT also has no effect since the potential is a continuous function of $w_{OPT,i}$. The situation when AVR completes a job is slightly more complicated. Observe that a job completes under AVR if and only if the size of the interval I_0 shrinks to 0, i.e. when the current time t_0 becomes equal to t_1, which shifts all the indices. At the moment this happens AVR has completed all the work allocated to I_0 and hence $w_{AVR,0} = 0$. Because all jobs are of type A, OPT has also completed the work allocated to I_1 so $w_{\mathrm{OPT},0} = 0$. Thus, the potential is continuous even in this case. (This is the only time we use that all the jobs are of type A.)

Arrival Case: The next case to consider is when a new job j arrives. First observe that adding a zero work job with deadline d_j does not change the value of the potential function ϕ. Thus, we may assume that the new job's deadline is t_k for some k. Let y be the density of the new job. Then the release increases the density of intervals $I_0, I_1, \ldots, I_{k-1}$ by y, increasing the weight of interval I_i by $y\tau_i$ for $0 \leq i \leq k-1$. This changes the potential function by

$$\begin{aligned}\Delta\phi = \alpha \sum_{i=0}^{k-1} \left(\frac{w_{AVR,i} + y\tau_i}{\tau_i}\right)^{\alpha-1} ((w_{AVR,i} + y\tau_i) - \alpha(w_{OPT,i} + y\tau_i)) \\ -\alpha \sum_{i=0}^{k-1} \left(\frac{w_{AVR,i}}{\tau_i}\right)^{\alpha-1} (w_{AVR,i} - \alpha w_{OPT,i}).\end{aligned} \tag{5}$$

This expression can be rearranged into

$$\sum_{i=0}^{k-1} \frac{\alpha}{\tau_i^{\alpha-1}} \Big((w_{AVR,i} + y\tau_i)^{\alpha-1}(w_{AVR,i} - \alpha w_{OPT,i} - (\alpha-1)y\tau_i) \\ - w_{AVR,i}^{\alpha-1}(w_{AVR,i} - \alpha w_{OPT,i})\Big)$$

By making the substitutions $q = w_{AVR,i}$, $\delta = y\tau_i$ and $r = w_{OPT,i}$ each term of this sum becomes a quantity shown to be at most 0 by Lemma 8.

Working case: We now consider times when no job arrives, and no jobs complete. Each s_i, including s_0, remains fixed during this time. We have to show

$$s_{AVR}(t_0)^\alpha - \alpha^\alpha s_{OPT}(t_0)^\alpha + \frac{d\phi(t)}{dt} \leq 0 \tag{6}$$

or equivalently,

$$s_0^\alpha - \alpha^\alpha s_{OPT}(t_0)^\alpha + \frac{d}{dt}(\alpha \sum_{i \geq 0} s_i^{\alpha-1}(w_{AVR,i} - \alpha w_{OPT,i})) \leq 0 \tag{7}$$

As AVR works, $w_{AVR,0}$ is decreasing at rate s_0, and $w_{AVR,i}$ remains fixed for all $i \geq 1$. Since OPT takes work from a single interval I_i, only one of the $w_{OPT,i}$ changes; let it be $w_{OPT,k}$. Then equation (7) is equivalent to

$$s_0^\alpha - \alpha^\alpha s_{OPT}(t_0)^\alpha + (-\alpha s_0^{\alpha-1} s_0 + \alpha^2 s_k^{\alpha-1} s_{OPT}(t_0)) \leq 0$$

Since a job active during one interval is also active in all earlier intervals, $s_k \leq s_0$ and it suffices to show that

$$(1-\alpha)s_0^\alpha + \alpha^2 s_0^{\alpha-1} s_{OPT}(t_0) - \alpha^\alpha s_{OPT}(t_0)^\alpha \leq 0$$

Substituting $z = s_0/s_{OPT}(t_0)$ gives

$$(1-\alpha)z^\alpha + \alpha^2 z^{\alpha-1} - \alpha^\alpha \leq 0 \tag{8}$$

Let $u(z)$ be the polynomial on the left hand side of inequality 8. Note that $u(0) = -\alpha^\alpha$ and $u(+\infty) = -\infty$. In addition, the derivative of $u(z)$ is 0 at only the point $z = \alpha$. Since $u(\alpha) = 0$, we conclude that $u(z)$ is non-positive for $z \geq 0$, which holds because of the definition of z. This establishes inequality 6. □

Lemma 4 and the argument of Yao, Demers, and Shenker [8] proves the $2^{\alpha-1}\alpha^\alpha$-competitiveness of AVR. We now give their argument for completeness.

Lemma 5. *[8] Among those instances on which* AVR *has it worst-case competitive ratio, there is a bitonic instance.*

Proof: Consider a worst-case instance I that is not bitonic. We explain how to transform I into another worst-case instance that is bitonic. There must be a job i that is of neither type A nor type B. By the definition of the types, there has to be some times s, u, with $s < u$, for which one of AVR or OPT is ahead of the other on job i at time s, but behind at time u. By the intermediate value theorem, there must be a time $t \in (s, u)$ where AVR and OPT have completed an equal amount of work w on job i. We say that the *lead changes* at such a time t. We now create a new instance I' from I by replacing job i with two jobs: one with work w released at time r_i with deadline t, and one with work $w_i - w$ released at time t with deadline d_i. It is easy to see that both AVR and OPT always run at the same speed in I' that they did in I. This transformation however reduces the number of lead changes by one. Since there can only be a bounded number of lead changes between YDS = OPT and AVR, a bounded number of applications of this transformation leads to a bitonic instance. □

Lemma 6. *[8] AVR is $2^{\alpha-1}\alpha^\alpha$-competitive on bitonic instances.*

Proof Sketch: Given a bitonic instance, let A be the set of type-A jobs and B be the others. Let AVR_A and AVR_B denote the energy attributable to A and B in the AVR schedule, respectively. Define OPT_A and OPT_B similarly with reference to the schedule OPT.

Next observe that the roles of type-A jobs and type-B jobs can be swapped by reversing time and swapping the release time and deadline for each job. Both YDS and AVR give the same schedule to the forward and backwards versions so Lemma 4 implies that AVR is simultaneously α^α-competitive with respect to energy attributable to type-A jobs and energy attributable to type-B jobs.

The proof follows by combining the schedules for the jobs of different types. The optimal cost is clearly at least $\text{OPT}_A + \text{OPT}_B$. To bound the cost of AVR, define $s_{AVR,A}(t)$ and $s_{AVR,B}(t)$ as the speed of AVR on type-A and type-B jobs respectively. Then the cost of AVR is at most

$$\begin{aligned}\int s_{AVR}(t)^\alpha dt &= \int \left(s_{AVR,A}(t) + s_{AVR,B}(t)\right)^\alpha dt \\ &\le \int 2^{\alpha-1}\left(s_{AVR,A}(t)^\alpha + s_{AVR,B}(t)^\alpha\right) dt \\ &= 2^{\alpha-1}\left(\text{AVR}_A + \text{AVR}_B\right) \\ &\le 2^{\alpha-1}\alpha^\alpha(\text{OPT}_A + \text{OPT}_B),\end{aligned}$$

which gives the desired ratio. □

Thus we reach our final theorem, which is an immediate consequence of Lemma 4, Lemma 5, and Lemma 6.

Theorem 7. *AVR is $2^{\alpha-1}\alpha^\alpha$-competitive.*

The following lemma from [2] was used in the proof of Lemma 4:

Lemma 8. *[2] Let $q, r, \delta \ge 0$ and $\alpha \ge 1$. Then $(q+\delta)^{\alpha-1}(q - \alpha r - (\alpha-1)\delta) - q^{\alpha-1}(q-\alpha r) \le 0$.*

Proof: The lemma is equivalent to showing that

$$(q-\alpha r)[(q+\delta)^{\alpha-1} - q^{\alpha-1}] - (q+\delta)^{\alpha-1}(\alpha-1)\delta \le 0$$

Since $[(q+\delta)^{\alpha-1} - q^{\alpha-1}] \ge 0$, it suffices to show that

$$q[(q+\delta)^{\alpha-1} - q^{\alpha-1}] - (q+\delta)^{\alpha-1}(\alpha-1)\delta \le 0$$

Let $\delta = zq$, which implies $z \ge 0$. The left hand side of the above becomes

$$q^\alpha[(1+z)^{\alpha-1} - 1] - q^\alpha[(1+z)^{\alpha-1}(\alpha-1)z]$$

Factoring out q^α and differentiating the rest with respect to z gives

$$\begin{aligned}&((\alpha-1)(1+z)^{\alpha-2}[1-(\alpha-1)z] + (1+z)^{\alpha-1}(-\alpha+1)) \\ &= ((\alpha-1)(1+z)^{\alpha-2}[1-(\alpha-1)z-(1+z)] \\ &= -\alpha(\alpha-1)z(1+z)^{\alpha-2}\end{aligned}$$

This is non-positive since $\alpha > 1$ and $z \geq 0$. Thus, the expression is maximized at $z = 0$, where it has value 0. This implies the result. $\square$

6 Conclusion

Even though AVR is not optimally competitive, one could imagine situations where a system designer might still adopt AVR because AVR is in some sense fair to each job. This is analogous to the reason that Processor Sharing (Round Robin) is adopted in some systems even though Processor Sharing is known not to have the best competitive ratio for the standard QoS measures.

Acknowledgments: We thank Don Coppersmith for helpful discussions.

References

1. Albers, S., Müller, F., Schmelzer, S.: Speed scaling on parallel processors. In: Proc. ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 289–298 (2007)
2. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. J. ACM 54(1) (2007)
3. Kwon, W.-C., Kim, T.: Optimal voltage allocation techniques for dynamically variable voltage processors. In: Proc. ACM-IEEE Design Automation Conf., pp. 125–130 (2003)
4. Li, M., Liu, B.J., Yao, F.F.: Min-energy voltage allocation for tree-structured tasks. Journal of Combinatorial Optimization 11(3), 305–319 (2006)
5. Li, M., Yao, A.C., Yao, F.F.: Discrete and continuous min-energy schedules for variable voltage processors. In: Proc. of the National Academy of Sciences USA, vol. 103, pp. 3983–3987 (2006)
6. Li, M., Yao, F.F.: An efficient algorithm for computing optimal discrete voltage schedules. SIAM J. on Computing 35, 658–671 (2005)
7. Pruhs, K.: Competitive online scheduling for server systems. SIGMETRICS Performance Evaluation Review 34(4), 52–58 (2007)
8. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Proc. IEEE Symp. Foundations of Computer Science, pp. 374–382 (1995)
9. Yun, H., Kim, J.: On energy-optimal voltage scheduling for fixed priority hard real-time systems. ACM Trans. on Embedded Computing Systems 2(3), 393–430 (2003)

Geometric Aspects of Online Packet Buffering: An Optimal Randomized Algorithm for Two Buffers*

Marcin Bienkowski[1] and Aleksander Mądry[1,2]

[1] Institute of Computer Science, University of Wroclaw, Poland
[2] CSAIL, MIT, Cambridge, MA, USA

Abstract. We study packet buffering, a basic problem occurring in network switches. We construct an optimal 16/13-competitive randomized online algorithm PB for the case of two input buffers of arbitrary sizes. Our proof is based on geometrical transformations which allow to identify the set of sequences incurring extremal competitive ratios. Later we may analyze the performance of PB on these sequences only.

Keywords: online algorithms, network problems, packet buffering.

1 Introduction

Nowadays, the performance of network backbones depends on the speed, with which network devices can switch data packets arriving at the input ports to the appropriate output ports. Since the traffic is usually bursty, the rate of arriving packets might be much higher than the rate with which the device can transmit them, and in result packets might get lost. This motivates the use of buffers attached to the input ports; these buffers can accumulate incoming packets and store them for later transmission. The capacity of buffers — although usually large — is limited, which makes buffer management techniques crucial for minimizing the data loss.

We study a basic problem in this context. We consider a network device which has m input ports and one output port. Each input port has an attached buffer which can store up to B packets; we assume that all packets are of unit size. Time is slotted into time steps. At any time step, any number of packets may arrive at the input ports and they are appended to the appropriate buffers. If a buffer cannot accommodate all the packets, the excess is lost. At any time step, the device can transmit one packet from one buffer; the buffer managing algorithm has to choose which buffer to send from. The scenario described above is typical for input-queued switches or routers. Additionally, this model, in which packets are equally important, is typical for current IP networks.

* Extended abstract. The full version of this paper is available online. Research supported by MNiSW grant number N206 001 31/0436, 2006–2008, MNiSW grant number N206 1723 33, 2007–2010, and by an Akamai Presidential Fellowship.

In our setting no information about the future is available to the algorithm. In particular, we make no probabilistic assumptions about the input. For analyzing the efficiency of our algorithms we use competitive analysis [13], and — on any input sequence — compare the throughput (the number of packets transmitted) of our algorithm and the optimal offline schedule. For any algorithm A and any sequence of packets arrival τ, we denote the throughput of A on τ by $T_{\mathrm{A}}(\tau)$. We call a deterministic algorithm $\textsc{Alg}$ c-competitive if for all sequences τ, it holds that $c \cdot T_{\mathrm{ALG}}(\tau) \geq T_{\mathrm{OPT}}(\tau)$, where $\textsc{Opt}$ denotes the optimal *offline* algorithm. Number c is called a *competitive ratio* of the algorithm $\textsc{Alg}$. If $\textsc{Alg}$ is a randomized algorithm, then in the definition above we replace $T_{\mathrm{ALG}}(\tau)$ with its expected value.

Previous Results. There are several results for the basic model described above. As the optimal competitive ratios can differ depending on the values B and m, the results address particular classes of these values.

First, we consider deterministic algorithms. The general upper bound holding for all values of B and m was given by Azar and Richter [2]. They proved that any deterministic *work-conserving* (i.e. serving a non-empty queue) algorithm is 2-competitive. They showed that for $B = 1$ no deterministic strategy can be better than $(2 - \frac{1}{m})$-competitive and presented a lower bound on the competitive ratio of $1.366 - \Theta(\frac{1}{m})$ which holds for any fixed B. Albers and Schmidt [1] improved that bound showing that for any fixed B and for large m the lower bound can be arbitrarily close to $e/(e-1) \approx 1.582$. They also showed an algorithm *Semi-Greedy* which is 1.944-competitive for $B \geq 2$ (see [12]). For $B = 2$ this algorithm is optimal and 1.857-competitive; for $B \to \infty$ the algorithm is 1.889-competitive. For $m = 2$ and $B \to \infty$, Schmidt [11] demonstrated a lower bound of $16/13 = 1.231$ and proved that a greedy algorithm achieves a ratio of $9/7 \approx 1.286$.

Randomized algorithms were also considered: Schmidt [11] showed a 3/2-competitive *Random Permutation* algorithm; the competitive ratio holds for any values of B and m. For the lower bound, define $h(n, k)$ as

$$h(n,k) = \frac{k+n}{k+1+\frac{(n-1)^{k+1}}{n^k}} \quad \text{and} \quad h(n) = \min_{k \in \mathbb{N}} h(n,k) \ . \tag{1}$$

A lower bound claimed by Albers and Schmidt in [1], whose proof can be found in [12], states that for any value of B the competitive ratio of any randomized algorithm is at least $h(m)$. This value is equal to $16/13$ for $m = 2$ and approaches 1.466 for $m \to \infty$.

Our Contribution and Paper Outline. In this paper, we present the first randomized online algorithm which — for the case of $m = 2$ buffers with arbitrary buffer size B — achieves the optimal competitive ratio of $16/13 \approx 1.231$.

Most papers on packet buffering concentrate around developing a smart algorithm and then comparing its behavior to the optimal one. Hence, the optimal algorithm is considered only in the analysis. We employ a different approach. In each step, we trace the set of possible states of the algorithm, which would so far serve the sequence in optimal manner. Then by keeping the state of our online algorithm as close to the center of this set as possible, we ensure that it performs well compared to the optimal solution. This technique bears some similarities to the well-known *work-function* technique, used for constructing many optimal or almost optimal online algorithms (for example for k-server [7] or page migration [3]).

Initially, we construct and analyze a deterministic algorithm PBF in a setting that allows PBF to have fractional number of packets in its buffers. We note that the lower bound on the competitive ratio of $h(2) = 16/13$ holds also for such model. The proof of PBF optimality consists of two parts. In Sect. 3.2, we show how the hardest sequences for PBF look like. We show that these input sequences (we call them *regular*) have very special structure, which we exploit to bound the competitiveness of PBF. We prove this by developing a geometric view on the packet buffering problem; such approach turns out to be surprisingly successful. Finally, using a potential function-like argument, in Sect. 3.3 we show that the performance ratio of PBF on any regular sequence is at most 16/13. We note that the idea of reductions of arbitrary sequences to the most difficult ones can be found in the previous papers, for example in [11].

As we mentioned above, PBF is a deterministic algorithm which is optimal in an extended, fractional model. We note that the most straightforward translation of this solution into a randomized non-fractional one does not work in our model. Instead, using techniques similar to randomized rounding [10], we construct a two-dimensional rounding technique, which yields an optimal algorithm PB.

Due to space limitations, the proofs of all technical lemmas appear in the full version of the paper.

Related Work. One of the most straightforward generalizations of the simple scenario considered is the model in which packets have values and the objective is to maximize the total value of packets sent. Although yet not commonly seen in practice, these *Differentiated Services* allow Internet Service Providers to assign different levels of *Quality of Service* to different data streams.

There are several results concerning the case of maintaining a single buffer, where packets have to be transmitted in FIFO order and where preemption (eviction of packets already in buffer) is allowed. Currently, the best deterministic *preemptive greedy* algorithm due to Englert and Westermann [4] achieves a competitive ratio of 1.732 and the best known lower bound for this problem, 1.419, is due to Kesselman et al [6]. There has also been work on a so-called *bounded-delay model*, in which no FIFO order is enforced but packets have deadlines (see e.g. [5]).

Azar and Richter [2] showed how to cope with multiple queues, presenting a general technique of transforming algorithms for single queue into multiple queue algorithms, losing factor 2 in the competitive ratio.

The packet buffering problems were also considered under some probabilistic assumptions on the input sequence (see e.g. [9]). However, there is an observed evidence that the nature of data traffic in networks is chaotic [8] and does not follow standard patterns like Poisson arrival model.

2 Preliminaries

First, let us formally define the input sequence. We transform a description of packets arrivals τ into a sequence of requests σ with more convenient form. For each time step in which there are no new incoming packets, we append a request IDLE to sequence σ. For a step in which there are new packets at input ports, say x_0 packets at buffer 0 and x_1 packets at buffer 1, we append $\text{ADD}(0)^{x_1}\,\text{ADD}(1)^{x_2}$ IDLE to σ.

By σ_t we understand the t-th element of σ and by $\sigma|_a^b$ the contiguous subsequence of σ starting at position (step) a and ending at b. Let $\sigma^t = \sigma|_1^t$. We say that the request σ_t is processed in *step* t. For any two sequences σ and σ' we denote their concatenation by $\sigma\sigma'$.

Semantics of Request Sequence. For any algorithm ALG, the *state* of its buffers at the end of a given step can be described by a pair $\boldsymbol{x}^{\text{ALG}} \in \{0, \ldots, B\} \times \{0, \ldots, B\}$, where the coordinates denote the number of packets in the respective buffers. For any request (sub)sequence σ, we use $\boldsymbol{x}^{\text{ALG}}(\sigma)$ to denote the state of ALG after it starts with empty buffers and serves the sequence σ. In particular, by $\boldsymbol{x}^{\text{ALG}}(\sigma^t)$ we mean the state of ALG after processing the first t steps of σ and $\boldsymbol{x}^{\text{ALG}}(\sigma^0) = (0, 0)$.

The semantics of the requests from σ sequence is straightforward. Fix any step t. For IDLE request, ALG may choose a non-empty buffer i and transmit one packet from it. Although the algorithm may also choose not to transmit a packet, any such algorithm can be transformed to a *work-conserving* one, which sends a packet whenever possible, and the competitiveness of the obtained algorithm is not worse. The way of choosing the buffer for transmission is called *pivoting rule*. Note that this rule is the only factor that determines the behavior of the algorithm.

If σ_t is an $\text{ADD}(i)$ request, a new packet is *added* to the buffer i. If the number of packets at the i-th buffer is already B, then the packet is immediately lost. Let $\ell^{\text{ALG}}(\sigma^t)$ be the number of packets that are actually added to the buffer in step t, $\ell^{\text{ALG}}(\sigma^t) \in \{0, 1\}$.

At the end of the input sequence, the algorithm empties all the buffers; we may assume that they are all transmitted in one batch.

Let $\mathcal{H} = [0, B] \times [0, B]$. We may view $\boldsymbol{x}^{\text{ALG}}$, the state of ALG, as a point from $\mathcal{H} \cap \mathbb{N}^2$. Then, the IDLE and ADD operations described above have obvious geometric interpretation. For any state $\boldsymbol{x}$, x_0 and x_1 denote the number of packets in the respective buffers, and $\|\boldsymbol{x}\| = x_0 + x_1$.

Fractional Model. It is now straightforward to generalize the above description to a *fractional model*, where the state of ALG can be any point from $\mathcal{H}$ (also the one with fractional coordinates). In particular, the algorithm serving IDLE request may choose to transmit fractional parts of packets from different buffers, with the only requirement that the total mass of the packets transmitted is at most 1. Note that the definition of $\ell^{\text{ALG}}(\sigma^t)$ can be extended to the fractional model in a straightforward manner.

Although we allow online algorithms to use fractional parts of the packets, to simplify our analysis we compare them to the optimal offline algorithm which still works in the standard model. We note that the lower bound of 16/13 holds in the fractional model, as well.

Competitiveness. For any sequence σ and any deterministic (not necessarily online) algorithm ALG, we define a function $\mathsf{loss}_{\text{ALG}}(\sigma)$ as the number of packets lost by the strategy ALG on σ, under the condition that ALG starts with empty buffers. For any algorithm ALG and two sequences σ and τ we define $\Delta_\sigma \mathsf{loss}_{\text{ALG}}(\tau) = \mathsf{loss}_{\text{ALG}}(\sigma\tau) - \mathsf{loss}_{\text{ALG}}(\sigma)$.

Obviously, the losses can occur only due to ADD requests which overflow some buffer. Therefore, if σ is the whole sequence, $\mathsf{loss}_{\text{ALG}}(\sigma) = \sum_{t:\sigma_t=\text{ADD}}(1 - \ell^{\text{ALG}}(\sigma^t))$. Let $S(\sigma)$ denote the total number of packets added in σ, i.e. the number of ADD requests. The throughput of ALG, denoted $T_{\text{ALG}}(\sigma)$, i.e. the number of packets transmitted by ALG, is then equal to $S(\sigma) - \mathsf{loss}_{\text{ALG}}(\sigma)$.

Consider a sequence σ. Let OPT be an optimal (offline) algorithm for the packet buffering problem, i.e. the one which minimizes the number of packets lost. We define the *performance ratio* of (an online) algorithm ALG on σ as

$$\mathcal{R}_{\text{ALG}}(\sigma) = \frac{T_{\text{OPT}}(\sigma)}{T_{\text{ALG}}(\sigma)} = \frac{S(\sigma) - \mathsf{loss}_{\text{OPT}}(\sigma)}{S(\sigma) - \mathsf{loss}_{\text{ALG}}(\sigma)} \ . \tag{2}$$

If Σ is any set of sequences, then $\mathcal{R}_{\text{ALG}}(\Sigma) = \sup_{\sigma\in\Sigma}\{R_{\text{ALG}}(\sigma)\}$. Let $\mathcal{Q}$ be the set of all possible sequences; then the competitive ratio can be defined as $\mathcal{R}_{\text{ALG}} = \mathcal{R}_{\text{ALG}}(\mathcal{Q})$. If $\mathcal{R}_{\text{ALG}} \le \alpha$, then we call ALG *$\alpha$-competitive.*

OPT State Space. As mentioned in the introduction, we would like to rely the behavior of our algorithm on tracing the state of some optimal off-line algorithm buffers in each step. Obviously, the complete knowledge about this state is not available to an on-line algorithm. Instead we will focus on extrapolating a set of possible OPT states which can be inferred from the already seen prefix of σ. In each step we trace a certain set $\mathcal{I}(\sigma^t)$ of possible states whose relation to an optimal solution is presented in Lemma 1.

We define set $\mathcal{I}$ inductively as $\mathcal{I}(\sigma^0) = \{(0,0)\}$, and

$$\mathcal{I}_{\text{pre}}(\sigma^t) = \begin{cases} (\mathcal{I}(\sigma^{t-1}) - (1,0)) \cup (\mathcal{I}(\sigma^{t-1}) - (0,1)) & \text{if } \sigma_t = \text{IDLE} \\ \mathcal{I}(\sigma^{t-1}) + (1,0) & \text{if } \sigma_t = \text{ADD}(0) \\ \mathcal{I}(\sigma^{t-1}) + (0,1) & \text{if } \sigma_t = \text{ADD}(1) \end{cases} , \tag{3}$$

$$\mathcal{I}(\sigma^t) = \begin{cases} \mathcal{I}_{\text{pre}}(\sigma^t) \cap \mathcal{H} & \text{if } \mathcal{I}_{\text{pre}}(\sigma^t) \cap \mathcal{H} \neq \emptyset \\ \mathcal{I}(\sigma^{t-1}) & \text{otherwise} \ . \end{cases} \tag{4}$$

An intuition behind the set $\mathcal{I}(\sigma^t)$ is that it contains states of all algorithms which try to greedily reduce their loss, i.e. postpone losing packets. A straightforward induction shows that the number of packets in each state from the set $\mathcal{I}$

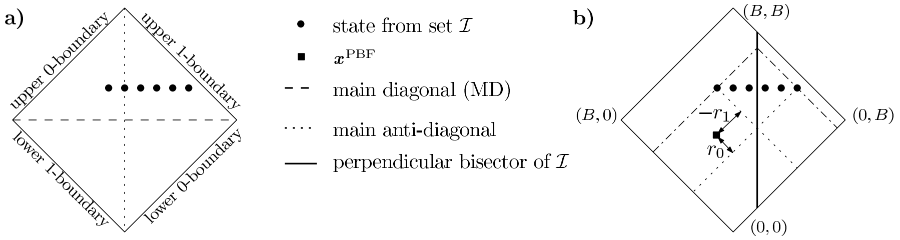


Fig. 1. Illustration of the set $\mathcal{I}$ and PBF parameters; $r_0 > 0$, $r_1 < 0$

is the same (we denote this number by $\|\mathcal{I}\|$) and $\mathcal{I}$ consists of non-fractional states contained in an anti-diagonal interval (see Fig. 1a). The following lemma shows a relation between the set $\mathcal{I}$ and optimal solutions.

Lemma 1. *There exists an algorithm A, such that $\boldsymbol{x}^A(\sigma^t) \in \mathcal{I}(\sigma^t)$ for any step t. Every algorithm with such property is optimal and loses a packet in step t on* Add *request if and only if $\mathcal{I}(\sigma^t) \neq \mathcal{I}_{\mathrm{pre}}(\sigma^t)$.*

The main implication of Lemma 1 is that we get a convenient description of the loss of an optimal algorithm. Namely, we can compute $\mathsf{loss}_{\mathrm{OPT}}(\sigma)$ by counting all Add requests, for which $\mathcal{I}(\sigma^t) \neq \mathcal{I}_{\mathrm{pre}}(\sigma^t)$.

3 Algorithm PBF

In this section, we present an algorithm PBF, which is optimal in the fractional model. Let

$$r_i(\sigma) = x_i^{\mathrm{PBF}}(\sigma) - \min_{z \in \mathcal{I}(\sigma)} z_i \qquad \text{for } i \in \{0,1\} \ . \tag{5}$$

Assume that the adversary decides to issue a maximum number of Add(i) requests without incurring a loss to Opt. Then r_i would be the number of packets lost by PBF. If $r_i \leq 0$, PBF cannot lose packets in this way (see r_1 in Fig. 1b). We note that r_i can be efficiently computed by an online algorithm (as $\mathcal{I}$ can be described by a few parameters). Let $\mathsf{bal}(\sigma) = r_0(\sigma) - r_1(\sigma)$ be called *balance*.

If PBF encounters Idle request in step t of σ, it computes a new shape of the set $\mathcal{I}(\sigma^t)$ first. Then it transmits a total mass of 1 packet, so that the resulting value of $|\mathsf{bal}(\sigma^t)|$ is as small as possible. This rule can be interpreted geometrically as choosing a new state $\boldsymbol{x}^{\mathrm{ALG}}(\sigma^t)$ as close to the perpendicular bisector (hence the abbreviation PB; F stands for fractional model) of the set $\mathcal{I}$ as possible.

3.1 Outline of the Proof

In the remaining part of this section, we prove the following theorem.

Theorem 1. *For any buffer size B,* PBF *on two buffers is* $16/13$*-competitive.*

Below we present the roadmap of the proof. As we mentioned in the introduction, the proof is divided into two parts. In the first one, we narrow down instances on which PBF has high competitive ratio. In the second part, we restrict our analysis only to these instances.

For any integers x,y, by a *block*, denoted $\mathcal{B}(x, y)$, we understand a subsequence $\textsc{Idle}^x\ \textsc{Add}(0)^y$. We also denote the sequence $\textsc{Add}(0)^B\ \textsc{Add}(1)^B$ by $\mathcal{A}$. By *main diagonal* (MD) we mean the diagonal of the square, which contains all fractional states $\boldsymbol{z}$ such that $\|\boldsymbol{z}\| = B$. We say that $\mathcal{I}$ is *above*, *at*, or *below* MD if $\|\mathcal{I}\|$ is, respectively, greater, equal, or less than B. We introduce the following classes of sequences.

Definition 1. *We denote the set of all sequences by $\mathcal{Q}$. We also define the following sets of sequences.*

– $\mathcal{Q}_N$*:* non-trivial. *$\sigma \in \mathcal{Q}_N$ if it ends with* Add *incurring loss to* PBF *and* $\boldsymbol{x}^{\mathrm{PBF}}(\sigma^t) \neq (0, 0)$ *for* $t > 0$*.*
– $\mathcal{Q}_P$*:* proper. *$\sigma \in \mathcal{Q}_P$ if it is non-trivial, starts with $\mathcal{A}$, and $\mathcal{I}(\sigma^t)$ is above MD for all $t \geq |\mathcal{A}|$.*
– $\mathcal{Q}_U$*:* uniform. *$\sigma \in \mathcal{Q}_U$ if σ is proper and after initial $\mathcal{A}$, consists only of* Add(0) *and* Idle *requests.*
– $\mathcal{Q}_R$*:* regular. *$\sigma \in \mathcal{Q}_R$ if σ is uniform and has a form $\mathcal{A}\,\mathcal{B}(x_1, y_1)\,\mathcal{B}(x_2, y_2) \ldots \mathcal{B}(x_n, y_n)$, where after each block $\mathcal{B}(x_i, y_i)$, $\boldsymbol{x}_0^{\mathrm{PBF}} = B$.*

The course of the proof is to show each of consecutive relations below.

$$\mathcal{R}_{\mathrm{PBF}} \leq \mathcal{R}_{\mathrm{PBF}}(\mathcal{Q}_N) \leq \mathcal{R}_{\mathrm{PBF}}(\mathcal{Q}_P) = \mathcal{R}_{\mathrm{GR}}(\mathcal{Q}_P) \leq \mathcal{R}_{\mathrm{GR}}(\mathcal{Q}_U) \leq \mathcal{R}_{\mathrm{GR}}(\mathcal{Q}_R) \quad (6)$$

$\mathcal{R}_{\mathrm{GR}}$ denotes the performance ratio of a natural Greedy algorithm (see [11]), which is defined formally later. We show these inequalities in Sect. 3.2. Then, in Sect. 3.3, we prove that $\mathcal{R}_{\mathrm{GR}}(\mathcal{Q}_R) \leq 16/13$. We note that by [1,12], the competitive ratio of any online algorithm Alg is at least $h(2) = \frac{16}{13}$, and therefore PBF is optimal and all the inequalities in (6) can be replaced by equalities.

3.2 Worst-Case Sequences

In this section, we consecutively prove all inequalities of (6) but the last one. In general, in order to show that $\mathcal{R}_{\mathrm{ALG}}(\mathcal{Q}_1) \leq \mathcal{R}_{\mathrm{ALG}}(\mathcal{Q}_2)$, we show that for any $\sigma \in \mathcal{Q}_1$, we can transform it to obtain a sequence $\widehat{\sigma} \in \mathcal{Q}_2$, such that the performance ratio of Alg does not decrease. Intuitively, $\widehat{\sigma}$ is more difficult for Alg than σ.

Changes in Balance. We start from several simple definitions and observations. We define $\mathsf{len}(\mathcal{I})$ as the length of the smallest interval containing set $\mathcal{I}$. This amount is equal to the number of $\mathcal{I}$ elements minus 1.

We conceptually divide each step into two parts. In the first one, the adversary issues a request and as a result the set $\mathcal{I}$ is changed. In case of an Add request,

$\boldsymbol{x}^{\mathrm{PBF}}$ is changed as well. In the second part, which is present only for Idle requests, PBF transmits some packets.

As a result of an Add(i) request, r_i may decrease by one. This can happen only if just before this request the set $\mathcal{I}$ *touches* (i.e. has non-empty intersection with) the upper i-boundary of $\mathcal{H}$, where *upper i-boundary* is $\mathcal{H} \cap \{(k_0, k_1) : k_i = B\}$. We call such an Add$(i)$ request a *hit*; such an Add reduces the value of $\mathsf{len}(\mathcal{I})$ by one.

If $\mathcal{I}$ is above MD, then in a step with Idle request, both r_i increase by 1, and therefore the balance remains unchanged. On the other hand, if $\mathcal{I}$ is at or below MD and it touches the lower i-boundary of $\mathcal{H}$, the corresponding r_i remains unchanged. If $\mathcal{I}$ touches only one boundary, the balance may therefore change by one. In total, upon an Idle request, in the first part of a step $\mathsf{len}(\mathcal{I})$ increases by 1 minus the number of lower boundaries $\mathcal{I}$ touches and in the second part PBF changes its state according to its pivoting rule. This observation leads to the following technical lemma.

Lemma 2. *For any sequence σ, there exists a non-trivial sequence $\widehat{\sigma}$, such that $\mathcal{R}_{\mathrm{PBF}}(\sigma) \leq \mathcal{R}_{\mathrm{PBF}}(\widehat{\sigma})$.*

Proper Sequences. Consider a non-trivial sequence σ. By the definition, at the end of σ set $\mathcal{I}$ touches an upper boundary (in the following informal description, we assume that it touches both boundaries). However, if we look from the adversary point of view, it is not obvious when this should happen for the first time. It is also not clear that keeping the set $\mathcal{I}$ below the main diagonal is not preferable — even if this constraints the growth of the set $\mathcal{I}$, the resulting constraints on PBF's behavior might be beneficial for the adversary.

We address the issues above by showing that proper sequences incur the worst performance ratio of PBF. Namely, we prove that for any sequence σ, there exists another sequence $\widehat{\sigma}$ with not smaller performance ratio, such that $\widehat{\sigma}$ starts with filling the buffers with packets and later it keeps the set $\mathcal{I}$ all the time above the diagonal.

We construct $\widehat{\sigma}$ on the basis of σ, so that after initial filling the buffers, $\widehat{\sigma}$ contains almost the same steps as σ. We can imagine that we have two instances of PBF running "in parallel" on σ and on $\widehat{\sigma}$. We show that it is possible to maintain an invariant that the set $\mathcal{I}$ and the point $\boldsymbol{x}^{\mathrm{PBF}}$ for $\widehat{\sigma}$ are equal to $\mathcal{I}$ and $\boldsymbol{x}^{\mathrm{PBF}}$ for σ translated by some vector. This invariant allows us to prove that the performance of PBF can only worsen by replacing σ with $\widehat{\sigma}$.

How do we create $\widehat{\sigma}$? If the request of σ does not change $\mathcal{I}$ and the spatial relation between $\mathcal{I}$ and $\boldsymbol{x}^{\mathrm{PBF}}$, we do not append anything to $\widehat{\sigma}$. Otherwise, if we have an Add request in σ, which is a hit, then this Add appended to $\widehat{\sigma}$ is a hit as well (as $\mathcal{I}(\widehat{\sigma})$ touches both boundaries). The only problem arises when we have an Idle request in σ occurring when $\mathcal{I}$ touches both lower boundaries, as in this case $\mathsf{len}(\mathcal{I})$ decreases. To simulate such change also on the instance $\widehat{\sigma}$, we introduce an additional request Lift. We justify this enhancement later in this section, by showing that the adversary does not need Lift to impose the worst competitive ratio.

Lift adds a half of a packet to both buffers of PBF. When it is issued in step t, it changes set $\mathcal{I}$ in a way opposite to the effect an Idle request would have. Namely, we extend the definition (3) by the following case.

$$\mathcal{I}_{\text{pre}}(\sigma^t) = (\mathcal{I}(\sigma^{t-1}) + (1,0)) \cup (\mathcal{I}(\sigma^{t-1}) + (0,1)) \quad \text{if } \sigma_t = \text{Lift} \tag{7}$$

Lemma 3. *For any non-trivial sequence σ, there exists a proper sequence $\widehat{\sigma}$ (possibly containing* Lift *requests) such that $\mathcal{R}_{\text{PBF}}(\sigma) \leq \mathcal{R}_{\text{PBF}}(\widehat{\sigma})$.*

Uniform Sequences. An important observation at this stage is that when we constrain our consideration only to proper sequences σ (possibly containing Lifts), PBF behaves like a Greedy algorithm. The set $\mathcal{I}$ always touches both upper boundaries of $\mathcal{H}$ (and in fact, the whole set $\mathcal{I}$ is therefore defined by a single variable $\|\mathcal{I}\|$). Thus, its perpendicular bisector always coincides with the main anti-diagonal of $\mathcal{H}$ and PBF just tries to move as close to it as possible, i.e. transmits packets to minimize the maximal level of packets in its buffers. This is exactly the pivoting rule of Greedy.

Now we want to further simplify the structure of the worst-case instances.

Lemma 4. *Let λ, λ' be two proper sequences possibly containing* Lift *requests, such that $x_0^{\text{GR}}(\lambda') \geq x_0^{\text{GR}}(\lambda)$, $x_0^{\text{GR}}(\lambda') \geq x_1^{\text{GR}}(\lambda')$, and $\|\boldsymbol{x}^{\text{GR}}(\lambda)\| \geq \|\boldsymbol{x}^{\text{GR}}(\lambda')\|$. Let τ be a sequence consisting only of* Add(0) *and* Idle *requests, such that $\lambda\tau$ and $\lambda'\tau$ are proper. Then $\Delta_{\lambda'}\mathsf{loss}_{\text{GR}}(\tau) - \Delta_{\lambda}\mathsf{loss}_{\text{GR}}(\tau) \geq \|\boldsymbol{x}^{\text{GR}}(\lambda')\| - \|\boldsymbol{x}^{\text{GR}}(\lambda)\|$.*

Lemma 5. *For any proper sequence σ, which may contain* Lift *requests, there exists a uniform sequence $\widehat{\sigma}$, such that $\mathcal{R}_{\text{GR}}(\sigma) \leq \mathcal{R}_{\text{GR}}(\widehat{\sigma})$.*

Proof. We show a series of transformations of σ, which eventually lead to a uniform sequence $\widehat{\sigma}$. Assume that σ contains some other request than Add(0) and Idle, and σ_t is the last such request. Then $\sigma|_{t+1}^{|\sigma|}$ consists only of Add(0) and Idle requests. If necessary, by swapping the labels of buffers in σ^t, we may ensure that $x_0^{\text{GR}}(\sigma^{t-1}) \geq x_1^{\text{GR}}(\sigma^{t-1})$. We call the resulting sequence $\widetilde{\sigma}$.

Let $\lambda = \sigma^t$, $\lambda' = \widetilde{\sigma}^{t-1}$ Add(0), and $\tau = \sigma|_{t+1}^{|\sigma|}$. Obviously, $\lambda\tau = \sigma$, $\lambda'\tau = \widetilde{\sigma}$, $S(\widetilde{\sigma}) = S(\sigma)$, As the set $\mathcal{I}$ changes in the same manner on Add and Lift requests, $\mathsf{loss}_{\text{OPT}}(\widetilde{\sigma}) = \mathsf{loss}_{\text{OPT}}(\sigma)$. We also have $\|\boldsymbol{x}^{\text{GR}}(\widetilde{\sigma}^{t-1})\| = \|\boldsymbol{x}^{\text{GR}}(\sigma^{t-1})\|$ and $\mathsf{loss}_{\text{GR}}(\widetilde{\sigma}^{t-1}) = \mathsf{loss}_{\text{GR}}(\sigma^{t-1})$. Since $x_0^{\text{GR}}(\widetilde{\sigma}^{t-1}) \geq x_1^{\text{GR}}(\widetilde{\sigma}^{t-1})$, it holds that $\Delta_{\widetilde{\sigma}^{t-1}}\mathsf{loss}_{\text{GR}}(\text{Add}(0)) \geq \Delta_{\sigma^{t-1}}\mathsf{loss}_{\text{GR}}(\sigma_t)$. Additionally, $\mathsf{loss}_{\text{GR}}(\lambda') - \mathsf{loss}_{\text{GR}}(\lambda) = \|\boldsymbol{x}^{\text{GR}}(\lambda)\| - \|\boldsymbol{x}^{\text{GR}}(\lambda')\| \geq 0$. Since λ, λ', and τ satisfy the requirements of Lemma 4, we get that $\mathsf{loss}_{\text{GR}}(\lambda'\tau) \geq \mathsf{loss}_{\text{GR}}(\lambda\tau)$ and thus $\mathcal{R}_{\text{GR}}(\widetilde{\sigma}) \geq \mathcal{R}_{\text{GR}}(\sigma)$.

After repeating the above operation at most $|\sigma|$ times, we end up with a desired sequence $\widehat{\sigma}$, which is actually equal to σ with all Add(1) and Lift requests replaced by Add(0). □

Regular Sequences. We cannot analyze uniform sequences easily, because Idle and Add(0) requests can be arbitrarily mixed. In order to alleviate this problem, we show how to change a uniform sequence into a regular one.

Lemma 6. *For any uniform sequence σ, there exists a regular sequence $\widehat{\sigma}$, such that $\mathcal{R}_{\mathrm{GR}}(\sigma) \leq \mathcal{R}_{\mathrm{GR}}(\widehat{\sigma})$.*

Proof. We denote a subsequence $\textsc{Add}(0)\,\textsc{Idle}$ by $\mathcal{F}$. We process σ from the beginning to the end, looking for $\mathcal{F}$. We show that if σ contains $\mathcal{F}$ and $x_0^{\mathrm{GR}} \leq B-1$ after processing $\textsc{Add}(0)$ request from $\mathcal{F}$, then such $\mathcal{F}$ can be removed from σ without decreasing the performance ratio. Moreover, after the removal the sequence remains proper. By applying this removal inductively to any occurrence of $\mathcal{F}$ in σ, we eventually get a regular $\widehat{\sigma}$.

Assume that $\sigma = \sigma_{\mathrm{prec}}\,\mathcal{F}\,\sigma_{\mathrm{succ}}$. We look separately at the change of the throughput of $\textsc{Opt}$ and $\textsc{Greedy}$. If $x_0^{\mathrm{GR}}(\sigma_{\mathrm{prec}}) \leq B-1$, then obviously the removal of $\mathcal{F}$ from σ does not change the state of $\textsc{Greedy}$ and decreases its throughput by 1. The change of T_{OPT} is twofold. First, it decreases by 1, since one $\textsc{Idle}$ was removed. Second, on σ_{succ} it may only increase because $\|\mathcal{I}(\sigma_{\mathrm{prec}})\| \geq \|\mathcal{I}(\sigma_{\mathrm{prec}}\,\mathcal{F})\|$. Moreover, in either case, $\|I\|$ can only increase after the removal, which implies that the new sequence is also proper. □

3.3 Performance Ratio on Regular Sequences

In this section we prove that the performance ratio of $\textsc{Greedy}$ on any regular sequence is at most 16/13. By the previous section, this will prove the competitiveness of PBF. We begin with an observation on the behavior of $\textsc{Greedy}$ on regular sequences.

Lemma 7. *Fix any regular sequence $\sigma = \mathcal{A}\mathcal{B}(x_1,y_1),\mathcal{B}(x_2,y_2)\ldots\mathcal{B}(x_n,y_n)$. Then $\boldsymbol{x}^{\mathrm{GR}}(\sigma) = (B, B-\gamma_i)$, where $\gamma_0 = 0$ and $\gamma_i \in [0,B)$ for all $i \leq n$. Moreover, we have the following recurrence relation for γ_i*

$$\gamma_i = \begin{cases} \gamma_{i-1} & \text{if } x_i \leq \gamma_{i-1} \\ \frac{\gamma_{i-1}+x_i}{2} & \text{if } x_i > \gamma_{i-1} \end{cases} .$$

Fix any $x \in [0,1)$. Let i be an integer such that $x \in [1-2^{-i}, 1-2^{-(i+1)})$. We define

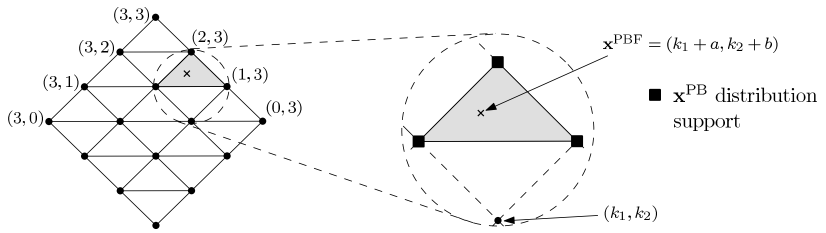
$$s(x) = i + \frac{x-(1-2^{-i})}{2^{-(i+1)}} . \tag{8}$$

Lemma 8. *$s(x)$ is continuous, piecewise linear, and monotonically increasing. Moreover, for any $0 \leq a < b \leq 1$, it holds that $s(\frac{a+b}{2}) - s(a) \leq b$ and $s(a) \geq \frac{16}{3} \cdot a - 2$.*

By a simple induction, we may prove the following bound.

Lemma 9. *For any regular sequence $\sigma = \mathcal{A}\mathcal{B}(x_1,y_1)\,\mathcal{B}(x_2,y_2)\,\ldots\,\mathcal{B}(x_n,y_n)$, and a corresponding sequence $\gamma_1,\gamma_2,\ldots,\gamma_n$, it holds that $\sum_{i=1}^{n} x_i/B \geq s(\gamma_n/B)$.*

Lemma 10. *For any regular sequence σ, $\mathcal{R}_{\mathrm{GR}}(\sigma) \leq 16/13$.*



(3, 3)
(3, 2)
(2, 3)
(3, 1)
(1, 3)
(3, 0)
(0, 3)
$\mathbf{x}^{\mathrm{PBF}} = (k_1 + a, k_2 + b)$
■ $\mathbf{x}^{\mathrm{PB}}$ distribution support
(k_1, k_2)

Fig. 2. Distribution of PB possible states

Proof. Let $\sigma = \mathcal{A}\,\mathcal{B}(x_1, y_1)\,\mathcal{B}(x_2, y_2)\,\ldots\,\mathcal{B}(x_n, y_n)$. This determines the sequence $\gamma_1, \gamma_2, \ldots, \gamma_n$. Since σ is non-trivial, both Opt and Greedy transmit packets during all $\sum_{i=1}^{n} x_i$ Idle requests of σ. Afterwards, $\boldsymbol{x}^{\mathrm{GR}} = (B - \gamma_n, B)$ and Opt has at most $2 \cdot B$ packets; these packets are transmitted at the end of σ. Therefore, the reciprocal of the performance ratio on σ is

$$\frac{(\sum_{i=1}^{n} x_i) + B + (B - \gamma_n)}{(\sum_{i=1}^{n} x_i) + B + B} = 1 - \frac{\gamma_n/B}{2 + \sum_{i=1}^{n} x_i/B} \geq 1 - \frac{\gamma_n/B}{2 + s(\gamma_n/B)} \geq \frac{13}{16} \ ,$$

where the last two inequalities follow from Lemmas 9 and 8, respectively. □

4 Randomization

In this section we describe a randomized algorithm PB, which works in a standard model and whose expected loss on a sequence σ is exactly the same as the loss of PBF on σ in the fractional model.

PB traces the current state of PBF. In case of Idle requests, PB tries to transmit a packet in such a way, that the expected number of packets in its buffers is equal to the actual number of packets in the buffers of PBF. We define PB algorithm implicitly, i.e. in each step we show what its probability distribution over possible states should be.

Definition 2. *Fix any (fractional) state of the buffers $\boldsymbol{x} = (k_0 + a, k_1 + b)$, where $k_0, k_1 \in \mathbb{N}$ and $a, b \in [0, 1]$. Define the following random variables:*

$$\triangledown(\boldsymbol{x}) = \begin{cases} (1,0) & \text{w.prob. } a \\ (0,1) & \text{w.prob. } b \\ (0,0) & \text{w.prob. } 1-a-b \end{cases} \qquad \triangle(\boldsymbol{x}) = \begin{cases} (1,0) & \text{w.prob. } 1-b \\ (0,1) & \text{w.prob. } 1-a \\ (1,1) & \text{w.prob. } a+b-1 \end{cases}$$

Let $\mu(\boldsymbol{x}) = (k_0, k_1) + \triangledown(\boldsymbol{x})$ if $a + b \leq 1$, and $\mu(\boldsymbol{x}) = (k_0, k_1) + \triangle(\boldsymbol{x})$ if $a + b \geq 1$.

We observe that $\mathbf{E}[\mu(\boldsymbol{x})] = \boldsymbol{x}$. For more intuitions about function μ, see Fig. 2. Each point from the square represents a legal state of an algorithm in the fractional model. Legal states of the algorithm in the standard model are represented by dots (points with integer coordinates). Then $\mu(\boldsymbol{x})$ is just a function which assigns probabilities to the vertices of a triangle enclosing $\boldsymbol{x}$.

Lemma 11. *It is possible to construct a randomized online algorithm* PB *for the standard model, such that* $\boldsymbol{x}^{\mathrm{PB}}(\sigma) = \mu(\boldsymbol{x}^{\mathrm{PBF}}(\sigma))$ *for any sequence* σ*. In effect,* $\mathsf{loss}_{\mathrm{PB}}(\sigma) = \mathsf{loss}_{\mathrm{PBF}}(\sigma)$*.*

Theorem 2. PB *is* $\frac{16}{13}$*-competitive for the packet buffering problem on two buffers.*

References

1. Albers, S., Schmidt, M.: On the performance of greedy algorithms in packet buffering. In: Proc. of the 36th ACM Symp. on Theory of Computing (STOC), pp. 35–44 (2004)
2. Azar, Y., Richter, Y.: Management of multi-queue switches in QoS networks. In: Proc. of the 35th ACM Symp. on Theory of Computing (STOC), pp. 82–89 (2003)
3. Chrobak, M., Larmore, L.L., Reingold, N., Westbrook, J.: Page migration algorithms using work functions. Journal of Algorithms 24(1), 406–415 (1993) In: Proc. of the 4th ISAAC, pp. 406–415 (1993)
4. Englert, M., Westermann, M.: Lower and upper bounds on FIFO buffer management in QoS switches. In: Proc. of the 14th European Symp. on Algorithms (ESA), pp. 352–363 (2006)
5. Englert, M., Westermann, M.: Considering suppressed packets improves buffer management in QoS switches. In: Proc. of the 18th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 209–218 (2007)
6. Kesselman, A., Mansour, Y., van Stee, R.: Improved competitive guarantees for QoS buffering. Algorithmica 43(1–2), 63–80 (2005) In: Proc. of the 11th ESA, pp. 361–372 (2003)
7. Koutsoupias, E., Papadimitriou, C.H.: On the k-server conjecture. Journal of the ACM 42(5), 971–983 (1995) In: Proc. of the 26th STOC, pp. 507–511 (1994)
8. Leland, W.E., Taqqu, M.S., Willinger, W., Wilson, D.V.: On the self-similar nature of ethernet traffic (extended version). IEEE/ACM Transactions on Networking 2(1), 1–15 (1994)
9. May, M., Bolot, J., Jean-Marie, A., Diot, C.: Simple performance models of differentiated services schemes for the internet. In: Proc. of the IEEE INFOCOM, pp. 1385–1394 (1999)
10. Raghavan, P., Tompson, C.D.: Randomized rounding: a technique for provably good algorithms and algorithmic proofs. Combinatorica 7(4), 365–374 (1987)
11. Schmidt, M.: Packet buffering: Randomization beats deterministic algorithms. In: Proc. of the 22nd Symp. on Theoretical Aspects of Computer Science (STACS), pp. 293–304 (2005)
12. Schmidt, M.: Online Packet Buffering. PhD thesis, Albert-Ludwigs-Universität Freiburg (2006)
13. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM 28(2), 202–208 (1985)

Maximizing the Minimum Load for Selfish Agents

Leah Epstein[1] and Rob van Stee[2,⋆]

[1] Department of Mathematics, University of Haifa, 31905 Haifa, Israel
lea@math.haifa.ac.il

[2] Max-Planck-Institut für Informatik, Saarbrücken, Germany
vanstee@mpi-inf.mpg.de

Abstract. We consider the problem of maximizing the minimum load for machines that are controlled by selfish agents, who are only interested in maximizing their own profit. Unlike the classical load balancing problem, this problem has not been considered for selfish agents until now.

For a constant number of machines, m, we show a monotone polynomial time approximation scheme (PTAS) with running time that is linear in the number of jobs. It uses a new technique for reducing the number of jobs while remaining close to the optimal solution. We also present an FPTAS for the classical problem, i.e., where no selfish agents are involved (the previous best result for this case was a PTAS) and use this to give a monotone FPTAS.

Additionally, we give a monotone approximation algorithm with approximation ratio $\min(m, (2+\varepsilon)s_1/s_m)$ where $\varepsilon > 0$ can be chosen arbitrarily small and s_i is the (real) speed of machine i. Finally we give improved results for two machines.

1 Introduction

In this paper, we are concerned with a fair allocation of jobs to parallel related machines, in the sense that each machine should contribute a 'reasonable amount' (compared to the other machines) to the processing of the jobs. Specifically, we are interested in maximizing the minimum load which is assigned to any machine. This value is also known as the *cover*, as it is the amount to which all machines are "covered". This problem has been studied in the past on identical [11,10,19] as well as related machines [7] and also in the online setting where jobs arrive one by one and need to be assigned without information about future jobs [6]. It is also closely related to the max-min fairness problem [9,15,8], where we want to distribute indivisible goods to players so as to maximize the minimum valuation.

In our case, the players (machines) have negative valuations for the jobs, since there is a cost incurred in running the jobs. So our goal becomes maximizing the

⋆ Research supported by the German Research Foundation (DFG). Work performed while the author was at the University of Karlsruhe, Germany.

minimum loss, i.e., making sure that the cost of processing is not distributed too unfairly. Moreover, the machines are controlled by selfish agents that only care about maximizing their individual profit (or minimizing their individual loss). The speeds of the machines are unknown to us, but before we allocate the jobs, the agents will give us bids which may or may not correspond to the real speeds of their machines.

Our goal in this paper will be to design *truthful mechanisms*, i.e., design games in such a way that truth telling is a dominant strategy for the agents: it maximizes the profit for each agent individually. This is done by introducing *side payments* for the agents. In a way, we reward them (at some cost to us) for telling us the truth. The role of the mechanism is to collect the claimed private data (bids), and based on these bids to provide a solution that optimizes our desired objective, and hand out payments to the agents. The agents know the mechanism and are computationally unbounded in maximizing their utility.

The seminal paper of Archer and Tardos [3] considered the general problem of one-parameter agents. The class of one-parameter agents contain problems where any agent i has a private value t_i and his valuation function has the form $w_i \cdot t_i$, where w_i is the work assigned to agent i. Each agent makes a bid depending on its private value and the mechanism, and each agent wants to maximize its own profit. The paper [3] shows that in order to achieve a truthful mechanism for such problems, it is necessary and sufficient to design a *monotone* approximation algorithm. An algorithm is monotone if for every agent, the amount of work assigned to it does not increase if its bid increases. More formally, an algorithm is monotone if given two vectors of length m, b, b' which represent a set of m bids, which differ only in one component i, i.e., $b_i > b'_i$, and for $j \neq i$, $b_j = b'_j$, then the total size of the jobs (the work) that machine i gets from the algorithm if the bid vector is b is never higher than if the bid vector is b'.

Using this result, monotone (and therefore truthful) approximation algorithms were designed for several classical problems, like scheduling on related machines to minimize the makespan [3,5,1,17], shortest path [4,13], set cover and facility location games [12], and combinatorial auctions [18,2].

Formal definition. Denote the number of jobs by n, and the size of job j by p_j $(j = 1, \ldots, n)$. Denote the number of machines by m, and the speed of machine i by s_i $(i = 1, \ldots, m)$. As stated, each machine belongs to a selfish user. The private value (t_i) of user i is equal to $1/s_i$, that is, the cost of doing one unit of work. The load on machine i, L_i, is the total size of the jobs assigned to machine i divided by s_i. The profit of user i is $P_i - L_i$, where P_i is the payment to user i by the payment scheme defined by Archer and Tardos [3]. Let b_{-i} denote the vector of bids, not including agent i. We write b (the total bid vector) also as (b_{-i}, b_i). Then the payment function for user i is defined as

$$P_i(b_{-i}, b_i) = h_i(b_{-i}) + b_i w_i(b_{-i}, b_i) - \int_0^{b_i} w_i(b_{-i}, u)du,$$

where $w_i(b_{-i}, b_i)$ is the work (total size of jobs) allocated to user i given the bid vector b and the h_i are arbitrary functions.

Our goal is to maximize $\min L_i$. This problem is NP-complete in the strong sense [14] even on identical machines. In order to analyze our approximation algorithms we use the approximation ratio. For an algorithm $\mathcal{A}$, we denote its cost by $\mathcal{A}$ as well. An optimal algorithm is denoted by OPT. The approximation ratio of $\mathcal{A}$ is the infimum $\mathcal{R}$ such that for any input, $\mathcal{A} \leq \mathcal{R} \cdot \text{OPT}$. If the approximation ratio of an offline algorithm is at most ρ we say that it is a ρ-approximation.

Previous results (non-selfish machines). For identical machines, Woeginger [19] designed a polynomial time approximation scheme (PTAS). He also showed that the greedy algorithm is m-competitive. This is optimal for deterministic online algorithms. Azar and Epstein [6] presented a randomized $O(\sqrt{m}\log m)$-competitive online algorithm and an almost matching lower bound of $O(\sqrt{m})$.

In [7], a PTAS was designed for related machines. For the semi-online case in which jobs arrive in non-increasing order, [6] gave an m-competitive algorithm called Biased-Greedy and showed that no algorithm could do better. For the case where jobs arrive in non-increasing order and also the optimal value is known in advance, [6] gave a 2-competitive algorithm Next Cover.

For unrelated machines, Bezáková and Dani [9] give several algorithms. One gives a solution value which is at most $\text{OPT} - p_{\max}$ less than the optimum, where $p_{\max}$ is the largest job size (on any machine). Note that this result may be close to zero. Two other algorithms have performance guarantee $n - m + 1$. Golovin [15] gave an algorithm which guarantees that at least a $(1-1/k)$ fraction of the machines receive jobs of total value at least OPT/k, for any integer k. In the same paper, he also gave an $O(\sqrt{n})$-approximation for the case of restricted assignment (each job can only be assigned to a subset of the machines, and has the same size on each allowed machine) where all job sizes are either 1 or some value X.

For the case of restricted assignment (without further restrictions on job sizes), Bansal and Sviridenko [8] provided an $O(\log\log m/\log\log\log m)$-approximation. Bezáková and Dani [9] showed that no polynomial-time algorithm can have a performance guarantee better than 2 unless P=NP. In particular, no PTAS is possible.

Our results. We present a *monotone* strongly polynomial time approximation scheme (PTAS) for a constant number of related machines. Its running time is linear in the number of jobs, n. To the best of our knowledge, the technique it uses for reducing the number of jobs while remaining close to the optimal solution is new. We then give a new result for non-selfish related machines (the classical problem) by presenting an FPTAS for it. We use this to give a monotone FPTAS with running time polynomial in n and ε and the logarithm of sum of job sizes.

Additionally, we present a monotone approximation algorithm which is based on Next Cover and achieves an approximation ratio of $\min(m, (2+\varepsilon)s_1/s_m)$. This algorithm is strongly polynomial-time for an arbitrary number of machines, and it is the first such algorithm that is monotone. It seems difficult to design a monotone approximation algorithm with a constant approximation ratio for an arbitrary number of machines. Finally, we study two monotone algorithms for two machines, and analyze their approximation ratios as a function of the speed ratio between them. These algorithms are very simple and in many cases faster than applying the PTAS or FPTAS on two machines.

Sorting. Throughout the paper, we assume that the jobs are sorted in order of non-increasing size ($p_1 \geq p_2 \geq \ldots \geq p_n$), except in Section 3, and the machines are sorted in a fixed order of non-decreasing bids (i.e. non-increasing speeds, assuming the machine agents are truthful, $s_1 \geq s_2 \geq \ldots \geq s_m$).

2 Unsuccessful Directions

To give a flavor of the problem, we begin by describing some algorithms that seem reasonable, but that have a very high approximation ratio and/or are not monotone. First we note that it is known that any deterministic algorithm which can be seen as a purely online algorithm (i.e., does not have any a-priori information on jobs, and cannot perform sorting), cannot have finite approximation ratio [6]. It follows from the same paper that algorithms which sort the jobs by non-increasing size but are otherwise online (i.e. after sorting, no information about later jobs is used apart from the fact that they will not be larger than the current job) cannot be better than m-competitive. Nor can online algorithms that only know the value of the optimal cover do better.

The following natural algorithms are either not monotone, or have an infinite approximation ratio.

– Least Processing Time (LPT). This algorithm does not even have finite approximation ratio. Given two machines of speeds 1 and 4, and two jobs of size 1, it will assign both jobs to the machine of speed 4. But then the cover is 0. Moreover, it is known that LPT is not monotone but an adaptation called LPT* is monotone [17]. However, the adaptation acts the same on this input and thus it cannot be used for the current problem.
– A greedy algorithm which sorts the jobs first, and assigns every job, in turn, to the least loaded machine, ignoring the effect of the new job on the schedule, has an infinite approximation ratio. This can be seen from the following example. There are two machines, of speeds 1 and M (for a large positive M) and two jobs of sizes M and 1. If the larger job is assigned to the slower machine and the smaller on to the faster machine, we get an approximation ratio of M.
– Biased-Greedy is a a special case of the previous algorithm which prefers faster machines in case of ties. As stated above, it cannot be better than m-competitive. Moreover, is not monotone. Consider an example with three machines of speeds $10, 9, 9$ and four jobs of sizes $3, 3, 2, 2$. One of the two slowest machines receives two jobs of size 2. If the speed of this machine increases to 10, it would only get one job of size 3.

– LPT-Cover. This is a natural variant of LPT for the covering problem. It orders the jobs by size as before, but now, assign each job to that machine where it improves the cover the most. In particular, as long as there are empty machines, assign jobs there. This algorithm assigns job arbitrarily to empty machines, therefore it is no better than the previous greedy algorithm. If it is defined to give preference to faster machines, then it acts as Biased-Greedy on the input stated above.

Approach-Average. To conclude this section, we state another direction that was not studied before and initially seems promising, but fails. Calculate $A = \sum_j p_j / \sum_i s_i$. Assign jobs (ordered by size) to a machine which after assignment of the job has load closest to A (which is an upper bound on OPT). This algorithm also has unbounded approximation ratio. Consider the following input. There are m machines, one of them has speed 1, the others have speed $1/m$. There are m jobs of size 1. It can be seen that a cover of (only) $1/m$ can be achieved. But A is slightly more than $m/2$, and the first $m/2$ jobs of size 1 will be assigned to the fast machine, which results in a load of zero on at least one slow machine.

3 PTAS for Constant m

To derive a PTAS, we would as usual like to reduce the number of options to be considered by rounding job sizes. However, a main problem here is that the rounding should be independent of the bids, since otherwise when one agent lies we get a different rounding and possibly a completely different set of jobs, making it unlikely to give a monotone assignment and certainly very hard to prove monotonicity. This was the main technical problem that we had to address in developing our PTAS. Due to space constraints, it is the only issue that we address here.

We construct an input for which we can find an optimal job assignment which is the smallest assignment lexicographically, and thus monotone. We build it in a way that the value of an optimal assignment for the adapted input is within a multiplicative factor of $1 - 3\varepsilon$ from the value of an optimal assignment for the original input. This is done by reducing the number of jobs of size no larger than OPT to a constant number (dependent on m and ε), using a method which is oblivious of the machine speeds.

Let $\Delta = 2m^2/\varepsilon^2 + m$. If the input consists of at most Δ jobs, then we are done. Otherwise, we keep the Δ largest such jobs as they are. This set is denoted by J_L. Let J_S be the rest of the jobs.

Let A be the total size of the jobs in J_S. Let a be the size of the largest job in J_S. If $A \leq 3a\Delta$, we combine jobs greedily to create mega-jobs of size in the interval $[a, 3a]$. One mega-job is created by combining jobs until the total size reaches at least a, this size does not exceed $2 \cdot a$. If we are left with a remainder of size less than a, it is combined into a previously created job. The resulting number of mega-jobs created from J_S is at most 3Δ.

Otherwise, we apply a "List Scheduling" algorithm with as input the jobs in J_S and Δ identical machines. These machines are only used to combine the jobs of J_s into Δ mega-jobs and should not be confused with the actual (m) machines in the input.

List Scheduling (LS) works by assigning the jobs one by one (in some order) to machines, each job is assigned to the machine with minimum load (at the moment the job is assigned). LS thus creates Δ sets of jobs and the maximum difference in size between two sets is at most a [16]. The jobs in each set are now combined into a mega-job. Thus we get Δ mega-jobs with sizes in the interval $[\frac{A}{\Delta} - a, \frac{A}{\Delta} + a]$. Since $\frac{A}{\Delta} \geq 3a$, we get that the ratio between the size of two such mega-jobs is no larger than 2.

In all three cases we get a constant number of jobs and mega-jobs.

Running time. We reduce the number of jobs to a constant. Note that we are only interested in identifying the Δ largest jobs. After this we merge all remaining jobs using a method based on their total size. These things can be done in time linear in n. Finally, once we have a constant number of jobs, we only need constant time to find an optimal solution for them. Thus our algorithm has running time which is linear in the number of jobs n.

4 FPTAS for Constant m

In the full version of this paper, we present a monotone fully polynomial-time approximation scheme for constant m. This scheme uses as a subroutine a non-monotone FPTAS which is described in Section 4.1. We explain how this subroutine can be used to create a monotone FPTAS in the full paper.

In the current problem, it can happen that some jobs are superfluous: if they are removed, the optimal cover that may be reached remains unchanged. Even though these jobs are superfluous, we need to take special care of these jobs to make sure that our FPTAS is monotone. In particular, we need to make sure that these superfluous jobs are always assigned in the same way, and not to very slow machines. We therefore needed to modify the FPTAS mechanism from [1] because we cannot simply use any "black box" algorithm as was possible in [1].

4.1 An FPTAS Which is Not Monotone

Choose ε so that $1/\varepsilon$ is an integer. We may assume that $n \geq m$, otherwise OPT $= 0$ and we assign all jobs to machine 1. In the proof of Lemma 2 we show that this assignment is monotone.

We give an algorithm which finds the optimal cover up to a factor of $1 - 2\varepsilon$. We can again use an algorithm which is an m-approximation [6], therefore we can assume we can find OPT within a factor of m. We scale the problem instance such that that algorithm returns a cover of size 1. Then we know that OPT $\in [1, m]$. We are now going to look for the highest value of the form $j \cdot \varepsilon$ ($j = 1/\varepsilon, 1/\varepsilon + 1, \ldots, m/\varepsilon$) such that we can find an assignment which is of value at least $(1-\varepsilon)j\varepsilon$. That is, we partition the interval $[1, m]$ into many small intervals of length ε. We want to find out in which of these intervals OPT is, and find an assignment which is at most one interval below it.

Given a value for j, we scale the input up by a factor of $\frac{n}{j\varepsilon^2} \geq \frac{m}{m\varepsilon} \geq 1$. Now the target value (the cover that we want to reach) for a given value of j is not $j\varepsilon$ but $S = n/\varepsilon$. Sort the machines by speed. For machines with the same speed, sort them according to some fixed external ordering. For job k and machine i, let $\ell_i^k = \lceil p_k/s_i \rceil$ $(k = 1, \ldots, n; i = 1, \ldots, m)$.

We use dynamic programming based on the numbers ℓ_i^k. A *load vector* of a given job assignment is an m-dimensional vector of loads induced by the assignment. Let $T(k, a)$ be a value between 0 and m for $k = 0, \ldots, n$ and an (integer!) load vector a. $T(k, a)$ is the maximum number such that job k is assigned to machine $T(k, a)$ and a load vector of a (or better) can be achieved with the jobs $1, \ldots, k$. If the vector a cannot be achieved, $T(k, a) = 0$. If a (or better) can be achieved, $T(k, a)$ is a number between 1 and m.

We initialize $T(0, 0) = m$, representing that a cover of 0 can be achieved without any jobs (this is needed for the dynamic program), and $T(0, a) = 0$ for any $a > 0$. For a load vector $a = (a_1, \ldots, a_m)$, $T(k, a)$ is computed from $T(k-1, a)$ by examining m values (each for a possible assignment of job k):

$$T(k,a) = \max\left(0, \left\{i \in \{1,\ldots,m\} \,\middle|\, a_i - \ell_i^k \geq 0 \wedge T(k-1,(a_{-i}, a_i - l_i^k)) > 0\right\}\right)$$

The notation (a_{-i}, x) represents the load vector in which the ith element of a has been replaced by x and all other elements are unchanged. Each value $T(k, a)$ is set only once, i.e., if it is nonzero it is not changed anymore. When a value $T(k, a)$ is set to a nonzero value x, we also set $T(k, (a_{-i}, a_i - y)) = x$ for every $y = 1, \ldots, l_i^k - 1$ such that $T(j, (a_{-i}, a_i - y)) = 0$. This represents the fact that although a load vector of precisely a cannot be achieved with this assignment, a load vector that dominates a (is at least as large in every element) can be achieved by assigning job k to machine $T(k, a)$.

The size of the table T for one value of k is $(S+1)^m$. The n tables are computed in total time $nmS(S + 1)^m = O(m(n/\varepsilon)^{m+2})$. (The factor S is from updating the table after setting some $T(k, a)$ to a nonzero value.) As soon as we find a value $k \leq n$ such that $T(k, S, \ldots, S) > 0$, we can determine the assignment for the first k jobs by going back through the tuples. Each time, we can subtract the last job from the machine where it was assigned according to the value of the tuple to find the previous load vector. If some element of the load vector drops below 0 due to this subtraction, we replace it by 0. If $k < n$, the last $n - k$ jobs are assigned to machine 1 (the fastest machine).

If $T(n, S, \ldots, S) = 0$ after running the dynamic program, the target value cannot be achieved. In this case we adjust our choice of j (using binary search) and try again. In this way, we eventually find the highest value of j such that all machines can be covered to $j\varepsilon$ using jobs that are rounded.

Note that the loss by rounding is at most n per machine (in the final scaled instance): if we replace the rounded job sizes by the actual job sizes as they were after the second scaling, then the loss is at most 1 per job, and there are at most n jobs on any machine. So the actual cover given by the assignment found by the dynamic program is at least $S - n$. Since the target value $S = n/\varepsilon$, we lose a factor of $1 - \varepsilon$ with regard to S. After scaling back (dividing by $n/(j\varepsilon^2)$

again) we have that the actual cover found is at least $(1 - \varepsilon)j\varepsilon$. On the other hand, due to the binary search a cover of $(j + 1)\varepsilon$ cannot be reached (not even with job sizes that are rounded up). This implies that our cover is at least $(1 - \varepsilon)(\text{OPT} - \varepsilon) \geq (1 - 2\varepsilon)\text{OPT}$ since $\text{OPT} \geq 1$.

5 Approximation Algorithm SNC for Arbitrary Values of m

In this section, we present an efficient approximation algorithm for an arbitrary number of machines m. Our algorithm uses Next Cover [6] as a subroutine. This semi-online algorithm is defined in Figure 1. Azar and Epstein [6] showed that if the optimal cover is known, Next Cover (NC) gives a 2-approximation. That is, for the guess $G = \text{OPT}/2$ it will succeed. NC also has the following property,

```
Input: guess value $G$, $m$ machines in a fixed order of non-increasing speeds, $n$ jobs in order of non-increasing sizes.
For every machine in the fixed order, starting from machine 1, allocate jobs to the machine according to the sorted order of jobs until the load is at least $G$.
If no jobs are left and not all machines reached a load level of $G$, report failure. If all machines reached a load of $G$, allocate remaining jobs (if any) to machine $m$, and report success.
```

Fig. 1. Algorithm Next Cover (NC)

which we will use later.

Lemma 1. *Suppose NC succeeds with guess G but fails with guess $G + \varepsilon$, where $\varepsilon \leq \frac{1}{3}G$. Then in the assignment for guess G, the work on machine m is less than $mw + \varepsilon$, where $w \geq G$ is the minimum work on any machine.*

Our algorithm Sorted Next Cover (SNC) works as follows. A first step is to derive a lower bound and an upper bound on the largest value which can be achieved for the input and m identical machines. To find these bounds, we can apply LPT (Longest processing Time), which assigns the sorted (in non-increasing order) list of jobs to identical machines one by one. Each job is assigned to the machine where the load after this assignment is minimal. It was shown in [11,10] that the approximation ratio of LPT is $\frac{4m-2}{3m-1} < \frac{4}{3}$. Thus we define A to be the value of the output assignment of LPT. We also define $L = \frac{A}{2}$ and $U = \frac{4}{3}A$. We have that A and U are clear lower an upper bounds on the optimal cover on identical machines. Since NC always succeeds to achieve half of an optimal cover, it will succeed with the value $G = L$. Since a cover of U is impossible, the algorithm cannot succeed with the value $G = U$. Throughout the algorithm, the values L and U are such that L is a value on which NC succeeds whereas U is a failure value. We perform a geometrical binary search. It is possible to prove using induction that if NC succeeds to cover all machines with a guess value G, then it succeeds to cover all machines using a smaller guess value $G' < G$. The induction is on the number of machines and the claim is that in order to achieve a cover of G' on the first i machines, NC uses the same subset or a smaller subset used to achieve G.

The algorithm has a parameter $\varepsilon \in (0, 1/2)$ that we can set arbitrarily. See Figure 2. Since the ratio between U and L is initially constant, it can be seen that the algorithm completes in at most $O(\frac{1}{\log(1+\varepsilon/2)})$ steps. The overall running time is $O(n(\log n + 1/\log(1 + \varepsilon/2))$ due to the sorting. Note that Steps 2 and 6 are only executed once.

```
Input: parameter $\varepsilon \in (0, 1/2)$, sorted set of jobs ($p_1 \geq \ldots \geq p_n$), sorted machine
bids ($b_1 \leq \ldots \leq b_m$).

 1. If there are less than $m$ jobs, assign them to machine 1 (the machine of speed
    $s_1$), output 0 and halt.
 2. Scale the jobs so that $\sum_{i=1}^{n} p_j = 1$. Run LPT on identical machines and denote
    the value of the output by $A$. Set $L = \frac{A}{2}$ and $U = \frac{4}{3}A$.
 3. Apply Next Cover on identical machines with the guess $G = \sqrt{U \cdot L}$.
 4. If Next Cover reports success, set $L = G$, else set $U = G$.
 5. If $U - L > \frac{\varepsilon}{2}L$, go to step 3, else continue with step 6.
 6. Apply Next Cover on identical machines with the value $L$. Next Cover partitions
    the jobs in $m$ subsets, each of total size of jobs at least $L$. Sort the subsets in
    non-increasing order and allocate them to the machines in non-increasing order
    of speed according to the bids.
```

Fig. 2. Algorithm Sorted Next Cover (SNC)

Lemma 2. *SNC is monotone.*

Proof. The subsets constructed in step 3 and 6 do not depend on the speeds of the machines. If a machine claims it is faster than it really is, the only effect is that it may get a larger subset. Similar if it is slower.

If the algorithm halts in step 1, then we again have a situation that jobs are partitioned into sets, and the sets are assigned in a sorted way. This is actually the output that steps 2–6 would produce if SNC was run with a guess value 0.

Theorem 1. *For any $0 < \varepsilon < 1$, SNC maintains an approximation ratio of* $\min(m, (2 + \varepsilon)s_1/s_m)$.

Proof. We start with the second term in the minimum. The load that SNC has on machine i is at least L/s_i, and Next Cover cannot find a cover above $U \leq (1+\varepsilon/2)L$ on identical machines. So the optimal cover on identical machines of speed 1 is at most $2(1+\varepsilon/2)L = (2+\varepsilon)L$. Thus the optimal cover on machines of speed s_m is at most $(2+\varepsilon)L/s_m$, and the optimal cover on the actual machines can only be lower since s_m is the smallest speed. We thus find a ratio of at most $((2 + \varepsilon)L/s_m)/(L/s_i) = (2 + \varepsilon)s_i/s_m \leq (2 + \varepsilon)s_1/s_m$.

We prove the upper bound of m using induction.

Base case: On one machine, SNC has an approximation ratio of 1.

Induction hypothesis: On $m-1$ machines, SNC has an approximation ratio of at most $m-1$.

Induction step: Recall that the jobs are scaled so that their total size is 1. Suppose each machine j has work at least $1/(jm)$ $(j=1,\ldots,m)$. Then the load on machine j is at least $1/(jms_j)$. However, the optimal cover is at most $1/(s_1+s_2+...+s_m) \le 1/(js_j+(m-j)s_m) \le 1/(js_j)$. Thus SNC maintains an approximation ratio of at most m in this case.

Suppose there exists a machine i in the assignment of SNC with work less than $1/(im)$. Consider the earliest (fastest) such machine i. Due to the resorting we have that the work on machines $i,\ldots,m$ is less than $1/(im)$. So the total work there is less than $(m-i+1)/(im)$. The work on the first $i-1$ machines is then at least $1-(m-i+1)/(im) = (im-m+i-1)/(im) = (i-1)(m+1)/(im)$ and the work on machine 1 is at least $(m+1)/(im)$. This is more than $m+1$ times the work on machine i.

We show that in this case there must exist a very large job, which is assigned to a machine by itself. Let L' and U' be the final values of L and U in the algorithm. Let w be the minimum work assigned to any machine for the guess value L'. Since SNC gives machine i work less than $1/(im)$, we have $w < 1/(im)$. We have $U'-L' \le \frac{\varepsilon}{2}L'$. SNC succeeds with L' and fails with U' and thus, since $\varepsilon \le \frac{1}{2}$ and by Lemma 1, machine m receives at most $mw + \frac{\varepsilon}{2}L' \le mw + \frac{1}{4}L' \le (m+\frac{1}{4})w \le (m+\frac{1}{4})/(im)$ running NC with the guess value L'. Moreover, NC stops loading any other machine in step 6 as soon as it covers L'.

We conclude that the only way that any machine can get work more than $(m+1)L'$ is if it gets a single large job. This means that in particular the first (largest) job has size $p_1 > (m+1)w \ge 3w \ge 3L'$. SNC assigns this job to its first machine, and the remaining work on the other machines.

To complete the induction step, compare the execution of SNC to the execution of SNC with as input the $m-1$ slowest machines and the $n-1$ smallest jobs. Denote the first SNC by SNC$_m$ and the second by SNC$_{m-1}$. We first show that SNC$_{m-1}$ fails on U'. Since $U' \le (1+\frac{\varepsilon}{2})w < 2w$, then SNC$_m$ assigns only p_1 to machine 1, and thus SNC$_{m-1}$ executes exactly the same on the other machines. Since machine 1 is covered, SNC$_m$ fails on some later machine, and then this also happens to SNC$_{m-1}$. Therefore, SNC$_{m-1}$ cannot succeed with U' or any larger value. A similar reasoning shows that SNC$_{m-1}$ succeeds with any guess that is at most L'. Finally, L' is at least the starting guess $A/2$. So $p_1 > 3L' \ge \frac{3}{2}A$ implies that LPT also puts only the first job on the first machine, since its approximation ratio is better than 4/3. Therefore, LPT gives the same guess value A for the original input on m machines as it would for the $n-1$ smallest jobs on $m-1$ machines. This means that SNC$_m$ and SNC$_{m-1}$ maintain the same values U and L throughout the execution, and then we can apply the induction hypothesis.

In the full paper, we show that the simple algorithm Round Robin has an approximation guarantee of m, so this algorithm can also be used in case the speed ratio is large. It should be noted that if we find an algorithm with a better guarantee than m, we cannot simply run both it and SNC and take the best assignment to get a better overall guarantee. The reason that this does not work is that this approach does not need to be monotone, even if this hypothetical new algorithm is monotone: we do not know what happens at the point where we switch from one algorithm to the other.

6 Algorithms for Small Numbers of Machines

We next consider the case of two machines. Even though previous sections give algorithms for this case with approximation ratio arbitrarily close to 1, we are still interested in studying the performance of SNC for this case. The main reason for this is that we hoped to get ideas on how to find algorithms with good approximation ratios for $m > 2$ machines that are more efficient than our approximation schemes. However, unfortunately, several obvious adaptations of SNC are not monotone, and it seems we will need more sophisticated algorithms for $m > 2$.

Our results for two machines are as follows. SNC has an approximation ratio of $\max\{\frac{3}{s+1}, \frac{2s}{s+1}\}$. A speed-aware variation of SNC has an approximation ratio of $\min\{1+\frac{s}{s+1}, 1+\frac{1}{s}\}$, which is better than that of SNC for $s \leq 1+\sqrt{2}$. Already on three machines, this algorithm is not monotone. Rounding speeds to a power of some number $a \geq \sqrt{2}$ does not give a monotone algorithm (and most likely it also does not help to round to powers of a smaller number). Rounding job sizes does not give a monotone algorithm already for two machines.

Acknowledgment. The authors would like to thank an anonymous referee who pointed out an error in an earlier version of our approximation scheme in Section 3, another referee who helped improve the presentation, and Motti Sorani for helpful discussions.

References

1. Andelman, N., Azar, Y., Sorani, M.: Truthful Approximation Mechanisms for Scheduling Selfish Related Machines. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 69–82. Springer, Heidelberg (2005)
2. Archer, A., Papadimitriou, C., Talwar, K., Tardos, E.: An approximate truthful mechanism for combinatorial auctions with single parameter agents. In: Proc. of 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 205–214 (2003)
3. Archer, A., Tardos, E.: Truthful mechanisms for one-parameter agents. In: Proc. 42nd Annual Symposium on Foundations of Computer Science, pp. 482–491 (2001)
4. Archer, A., Tardos, E.: Frugal path mechanisms. In: Proc. of 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 991–999 (2002)
5. Auletta, V., De Prisco, R., Penna, P., Persiano, G.: Deterministic truthful approximation mechanisms for scheduling related machines. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 608–619. Springer, Heidelberg (2004)

6. Azar, Y., Epstein, L.: On-line machine covering. In: Burkard, R.E., Woeginger, G.J. (eds.) ESA 1997. LNCS, vol. 1284, pp. 23–36. Springer, Heidelberg (1997)
7. Azar, Y., Epstein, L.: Approximation schemes for covering and scheduling on related machines. In: Jansen, K., Rolim, J.D.P. (eds.) APPROX 1998. LNCS, vol. 1444, pp. 39–47. Springer, Heidelberg (1998)
8. Bansal, N., Sviridenko, M.: The Santa Claus Problem. In: Proc. of 38th Annual ACM Symposium on Theory of Computing (STOC), pp. 31–40 (2006)
9. Bezáková, I., Dani, V.: Nobody left behind: fair allocation of indivisible goods. ACM SIGecom Exchanges, 5.3 (2005)
10. Csirik, J., Kellerer, H., Woeginger, G.J.: The exact LPT-bound for maximizing the minimum completion time. Operations Research Letters 11, 281–287 (1992)
11. Deuermeyer, B.L., Friesen, D.K., Langston, M.A.: Scheduling to maximize the minimum processor finish time in a multiprocessor system. SIAM J. Discrete Methods 3, 190–196 (1982)
12. Devanur, N.R., Mihail, M., Vazirani, V.V.: Strategyproof cost-sharing mechanisms for set cover and facility location games. In: ACM Conference on E-commerce, pp. 108–114 (2003)
13. Elkind, E., Sahai, A., Steiglitz, K.: Frugality in path auctions. In: Proc. of 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 701–709 (2004)
14. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the theory of NP-Completeness. Freeman and Company, New York (1979)
15. Golovin, D.: Max-min fair allocation of indivisible goods. Technical Report, Carnegie Mellon University, CMU-CS-05-144 (2005)
16. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell System Technical J. 45, 1563–1581 (1966)
17. Kovacs, A.: Fast monotone 3-approximation algorithm for scheduling related machines. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 616–627. Springer, Heidelberg (2005)
18. Mu'alem, A., Nisan, N.: Truthful approximation mechanisms for restricted combinatorial auctions. In: Proc. of the 18th National Conference on Artificial Intelligence and 14th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI), pp. 379–384 (2002)
19. Woeginger, G.J.: A polynomial time approximation scheme for maximizing the minimum machine completion time. Operations Research Letters 20(4), 149–154 (1997)

Approximate Polynomial gcd: Small Degree and Small Height Perturbations

Joachim von zur Gathen[1] and Igor E. Shparlinski[2]

[1] B-IT, Universität Bonn
53113 Bonn, Germany
gathen@bit.uni-bonn.de

[2] Department of Computing, Macquarie University
NSW 2109, Australia
igor@ics.mq.edu.au

Abstract. We consider the following computational problem: we are given two coprime univariate polynomials f_0 and f_1 over a ring $\mathcal{R}$ and want to find whether after a small perturbation we can achieve a large gcd. We solve this problem in polynomial time for two notions of "large" (and "small"): large degree (when $\mathcal{R} = \mathbb{F}$ is an arbitrary field, in the generic case when f_0 and f_1 have a so-called normal degree sequence), and large height (when $\mathcal{R} = \mathbb{Z}$).

Keywords: Euclidean algorithm, gcd, approximate computation.

1 Introduction

Symbolic (exact) computations of the gcd of two univariate polynomials form a well-developed topic of computer algebra. These methods are not directly applicable when the coefficients are "inexact" real numbers, maybe coming from physical measurements. The appropriate model here is to ask for a "large" gcd, allowing "small" additive perturbations of the inputs. Numerical analysis provides several ways of formalizing this, and "approximate gcd" computations are an emerging topic of computer algebra with a growing literature. We only point to Bini & Boito (2007) and its references.

The present paper considers two "exact" notions of approximate gcds. Namely, let $f_0, f_1 \in \mathbb{F}[x]$ be two univariate polynomials over a field $\mathbb{F}$, both of degree at most n, and d and e integers. We are interested in perturbations $u_0, u_1 \in \mathbb{F}[x]$ of degree at most e such that $\deg \gcd(f_0 + u_0, f_1 + u_1) \geq d$. We show that if $e < \min\{2d - n, n - d\}$, then the problem has at most one solution, and if one exists, we can find it in polynomial time. Then we also consider polynomials over $\mathbb{Z}$ and obtain similar results for perturbations $v \in \mathbb{Z}[x]$ of small height that achieve a $\gcd(f_0, f_1 + v)$ of large height (without any restrictions on their degree except that $\deg v \leq n$).

These results are natural polynomial analogues of those obtained recently by Howgrave-Graham (2001).

We prove that our algorithms solve the problem under rather restrictive assumptions. It remains an open question whether either a variant or some other algorithm can tackle a larger set of input values.

We also remark that finding multidimensional analogues, that is, constructing algorithms to find "small" perturbations $u_0, \ldots, u_{s-1}$ of $f_0, \ldots, f_{s-1}$ such that $\gcd(f_0 + u_0, \ldots, f_{s-1} + u_{s-1})$ is "large" (in both number and polynomial cases) is another interesting direction of research.

2 Gcd of Large Degree

We write $f \operatorname{quo} g$ and $f \operatorname{rem} g$ for the quotient and remainder on division of f by nonzero g. Thus $f = (f \operatorname{quo} g) \cdot g + (f \operatorname{rem} g)$ and $\deg(f \operatorname{rem} g) < \deg g$.

The *degree sequence* of two univariate polynomials $f_0, f_1 \in \mathbb{F}[x]$ is the sequence of degrees $\deg f_0, \deg f_1, \deg f_2, \ldots$ of the remainders $f_0, f_1, f_2, \ldots$ in the Euclidean algorithm. Usually, but not always, $\deg f_{i-1} = 1 + \deg f_i$, and we say that f_0, f_1 *have a normal degree sequence* if that is the case for all i. We denote by M a polynomial multiplication time over $\mathbb{F}$, so that two polynomials of degree at most n can be multiplied with $O(\mathsf{M}(n))$ operations in $\mathbb{F}$. We may use $\mathsf{M}(n) = n \log n \log\log n$. In particular $\mathsf{M}(n) \in O\tilde{~}(n)$, where as usual $A \in O\tilde{~}(B)$ means that $|A| \leq c_1 B(\log(B+2))^{c_2}$ for some constants $c_1, c_2 > 0$; see von zur Gathen & Gerhard (2003, Chapter 8).

For our first result, we consider a field $\mathbb{F}$ and univariate polynomials $f_0, f_1 \in \mathbb{F}[x]$. We ask for perturbations $u_0, u_1 \in \mathbb{F}[x]$ of small degree so that the perturbed polynomials have a gcd of large degree. More precisely, we also have integers e_0, e_1, d, and we consider the set

$$\mathcal{U} = \{(u_0, u_1) \in \mathbb{F}[x]^2 \colon \deg u_i \leq e_i \text{ for } i = 0, 1,\ \deg\gcd(f_0 + u_0, f_1 + u_1) = d\}. \tag{1}$$

If e_i is negative, then the condition is meant to imply that $u_i = 0$. As an example, we can take $f_1, g, u_0 \in \mathbb{F}[x]$ of degrees n_1, m, e_0, respectively, with $e_0 < n_1 < m$, and $f_0 = gf_1 - u_0$, $d = n_1$, and $e_1 = n_1 - m - 1$. Then $\mathcal{U} = \{(u_0, 0)\}$, and the hypotheses in the theorem below are satisfied.

The algorithm below executes the Extended Euclidean Algorithm (EEA) for (f_0, f_1). It produces a finite series of "lines" (r_j, s_j, t_j) such that $s_j f_0 + t_j f_1 = r_j$, where $\deg r_j \leq n$ is strictly decreasing with growing j (see von zur Gathen & Gerhard 2003, Section 3.2). We have $s_1 = t_0 = 0$, and all other s_i and t_i are nonzero. Furthermore, since $\deg s_j$ and $\deg t_j$ are strictly increasing (see von zur Gathen & Gerhard 2003, Lemma 3.10), there is at most one "line" (r, s, t) with a prescribed degree for s (or t). We denote as $\operatorname{lc}(f)$ the leading coefficient of a polynomial f.

Algorithm 2. Approximate gcd of large degree.

Input: $f_0, f_1 \in \mathbb{F}[x]$ monic of degrees $n_0 > n_1$, respectively, coprime and with a normal degree sequence. Furthermore, integers d, e_0, e_1 with $d > 0$ and

$$e_0 < \min\{2d - n_1, n_0 - d\}, e_1 < \min\{2d - n_0, n_1 - d\}.$$

Output: $\mathcal{U}$ as in (1).

```
1. Execute the EEA with input $(f_0, f_1)$.
2. Check if the EEA computes $(r, s, t)$ with $sf_0 + tf_1 = r$ and $n_0 - \deg t = n_1 - \deg s = d$. If not, return $\mathcal{U} = \varnothing$.
3. Otherwise, if $s = 0$, then let $u_0 = -(f_0 \text{ rem } f_1)$ and return $\mathcal{U} = \{(u_0, 0)\}$ if $\deg u_0 \le e_0$, and else $\mathcal{U} = \varnothing$. If $t = 0$, then return $\mathcal{U} = \varnothing$.
4. {We now have $sf_0 + tf_1 = r$ and $st \ne 0$.} Compute
       $h_0 = f_0 \text{ quo } t,$
       $h_1 = f_1 \text{ quo } s.$
   If $h_0$ and $h_1$ are not associates, return $\mathcal{U} = \varnothing$.
5. Else, compute
       $h = \mathrm{lc}(h_0)^{-1} h_0,$
       $\alpha = \mathrm{lc}(t)^{-1},$
       $q_0 = \alpha t,$
       $q_1 = -\alpha s,$
       $u_i = q_i h - f_i \text{ for } i = 0, 1.$
6. If $\deg u_i \le e_i$ for $i = 0, 1$, then return $\mathcal{U} = \{(u_0, u_1)\}$, else return $\mathcal{U} = \varnothing$.
```

Theorem 3. *Let f_0, f_1, $n = n_0$, n_1, d, e_0, e_1 satisfy the input specification of Algorithm 2. Then the set $\mathcal{U}$ contains at most one element, and Algorithm 2 computes it with $O(\mathsf{M}(n) \log n)$ operations in $\mathbb{F}$.*

Proof. We have noted above that there is at most one "line" (r, s, t) in the EEA with $sf_0 + tf_1 = r$ and $n_0 - \deg t = n_1 - \deg s = d$. If there is no such line, then our algorithm returns $\mathcal{U} = \varnothing$. Otherwise we take that line.

We first have to check that any (u_0, u_1) returned by the algorithm is actually in the set $\mathcal{U}$. This is clear in Step 3. For an output in Step 6, we note that

$$\gcd(f_0 + u_0, f_1 + u_1) = \gcd(q_0 h, q_1 h) = h \gcd(s, t) = h,$$

since $\gcd(s, t) = 1$ (see von zur Gathen & Gerhard 2003, Lemma 3.8 (v)),

$$\deg h = \deg h_0 = \deg f_0 - \deg t = d,$$

and indeed $(u_0, u_1) \in \mathcal{U}$.

To show correctness of the algorithm it remains to show that if $\mathcal{U} \ne \varnothing$, then the algorithm indeed returns this set $\mathcal{U}$, and that $\mathcal{U}$ has at most one element.

So we now suppose that $\mathcal{U} \ne \varnothing$, let $(u_0, u_1) \in \mathcal{U}$, and $h = \gcd(f_0 + u_0, f_1 + u_1)$, so that $\deg h = d$. One first checks that the algorithm deals correctly with the two special cases $d = n_0$ and $d = n_1$. In the other cases, there exist uniquely determined $q_0, q_1 \in \mathbb{F}[x]$ such that

$$f_i = q_i h - u_i \quad \text{for} \quad i = 0, 1, \tag{4}$$

since $\deg u_i < 2d - n_{1-i} < d = \deg h$. Eliminating h from these two equations, we find

$$q_1 f_0 - q_0 f_1 = q_0 u_1 - q_1 u_0, \tag{5}$$

and call this polynomial $g = q_0 u_1 - q_1 u_0$. We have $\deg q_0 = n_0 - d < n_0$. Now g is nonzero, because otherwise f_0 would divide q_0, a polynomial of smaller degree than f_0, which would imply that $q_0 = 0$, a contradiction.

We have

$$\deg q_0 + \deg g \leq n_0 - d + \max\{(n_0 - d) + e_1, (n_1 - d) + e_0\} < n_0,$$

since $e_i < 2d - n_{1-i}$ for $i = 0, 1$.

Thus (5) satisfies the degree inequalities of the EEA, and by the well-known uniqueness property of polynomial continued fractions (see, for example, von zur Gathen & Gerhard (2003, Lemma 5.15)), there exist a remainder r and corresponding Bézout coefficients s, t in the EEA for f_0 and f_1, and nonzero $\alpha \in \mathbb{F}[x]$ so that

$$sf_0 + tf_1 = r \text{ and } (g, q_1, -q_0) = \alpha(r, s, t).$$

Furthermore, since the Euclidean degree sequence is normal, α is a constant. We have $n_0 - \deg q_0 = n_0 - \deg t = d$, similarly $n_1 - \deg q_1 = d$, and $\deg u_i \leq e_i < n_i - d = \deg q_i$, so that u_i equals the remainder of f_i on division by q_i, for $i = 0, 1$. It follows from (4) that indeed (u_0, u_1) is returned by the algorithm.

In particular, since at most one (u_0, u_1) is returned by the algorithm and it equals each element of $\mathcal{U}$ (if $\mathcal{U} \neq \varnothing$), $\mathcal{U}$ contains at most one element.

The cost for computing a single line in the Extended Euclidean Scheme is $O(\mathsf{M}(n) \log n)$; see von zur Gathen & Gerhard (2003, Algorithm 11.4). All other operations are not more expensive. $\square$

In particular the cost of Algorithm 2 is in $O\tilde{~}(n)$.

Figure 1 indicates at the bottom the triangle of values in the e_0-d-plane satisfying the restriction required for e_0, with large $n_0 = n_1 + 1$. There are trivial solutions $u_i = -f_i \operatorname{rem} h$ for $i = 0, 1$ when $e_0, e_1 \geq d - 1$, for any h of degree d; these form the area above the diagonal. We ran experiments with "random" polynomials, with and without a planted perturbed gcd. Values in the bottom triangle were, of course, correctly dealt with. We also ran the algorithm without any of the bounds d, e_0, e_1. Then it would typically compute $(u_0, u_1) \in \mathcal{U}$ with $e_0 = n_0 - d$ and $1 \leq d \leq n_1$, which is the dotted line in Figure 1. Planted gcds with $d < n_0/2$ were usually not detected.

3 Gcd of Large Height

We now look at the same problem in a different setting which we consider only for polynomials over $\mathbb{Z}$ (although it can be extended to polynomials over other fields

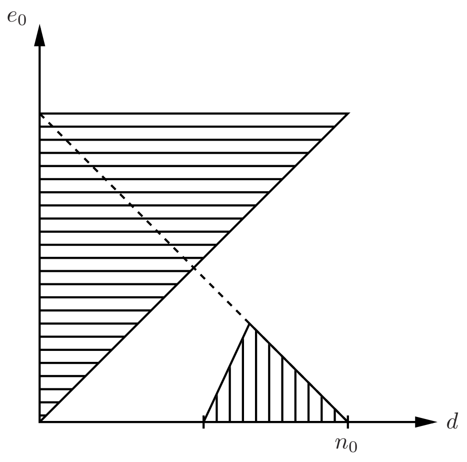


Fig. 1. The three areas – bottom triangle, half-plane, dotted line – are explained in the text

and rings). Namely, we consider the case where the height $H(f) = \max\{|f_j|\colon 0 \leq j \leq n\}$ of a polynomial

$$f = \sum_{j=0}^{n} f_j x^j \in \mathbb{Z}[x]$$

is the measure of interest.

We first need to know that a large polynomial takes a small value only very rarely. Our bound is in fact the same as for the number of roots of the polynomial.

Lemma 6. *Let $h \in \mathbb{Z}[x]$ have degree $d \geq 3$, let $A \geq 2$ be an integer, and*

$$\mathcal{A} = \{a \in \mathbb{Z}\colon -A \leq a \leq A,\ |h(a)| \leq H(h)2^{-d}A^{-d^2}\}.$$

Then $\#\mathcal{A} \leq d$.

Proof. Let $a_0, \ldots, a_d \in \{-A, \ldots, A\}$ be $d+1$ distinct integers, and let $V = (a_i^j)_{0\leq i,j\leq d}$ be the corresponding $(d+1) \times (d+1)$ Vandermonde matrix. Each column of V has L_2-norm at most

$$(\sum_{0\leq i\leq d} A^{2i})^{1/2} \leq 2^{1/2}A^d.$$

We write $h = h_d x^d + \cdots + h_1 x + h_0$. Then

$$V \cdot (h_0, \ldots, h_d)^T = (h(a_0), \ldots, h(a_d))^T$$

The determinant of V is a nonzero integer, therefore from Cramer's rule and Hadamard's inequality we find

$$\begin{aligned} H(h) = \max_{0\le k\le d} |h_k| &\le (2^{1/2}A^d)^d \left(\sum_{0\le j\le d} h(a_j)^2\right)^{1/2} \\ &\le (d+1)^{1/2}(2^{1/2}A^d)^d \max_{0\le j\le d} |h(a_j)| \le 2^d A^{d^2} \max_{0\le j\le d} |h(a_j)|, \end{aligned}$$

which proves the claim. □

The bound of Lemma 6 can be improved slightly by estimating the determinant of V more carefully.

We also need the following statement which has essentially been proved in Howgrave-Graham (2001). For the sake of completeness we present a succinct proof. The gcd of two integers, at least one of which is nonzero, is taken to be positive.

Lemma 7. *Let F_0 and F_1 be integers. Then the set of all integers V with $|V| < |F_1|$ and*

$$\gcd(F_0, F_1 + V) \ge 2\sqrt{|F_0 V|}$$

can be computed in time polynomial in $\log(|F_0 F_1| + 1)$.

Proof. For an integer V we write

$$\Delta = \gcd(F_0, F_1 + V), \qquad G_0 = \frac{F_0}{\Delta}, \qquad G_1 = \frac{F_1 + V}{\Delta}.$$

We have $|F_1 + V| < 2|F_1|$. Then one verifies that

$$F_0G_1 - F_1G_0 = \frac{F_0 V}{\Delta} = \frac{(F_1 + V_1)(F_0V_1 - F_1V_0)}{G_1\Delta^2}.$$

Hence

$$\left|\frac{F_0}{F_1} - \frac{G_0}{G_1}\right| \le \frac{2|F_1|(|F_0V|)}{|F_1|G_1^2\Delta^2} \le \frac{1}{2G_1^2}.$$

Thus G_0/G_1 is one of the convergents in the continued fraction expansion of F_0/F_1, and can be found in polynomial time. Thus $\Delta = F_0/G_0$ can take only polynomially many values. For each of them, we verify whether $V = G_1\Delta - F_1$ satisfies the condition of the lemma. □

The gcd of polynomials f_0 and f_1 in $\mathbb{Z}[x]$ is monic if one of f_0 or f_1 is. We now consider for given $f_0, f_1 \in \mathbb{Z}[x]$ and integers D, E the set

$$\mathcal{V} = \{v \in \mathbb{Z}[x] \,:\, H(v) \le E,\ H(\gcd(f_0, f_1 + v)) \ge D\}. \tag{8}$$

Algorithm 9. Approximate gcd of large height.

Input: $f_0, f_1 \in \mathbb{F}[x]$ monic of degrees $n \geq n_1$ and heights H_0 and H_1, respectively, and such that $\gcd(f_0, f_1) = 1$. Furthermore, we are given a positive $\varepsilon < 1$ and positive integers D and E.

Output: $\mathcal{V}$ as in (8).

1. Initialize $\mathcal{V} = \varnothing$. Put $A = \lceil 4\varepsilon^{-1}n^2 \rceil$ and choose $n+1$ distinct integers $a_0, \ldots, a_{n+1}$ uniformly at random in the interval $\{-A, \ldots, A\}$.
2. Evaluate $f_i(a_j)$ for $j = 0, \ldots, n$ and $i = 0, 1$.
3. For each $j = 0, \ldots, n$, compute continued fraction expansions of $f_0(a_j)/f_1(a_j)$ and find the set of all V_j with
$$\gcd\left(f_0(a_j), f_1(a_j) + V_j\right) \geq D2^{-n}A^{-n^2}.$$
4. For each possible choice $(V_0, \ldots, V_n)$ compute the unique interpolation polynomial $v \in Q[x]$ of degree at most n with $v(a_j) = V_j$ for all j. If v satisfies the conditions in (8), then add v to $\mathcal{V}$.
5. Return $\mathcal{V}$.

Theorem 10. *Let f_0, f_1, ε, D, E be inputs to Algorithm 9. If*
$$E < H_1 2^{-n-1}(4\varepsilon^{-1}n^2 + 1)^{-n^2-n}$$
and
$$D \geq 2^{n+2}(4\varepsilon^{-1}n^2 + 1)^{n^2+n}(H_0E)^{1/2},$$
then Algorithm 9 computes $\mathcal{V}$ with probability $1 - \varepsilon$ in time polynomial in $(\log(DH_1\varepsilon^{-1}))^n$.

Proof. Let $v \in \mathcal{V}$ as in (8), $h = \gcd(f_0, f_1 + v)$, and $d = \deg h$. We want to show that with probability at least $1 - \varepsilon$, v is found in step 4.

For $a_0, \ldots, a_n$ chosen in step 1, by Lemma 6 we see that with probability at least
$$\left(1 - \frac{4n}{2A+1}\right)^n > \left(1 - \frac{\varepsilon}{2n}\right)^n > 1 - \varepsilon,$$
we have simultaneously
$$|h(a_j)| \geq H(h)2^{-d}A^{-d^2} \geq D2^{-n}A^{-n^2} \qquad \text{and} \qquad |f_i(a_j)| \geq H_i 2^{-n}A^{-n^2}$$
for each $j = 0, \ldots, n$ and $i = 0, 1$, since each a_j has to avoid the at most $d + 2n \leq 3n$ "small" values of h, f_0 and f_1, and also the values $a_0, \ldots, a_{j-1}$. We also have
$$|f_1(a_j)| \geq H_1 2^{-n}A^{-n^2} > 2EA^n \geq |v(a_j)|$$
for each j, so that $f_1(a_j) + v(a_j) \neq 0$. Since the value of a polynomial gcd divides the gcd of the polynomial values, we find
$$\gcd\left(f_0(a_j), f_1(a_j) + v(a_j)\right) \geq |h(a_j)| \geq D2^{-n}A^{-n^2}.$$

On the other hand,

$$|f_i(a_j)| \le 2H_iA^n \qquad \text{and} \qquad |v(a_j)| \le 2EA^n$$

for each $j = 0, \ldots, n$ and $i = 0, 1$. Thus, under the conditions of the theorem we have

$$\begin{aligned} 2(|f_0(a_j)v(a_j)|)^{1/2} &\le (16H_0EA^{2n})^{1/2} \\ &\le (16D^2 2^{-2n-4}(4\varepsilon^{-1}n^2+1)^{-2n^2-2n}A^{2n})^{1/2} \\ &\le (D^2 2^{-2n}A^{-2n^2})^{1/2} = D2^{-n}A^{-n^2}. \end{aligned}$$

The above inequalities show that Lemma 7 applies and step 3 indeed finds the value $V_j = v(a_j)$. Thus Algorithm 9 works correctly. For any j, the set of all V_j in step 3 can be computed in time polynomial in $n \log(H_0H_1\varepsilon^{-1})$, by Lemma 7. Finally, the number of possibilities for the vector $(V_0, \ldots, V_n)$ is polynomial in $(\log DH_1\varepsilon^{-1})^n$. □

Acknowledgements

The first author's work was supported by the B-IT Foundation, and the second author's by ARC grant DP0556431. Thanks go to Daniel Loebenberger for help with the figure, and to Mark Giesbrecht.

References

1. Bini, D.A., Boito, P.: Structured Matrix-Based Methods for Polynomial ϵ-gcd: Analysis and Comparisons. In: Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation ISSAC 2007, Waterloo, Ontario, Canada, pp. 9–16 (2007)
2. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra, 2nd edn. Cambridge University Press, Cambridge (2003); 1st edn. (1999), http://www-math.upb.de/~aggathen/mca/
3. Howgrave-Graham, N.: Approximate integer common divisor. In: Silverman, J.H. (ed.) Cryptography and Lattices: International Conference, CaLC 2001, Providence. LNCS, vol. 2146, pp. 51–66. Springer, Heidelberg (2001), http://www.springerlink.com/content/ak783wexe7ghp5db/

Pseudorandom Graphs from Elliptic Curves

Igor E. Shparlinski*

Department of Computing, Macquarie University
NSW 2109, Australia
igor@ics.mq.edu.au

Abstract. Most of the constructions of pseudorandom graphs are based on additive or multiplicative groups of elements of finite fields. As a result the number of vertices of such graphs is limited to values of prime powers or some simple polynomial expressions involving prime powers. We show that elliptic curves over finite fields lead to new constructions of pseudorandom graphs with a new series of parameters. Accordingly, the number of vertices of such graphs can take most of positive integer values (in fact, any positive value under some classical conjectures about the gaps between prime numbers).

Keywords: Pseudorandom graph, elliptic curve, exponential sum.

1 Introduction

1.1 Motivation

Let $\mathcal{V}_{\mathfrak{G}}$ denote the set of vertices of an undirected graph $\mathfrak{G}$. Given two disjoint subsets $\mathcal{U}, \mathcal{W} \subseteq \mathcal{V}_{\mathfrak{G}}$ we define $e_{\mathfrak{G}}(\mathcal{U}, \mathcal{W})$ as the number of edges which lead from vertices in $\mathcal{U}$ to vertices in $\mathcal{W}$. We also write $E_{\mathfrak{G}}$ for the number of edges in $\mathfrak{G}$.

It is natural to expect that for a graph with a reasonably uniform distribution of edges the quantity

$$\Delta(\mathfrak{G}) = \max_{\substack{\mathcal{U},\mathcal{W}\subseteq\mathcal{V}_{\mathfrak{G}}\\ \mathcal{U}\cap\mathcal{V}=\emptyset}} \left| e_{\mathfrak{G}}(\mathcal{U}, \mathcal{W}) - \frac{E_{\mathfrak{G}}}{(\#\mathcal{V}_{\mathfrak{G}})^2} \#\mathcal{U}\#\mathcal{W} \right|,$$

which is called the *discrepancy* of $\mathfrak{G}$, is small compared to $E_{\mathfrak{G}}$. In particular, with high probability, this property holds for various families of random graphs. Accordingly, graphs for which this is the case, are called *pseudorandom*, see [6,9] for surveys of pseudorandom graphs, their various equivalent definitions and applications.

We note that for d-regular graphs $\mathfrak{G}$, which are of our main interest, the definition of the discrepancy simplifies as

$$\Delta(\mathfrak{G}) = \max_{\mathcal{U},\mathcal{W}\subseteq\mathcal{V}_{\mathfrak{G}}} \left| e_{\mathfrak{G}}(\mathcal{U}, \mathcal{W}) - \frac{d}{n} \#\mathcal{U}\#\mathcal{W} \right|,$$

* This work was supported in part by ARC grant DP0556431.

where $n = \#\mathcal{V}_{\mathfrak{G}}$ is number of vertices of $\mathfrak{G}$.

Since the adjacency matrix of an undirected graph is symmetric, its eigenvalues are real and we order them as $\lambda_1 \geq \ldots \geq \lambda_n$. If $\mathfrak{G}$ is d-regular then $\lambda_1 = d$. We also note that $\lambda_n = -d$ if and only if $\mathfrak{G}$ is bipartite. Moreover, the second largest absolute value of the eigenvalues, which we denote as

$$\rho(\mathfrak{G}) = \max\{|\lambda_j| \ : \ |\lambda_j| < d, \ j = 1, \ldots, n\}$$

plays an exceptionally important role in the theory of pseudorandom graphs, see [2,6,9]. For example,

$$\left| e_{\mathfrak{G}}(\mathcal{U}, \mathcal{W}) - \frac{d}{n} \#\mathcal{U} \#\mathcal{W} \right| \leq \rho(\mathfrak{G}) \sqrt{\#\mathcal{U} \#\mathcal{W}}, \qquad (1)$$

see [6, Lemma 2.5] or [9, Theorem 2.11]. We also refer to [2] for a detailed study of relations between $\rho(\mathfrak{G})$ and various properties of $\mathfrak{G}$.

Several explicit constructions of pseudorandom graphs are known, see [6,9] and references therein. Typically such constructions are based on additive or multiplicative groups (and their combinations) of elements of a finite field, or sometimes on matrix groups or supersingular elliptic curves over finite fields. For example, such are all known constructions of *Ramanujan graphs* which are d-regular graphs $\mathfrak{G}$ with $\rho(\mathfrak{G}) \leq 2\sqrt{d-1}$ which is asymptotically optimal, see the recent survey [11]. Accordingly, this leads to cardinalities n which are prime powers q, or shifted prime powers $q \pm 1$, or some simple polynomial expressions of them (for example such as $q^t(q-1)$ with an integer $t \geq 2$ for the so-called projective norm graphs of [1]). The choice of possible values of the total number of edges $E_{\mathfrak{G}}$ is usually quite constrained too.

1.2 Our Results

Here we use arbitrary elliptic curves over finite fields to produce a new construction of d-regular pseudorandom Cayley graphs with a new series of parameters. In particular, we estimate $\rho(\mathfrak{G})$ for these graphs $\mathfrak{G}$, which together with (1) implies a bound on $\Delta(\mathfrak{G})$. We also obtain another bound which gives a more precise estimate on $e_{\mathfrak{G}}(\mathcal{U}, \mathcal{W})$ in the case when one of the sets is rather thin.

Our construction works for every n which is the number of $\mathbb{F}_p$-rational points of some elliptic curve over a field of p elements $\mathbb{F}_p$, where p is an arbitrary prime.

By the classical results of Deuring [4] for any prime $p > 3$ and integer n in the interval $[p + 1 - 2p^{1/2}, p + 1 + 2p^{1/2}]$ is taken as the number of $\mathbb{F}_p$ -rational points of some elliptic curve over $\mathbb{F}_p$.

Note that the probability that an integer n is not in such an interval for some prime p is very small. More precisely, for $x \to \infty$, the number of such integers $n \leq x$ is $O(x^{2/3+o(1)})$ which follows via partial summation from the estimate

$$\sum_{\substack{p_j \leq z \\ p_{j+1} - p_j > z^{1/2}}} (p_{j+1} - p_j) \leq z^{2/3+o(1)}$$

which is given in [10], where p_j the jth prime number, $j = 1, 2, \ldots$, see also [12]. By the classical Cramer conjecture [3],

$$p_{j+1} - p_j = O\left((\log p_j)^2\right).$$

Certainly under this conjecture or even its much more relaxed version

$$p_{j+1} - p_j < 4p_j^{1/2}. \tag{2}$$

every positive integer n presents a cardinality of some elliptic curve over a finite field. On the other hand, even the Riemann Hypothesis falls a little short of (2) (at least without some additional arguments).

1.3 Outline of the Paper and Notation

We provide all necessary facts on elliptic curves in Section 2.1 and this is enough to understand our construction given in Section 3.1.

The proofs of our bounds of the second largest eigenvalue and other estimates, which are given in Section 3.2 are based on some recent bounds exponential sums over points of elliptic curves, which we present in Section 2.2.

Throughout the paper, the implicit constants in the symbols 'O' and '$\ll$' and '$\gg$' may occasionally depend on an integer parameter ν and are absolute otherwise (we recall that $U = O(V)$ and $U \ll V$ are both equivalent to the inequality $|U| \le cV$ with some constant $c > 0$).

We always use p to denote a prime number.

We also assume that $\mathbb{F}_p$ is represented by the set $\{0, \ldots, p-1\}$.

2 Preparation

2.1 Background on Elliptic Curves

Since considering elliptic curves over non-prime fields does not substantially extend the set of parameters our construction produces we only consider elliptic curves $\mathbf{E}$ over $\mathbb{F}_p$ where p is a prime. In this case, $\mathbf{E}$ can be given an affine *Weierstrass equation* of the form

$$y^2 = x^3 + ax + b, \tag{3}$$

with coefficients $a, b \in \mathbb{F}_p$, such that $4a^3 + 27b^2 \ne 0$.

It is known, see [14], that the set $\mathbf{E}(\mathbb{F}_p)$ of $\mathbb{F}_p$-rational points of $\mathbf{E}$ forms an *Abelian group* under an appropriate composition rule, which we call *addition* and denote $\oplus$, and with the point at infinity $\mathcal{O}$ as the neutral element (we also use $\ominus$ in its obvious meaning as the operation inverse to $\oplus$). Thus, given a point $Q \in \mathbf{E}(\mathbb{F}_p)$ and an integer m we write mQ for the sum of m copies of Q. We also recall that

$$|\#\mathbf{E}(\mathbb{F}_p) - p - 1| \le 2p^{1/2},$$

where $\#\mathbf{E}(\mathbb{F}_p)$ is the number of $\mathbb{F}_p$-rational points, including the point at infinity $\mathcal{O}$. Given a point $Q \in \mathbf{E}(\mathbb{F}_p)$ with $Q \neq \mathcal{O}$ we denote by $x(Q)$ and $y(Q)$ its affine components, $Q = (x(Q), y(Q))$. The negative of $Q = (x(Q), y(Q))$ is given by $-Q = (x(Q), -y(Q)) = \mathcal{O} \ominus Q$.

For a prime ℓ denote by $\mathbf{E}[\ell]$ the group of ℓ-torsion points on $\mathbf{E}$, that is the set of points Q on $\mathbf{E}$, defined over the algebraic closure of $\mathbb{F}_p$ for which $\ell Q = \mathcal{O}$. Then for $\ell \neq p$ we have

$$\#\mathbf{E}[\ell] = \ell^2.$$

Let $\mathcal{X}_{\mathbf{E}}$ be the group of characters on $\mathbf{E}(\mathbb{F}_p)$ (considered as an Abelian group). It is known that $\mathbf{E}(\mathbb{F}_p)$ is of rank at most 2 and thus is isomorphic to $\mathbb{Z}/M \times \mathbb{Z}/L$ for unique positive integers M and L with $L \mid M$ and $\#\mathbf{E}(\mathbb{F}_p) = ML$. Thus one can find points $G_1, G_2 \in \mathbf{E}(\mathbb{F}_p)$ of orders M and L, respectively, and such that any point in $\mathbf{E}(\mathbb{F}_p)$ can be written uniquely in the form $mG_1 + \ell G_2$ with $1 \le m \le M$ and $1 \le \ell \le L$. Now $\mathcal{X}_{\mathbf{E}}$ can be explicitly described as

$$\mathcal{X}_{\mathbf{E}} = \{\chi \mid \chi(mG_1 + \ell G_2) = \exp(2\pi i a m/M)\exp(2\pi i a\ell/L),\\ 0 \le a < M,\ 0 \le b < L\}$$

(this set does not depend on the choice of generators G_1 and G_2). The trivial character χ_0 corresponds to $a = b = 0$.

For an elliptic curve $\mathbf{E}$ over $\mathbb{F}_p$ and a positive integer h, we use $\mathbf{E}(\mathbb{F}_p; h)$ to denote the set of points $Q \in \mathbf{E}(\mathbb{F}_p)$ with $x(Q) \in \{0, \ldots, h-1\}$. We need the following special case of much more general results of [5,15].

Lemma 1. *Let p be a prime. Then for any integer h with $1 \le h < p$, we have $\#\mathbf{E}(\mathbb{F}_p; h) = h + O(p^{1/2}\log p)$.*

We also recall, that for any $\varepsilon > 0$ and sufficiently large p, for the set $\mathbf{E}(\mathbb{F}_p; h)$ generates the whole group $\mathbf{E}(\mathbb{F}_p)$ provided $h \ge p^{1/2+\varepsilon}$, see [8].

2.2 Exponential Sums

For a prime p we denote

$$\psi_p(z) = \exp(2\pi i z/p),$$

which is an additive character of $\mathbb{F}_p$. We have the following orthogonality relation

$$\frac{1}{p}\sum_{r=-(p-1)/2}^{(p-1)/2} \psi_p(rz) = \begin{cases} 1, & \text{if } z \equiv 0 \pmod p, \\ 0, & \text{if } z \not\equiv 0 \pmod p, \end{cases} \tag{4}$$

which is used to express various characteristic functions and thus to relate various counting questions to exponential sums.

The identity (4) is very often complemented by the bound

$$\sum_{z=W+1}^{W+Z} \psi_p(rz) \ll \min\{Z, p/|r|\} \tag{5}$$

which holds for any integers r, W and $Z \geq 1$ with $0 < |r| \leq p/2$, see [7, Bound (8.6)].

We need to recall some bounds of exponential sums over points of elliptic curves.

Our basic tool is the following result from [8].

Lemma 2. *Let p be a prime and let $\mathbf{E}$ be an elliptic curve over $\mathbb{F}_p$. For any nontrivial character $\chi \in \mathcal{X}_{\mathbf{E}}$ and an integer r with $\gcd(r,p)=1$ the bound*

$$\sum_{\substack{Q\in\mathbf{E}(\mathbb{F}_p)\\ Q\neq\mathcal{O}}} \chi(Q)\psi_p\left(rx(Q)\right) \ll p^{1/2}$$

holds.

We also need the following result from [13].

Lemma 3. *Let p be a prime and let $\mathbf{E}$ be an elliptic curve over $\mathbb{F}_p$. For any subsets $\mathcal{Q},\mathcal{R} \subseteq \mathbf{E}(\mathbb{F}_p)$ and an integer r with $\gcd(r,p)=1$, the bound*

$$\sum_{\substack{Q\in\mathcal{Q},R\in\mathcal{R}\\ Q\neq R}} \psi_p\left(rx(Q\ominus R)\right) \ll (\#\mathcal{Q})^{1-1/2\nu}(\#\mathcal{R})^{1/2}p^{1/2\nu} + (\#\mathcal{Q})^{1-1/2\nu}\#\mathcal{R}p^{1/4\nu}$$

holds for any integer $\nu \geq 1$, where the implied constant depends only on ν.

As in [13], we remark that for any $\varepsilon \in (0,1/2)$, taking $\nu = \lceil 1/\varepsilon \rceil$, Lemma 3 gives a nontrivial bound provided $\#\mathcal{Q} > q^{1/2+\varepsilon}$ and $\#\mathcal{R} > q^{\varepsilon}$. Certainly $\#\mathcal{Q}$ and $\#\mathcal{R}$ can be interchanged on the right hand side of that bound too (it is enough to apply Lemma 3 to the new set $-\mathcal{Q}$ and $-\mathcal{R}$).

3 Main Results

3.1 Construction

Let $p \geq 3$ be a prime and let $\mathbf{E}$ be an elliptic curve over $\mathbb{F}_p$. For any integer h with $1 \leq h < p$, we denote by $\widetilde{\mathbf{E}}(\mathbb{F}_p;h)$ obtained from $\mathbf{E}(\mathbb{F}_p;h)$ by removing points of order 2, that is,

$$\widetilde{\mathbf{E}}(\mathbb{F}_p;h) = \mathbf{E}(\mathbb{F}_p;h) \setminus \mathbf{E}[2].$$

Finally, given a positive integer $k < 0.5\#\left(\mathbf{E}(\mathbb{F}_p) \setminus \mathbf{E}[2]\right)$, we find a largest h with

$$2k \geq \#\widetilde{\mathbf{E}}(\mathbb{F}_p;h)$$

and choose a set $\mathcal{S}$ with

$$\#\mathcal{S} = k, \qquad \mathcal{S} \cap -\mathcal{S} = \emptyset, \qquad \widetilde{\mathbf{E}}(\mathbb{F}_p;h) \subseteq (\mathcal{S} \cup -\mathcal{S}) \subseteq \widetilde{\mathbf{E}}(\mathbb{F}_p;h+1), \tag{6}$$

where $-\mathcal{S}$ denotes the set of points $-P$ with $P \in \mathcal{S}$.

We define the graph $G_p(\mathbf{E},\mathcal{S})$ as the graph whose vertices are labeled by the $n = \#\mathbf{E}(\mathbb{F}_p)$ points of $\mathbf{E}(\mathbb{F}_p)$ and two distinct vertices Q and R are connected if and only if $Q \ominus R \in \mathcal{S}$ or $R \ominus Q \in \mathcal{S}$.

Since $\mathcal{S} \cap \mathbf{E}[2] = \emptyset$, we see that only one of the points P and $-P$ may belong to $\mathcal{S}$. Therefore $G_p(\mathbf{E},\mathcal{S})$ is a $2k$-*regular Cayley graph* on $n = \#\mathbf{E}(\mathbb{F}_p)$ vertices.

3.2 Estimates

Now we are prepared to formulate our main estimate on the second largest eigenvalue of the graphs $G_p(\mathbf{E}, \mathcal{S})$.

Theorem 1. *Let $p \geq 3$ be a prime and let $\mathbf{E}$ be an elliptic curve over $\mathbb{F}_p$. Then for any set S with (6) the graph $\mathfrak{G} = G_p(\mathbf{E}, \mathcal{S})$ is a $2k$-regular Cayley graph on $n = \#\mathbf{E}(\mathbb{F}_p)$ vertices for which*

$$\rho(\mathfrak{G}) \ll n^{1/2} \log n.$$

Proof. The eigenvalues of $\mathfrak{G}$ are given by the character sums

$$\sum_{P \in \mathcal{S} \cup -\mathcal{S}} \chi(P), \qquad \chi \in \mathcal{X}_{\mathfrak{G}},$$

see [6, Proposition 11.7] (see also [9, Section 3]), where the trivial character χ_0 corresponds to the trivial eigenvalue $2k$.

Clearly there are at most 2 points $Q \in \mathbf{E}(\mathbb{F}_p)$ with $x(Q) = h$. Thus

$$\#\left((\mathcal{S} \cup -\mathcal{S}) \setminus \widetilde{\mathbf{E}}(\mathbb{F}_p; h)\right) \leq \#\left(\widetilde{\mathbf{E}}(\mathbb{F}_p; h+1) \setminus \widetilde{\mathbf{E}}(\mathbb{F}_p; h)\right) \leq 2.$$

We also have

$$\#\left(\mathbf{E}(\mathbb{F}_p; h) \setminus \widetilde{\mathbf{E}}(\mathbb{F}_p; h)\right) \leq \#\mathbf{E}[2] \leq 4$$

from which we deduce

$$\#\left(\mathbf{E}(\mathbb{F}_p; h) \setminus (\mathcal{S} \cup -\mathcal{S})\right) \leq 6. \tag{7}$$

Therefore

$$\sum_{P \in \mathcal{S} \cup -\mathcal{S}} \chi(P) = \sum_{P \in \mathcal{S}} (\chi(P) + \chi(-P)) = \sum_{P \in \mathbf{E}(\mathbb{F}_p; h)} \chi(P) + O(1). \tag{8}$$

Using the identity (4), we write

$$\begin{aligned} \sum_{\substack{P \in \mathbf{E}(\mathbb{F}_p; h) \\ P \neq \mathcal{O}}} \chi(P) &= \sum_{P \in \mathbf{E}(\mathbb{F}_p)} \chi(P) \sum_{z=0}^{h-1} \frac{1}{p} \sum_{r=-(p-1)/2}^{(p-1)/2} \psi_p\left(r\left(x(P) - z\right)\right) \\ &= \frac{1}{p} \sum_{r=-(p-1)/2}^{(p-1)/2} \sum_{\substack{P \in \mathbf{E}(\mathbb{F}_p) \\ P \neq \mathcal{O}}} \chi(P) \psi_p\left(r x(P)\right) \sum_{z=0}^{h-1} \psi_p\left(-rz\right). \end{aligned}$$

For a nontrivial character $\chi \neq \chi_0$ we now use Lemma 2 and then the bound (5), getting

$$\begin{aligned} \sum_{\substack{P \in \mathbf{E}(\mathbb{F}_p; h) \\ P \neq \mathcal{O}}} \chi(P) &\ll p^{-1/2} \sum_{r=-(p-1)/2}^{(p-1)/2} \left| \sum_{z=0}^{h-1} \psi_p\left(-rz\right) \right| \\ &\ll p^{1/2} \sum_{r=-(p-1)/2}^{(p-1)/2} \frac{1}{|r|} \ll p^{1/2} \log p. \end{aligned}$$

Substituting this bound in (8) and using that $n = p + O(p^{1/2})$, we conclude the proof. $\square$

Combining Theorem 1 with (1), we deduce that for the graph $\mathfrak{G} = G_p(\mathbf{E}, \mathcal{S})$ and any disjoint subsets $\mathcal{Q}, \mathcal{R} \subseteq \mathbf{E}(\mathbb{F}_p)$ we have

$$e_{\mathfrak{G}}(\mathcal{Q}, \mathcal{R}) - \frac{2k}{n}\#\mathcal{Q}\#\mathcal{R} \ll \sqrt{n\#\mathcal{Q}\#\mathcal{R}}\log n. \qquad (9)$$

In particular

$$\Delta(\mathfrak{G}) \ll n^{3/2}\log n$$

which is better than the trivial bound $\Delta(\mathfrak{G}) \le kn$ when $k = \#\mathcal{S} \ge n^{1/2+\varepsilon}$ for any fixed $\varepsilon > 0$ and sufficiently large n. As we have remarked in Section 2.1, the graph $\mathfrak{G}$ is connected under this condition.

We now obtain another bound on $e_{\mathfrak{G}}(\mathcal{Q}, \mathcal{R})$ which improves (9) when one of the sets $\mathcal{Q}$ and $\mathcal{R}$ is small.

Theorem 2. *Let $p \ge 3$ be a prime and let $\mathbf{E}$ be an elliptic curve over $\mathbb{F}_p$. Then for any set S with (6) the graph $\mathfrak{G} = G_p(\mathbf{E}, \mathcal{S})$ is a $2k$-regular Cayley graph on $n = \#\mathbf{E}(\mathbb{F}_p)$ vertices and for any disjoint subsets $\mathcal{Q}, \mathcal{R} \subseteq \mathbf{E}(\mathbb{F}_p)$ we have*

$$\begin{aligned} e_{\mathfrak{G}}(\mathcal{Q}, \mathcal{R}) &- \frac{2k}{n}\#\mathcal{Q}\#\mathcal{R} \\ &\ll \left((\#\mathcal{Q})^{1-1/2\nu}(\#\mathcal{R})^{1/2}n^{1/2\nu} + (\#\mathcal{Q})^{1-1/2\nu}\#\mathcal{R}n^{1/4\nu}\right)\log n \end{aligned}$$

that for any integer $\nu \ge 1$.

Proof. We have $e_{\mathfrak{G}}(\mathcal{Q}, \mathcal{R}) = e_{\mathfrak{G},0}(\mathcal{Q}, \mathcal{R}) + e_{\mathfrak{G},1}(\mathcal{Q}, \mathcal{R})$ where

$$\begin{aligned} e_{\mathfrak{G},0}(\mathcal{Q}, \mathcal{R}) &= \#\{(Q,R) \in \mathcal{Q} \times \mathcal{R} \ : \ Q \ominus R \in \mathcal{S}\}, \\ e_{\mathfrak{G},1}(\mathcal{Q}, \mathcal{R}) &= \#\{(Q,R) \in \mathcal{Q} \times \mathcal{R} \ : \ R \ominus Q \in \mathcal{S}\}. \end{aligned}$$

Certainly both quantities can be considered analogously, so we only concentrate on $e_{\mathfrak{G},0}(\mathcal{Q}, \mathcal{R})$.

Using (7) we write

$$e_{\mathfrak{G},0}(\mathcal{Q}, \mathcal{R}) = \sum_{\substack{(Q,R)\in\mathcal{Q}\times\mathcal{R} \\ Q\ominus R\in\mathcal{S}}} 1 = \sum_{\substack{(Q,R)\in\mathcal{Q}\times\mathcal{R} \\ Q\ominus R\in\mathbf{E}(\mathbb{F}_p;h)}} 1 + O\left(\min\{\#\mathcal{Q}, \#\mathcal{R}\}\right). \qquad (10)$$

Now, using the identity (4), we obtain

$$\begin{aligned} \sum_{\substack{(Q,R)\in\mathcal{Q}\times\mathcal{R} \\ Q\ominus R\in\mathbf{E}(\mathbb{F}_p;h)}} 1 &= \sum_{Q\in\mathcal{Q}, R\in\mathcal{R}} \sum_{z=0}^{h-1} \frac{1}{p} \sum_{r=-(p-1)/2}^{(p-1)/2} \psi_p\left(r\left(x(Q\ominus R) - z\right)\right) \\ &= \frac{1}{p} \sum_{r=-(p-1)/2}^{(p-1)/2} \sum_{Q\in\mathcal{Q}, R\in\mathcal{R}} \psi_p\left(rx(Q\ominus R)\right) \sum_{z=0}^{h-1} \psi_p\left(-rz\right) \\ &= \frac{h}{p}\#\mathcal{Q}\#\mathcal{R} + W, \end{aligned}$$

where

$$W = \frac{1}{p} \sum_{0<|r|\le (p-1)/2} \sum_{Q\in\mathcal{Q}, R\in\mathcal{R}} \psi_p\left(rx(Q\ominus R)\right) \sum_{z=0}^{h-1} \psi_p\left(-rz\right).$$

Applying Lemma 3 and then the bound (5) we derive that for any integer $\nu \ge 1$,

$$\begin{aligned} |W| &\ll \frac{1}{p}\left((\#\mathcal{Q})^{1-1/2\nu}(\#\mathcal{R})^{1/2}p^{1/2\nu} + (\#\mathcal{Q})^{1-1/2\nu}\#\mathcal{R}p^{1/4\nu}\right) \\ &\qquad\qquad \sum_{0<|r|\le (p-1)/2} \left|\sum_{z=0}^{h-1}\psi_p\left(-rz\right)\right| \\ &\ll \left((\#\mathcal{Q})^{1-1/2\nu}(\#\mathcal{R})^{1/2}p^{1/2\nu} + (\#\mathcal{Q})^{1-1/2\nu}\#\mathcal{R}p^{1/4\nu}\right)\log p. \end{aligned}$$

Thus from (10), using the trivial bound

$$\min\{\#\mathcal{Q}, \#\mathcal{R}\} \le \sqrt{\#\mathcal{Q}\#\mathcal{R}} \le (\#\mathcal{Q})^{1-1/2\nu}(\#\mathcal{R})^{1/2}p^{1/2\nu},$$

we obtain

$$\begin{aligned} e_{\mathfrak{G},0}(\mathcal{Q},\mathcal{R}) &- \frac{h}{p}\#\mathcal{Q}\#\mathcal{R} \\ &\ll \left((\#\mathcal{Q})^{1-1/2\nu}(\#\mathcal{R})^{1/2}p^{1/2\nu} + (\#\mathcal{Q})^{1-1/2\nu}\#\mathcal{R}p^{1/4\nu}\right)\log p. \end{aligned}$$

Since $e_{\mathfrak{G},1}(\mathcal{Q},\mathcal{R})$ satisfies the same bound, we derive

$$\begin{aligned} e_{\mathfrak{G}}(\mathcal{Q},\mathcal{R}) &- \frac{2h}{p}\#\mathcal{Q}\#\mathcal{R} \\ &\ll \left((\#\mathcal{Q})^{1-1/2\nu}(\#\mathcal{R})^{1/2}n^{1/2\nu} + (\#\mathcal{Q})^{1-1/2\nu}\#\mathcal{R}n^{1/4\nu}\right)\log n. \end{aligned}$$

By Lemma 1 and (7) we have

$$k = \#\mathbf{E}(\mathbb{F}_p; h) + O(1) = h + O(p^{1/2}\log p) = h + O(n^{1/2}\log n).$$

It remains to note that

$$\frac{2h}{p} = \frac{2k + O(n^{1/2}\log n)}{n + O(n^{1/2})} = \frac{2k}{n} + O(kn^{-3/2}\log n) = \frac{2k}{n} + O(n^{-1/2}\log n)$$

and also that $\#\mathcal{Q}\#\mathcal{R}n^{-1/2} \le (\#\mathcal{Q})^{1-1/2\nu}\#\mathcal{R}n^{1/4\nu}$. □

Exactly as Lemma 3, Theorem 2 is nontrivial provided $\#\mathcal{Q} > n^{1/2+\varepsilon}$ and $\#\mathcal{R} > n^{\varepsilon}$ for any fixed $\varepsilon > 0$ and sufficiently large n (as before, $\mathcal{Q}$ and $\mathcal{R}$ can be interchanged).

4 Remarks

Different Weierstrass equations (3) may define isomorphic curves. However it seems that the corresponding graphs $G_p(\mathbf{E}, \mathcal{S})$ are not isomorphic. It would be interesting to estimate the total number of non-isomorphic graphs our construction produces over all curves $\mathbf{E}$ over $\mathbb{F}_p$ with $\#E(\mathbb{F}_p) = n$ and the corresponding sets $\mathcal{S}$ with $\#\mathcal{S} = k$.

We also hope that our graphs, combined with some ideas of constructions of [16,17], may lead to new strong hash functions.

References

1. Alon, N., Rónyai, L., Szabó, T.: Norm-graphs: Variations and applications. J. Combin. Theory, Ser. B 76, 280–290 (1999)
2. Chung, F.R.K.: Spectral graph theory. Amer. Math. Soc., Providence, RI (1997)
3. Cramer, H.: On the order of magnitude of the difference between consecutive prime numbers. Acta Arith. 2, 23–46 (1936)
4. Deuring, M.: Die Typen der Multiplikatorenringe elliptischer Funktionenkörper. Abh. Math. Sem. Hansischen Univ. 14, 197–272 (1941)
5. Granville, A., Shparlinski, I.E., Zaharescu, A.: On the distribution of rational functions along a curve over $\mathbb{F}_p$ and residue races. J. Number Theory 112, 216–237 (2005)
6. Hoory, S., Linial, N., Wigderson, A.: Expander graphs and their applications. Bull. Amer. Math. Soc. 43, 439–561 (2006)
7. Iwaniec, H., Kowalski, E.: Analytic number theory. Amer. Math. Soc., Providence, RI (2004)
8. Kohel, D.R., Shparlinski, I.E.: Exponential sums and group generators for elliptic curves over finite fields. In: Bosma, W. (ed.) ANTS 2000. LNCS, vol. 1838, pp. 395–404. Springer, Heidelberg (2000)
9. Krivelevich, M., Sudakov, B.: Pseudo-random graphs. In: More Sets, Graphs and Numbers. Bolyai Society Mathem. Studies 15, pp. 199–262. Springer, Heidelberg (2006)
10. Matomäki, K.: Large differences between consecutive primes. Quart. J. Math. 58, 489–518 (2007)
11. Murty, M.R.: Ramanujan graphs. J. Ramanujan Math. Soc. 18, 1–20 (2003)
12. Peck, A.S.: Differences between consecutive primes. Proc. London Math. Soc. 76, 33–69 (1998)
13. Shparlinski, I.E.: Bilinear character sums over elliptic curves. Finite Fields and Their Appl. (to appear)
14. Silverman, J.H.: The arithmetic of elliptic curves. Springer, Berlin (1995)
15. Vajaitu, M., Zaharescu, A.: Distribution of values of rational maps on the $\mathbb{F}_p$-points on an affine curve. Monatsh. Math. 136, 81–86 (2002)
16. Tillich, J.-P., Zémor, G.: Hashing with SL_2. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 40–49. Springer, Heidelberg (1994)
17. Zémor, G.: Hash functions and Cayley graphs. Designs, Codes and Cryptography 4, 381–394 (1994)

Speeding-Up Lattice Reduction with Random Projections (Extended Abstract)

Ali Akhavi[1] and Damien Stehlé[2]

[1] Université de Caen/GREYC, Bd Maréchal Juin, F-14032 Caen Cedex, France
ali.akhavi@info.unicaen.fr
http://users.info.unicaen.fr/~akhavi

[2] CNRS/LIP/INRIA/ENS/UCBL, 46 allée d'Italie, F-69364 Lyon Cedex 07, France
damien.stehle@ens-lyon.fr
http://perso.ens-lyon.fr/damien.stehle

Abstract. Lattice reduction algorithms such as LLL and its floating-point variants have a very wide range of applications in computational mathematics and in computer science: polynomial factorization, cryptology, integer linear programming, *etc.* It can occur that the lattice to be reduced has a dimension which is small with respect to the dimension of the space in which it lies. This happens within LLL itself. We describe a randomized algorithm specifically designed for such rectangular matrices. It computes bases satisfying, with very high probability, properties similar to those returned by LLL. It significantly decreases the complexity dependence in the dimension of the embedding space. Our technique mainly consists in randomly projecting the lattice on a lower dimensional space, by using two different distributions of random matrices.

1 Introduction

A *lattice* L is a set of integer linear combinations of some linearly independent vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d \in \mathbb{R}^n$. These vectors are called a *basis* of the lattice. A given lattice has infinitely many bases, but their cardinality d is always the same: it is called the lattice *dimension*. The dimension n of the basis vectors is called the *degree* of the lattice. The degree of a given lattice cannot be smaller than its dimension. In this article, we are interested in lattices whose degrees are much higher than their dimensions: we will informally call *rectangular* such lattices. When the degree and the dimension match, the lattice is *full-dimensional*.

Lattices are an algorithmic tool that proved crucial in many areas in computer science and mathematics, ranging from cryptology [12,1,19] to computer arithmetic [6,7,24] and algorithmic number theory [20,11]. They became popular in 1982, when Arjen Lenstra, Hendrik Lenstra Jr, and László Lovász introduced the renowned algorithm now known under the acronym LLL [17]. Given a lattice basis made of integer vectors, the LLL algorithm discloses a short non-zero lattice vector in time polynomial in the bit-size of the input. This algorithm has complexity $O(d^5 n \log^3 B)$, where B is the maximum of the norms of the input vectors. The practical variants of LLL rely on floating-point arithmetic (for the underlying Gram-Schmidt orthogonalization), and the best fully proved such variant is due to Nguyen and Stehlé [18]. The so-called L^2 algorithm has bit-complexity $O(d^4 n \log B(d + \log B))$. We will consider this variant here, though the technique we introduce works with any other variant of LLL.

Our main result is to provide a randomized algorithm taking as input a lattice basis and computing another basis such that with overwhelming probability (e.g., as d grows to infinity) this basis satisfies properties similar to those returned by LLL. If one neglects all terms polynomial in $\log d, \log n$ and $\log \log B$, then it runs in time $\tilde{O}(d^2(d^3 + n) \log B(d + \log B))$: the cost dependence in the degree of the lattice is considerably weakened. Moreover, the bit-size of the integers involved in the algorithm is essentially the same as the bit-size of the initial basis. A simpler strategy than the one we develop is based on the Gram matrix (the symmetric matrix of the pairwise inner-products): one can compute the LLL-transformation by reducing the Gram matrix. It is deterministic and decreases the cost dependence in n, but it suffers from two drawbacks: the bit-sizes of the entries of the Gram matrix are essentially twice bigger than the ones of the input basis and the floating-point inaccuracies can be significantly larger if we start from the Gram matrix. Strong heuristics [22] tend to show that one can use half the precision required by the L^2 algorithm by disregarding the Gram matrix.

Rectangular lattices arise in the two following situations. First of all, they sometimes occur in Coppersmith's methods to find small roots of polynomials over the integers and modulo an integer [12]. These methods have numerous applications in cryptology. The involved lattice bases are full-dimensional but highly structured. This structure sometimes creates situations where subsets of the input basis vectors suffice to provide the short vectors found by LLL. The number of vectors to be considered can be drastically decreased, while their embedding dimension remains constant, thus creating rectangular lattices. This arises in [4], where Coppersmith's method is used to cryptanalyse RSA when the secret exponent is unusually small, and in [23], where it is used to find bad cases for the rounding of mathematical functions, in the field of computer arithmetic. In [4], the ratio between the degree and the dimension is constant, while in [23] the degree grows as the square of the dimension. Another context where rectangular lattices arise is the LLL algorithm itself (and most of its variants, including L^2), even for full-dimensional lattices. In LLL, the basis is reduced incrementally. There is a main loop whose main parameter is an index k. The meaning of this index is that in the current basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d)$, the vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{k-1}$ are already LLL-reduced and one is trying to extend this property to $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k$. At the beginning of the execution, the index k is set to 2, while at the end it reaches $d + 1$. As long as the index k has not been beyond some arbitrary k_0, we are in fact applying LLL to the vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{k_0} \in \mathbb{Z}^n$. The smaller the considered k_0, the more rectangular the lattice being reduced. Our technique may be used within LLL to speed it up by a constant factor.

To achieve the result, we develop a few tools, which may be of independent interest. Firstly, we decrease the degree of the lattice by applying a random projection technique: we multiply the $n \times d$ input basis matrix by a random $d \times n$

matrix, and show that by reducing the randomly projected lattice we get useful information for the initial lattice with very high probability. This resembles the famous Johnson-Lindenstrauss theorem [15], which shows that one can randomly map N vectors in a $O(\log N)$-dimensional space without modifying significantly the pairwise distances between the vectors. We cannot directly apply such a method since we do not consider the input vectors solely, but their infinitely many integer linear combinations (i.e., in our case N would be infinite). Moreover, we need to keep the Euclidean structure of the initially spanned vector space. In particular, we do not decrease the degree of our lattice below its dimension.

In this paper we consider two random projections. In both models each row of the projection matrix are chosen independently with a common distribution μ_n. In the first model, called the Gaussian model, $\nu_n = \mathcal{N}(0, I_n)$, the standard normal distribution. In other words, each entry of the projection matrix is sampled independently with the standard normal distribution $\mathcal{N}(0, 1)$. These random matrices have been studied extensively and we will rely on a result about their condition numbers, due to Chen and Dongarra [9]. In the second random model, called the unit ball model, ν_n is the uniform distribution inside $\mathcal{B}_n(\mathbf{0}, 1)$, the n-dimensional ball of radius 1 that is centered in $\mathbf{0}$. So each row of the projection matrix is sampled uniformly and independently inside $\mathcal{B}_n(\mathbf{0}, 1)$. Such random matrices have been already studied in [13,2,3]. We will rely on some of the results of these papers. Notice that Rouault [21] recently studied the asymptotic behavior of the determinant of the lattice generated by the rows of a rectangular random matrix with both distributions that we consider here.

All the proofs in this paper are done in continuous random models, i.e. entries of our random matrices are real numbers, which is unsuitable to devise an algorithm. In practice, random matrices are generated with the associated discretised law. Due to space limitation, we chose to skip these difficulties and to to describe them in the full version of the paper.

We performed tests on our reduction technique. They worked very well for many different classes of random projections, including easily samplable ones (such as entries chosen independently and uniformly in $\{-1, 0, 1\}$). As theoretically predicted, the speed-ups can be made arbitrarily large by increasing the ratio between the lattice degree and the lattice dimension.

Related work. Chen and Storjohann [10] introduced in 2005 a probabilistic technique to compute a reduced basis of a lattice given by a generating family: one is given more vectors than the lattice dimension. Our work can be seen as dual to theirs. We deal with vertically rectangular matrices by multiplying them on the left by a random matrix, whereas they deal with horizontally rectangular matrices by multiplying them on the right by a random matrix. They use the arithmetic structure of the lattice whereas we consider its geometric embedding. The two techniques may be used together.

Road-map of the paper. In Section 2, we provide the necessary background on lattices. In Section 3, the core of the paper, we describe our randomized algorithm and perform its complexity analysis. Section 4 is devoted to show its correctness with two different sources of random matrices. Finally, in Section 5, we describe our experiments.

Notations. All costs are given for the bit-complexity model and we assume that we have a perfect source of random bits. We use only naive arithmetic and naive linear algebra. The results may be improved by using fast arithmetic and fast linear algebra. We let $\mathcal{B}_n(\boldsymbol{a}, R)$ denote the n-dimensional ball of radius R centered in $\boldsymbol{a}$. If B is a matrix, we denote by $L(B)$ the lattice spanned by its columns. We denote by $\|B\|_2$ the matrix norm induced by the Euclidean norm, also called the spectral. The maximum of the absolute values of B's entries is the usual max norm denoted by $\|B\|$.

2 Some Reminders on Lattices

We refer to [8] and [11] for comprehensive introductions to lattices and their computational aspects. We give below only the material that is necessary to the description and proof of our probabilistic reduction technique.

Let $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d \in \mathbb{R}^n$ be linearly independent vectors. Their *Gram-Schmidt orthogonalization* is defined as follows: the vector $\boldsymbol{b}_i^*$ is the component of the vector $\boldsymbol{b}_i$ which is orthogonal to the linear span of the vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i-1}$. We have $\boldsymbol{b}_i^* = \boldsymbol{b}_i - \sum_{j=1}^{i-1} \frac{r_{j,i}}{r_{j,j}} \boldsymbol{b}_j^*$ where $r_{j,i} = \frac{\langle \boldsymbol{b}_i, \boldsymbol{b}_j^* \rangle}{\|\boldsymbol{b}_j^*\|}$. If B is a full-rank $n \times d$ matrix, its *QR-factorization* is the unique pair of matrices (Q, R) such that $B = Q \cdot R$, Q is an $n \times d$ matrix made of orthonormal column vectors and R is an upper triangular $d \times d$ matrix with positive diagonal coefficients. The Gram-Schmidt orthogonalization and the QR-factorization of the matrix made of the $\boldsymbol{b}_i$'s are closely related: the i-th column of Q is $\frac{\boldsymbol{b}_i^*}{\|\boldsymbol{b}_i^*\|}$ and the matrix R is made of the $r_{i,j}$'s.

Let $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$ and $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_d$ be two bases of the same lattice. If B and C are the matrices whose columns are the $\boldsymbol{b}_i$'s and $\boldsymbol{c}_i$'s, then there exists a $d \times d$ integer matrix T of determinant ± 1 such that $B = C \cdot T$. Such a matrix is called *unimodular*. Moreover, if two matrices can be obtained one another by unimodular matrices, their columns span the same lattice. Let L be a lattice. The length of any shortest non-zero vector is called the lattice *minimum* and denoted by $\lambda(L)$.

Consider the $\boldsymbol{b}_i$'s as a basis of a lattice L. The *determinant* of L is defined by $\det L = \prod_{i=1}^d \|\boldsymbol{b}_i^*\|$. This does not depend on the choice of the basis. Hadamard's inequality gives that $\det L \leq \prod_{i=1}^d \|\boldsymbol{b}_i\|$. Let $\delta \in (1/4, 1]$ and $\eta \in [1/2, \sqrt{\delta})$. The $\boldsymbol{b}_i$'s are said (δ, η)-*LLL-reduced* if for any $i < j$, we have $|r_{i,j}| \leq \eta \cdot r_{i,i}$, and for any i, we have $\delta \cdot r_{i-1,i-1}^2 \leq r_{i,i}^2 + r_{i-1,i}^2$. When introduced in [17], LLL-reduction referred to the pair $(3/4, 1/2)$. The vectors of a LLL-reduced basis are relatively short. In particular, we have $\|\boldsymbol{b}_1\| \leq (\delta - \eta^2)^{-\frac{d-1}{4}} (\det L)^{\frac{1}{d}}$ and $\prod_{i=1}^d \|\boldsymbol{b}_i\| \leq (\delta - \eta^2)^{-\frac{d(d-1)}{4}} (\det L)$. We refer to [18] for a proof of this fact and for the cost of the algorithm mentioned in the following theorem. The property on the unimodular transformation matrix is classical and a proof can be found in [16].

Theorem 1. *Let $\eta \in (1/2, 1)$ and $\delta \in (\eta^2, 1)$. There exists an algorithm such that when given as input any linearly independent vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d \in \mathbb{Z}^n$ it computes a (δ, η)-LLL-reduced basis of the lattice they span in time $O(d^4 n(d + \log B) \log B)$, where $B = \max_i \|\boldsymbol{b}_i\|$. Furthermore, the bit-lengths of the entries of the transformation matrix are bounded by $O(d \log B)$.*

3 Probabilistic Reduction of Rectangular Lattices

3.1 High-Level Description of the Algorithm

We are given an $n \times d$ integer matrix B and try to find a small integer linear combination of its columns. Instead of applying an LLL-type algorithm directly, we apply a random $d \times n$ dimensional projection P to the matrix and LLL-reduce the $d \times d$ projected matrix $B' = P \cdot B$. By doing so, we decrease the cost with respect to n. We wish that with high probability the unimodular transformation T obtained by LLL-reducing B' somehow reduces B as well. Figure 1 sums up the general method. The top arrow is computationally expensive and is approximately and probabilistically simulated by the succession of plain arrows, that are cheaper. The main result of the paper is the theorem following the description of the algorithm.

$$\begin{tikzcd}
B \arrow[r, dashed, "\text{Direct LLL}"'] \arrow[d] & C = B \cdot T \\
B' = P \cdot B \arrow[r, "\text{LLL}"] & C' = B' \cdot T \arrow[u]
\end{tikzcd}$$

Fig. 1. High-level description of the algorithm of Figure 2

Input: A lattice basis $B = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d) \in \mathbb{Z}^{n \times d}$.
Output: Another basis of the same lattice, hopefully made of short vectors.
Parameters: (δ, η) such that $\eta \in (1/2, 1)$ and $\delta \in (\eta^2, 1)$.
1. Generate a random $d \times n$ matrix P with a fixed distribution.
2. Compute $B' = P \cdot B$.
3. Compute $C' = \text{LLL}_{\delta,\eta}(B')$.
4. Compute $T = (B')^{-1} \cdot C'$.
5. Return $B \cdot T$.

Fig. 2. Probabilistic reduction of a rectangular lattice

Theorem 2. *Let $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d) \in \mathbb{Z}^{n \times d}$ be a basis of a lattice L with $B = \max \|\boldsymbol{b}_i\|$. The algorithm of Figure 2 will compute a basis $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_d)$ of L with the expected running time:*

$$O\left(d^5 \log nB(d+\log nB) + d^2 n \log nB(\log nB + d \log\log nB)\right).$$

If the entries of the random matrix P are independent Gaussian random variables, then for all $x<1$ then with probability greater than $1-x$,

1. *The vector $\boldsymbol{c}_1$ satisfies $\|\boldsymbol{c}_1\| \leq \frac{2^8 d^2}{x^3}(\delta-\eta^2)^{-\frac{d-1}{4}} \cdot (\det L)^{\frac{1}{d}}$.*
2. *The basis $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_d)$ satisfies $\prod_{i\leq d} \|\boldsymbol{c}_i\| \leq \left(\frac{2^8 d^2}{x^3}(\delta-\eta^2)^{-\frac{d-1}{4}}\right)^d \cdot (\det L)$.*

If the rows of the random matrix P are independent random vectors each one picked up uniformly inside the n dimensional unit ball then for any d, there exists $n_0(d)$ such that for any $n \geq n_0(d)$, with probability greater than $1-2^{-d}$,

1. *The vector $\boldsymbol{c}_1$ satisfies $\|\boldsymbol{c}_1\| \leq 2^{4d}(\det L)^{1/d}$.*
2. *The basis $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_d)$ satisfies $\prod_{i\leq d} \|\boldsymbol{c}_i\| \leq 2^{4d^2}(\det L)$.*

Notice that one can take $x = 2^{-d}$ and obtain that with probability exponentially close to 1 the output still satisfies properties similar to what would have been returned by LLL. On both model the length of the first vector may also be expressed as an approximation of the first minimum of the lattice by a factor similar to what would have been returned by LLL. Subsection 3.2 proves the complexity statement and Section 4 the correctness in the continuous models.

3.2 Complexity Analysis

We now prove the complexity statements of Theorem 2. We assume the reader is familiar with the Chinese Remainder Theorem (CRT for short). We refer to [14] for an introduction to the CRT.

From the previous subsection, we know that Step 1 of the algorithm of Figure 2 costs $O(dn \log n)$ bit operations. Step 2 is a multiplication of a $d \times n$ matrix whose entries have length $O(\log n)$ with an $n \times d$ matrix whose entries have length $O(\log B)$. The entries of the $d \times d$ matrix B' have length $O(\log nB)$. The matrix multiplication is performed with the CRT. One takes $O\left(\frac{\log nB}{\log\log nB}\right)$ prime numbers, each of them of length $O(\log\log nB)$. The construction of the primes is computationally negligible. The matrix multiplications modulo the primes cost $O(d^2 n \log nB \log\log nB)$. The conversions of the input matrices into matrices modulo the primes cost $O(dn \log^2 nB)$, whereas the conversion of the output matrices modulo the primes into the integer matrix B' costs $O(d^2 \log^2 nB)$. Theorem 1 gives us that Step 3 costs $O\left(d^5 \log nB(d+\log nB)\right)$. At Step 4, we use again the CRT. Thanks to Theorem 1, we know that the entries of the matrix T have length $O(d \log nB)$. By an analysis similar to the one developed for Step 2, we get that the cost is bounded by $O\left(d^4 \log nB(\log d + \log\log nB) + d^4 \log^2 nB\right)$. At Step 5, we have to multiply an $n \times d$ matrix whose entries have length $O(\log B)$ with a $d \times d$ matrix whose entries have length $O(d \log nB)$. We split each entry of the matrix T into d blocks of roughly $\Theta(\log nB)$ bits, which gives rise to d matrices of dimensions $d \times d$ and whose entries have length $O(\log nB)$. We thus have d

balanced matrix multiplications to perform. For each of them we use the CRT. The overall bit-cost of this step is $O\left(d^3 n \log nB \log\log nB + d^2 n \log^2 nB\right)$. This concludes the proof for the bit-complexity bound of the algorithm of Figure 2 claimed by Theorem 2.

4 Probabilistic Correctness in Two Continuous Models

We consider an input basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d)$ given by an $n \times d$ matrix B. Let $B = Q_B R_B$ be its QR-factorization. Let P be a $d \times n$ random matrix, either from the Gaussian model or from the unit ball model. Let $B' = P \cdot B$ and P' the $d \times d$ matrix $P \cdot Q_B$. We are to show that, with high probability, if an integer linear combination of the columns of $B' = P' R_B$ is a short vector of the lattice $L(B')$, then the same combination of columns of B will be a short vector in $L(B)$. Let $\boldsymbol{c}' \in L(B')$ be defined by $\boldsymbol{c}' = B'\boldsymbol{x} = P' R_B \boldsymbol{x}$, with $\boldsymbol{x} \in \mathbb{Z}^d$. Let $\boldsymbol{c}$ be defined by the same linear combination of the $\boldsymbol{b}_i$'s: $\boldsymbol{c} = B\boldsymbol{x} = Q_B R_B \boldsymbol{x}$.

Our goal is to compare the ratios $\frac{\|\boldsymbol{c}'\|}{(\det L(B'))^{1/d}}$ and $\frac{\|\boldsymbol{c}\|}{(\det L(B))^{1/d}}$. Lemma 1 provides an upper bound to $\det L(B')/\det L(B)$ which holds with high probability. Moreover if $\boldsymbol{c}' \in L(B')$ is the first vector of the basis output by LLL, then $\|\boldsymbol{c}\| \leq 2^{O(d)} (\det L(B'))^{\frac{1}{d}}$. To compare $\|\boldsymbol{c}'\|$ and $\|\boldsymbol{c}\|$, we proceed as follows.

Since the columns of Q_B are orthonormal, we have $\|\boldsymbol{c}\| = \|R_B \boldsymbol{x}\|$. We get $\|\boldsymbol{c}\| = \|(P')^{-1}\boldsymbol{c}'\| \leq \left\|(P')^{-1}\right\|_2 \cdot \|\boldsymbol{c}'\|$. Lemma 3 provides an upper bound to $\left\|(P')^{-1}\right\|_2$ which also holds with high probability in the Gaussian model.

Similarly, if $(\boldsymbol{c}'_1, \ldots, \boldsymbol{c}'_d)$ is an LLL-reduced basis of $L(B')$, then $\prod_{i \leq d} \|\boldsymbol{c}'_i\| \leq 2^{O(d^2)} \det L(B')$. If $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_d)$ is the basis of $L(B)$ where any $\boldsymbol{c}_i$ is expressed in terms of the input basis B with the same integer linear combination than $\boldsymbol{c}'_i$ in terms of B', then: $\prod_{i=1}^d \|\boldsymbol{c}_i\| = \prod_{i=1}^d \|(P')^{-1}\boldsymbol{c}'_i\| \leq \left\|(P')^{-1}\right\|_2^d \cdot \prod_{i=1}^d \|\boldsymbol{c}'_i\|$.

To achieve computations in the unit ball model, we decompose once more the matrix P': let $P' = R_{P'} Q_{P'}$ be the transpose of the QR-decomposition of $(P')^t$. Since the rows of $Q_{P'}$ are orthonormal, we have $\|\boldsymbol{c}\| = \|Q_{P'} R_B \boldsymbol{x}\|$. We get $\|\boldsymbol{c}\| = \|(R_{P'})^{-1}\boldsymbol{y}\| \leq d\left\|(R_{P'})^{-1}\right\| \cdot \|\boldsymbol{c}'\|$. Analogously to the previous case, Lemma 5 provides an upper bound to $\left\|(R_{P'})^{-1}\right\|$ in the unit ball model. There is also an analogous upper bound for $\prod_{i=1}^d \|\boldsymbol{c}_i\|$ that is $d^d \left\|(R_{P'})^{-1}\right\|^d \prod_{i=1}^d \|\boldsymbol{c}_i\|$.

Notice that in Theorem 2, one could also compare the first vector output by our algorithm with the first minimum of the lattice (as it is usually done in the LLL case): we use the facts that $|\boldsymbol{c}_1\| \leq \|(P')^{-1}\|_2 \cdot \|\boldsymbol{c}'_1\|$, $\|\boldsymbol{c}'_1\| \leq 2^{O(d)} \cdot \lambda(L'(B))$ and $\lambda(L'(B)) \leq \|P'\|_2 \cdot \lambda(L(B))$. For the last inequality, consider $\boldsymbol{s} \in \mathbb{Z}^d$ such that $\|B\boldsymbol{s}\| = \lambda(L(B))$. For the same reasons as above, $\|B'\boldsymbol{s}\| \leq \|P'\|_2 \cdot \|B\boldsymbol{s}\|$. It now suffices to see that $\lambda(L(B')) \leq \|B'\boldsymbol{s}\|$.

Lemma 1. *For any $\lambda > 1$, the following holds with probability at least $1 - 1/\lambda^2$:*

(i) In the Gaussian model, $(\det L(B'))^2 \leq d^d \cdot (1 + 3\lambda) \cdot (\det L(B))^2$.
(ii) In the unit ball model, $(\det L(B'))^2 \leq \frac{d!}{(n+2)^d} \cdot (1 + 2d\lambda) \cdot (\det L(B))^2$.

Proof. We have $B' = P \cdot B = P \cdot Q_B \cdot R_B$, which gives that $\det L(B') = \det(P \cdot Q_B) \det R_B = \det(P \cdot Q_B) \det L(B)$. It thus suffices to focus on the determinant of the $d \times d$ matrix $P' = P \cdot Q_B$.

Notice first that the matrix Q_B can be extended to an $n \times n$ orthogonal matrix $Q'_B = (Q_B|\cdot)$. We are interested in $P \cdot Q_B$, i.e., the $d \times d$ left sub-matrix of $P \cdot Q'_B$. Since the both distributions of P that we consider are invariant under right multiplication by an orthonormal matrix, the random matrices P and $P \cdot Q'_B$ follow the same distribution. The distribution of $P \cdot Q_B$ is thus the same as the distribution of the left $d \times d$ sub-matrix of P, denoted by P_l.
Proof of (i). The random matrix P is Gaussian. Let the rows of the $d \times d$ left sub-matrix of P be denoted by $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_d$. Thanks to Hadamard's inequality, we have $\det P' \overset{(d)}{=} \det P_l \leq \prod_{i=1}^{d} \|\boldsymbol{p}_i\|$. Let X be $\prod_{i=1}^{d} \|\boldsymbol{p}_i\|$.

Any $\|\boldsymbol{p}_i\|^2$ is the sum of d squared independent Gaussians. Thus $\mathbb{E}(\|\boldsymbol{p}_i\|^2) = d$ and $\mathbb{E}(\|\boldsymbol{p}_i\|^4) = d(d+2)$. Since they are independent, one gets:

$$\mathbb{E}(X^2) = d^d \quad \text{and} \quad \sigma(X^2) = \mathbb{E}(X^2) \cdot \sqrt{f(d)},$$

where $f(d) = \left(\frac{d+2}{d}\right)^d - 1 \leq 9$. Chebyshev's inequality gives that for $\lambda > 0$:

$$\mathbb{P}\left\{X^2 - \mathbb{E}(X^2) > 3\lambda \mathbb{E}(X^2)\right\} \leq 1/\lambda^2.$$

Proof of (ii). Now P is distributed under the unit ball model. Let H be a d-dimensional linear subspace. Consider the distribution of the orthogonal projections of the rows of P onto H. Since the distribution of the rows of P is invariant under rotation, the distribution of their orthogonal projections is the same no matter onto which subspace H the projection is performed. Let us pick up $n-d$ additional vectors $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{n-d}$ in the n-dimensional unit ball and let H be the orthogonal coset of the (almost surely $(n-d)$-dimensional) space spanned by these additional vectors: $H =< \boldsymbol{p}_1, \ldots, \boldsymbol{p}_{n-d} >^{\perp}$. Let the rows of P be denoted by $\boldsymbol{p}_{n-d+1}, \ldots, \boldsymbol{p}_n$. Let us denote by $\boldsymbol{p}_1^*, \ldots, \boldsymbol{p}_n^*$ the Gram-Schmidt orthogonalization of the random vectors $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n$. We then have $\det(P \cdot Q_B) = \prod_{i=n-d+1}^{n} \|\boldsymbol{p}_i^*\|$. Let X be the random variable corresponding to $\det(P \cdot Q)$. It is proved in [13] that the $\|\boldsymbol{p}_i^*\|^2$'s are independent random variables and that their distribution is given by $\|\boldsymbol{p}_i^*\|^2 \overset{(d)}{=} \beta\left(\frac{n-i+1}{2}, \frac{i+1}{2}\right)$. The Beta law is classical in probability theory and its moments are well known:

$$\mathbb{E}(\|\boldsymbol{p}_i^*\|^2) = \tfrac{n-i+1}{n+2} \quad \text{and} \quad \mathbb{E}(\|\boldsymbol{p}_i^*\|^4) = \tfrac{(n-i+1)(n-i+3)}{(n+4)(n+2)}.$$

Then the independence of $\|\boldsymbol{p}_i^*\|^2$'s leads to:

$$\mathbb{E}(X^2) = d!/(n+2)^d \quad \text{and} \quad \sigma^2(X^2) = \mathbb{E}(X^2) \cdot \sqrt{f(d,n)},$$

where $f(d,n) = \frac{(d+1)(d+2)}{2}\left(\frac{n+2}{n+4}\right)^d - 1$. By routine computation, one sees that $\sqrt{f(d,n)} \leq 2d$ and conclude thanks to Bienaymé's inequality. □

4.1 Probabilistic Correctness in the Gaussian Model

The correctness claims of Theorem 2 derive from Lemmas 1 and 3. To bound the quantity $\|(P')^{-1}\|$, we use the following result on the condition number of a Gaussian random matrix.

Lemma 2 ([9]). *Let κ be the condition number of the matrix P', i.e., $\|P'\| \cdot \|(P')^{-1}\|$. Then for any $\lambda \geq 1$, the probability that $\kappa > \lambda d$ is smaller than $4/\lambda$.*

The last ingredient to the proof of correctness of theorem 2 is the following.

Lemma 3. *Let $t < 1$. Then $\left\|(P')^{-1}\right\| \leq 32d/t^2$ holds with probability greater than $1 - t$.*

Proof. Let $x < 1/2$. We upper-bound by 1 the density function of the first entry of P'. So with probability greater than $1 - 2x$, we have $\|P'\|_2 \geq \|P'\|_2 \geq x$. By using Lemma 2, we obtain that with probability greater than $1 - 2x - 4/\lambda$ we have $\|(P')^{-1}\|_2 \leq \lambda d/x$. Setting $x = t/4$ and $\lambda = 8/t$ provides the result. □

By using Lemmas 1 and 3, we see that, with probability greater than $1-t-1/\lambda^2$, the first vector computed by the algorithm of Figure 2 satisfies:

$$\|\boldsymbol{c}_1\| \leq (\delta - \eta^2)^{-\frac{d-1}{4}} \frac{32 \cdot d^{\frac{3}{2}} (1 + 3\lambda)^{\frac{1}{2d}}}{t^2} \cdot (\det L(B))^{\frac{1}{d}}.$$

By choosing $\lambda = \sqrt{2/x}$ and $t = x/2$, we obtain the result claimed in Theorem 2.

4.2 Probabilistic Correctness in the Unit Ball Model

The correctness claims of Theorem 2 derive from Lemmas 1 and 5.

Lemma 4. *Suppose that $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n$ are n vectors chosen independently and uniformly in the n-dimensional unit ball. Then for any $d \leq n$ and any $v < \frac{1}{4\sqrt{n}}$:*

$$\mathbb{P}\{\min_{1 \leq k \leq d} \|\boldsymbol{p}^*_{n-d+k}\| \leq v\} \leq 4\sqrt{n}v.$$

Proof. Let $\ell_i = \|\boldsymbol{p}^*_i\|$. The distributions of the ℓ_i's are given by [13]:

$$\mathbb{P}[l_{n-d+k} \leq v] = \frac{2}{B\left(\frac{d-k+1}{2}, \frac{n-d+k+1}{2}\right)} \int_0^v u^{d-k} (1 - u^2)^{\frac{n-d+k-1}{2}} du.$$

Since $1 - u^2 \leq 1$, the integral smaller than v^{d-k+1}. Rewriting the denominator in terms of the Gamma function, we get $\mathbb{P}[\ell_{n-d+k} \leq v] \leq \frac{2\ \Gamma\left(\frac{n+2}{2}\right)\ v^{d-k+1}}{\Gamma\left(\frac{d-k+1}{2}\right)\Gamma\left(\frac{n-d+k+1}{2}\right)}$. Using classical properties of the Gamma function, we obtain

$$\mathbb{P}[\ell_{n-d+k} \leq v] \leq 2\left(\frac{nv^2}{2}\right)^{\frac{d-k+1}{2}} \text{ and } \mathbb{P}[\min_{1 \leq k \leq d} \ell_{n-d+k} \leq v] \leq 2\sum_{k=1}^{d}\left(\frac{n\ v^2}{2}\right)^{\frac{d-k+1}{2}}.$$

Finally, if $nv^2 \leq 1/2$, we have $\mathbb{P}\left[\min_{1 \leq k \leq d} \ell_{n-d+k} \leq v\right] \leq 4\sqrt{n}v$. □

Lemma 5. *Let P be a random matrix chosen as previously. Let $P = RQ$ be the transpose of the QR-decomposition of P^t. Let u and v be two reals satisfying $u < 1\sqrt{3}$ and $v < \frac{1}{4\sqrt{n}}$. For any d there exists n_1 such that for all $n \geq n_1(d,u)$, with probability greater than $1 - d^2(\frac{u^2}{1+u^2})^d - 4\sqrt{n}v$, we have:*

$$\|R^{-1}\| \leq \frac{1}{v}\ (1+\frac{1}{u})^d.$$

Proof (Sketch). First, notice that as explained in the proof of Lemma 1, the rows of $P' = P \cdot Q_B$ have the same distributions as the projections $\boldsymbol{p}^*_{n-d+1}, \boldsymbol{p}_{n-d+2}[n-d+1], \ldots, \boldsymbol{p}_n[n-d+1]$ of n vectors $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n$ chosen independently and uniformly in $\mathcal{B}_n(\mathbf{0}, 1)$ in the orthogonal of the span of the $n-d$ first ones. Let us denote ℓ_i the norm of $\boldsymbol{p}^*_i$. The proof, available in the full version, the previous lemma and classical bounds on the Gamma and Beta functions together with the following tools:

– an asymptotic equivalent for $\mathbb{P}[\ell_{n+j}/\ell_{n+i} < v]$ when n grows to infinity and i and j are two constants. This is available in [2](using the Laplace method for evaluating integrals asymptotically)
– an explicit expression of the coefficients of R_P^{-1} as a function of the coefficients $r_{i,j}$ (using the fact that the matrix R_P is lower triangular and so is R_P^{-1} as well) □

By using Lemmas 1 and 5 after routine computations we see that, with probability greater than $1 - 4\sqrt{n}v - d^2(\frac{u^2}{1+u^2})^d - \lambda^{-2}$, the first vector computed by the algorithm of Figure 2 satisfies:

$$\|\boldsymbol{c}_1\| \leq (\delta - \eta^2)^{-\frac{d-1}{4}} \cdot v \cdot \left(1 + \frac{1}{u}\right)^{d-1} \frac{4 \cdot d^{\frac{3}{2}} \cdot \lambda^{\frac{1}{2d}}}{\sqrt{n+2}} \ (\det L(B))^{\frac{1}{d}}.$$

Finally we choose $\lambda = 2^{d/2}$, $u = \sqrt{1/8}$, $v = 1/(2^d\sqrt{n})$.

5 Experimental Data

In this section, we report experiments supporting the validity of our method. The experiments are very promising in the sense that the random projection technique seems to work with a wide range of random matrices and seems to perform better than what we proved. Indeed, the output bases are not only made of vectors of small lengths, but LLL terminates very quickly given them as input.

The experiments were performed using Magma [5] V2.14 on an AMD Opteron 2.40GHz. Each figure corresponds to an average over at least ten samples. We used the LLL routine with the default options ($\delta = 0.75, \eta = 0.51$). Magma's LLL is based on the floating-point L^2 algorithm [18]. The Magma code corresponding to our experiments is available under the GPL at: `http://perso.ens-lyon.fr/damien.stehle/DIMREDUCTION.html`. We considered the following families of random projections.

– $\mathcal{R}_1(N)$: each vector is sampled independently in the sphere $\mathcal{B}_n\left(\mathbf{0}, 10^N\right)$. The computations are performed with decimal precision N. The sampling would be uniform if the computations were performed with infinite precision.
– $\mathcal{R}_2(N)$: each entry is Gaussian variate approximated to decimal precision N.
– $\mathcal{R}_3(N)$: each entry is taken uniformly and independently in $\mathbb{Z} \cap \left[-2^N, 2^N\right)$.
– $\mathcal{R}_4$: each matrix entry is taken uniformly and independently in $\{-1, 1\}$.

The matrices to be reduced are generated in the following way. We first create a $d \times d$ random matrix of the following shape:

$$\begin{pmatrix} x_1 & x_2 & \dots & x_d \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix},$$

where the x_i's are chosen uniformly and independently in $[0, B]$ for some fixed B. When B is large enough, the columns form lattice bases that are far from being reduced. To obtain $n \times d$ lattice bases, we multiply them by matrices sampled from $\mathcal{R}_3(100)$. This provides rectangular bases that are far from being reduced with large and balanced entries. We tested our technique with varying parameters d, n and B and for the classes of random projections described above. We also measured the time LLL takes on the output basis. We compared our technique with the direct LLL approach and with the Gram matrix approach described in the introduction (LLL-reducing the Gram matrix and applying the computed transformation to the input basis). We also compared the lengths of the first vectors of the outputs. The results are described in Figures 3 and 4.

d	20	30	40	50	20	30	40	50
Direct LLL	0.62	8.47	13.6	23.9	1.30	15.8	92.7	341.0
Gram-based approach	0.40	2.26	8.41	25.3	0.52	3.70	16.3	70.7
Random projection approach	0.22	1.19	4.20	13.1	0.25	1.42	5.19	24.8
Direct LLL on the output basis	0.01	0.03	0.07	0.10	0.02	0.09	0.41	1.22

Fig. 3. Timings in seconds of the different LLL approaches for rectangular lattices, when the random matrix is chosen from $\mathcal{R}_4$ and $n = 5d, B = 2^{100 \cdot d}$, (first four columns) and $n = d^2/2, B = 2^{100 \cdot d}$ (last four columns)

Figure 3 shows that the random projection technique can be significantly faster than the direct technique, in particular when n is much larger than d, even if one includes the running-time of LLL on the output basis. Figure 4 shows that the output quality is similar to that of the direct LLL approach. The vector found by the random projection method is most often longer than the one computed by the direct LLL approach, but the ratio remains small. The technique seems to provide reasonably short vectors for all the afore-mentioned families of projections.

d	10	20	30	40	50
$\mathcal{R}_1(100)$	2.34/0.99	3.24/1.06	2.95/1.18	3.90/0.99	5.55/0.88
$\mathcal{R}_2(100)$	3.04/1.02	12.9/1.00	4.13/0.98	4.57/1.00	4.19/0.94
$\mathcal{R}_2(1000)$	3.02/1.04	3.07/0.87	4.40/1.12	5.55/1.16	5.02/1.07
$\mathcal{R}_3(3)$	3.54/1.03	6.67/1.04	3.10/1.02	6.52/0.98	6.21/0.95
$\mathcal{R}_3(10)$	2.98/0.96	2.97/0.99	4.37/1.09	5.58/1.03	3.70/0.99
$\mathcal{R}_4$	3.91/0.96	3.73/1.06	7.00/1.00	4.20/1.03	3.49/1.00

Fig. 4. Ratios between the lengths of the first output vectors after the random projection technique (respectively after LLL on the output basis) and after the direct LLL approach (left of each entry, respectively right of each entry), with $n = 3d$ and $B = 2^{100 \cdot d}$

Acknowledgments. We are grateful to Richard Brent, Philippe Flajolet, Guillaume Hanrot, Luis Pardo, Brigitte Vallée and Gilles Villard for helpful discussions. This work was partially funded by the LaRedA project of the Agence Nationale de la Recherche. It was initiated while the first author was hosted within the computer science laboratory of the University of Paris 7 (LIAFA) and completed while the second author was hosted within the Magma group at the University of Sydney.

References

1. Ajtai, M., Dwork, C.: A public-key cryptosystem with worst-case/average-case equivalence. In: Proc. of STOC 1997, pp. 284–293. ACM, New York (1997)
2. Akhavi, A.: Random lattices, threshold phenomena and efficient reduction algorithms. TCS 287(2), 359–385 (2002)
3. Akhavi, A., Marckert, J.-F., Rouault, A.: On the reduction of a random basis. In: Proc. of the ANALCO 2007, New Orleans, SIAM, Philadelphia (2007)
4. Boneh, D., Durfee, G.: Cryptanalysis of RSA with private key d less than $N^{0.292}$. IEEE Trans Inform Theor 46(4), 233–260 (2000)
5. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I. The user language. JSC 24(3–4), 235–265 (1997)
6. Brisebarre, N., Chevillard, S.: Efficient polynomial L-approximations. In: Proc. of ARITH'18, pp. 169–176. IEEE, Los Alamitos (2007)
7. Brisebarre, N., Hanrot, G.: Floating-point L2-approximations to functions. In: Proc. of ARITH'18, pp. 177–186. IEEE, Los Alamitos (2007)
8. Cassels, J.W.S.: An Introduction to the Geometry of Numbers. Springer, Heidelberg (1971)
9. Chen, Z., Dongarra, J.: Condition numbers of gaussian random matrices. SIAM J Matrix Anal A 27(3), 603–620 (2005)
10. Chen, Z., Storjohann, A.: A BLAS based C library for exact linear algebra on integer matrices. In: Proc. of ISSAC 2005, pp. 92–99. ACM, New York (2005)
11. Cohen, H.: A Course in Computational Algebraic Number Theory. Springer, Heidelberg (1995)
12. Coppersmith, D.: Small solutions to polynomial equations, and low exponent RSA vulnerabilities. J. of Cryptology 10(4), 233–260 (1997)

13. Daudé, H., Vallée, B.: An upper bound on the average number of iterations of the LLL algorithm. TCS 123(1), 95–115 (1994)
14. von zur Gathen, J., Gerhardt, J.: Modern Computer Algebra. Cambridge University Press, Cambridge (2003)
15. Johnson, W.B., Lindenstrauss, J.: Extension of Lipschitz mappings into a Hilbert space. Comm Contemp Math 26, 189–206 (1984)
16. Koy, H., Schnorr, C.P.: Segment LLL-reduction of lattice bases. In: Silverman, J.H. (ed.) CaLC 2001. LNCS, vol. 2146, pp. 67–80. Springer, Heidelberg (2001)
17. Lenstra, A.K., Lenstra Jr., H.W., Lovász, L.: Factoring polynomials with rational coefficients. Math Ann 261, 513–534 (1982)
18. Nguyen, P., Stehlé, D.: Floating-point LLL revisited. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 215–233. Springer, Heidelberg (2005)
19. Nguyen, P., Stern, J.: The two faces of lattices in cryptology. In: Silverman, J.H. (ed.) CaLC 2001. LNCS, vol. 2146, pp. 146–180. Springer, Heidelberg (2001)
20. Odlyzko, A.M., te Riele, H.J.J.: Disproof of Mertens conjecture. J reine angew Math 357, 138–160 (1985)
21. Rouault, A.: Asymptotic behavior of random determinants in the laguerre, gram and jacobi ensembles. Latin American Journal of Probability and Mathematical Statistics (ALEA) 3, 181–230 (2007)
22. Schnorr, C.P.: Progress on LLL and lattice reduction. In: Proc. of the LLL+25 conference (to appear)
23. Stehlé, D.: On the randomness of bits generated by sufficiently smooth functions. In: Hess, F., Pauli, S., Pohst, M. (eds.) ANTS 2006. LNCS, vol. 4076, pp. 257–274. Springer, Heidelberg (2006)
24. Stehlé, D., Lefèvre, V., Zimmermann, P.: Searching worst cases of a one-variable function. IEEE Trans Comp. 54(3), 340–346 (2005)

Sparse Approximate Solutions to Semidefinite Programs

Elad Hazan

IBM Almaden Research Center
650 Harry Road, San Jose, 95120 CA, USA
hazan@us.ibm.com

Abstract. We propose an algorithm for approximately maximizing a concave function over the bounded semi-definite cone, which produces **sparse** solutions. Sparsity for SDP corresponds to low rank matrices, and is a important property for both computational as well as learning theoretic reasons. As an application, building on Aaronson's recent work, we derive a linear time algorithm for Quantum State Tomography.

1 Introduction

In this paper we describe a simple algorithm for approximately solving a special case of Semi-Definite Programs (SDP), in which the goal is to maximize a concave function in the bounded semi-definite cone, which also produces a sparse solution. Our notion of sparsity is not the usual one, i.e. small number of non-zero entries in the solution matrix. Rather, the notion of sparsity in the SDP setting is low rank, coupled with a Cholesky composition representation of the solution. That is, our solutions will be of the form $X = VV^\top \in \mathbb{R}^{n\times n}$, and the solution X is represented by the matrix $V \in \mathbb{R}^{n\times k}$. This notion of sparsity is computationally appealing: the time to compute vector-matrix products with the matrix X, as well as the space required to store it is $O(nk)$.

Our algorithm and its analysis are different from previous work on approximation algorithms for SDP. Unlike previous approximate SDP approaches, the algorithm is **not** based on the multiplicative weights method and its extensions. Rather, the main ingredient is an extension of the Frank-Wolfe [FW56] algorithm [1] for optimizing a single function over the bounded PSD cone. The analysis of this part crucially depends on the dual SDP, and goes beyond the use of online learning techniques which were prevalent in all previous approaches.

The algorithm has the appealing property of proceeding in iterations, such that in each iteration the solution is provably better than the preceding one, and has rank at most one larger. Thus after k iterations the algorithm attains a $\frac{1}{k}$-approximate [2] solution with rank at most k. The previous algorithm of

[1] Whose analysis was recently revisited, simplified and extended in [Cla]. The latter paper inspired this work.

[2] A ε-approximate solution to a SDP over the bounded cone is a PSD matrix with trace equal to one, which satisfies all constraints up to an additive term of ε.

[AHK05], which was based on the multiplicative weights method, also produces sparse solutions, but requires $\Omega(k^2)$ iterations to achieve the same approximation guarantee. Besides elementary computation, in each iteration our algorithm performs at most one eigenvector computation (which can be replaced by two approximate eigenvector computations).

Next, we apply our method to the problem of Quantum State Tomography. In this problem the goal is to approximately learn a mixed quantum state given access to independent measurements. Recently, Aaronson [Aar] showed how a combination of techniques from learning theory and quantum computation can be used to learn an n-bit mixed quantum state with a linear number of measurements (an exponential improvement over previous techniques). We show how to approximately solve Aaronson's proposed SDP in linear time.

This application is particularly suited for our method for the following reasons: First, Aaronson proved that an approximate solution to his SDP guarantees low generalization error. Thus solving the SDP approximately (rather than exactly) is part of the problem statement and suited for approximation techniques. Even more so, an exact solution corresponds to an exact match of the hypothesis to the data, and could lead to "overfitting". Second, a widely accepted principle known as "Occam's Razor" roughly states that simple hypothesis are preferable. Our algorithm generates a simple hypothesis, taking low-rank is a notion of simplicity. Third, the state matrix of an n-cubit state is of size 2^n. For even a small number of qubits the size of the matrices is too large to effective deploy any algorithm with super-linear complexity. This rules out interior point methods. For this reason also, exploiting the sparsity of the input is absolutely necessary.

Finally, we show how to extend the optimization routine over the bounded semi-definite cone in order to approximately solve general SDPs. However, for general SDPs we do not improve the running time over existing techniques. Whereas for optimization over the bounded SDP cone our algorithm produces an ε-approximation in $O(\frac{1}{\varepsilon})$ iterations, which improves upon all existing approximation methods, for general SDPs we need $O(\frac{1}{\varepsilon^2})$ iterations.

1.1 Existence of Sparse Solutions

In the first part of the paper we focus on the problem of maximizing a single concave function over the cone of semi-definite matrices with trace bounded by one. A natural question concerning sparse solutions is what is the minimal rank required of a solution which is "close" to the optimum. Obviously, if the concave objective function happens to be linear, then the problem reduces to an eigenvector computation, and the optimal solution is of rank one. However, this is not the case in general, and even for quadratic functions the optimal solution can have full rank.

It follows from the work of Clarkson [Cla], that in order to attain an ε-additive approximation (in a sense that will be made precise below), the rank of the approximation needs to be at least $\Omega(\frac{1}{\varepsilon})$. However it does not follow from previous work that there even *exists* an ε-approximate solution with such rank.

The best bound we know of is from [AHK05], which implies the existence of an ε-approximate solution with rank $O(\frac{1}{\varepsilon^2})$.

In this paper we give a constructive proof that an ε-approximate solution with rank $O(\frac{1}{\varepsilon})$ exists by giving an efficient algorithm to find it (see Theorem 1 below).

2 Preliminaries

For two matrices $A, B \in \mathbb{C}^{n\times n}$ we denote by $A \bullet B$ the inner product of these matrices when thought of as vectors in $\mathbb{C}^{n^2}$, i.e. $A \bullet B = \sum_{ij} A_{ij} \cdot B_{ij} = \mathbf{Tr}(AB)$. All results in this paper apply to Hermitian matrices, and this generality is important for our application to Quantum State Tomography. However, for simplicity the reader may think of real matrices.

A matrix $A \in \mathbb{C}^{n\times n}$ is positive semi-definite (PSD), denoted by $A \succeq 0$, if and only if all its eigenvalues are non-negative real numbers. We write $A \succeq B$ if the matrix $A - B \succeq 0$ is PSD. In the analysis we require the following basic characterization of PSD matrices.

Fact 2.1 *A matrix X is PSD if and only if $X \bullet V \succeq 0$ for all $V \succeq 0$.*

Consider a general SDP feasibility problem (as is standard, the optimization version is reduced to feasibility by binary search)

$$\begin{aligned} \forall i \in [m]\ .\ A_i \bullet X &\le b_i \\ X &\in \mathcal{P} \end{aligned} \tag{1}$$

We say that a matrix $X \succeq 0$ is ε-approximate solution to (1) if it satisfies $\forall i\ .\ A_i \bullet X - b_i \le \varepsilon$.

Quantum State Tomography. In the problem of quantum state tomography (QST), the goal is to learn an approximate description of a certain quantum state given access to independent measurements of the state. Recently, Aaronson [Aar] showed that a small set of measurements (linear in the number of qubits) can be used to accurately learn a state in this model.

The problem of finding a mixed n-qubit state which approximately agrees with the measurements can be cast as the following optimization problem. The input is m Hermitian matrices $E_1, ..., E_m \in \mathbb{C}^{N\times N}$ (for the QST problem we denote $N = 2^n$) with eigenvalues in $[0, 1]$, and m real numbers $b_1, ..., b_m \in [0, 1]$. The goal is to find a Hermitian positive semidefinite matrix X that satisfies (for a given constant $\eta > 0$)

$$\begin{aligned} \forall i\ .\ |E_i \bullet X - b_i| \le \eta \\ X \succ 0\ ,\ \mathbf{Tr}(X) = 1 \end{aligned} \tag{2}$$

As alternative (and equivalent up to the approximation parameter) formulation, which may even be more interesting for both theoretical reasons as well as applications, is

$$\sum_i (E_i \bullet X - b_i)^2 \leq \eta \tag{3}$$
$$X \succ 0 \ , \ \mathbf{Tr}(X) = 1$$

As Aaronson notes, the above formulation is a convex program with semi-definite constraints, and thus solvable in polynomial time using interior point methods [NN94, Ali95] or the ellipsoid method [GLS88]. However, the exponential size of the measurement matrices quickly render any super-linear method impractical.

3 A Sparse Approximate SDP Solver

Let $\mathcal{P} = \{X \succeq 0 \ , \ \mathbf{Tr}(X) = 1\}$ be the cone of SDP matrices with trace equals one. This convex set is a natural generalization of the simplex, and is the set of all quantum distributions. In this section we consider the the following optimization problem:

$$\max f(X) \tag{4}$$
$$X \in \mathcal{P}$$

Besides an interesting problem by its own right, we show in the next section how to use the algorithm we develop in this section to solve general SDP. The following simple and efficient algorithm always maintains a sparse PSD matrix, rank at most k after k iterations. The algorithm can be viewed as a generalization of the Frank-Wolfe algorithm for optimizing over the simplex, which was recently revisited by Clarkson [Cla].

Algorithm 1. SparseApproxSDP

```
1: Let $f$ be a given concave function, with curvature constant $C_f$ as defined below.
   Initialize $X_1 = v_0 v_0^\top$ for an arbitrary rank one matrix $v_0 v_0^\top$ (with trace one).
2: for $k = 1, ..\infty$ do
3:     Let $\varepsilon_k = \frac{C_f}{k^2}$. Compute $v_k \leftarrow \text{ApproxEV}(\nabla f(X_k), \varepsilon_k)$.
4:     Let $\alpha_k = \frac{1}{k}$.
5:     Set $X_{k+1} = X_k + \alpha_k(v_k v_k^\top - X_k)$.
6: end for
```

The procedure ApproxEV used in algorithm 1 is an approximate eigenvalue solver. It guarantees the following: for a negative semidefinite matrix M and any $\varepsilon > 0$, the vector $x = \text{ApproxEV}(M, \varepsilon)$ satisfies $x^\top M x \geq \lambda_{\max}(M) - \varepsilon$. At this point the reader may think of this procedure as an exact maximum eigenvalue computation. In the end of this section we prove that ApproxEV can be made to have running time which is linear in the number of non-zero entries of M.

A crucial property of the function f which effects the convergence rate is its curvature constant, defined by Clarkson [Cla], as follows.

Definition 1. *Define the curvature constant of f as*

$$C_f \triangleq \sup_{x,z\in\mathcal{P} \quad y=x+\alpha(z-x)} \frac{1}{\alpha^2}[f(x)-f(y)+(y-x)^\top \nabla f(x)]$$

Note that C_f is upper bounded by the largest eigenvalue of the Hessian of $-f$.

Our main performance guarantee is given by the following Theorem. Henceforth let X^* denote the optimal solution to (4).

Theorem 1. *Let C_f be defined as in the following section. Then the iterates X_k of Algorithm 1 satisfy forall $k > 1$*

$$\forall X^* \in \mathcal{P} \ . \ f(X_k) \geq f(X^*) - \frac{4C_f}{k}$$

REMARK: A similar guarantee can be given for α_k chosen greedily by binary search (instead of $\frac{1}{k}$).

Before proving this theorem, we define some notation.

– Denote by
$$z(X) = \max_{\|v\|=1} v^\top \nabla f(X) v = \lambda_{max}(\nabla f(X))$$
the largest eigenvalue of the gradient of f at X.
– Let $w(X) = z(X) + f(X) - X \bullet \nabla f(X)$. As we show in Lemma 1 below, $w(X)$ is the dual objective to optimization problem (4).
– We also denote by $h(X) = \frac{f(X^*)-f(X)}{4C_f}$ the normalized distance to the optimum at X (in value) and by $g(X) = \frac{w(X)-f(X)}{4C_f}$ the duality gap at X.

Lemma 1. *Weak duality:*

$$w(X) \geq w(X^*) \geq f(X^*) \geq f(X)$$

Proof. It suffices to prove that the formulation:

$$\min_{X\in\ \mathbb{S}} w(X)$$

(here $\mathbb{S}$ is the set of all matrices in $\mathbb{R}^{n\times n}$) is the dual of (4). To see that, let's write the Lagrangian relaxation of (4), which is equivalent to $-\min_{X\in\mathcal{P}} -f(X)$ (for this formulation recall Fact 2.1):

$$-\max_{V\succeq 0, z\in\mathbb{R}} \min_{X\in\mathbb{S}} -f(X) - X\bullet V + z(X\bullet I - 1)$$

The optimum is obtained when all derivatives w.r.t x are zero, i.e.

$$-\nabla f(X) - V + zI = 0$$

which implies $V = zI - \nabla f(X)$. There is also the obvious constraint that $0 \preceq V = zI - \nabla f(X)$. Plugging back we get that the optimization problem becomes

$$\begin{aligned}
&-\max_{z\in\mathbb{R}} \min_{X\in\mathbb{S}} -f(X) - X \bullet (zI - \nabla f(X)) + z(X \bullet I - 1) = \\
&-\max_{z\in\mathbb{R}} \min_{X\in\mathbb{S}} -f(X) + X \bullet \nabla f(X) - z = \\
&-\max_{z\in\mathbb{R}, X\in\mathbb{S}} -f(X) + X \bullet \nabla f(X) - z = \\
&\min_{z\in\mathbb{R}, X\in\mathbb{S}} f(X) - X \bullet \nabla f(X) + z
\end{aligned}$$

Subject to $zI \geq \nabla f(X)$, or $z \geq \lambda_{max}\{\nabla f(X)\}$. So w.l.o.g we have $z = z(X) = \lambda_{max}\{\nabla f(X)\}$ and the above becomes $\min_{X\in\mathbb{S}} w(X)$.

In particular, the above implies that $g(X) \geq h(X)$. We can now prove the theorem:

Proof (Proof of Theorem 1). By the definitions above we have

$$v_k^\top \nabla f(X_k) v_k \geq z(X_k) - \varepsilon_k = w(X_k) - f(X_k) + X_k \bullet \nabla f(X_k) - \varepsilon_k$$

Therefore

$$\begin{aligned}
(v_k v_k^\top - X) \bullet \nabla f(X_k) &= v_k^\top \nabla f(X_k) v_k - X \bullet \nabla f(X_k) \\
&\geq w(X_k) - f(X_k) - \varepsilon_k
\end{aligned} \tag{5}$$

Now,

$$\begin{aligned}
f(X_{k+1}) &= f(X_k + \alpha_k(v_k v_k^\top - X_k)) \\
&\geq f(X_k) + \alpha_k(v_k v_k^\top - X_k) \bullet \nabla f(X_k) - \alpha_k^2 C_f \quad \text{by definition of } C_f \\
&\geq f(X_k) + \alpha_k(w(X_k) - f(X_k)) - \alpha_k^2 C_f - \varepsilon_k \quad \text{by (5)} \\
&= f(X_k) + 4C_f g(X_k)\alpha_k - C_f \alpha_k^2 - \varepsilon_k \\
&\leq f(X_k) + 4C_f h(X_k)\alpha_k - C_f \alpha_k^2 - \varepsilon_k
\end{aligned}$$

By definition of $h(X_k)$ and Lemma 1 this implies

$$h(X_{k+1}) \leq h(X_k) - \alpha_k h(X_k) + \frac{1}{4}\alpha_k^2 + \frac{\varepsilon_k}{4C_f}$$

Let $\varepsilon_k = \frac{C_f}{k^2}$.

Now we prove inductively that $h(x_k) \leq \frac{1}{k}$. In the first iteration we assume w.l.o.g that it's true, else we could have taken $\alpha_1 = 1$, and get that $h(X_2) \leq \frac{1}{2}$, now rename k. So by taking $\alpha_k = \frac{1}{k}$ we have

$$\begin{aligned}
h(X_{k+1}) &\leq h(X_k)(1 - \alpha_k) + \frac{1}{4}\alpha_k^2 + \frac{\varepsilon_k}{4C_f} \\
&\leq \frac{1}{k} - \frac{1}{k^2} + \frac{1}{4k^2} + \frac{1}{4k^2} \leq \frac{1}{k}
\end{aligned}$$

3.1 Using Approximate Eigenvector Computations

We now describe how to efficiently implement an eigenvector computation procedure using the Lanczos algorithm with a random start. This was first shown by [AHK05].

Lemma 2. *Given a NSD matrix M with $\tilde{N}$ non-zero entries and eigenvalues in the range $[-C, 0]$, there exists an algorithm which produces in time $\tilde{O}(\frac{\tilde{N}\sqrt{C}}{\sqrt{\varepsilon}})$ a vector x such that*

$$x^\top M x \geq \lambda_{\max}(M) - \varepsilon$$

Proof. We need Theorem 3.2(a) of Kuczyński and Woźniakowski [KW92]:

Lemma 3. *Let $M \in \mathbb{R}^{n \times n}$ be a positive semidefinite matrix with $\tilde{N}$ non-zero entries. Then with high probability, the Lanczos algorithm produces in $O(\frac{\log(n)}{\sqrt{\gamma}})$ iterations a unit vector x such that $\frac{x^T M x}{\lambda_{\max}(M)} \geq 1 - \gamma$.*

Note that each iteration of the Lanczos algorithm takes $\tilde{O}(\tilde{N})$ time. Now let $M_2 = CI + M$. Notice that M_2 is positive semidefinite, and $\lambda_i(M_2) = C + \lambda_i(M)$. We apply Lemma 3 with $\gamma = \frac{\varepsilon}{C}$ to obtain in time $\tilde{O}(\frac{\tilde{N}}{\sqrt{\gamma}})$ a unit vector x such that:

$$\frac{C + x^T M x}{C + \lambda_{\max}(M)} = \frac{x^T M_2 x}{\lambda_{\max}(M_2)} \geq 1 - \frac{\varepsilon}{C}$$

Simplifying this gives the lemma statement.

Combining the pieces together we obtain:

Theorem 2. *Let C_k be a bound on the absolute value of the largest eigenvalue of $\nabla f(X_k)$, $C = \max_k\{C_k\}$ and $\tilde{N}$ be the maximal number of non-zero entries in $\nabla f(X)$. Algorithm 1 returns a δ-approximate solution in time*

$$\tilde{O}\left(\frac{C_f(n + T_{GD})}{\delta} + \frac{\tilde{N}\sqrt{C}C_f^{1.5}}{\delta^2}\right)$$

Where T_{GD} is the time to compute $\nabla f(X_k)$.

Proof. By Theorem 1, we have $f(X_k) \geq f(X^*) - \frac{4C_f}{k}$, hence it suffices to have $k = \frac{4C_f}{\delta}$ iterations. Besides the calls to ApproxEV, the computation required in each iteration is the computation of the gradient (in time T_{GD}) and other elementary computations which can be carried out in time $O(n)$. By lemma 2, the k'th invokation of ApproxEV runs in time $\tilde{O}(\frac{\tilde{N}\sqrt{C_k}}{\sqrt{\varepsilon_k}}) = \tilde{O}(k\tilde{N}\sqrt{C_k/C_f})$. The total running time thus comes to

$$\frac{4C_f}{\delta} \cdot (n + T_{GD}) + \sum_{i=1}^{k} \tilde{O}(\frac{i \cdot \tilde{N}\sqrt{C_i}}{\sqrt{C_f}})$$

4 Solving General SDPs

In this section we apply the sparse SDP solver to approximately solve general a SDP in the form 1. We note that the results in this section do not improve over [AHK05] in terms of running time (whereas for the case of optimizing a single function over the bounded semi-definite cone we do obtain an an improvement in running time), and are given here only to illustrate how similar results can be obtained through a very different analysis. In fact, the algorithm below is almost identical to the one obtained in [AHK05] via Multiplicative Weights techniques.

Algorithm 2. Fast SDP

```
1: Input: set of constraints given by $A_1, ..., A_m \in \mathbb{C}^{N\times N}$ and $b_1, ..., b_m \in \mathbb{R}$. Desired
   accuracy $\varepsilon$. Let $\omega = \max_i\{\lambda_{\max}(A_i)\}$.
2: Let $M = \frac{\log m}{\varepsilon}$
3: Apply algorithm 1 to the following function for $k$ rounds, with $k = \frac{1}{\varepsilon}$.
       $$f(X) = -\frac{1}{M}\log\left(\sum_{i=1}^{m} e^{M\cdot(A_i\bullet X - b_i)}\right)$$
4: if $f(X_k) < -\varepsilon$ return FAIL. Else, return $X_k$.
```

Lemma 4. *A matrix X for which $f(X) \geq -\varepsilon$ is a ε-approximate solution to QST.*

Proof. Consider the function (for $M = \frac{\log m}{\varepsilon}$ and $y \in \mathbb{R}^m$)

$$\Phi(y) = \frac{1}{M}\log(\sum_i e^{M\cdot y_i})$$

It is a well known fact (see [GK94]) that for $M \geq 0$, we have

$$\lambda(x) \leq \Phi(x) \leq \lambda(x) + \frac{\log m}{M}$$

Where

$$\lambda(x) \triangleq \max_i y_i$$

Therefore, if X satisfies $f(X) \geq -\varepsilon$, we have $\Phi(X) \leq \varepsilon$ for $y_i = A_i \bullet X - b_i$. This implies that $\max_i y_i \leq \Phi(x) \leq \varepsilon$ and hence

$$\forall i \;.\; A_i \bullet X - b_i \leq \varepsilon$$

Lemma 5. *The function f above is concave, and its C_f value is bounded by $\frac{\omega^2 \log m}{\varepsilon}$. The value C is bounded by $C \leq \omega$.*

Proof. See [BV04] for a proof that that $-f$ is convex.

Let us now define $z(X) \in \mathbb{R}^{2m}$ as the vector such that $z(X)_i = e^{M\cdot(A_i \bullet X - b_i)}$ for $i \in [m]$. In the following we just say z when there's no ambiguity as for the X in question.

Differentiating $g(X) = -f(X)$ with respect to X we get

$$M \cdot \nabla g(X) = \frac{1}{1^\top z} \sum_{i=1}^{m} z_i \cdot MA_i \tag{6}$$

So we can bound the parameter C by

$$C \leq \lambda_{\max}(\nabla g) = \lambda_{\max}(\frac{1}{1^\top z} \sum_{i=1}^{m} z_i A_i) \leq \omega$$

And taking the second derivative we get

$$M \cdot \nabla^2 g(X) = -\frac{1}{(1^\top z)^2} \sum_{i,j=1}^{m} z_i z_j A_i \bullet A_j \cdot M^2 + \frac{1}{1^\top z} \sum_i z_i A_i \bullet A_i \cdot M^2$$

Hence,

$$\nabla^2 g(X) \preceq M \cdot \frac{1}{1^\top z} \sum_i z_i A_i \bullet A_i \preceq M \cdot \omega^2 I$$

and therefore $\lambda_{\max}(\nabla^2 g(X)) \leq M\omega^2$. Since C_f is bounded by the absolute value of the smallest eigenvalue of $\nabla^2 f$ on $\mathcal{P}$, this gives the Lemma.

Theorem 3. *If SDP (1) is feasible, then Algorithm 2 returns a ε-approximate solution, else it returns FAIL. The algorithm performs $\frac{\omega^2 \log m}{\varepsilon^2}$ approximate eigenvalue computations and has total running time of*

$$\tilde{O}\left(\frac{\omega^2 n}{\varepsilon^2} + \tilde{N} \cdot \frac{\omega^{3.5}}{\varepsilon^{3.5}}\right)$$

Where $\tilde{N}$ is the total number of non-zero entries in $A_1, ..., A_m$.

Proof. For the feasible solution X^*, we have $y_i^* = A_i \bullet X^* - b_i \leq 0$. Hence,

$$f(X^*) = -\frac{1}{M} \log(\sum_i e^{My_i^*}) \geq -\frac{1}{M}\log(m) = -\varepsilon$$

Therefore, we get a $\delta = \varepsilon$ approximation after $\frac{C_f}{\varepsilon} \leq \frac{\omega^2 \log m}{\varepsilon^2}$ iterations, and have $f(X_k) \geq f(X^*) - \varepsilon \geq -2\varepsilon$, which according to Lemma 4 is a 2ε approximation solution to QST.

According to Theorem 2 the total running time is bounded by

$$\tilde{O}\left(\frac{C_f(n + T_{GD})}{\delta} + \frac{\tilde{N}\sqrt{C}C_f^{1.5}}{\delta^2}\right)$$

The gradient of f is (see equation (6))

$$\nabla f(X_k) = -\frac{1}{1^\top z}\sum_{i=1}^{m} z_i \cdot A_i$$

This is a convex combination of the matrices $\{-A_i\}$ according to the distribution z. Note that the distribution z at iteration k, denoted z_k, can be obtained from z_{k-1} in time $\tilde{O}(\tilde{N})$ since $X_k = (1-\alpha_k)X_{k-1} + \alpha_k v_k v_k^\top$. Therefore $T_{GD} = \tilde{O}(\tilde{N})$. Also, in our case $\delta = \varepsilon$, and C_f, C are bounded as in lemma 5.

Plugging these bounds the lemma is obtained.

5 QST in Linear Time

The application to QST is straightforward. To solve formulation (3), apply Theorem 1 to obtain the corollary below. We note that the curvature constant C_f is bounded by the largest eigenvalue of the Hessian, which is bounded by $C_f \leq 2\sum_{i=1}^{m} \lambda_{\max}(E_i)^2$.

Corollary 1. *Algorithm 1 returns a ε-approximate solution to an instance of QST in $\frac{4C_f}{\varepsilon}$ iterations.*

As for formulation (2), we have $2m$ constraints of the form $E_i \bullet X - b_i \leq \eta$ and $b_i - E_i \bullet X \leq \eta$. The parameter ω is bounded by the maximum eigenvalue of E_i, which is just 1. Also recall that for QST the dimension of the matrices E_i is $N = 2^n$. By Theorem 3 we conclude

Corollary 2. *Algorithm 2 returns a ε-approximate solution to an instance of QST in $\frac{\log m}{\varepsilon^2}$ iterations and has total running time of*

$$\tilde{O}\left(\frac{N}{\varepsilon^2} + \frac{\tilde{N}}{\varepsilon^{3.5}}\right)$$

Where $\tilde{N}$ is the total number of non-zero entries in the matrices $E_i, ..., E_m$.

Using (standard) clever implementation techniques, one can remove the dependence on $N = 2^n$ completely, and obtain a running time of

$$\tilde{O}\left(\frac{\tilde{N}}{\varepsilon^{3.5}}\right)$$

References

[Aar] Aaronson, S.: The learnability of quantum states. arXiv:quant-ph/0608142v3

[AHK05] Arora, S., Hazan, E., Kale, S.: Fast algorithms for approximate semide.nite programming using the multiplicative weights update method. In: 46th IEEE FOCS, pp. 339–348 (2005)

[Ali95] Alizadeh, F.: Interior point methods in semidefinite programming with applications to combinatorial optimization. SIAM J. Optim. 5(1), 13–51 (1995)

[BV04] Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, New York, NY, USA (2004)

[Cla] Clarkson, K.L.: Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. In: SODA 2008: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms,

[FW56] Frank, M., Wolfe, P.: An algorithm for quadratic programming. Naval Research Logistics Quarterly 3, 149–154 (1956)

[GK94] Grigoriadis, M.D., Khachiyan, L.G.: Fast approximation schemes for convex programs with many block and coupling constraints. SIAM Journal on Optimization 4, 86–107 (1994)

[GLS88] Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization. Springer, Heidelberg (1988)

[KW92] Kuczyński, J., Woźniakowski, H.: Estimating the largest eigenvalue by the power and lanczos algorithms with a random start. SIAM Journal on Matrix Analysis and Applications 13(4), 1094–1122 (1992)

[NN94] Nesterov, Y., Nemirovskii, A.: Interior Point Polynomial Methods in Convex Programming: Theory and Applications. Society for Industrial and Applied Mathematics, Philadelphia (1994)

On the Facets of Mixed Integer Programs with Two Integer Variables and Two Constraints

Gérard Cornuéjols[1,⋆] and François Margot[2,⋆⋆]

[1] Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA 15213, USA and LIF, Université d' Aix-Marseille, Faculté des Sciences de Luminy, France
gc0v@andrew.cmu.edu

[2] Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA 15213, USA
fmargot@andrew.cmu.edu

Abstract. In this paper we consider an infinite relaxation of the mixed integer linear program with two integer variables and two constraints, and we give a complete characterization of its facets. We then derive an analogous characterization of the facets of the underlying finite integer program.

Keywords. integer programming, intersection cut, group relaxation.

1 Introduction

We consider the mixed 2-integer-variable linear program with two constraints

$$\begin{aligned} &x = f + \textstyle\sum_{j=1}^{k} r^j s_j \\ &x \in \mathbb{Z}^2 \\ &s \in \mathbb{R}^k_+ \end{aligned} \tag{1}$$

where $f \in \mathbb{Q}^2 \setminus \mathbb{Z}^2$, $k \geq 1$, and $r^j \in \mathbb{Q}^2$. Let $R_f(r^1, \ldots, r^k)$ be the convex hull of all vectors $s \in \mathbb{R}^k_+$ such that $f + \sum_{j=1}^{k} r^j s_j$ is integral. $R_f(r^1, \ldots, r^k)$ is a polyhedron (We refer the reader to [11] for standard definitions). Model (1) was considered by Andersen, Louveaux, Weismantel and Wolsey [1]. They showed that the nontrivial facets of $R_f(r^1, \ldots, r^k)$ are necessarily defined by split inequalities or intersection cuts (Balas [2]) arising from triangles or quadrilaterals in $\mathbb{R}^2$. A goal of this paper is to give a converse to the result in [1]: which splits, triangles and quadrilaterals actually define facets of $R_f(r^1, \ldots, r^k)$?

Gomory and Johnson [8] suggested relaxing the k-dimensional space of variables $s = (s_1, \ldots, s_k)$ to an infinite-dimensional space, where the variables s_r are defined for any $r \in \mathbb{Q}^2$. We get the *infinite program with two integer variables and two constraints*

$$\begin{aligned} &x = f + \textstyle\sum r s_r \\ &x \in \mathbb{Z}^2 \\ &s \geq 0 \quad \text{with finite support.} \end{aligned} \tag{2}$$

⋆ Supported by NSF grant CMMI-0653419, ONR grant N00014-97-1-0196 and ANR grant BLAN06-1-138894.

⋆⋆ Supported by ONR grant N00014-97-1-0196.

The vector $s = (s_r)_{r \in \mathbb{Q}^2}$ is said to have *finite support* if $s_r \neq 0$ for a finite number of $r \in \mathbb{Q}^2$. Let R_f be the convex hull of all vectors $s \geq 0$ with finite support such that $f + \sum r s_r$ is integral. Note that the polyhedron $R_f(r^1, \ldots, r^k)$ is the face of R_f obtained by setting $s_r = 0$ for all $r \in \mathbb{Q}^2 \setminus \{r^1, \ldots, r^k\}$. The motivation for working with R_f instead of $R_f(r^1, \ldots, r^k)$ is that it only has one parameter, namely f, and therefore the results are cleaner. Moreover results obtained for R_f can be carried over to the model used in [1].

We say that an inequality is *valid* for R_f (resp. $R_f(r^1, \ldots, r^k)$) if it is satisfied by all vectors in R_f (resp. $R_f(r^1, \ldots, r^k)$). Inequalities $s_i \geq 0$ are called *trivial* valid inequalities. In this paper, we discuss only nontrivial valid inequalities. The solution $s = 0$ is not feasible for R_f. Any valid inequality for R_f that cuts off the vector $s = 0$ is of the form

$$\sum \psi(r) s_r \geq 1 \tag{3}$$

where $\psi : \mathbb{Q}^2 \to \mathbb{R} \cup \{+\infty\}$ and, as above, we only consider vectors s with finite support. To avoid ambiguity, the product $+\infty \cdot 0$ is defined as 0.

Any valid inequality for R_f yields a valid inequality for $R_f(r^1, \ldots, r^k)$ by simply restricting it to the space $r^1, \ldots, r^k$. Furthermore, a full description of the polyhedron $R_f(r^1, \ldots, r^k)$ is obtained from the set of valid inequalities for R_f by adding the constraints $s_r = 0$ for $r \neq r^1, \ldots, r^k$. Therefore we will assume in the remainder that valid inequalities for $R_f(r^1, \ldots, r^k)$ are restrictions of valid inequalities for R_f.

An inequality $\sum \psi(r) s_r \geq 1$ valid for R_f is *minimal* if there is no valid inequality $\sum \psi'(r) s_r \geq 1$ where $\psi' \leq \psi$ and $\psi'(r) < \psi(r)$ for at least one $r \in \mathbb{Q}^2$. We also say that such a function ψ is *minimal.* The following result was proved in [3].

Theorem 1. *A minimal valid function ψ is nonnegative homogeneous piecewise linear and convex. Furthermore, the closure of the set*

$$B_\psi := \{x \in \mathbb{Q}^2 : \ \psi(x - f) \leq 1\}. \tag{4}$$

is a full-dimensional polyhedron with 2, 3 or 4 edges, it contains no integral point in its interior but each edge contains an integral point in its relative interior.

A function ψ is *positively homogeneous* if $\psi(\lambda r) = \lambda \psi(r)$ for all $\lambda \geq 0$. Since ψ is always nonnegative in this paper, we simply say *homogeneous* to mean positively homogeneous. We will also simply say in the *interior* of an edge to mean in the *relative interior* of that edge.

The point f is in B_ψ since $\psi(0) = 0$. When f is in the interior of B_ψ, then ψ is continuous and B_ψ is closed (we call this the *nondegenerate* case), see Figure 1. In this case, the boundary of B_ψ is the set of points $x \in \mathbb{Q}^2$ that satisfy $\psi(x - f) = 1$. Thus, the knowledge of f and of the boundary of B_ψ together with the homogeneity of ψ is enough to compute the value of $\psi(r)$ for any vector $r \in \mathbb{Q}^2 \setminus \{0\}$: If $f + \lambda r$ is a point on the boundary of B_ψ for some $\lambda > 0$, we get that $\psi(r) = 1/\lambda$. Otherwise, if there is no such λ, r is an unbounded direction of

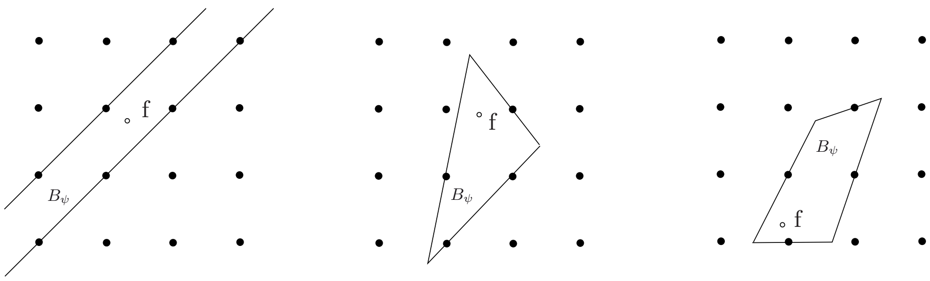


Fig. 1. Representation of B_ψ for nondegenerate cases

B_ψ and $\psi(r) = 0$. We use the graphic representation of B_ψ to describe ψ when possible. The inequalities corresponding to the three cases of Figure 1 will be called *split*, *triangle* and *quadrilateral* inequalities. They are special case of the *intersection cuts* of Balas [2]. Solid lines in Figure 2 give level curves of $\psi(r)$ with values 0 and 1 for the three examples of Figure 1.

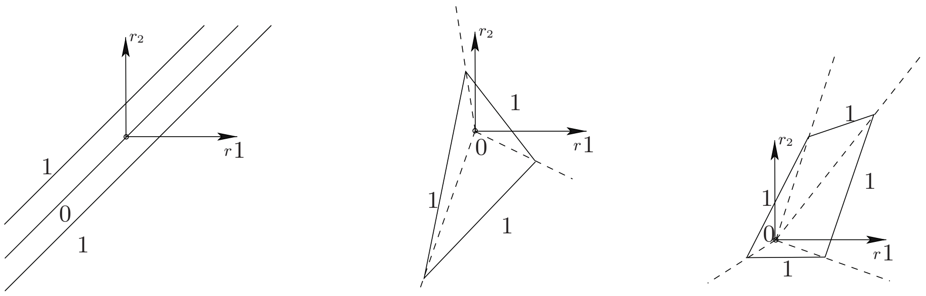


Fig. 2. Level curves of $\psi(r)$ for nondegenerate cases

When f is a vertex of clB_ψ (the closure of B_ψ) or when f lies on one of its edges, ψ is neither continuous nor finite everywhere [3] (*degenerate case*), see Figure 3. In particular, for any direction $r \neq 0$ such that the half-line $L_r = \{x = f + \lambda r \text{ for } \lambda > 0\}$ is outside clB_ψ, we have $\psi(r) = +\infty$. For the directions such that the half-line L_r goes through the interior of clB_ψ, let $f + \lambda r$ be the point where L_r intersects the boundary of clB_ψ; then we get $\psi(r) = 1/\lambda$. Finally, when L_r supports an edge of clB_ψ, let $y = f + \lambda r$ be the first integral point encountered on L_r starting from f and let $x = f + \mu r$ be the first vertex of clB_ψ encountered (if any); if y is encountered first, we get $\psi(r) = 1/\lambda$ and if x is encountered first, we get $\psi(r) = 1/\mu$. There are five different degenerate inequalities, depending of the type of set clB_ψ and the position of f on its faces: *degenerate split*, *vertex-degenerate triangle*, *edge-degenerate triangle*, *vertex-degenerate quadrilateral* and *edge-degenerate quadrilateral* inequalities. Solid lines in Figure 4 give level curves of $\psi(r)$ with values 0 and 1 for the three examples of Figure 3.

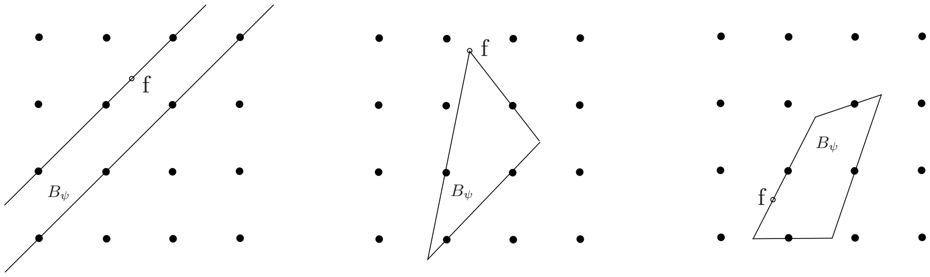


Fig. 3. Representation of B_ψ for degenerate cases

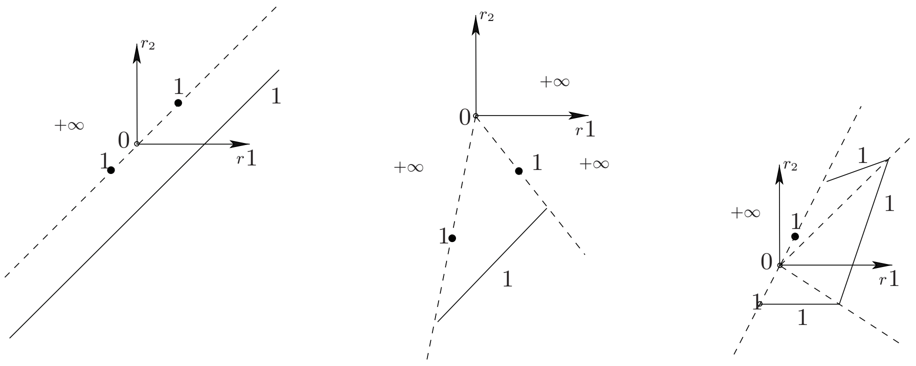


Fig. 4. Level curves of $\psi(r)$ for degenerate cases

Note that Dey et al. [5] showed in a more general context that, if $\psi(r) < +\infty$ everywhere, then ψ is continuous, and therefore ψ is nondegenerate.

Polyhedra with no integral point in their interior but with an integral point in the relative interior of each facet are called *maximal lattice-free* [9]. The complete list of all maximal lattice-free convex sets in the plane is known:

Theorem 2. *[9] A maximal lattice-free convex set in the plane (x_1, x_2) is one of the following:*

i) An irrational line $ax_1 + bx_2 = c$, where a/b is irrational and $c \notin a\mathbb{Z} + b\mathbb{Z}$;
ii) A strip $c \le ax_1 + bx_2 \le c + 1$ where a and b are coprime integers and c is an integer;
iii) A triangle with an integral point in the interior of each of its edges;
iv) A quadrilateral containing exactly four integral points, with exactly one of them in the interior of each of its edges; Moreover, these four integral points are vertices of a parallelogram of area 1.

The polyhedra referred to in Theorem 1 correspond to the last three cases in Theorem 2. The first case does not play a role here as we only consider rational vectors f and r in the definition of R_f.

A valid inequality $\sum \psi(r)s_r \geq 1$ for R_f *defines a facet* of R_f if there does not exist two distinct valid inequalities $\sum \psi_j(r)s_r \geq 1$, $j = 1, 2$, such that $\psi = \frac{1}{2}\psi_1 + \frac{1}{2}\psi_2$. Note that, although we only use nontrivial inequalities in this definition, including them would give an equivalent definition. By extension, we also say that such a function ψ *defines a facet* of R_f. Gomory [7] recently raised the question of describing the facets of R_f. In this paper, we give a complete characterization of the facets of R_f.

A valid inequality $\sum_{i=1}^{k} \psi_j(r^i)s_i \geq 1$ for $R_f(r^1, \ldots, r^k)$ *defines a facet* of $R_f(r^1, \ldots, r^k)$ if two distinct valid inequalities $\sum_{i=1}^{k} \psi_j(r^i)s_i \geq 1$, $j = 1, 2$, do not exist such that $\psi(r^i) = \frac{1}{2}\psi_1(r^i) + \frac{1}{2}\psi_2(r^i)$ for $i = 1, \ldots, k$. This definition of a facet of $R_f(r^1, \ldots, r^k)$ is consistant with the usual definition of a facet of a polyhedron only if the polyhedron is full dimensional. The next lemma shows that this is the case.

Lemma 1. *If $R_f(r^1, \ldots, r^k)$ is non empty, then it is full dimensional.*

Proof. The recession cone of $R_f(r^1, \ldots, r^k)$ is $\mathbb{R}^k_+$.

The paper is organized as follows. Section 2 explains how results for the facets of R_f can be used to derive results for the facets of $R_f(r^1, \ldots, r^k)$. Section 3 shows that split inequalities are always facet defining for R_f, and that degenerate split inequalities also are. Section 4 deals with triangle and quadrilateral inequalities. It shows that triangle inequalities are always facets of R_f and it gives a necessary and sufficient condition for a quadrilateral inequality to define a facet. For the remaining degenerate cases, vertex-degenerate and edge-degenerate triangle inequalities are facet defining only if a condition on the integral points in the boundary of B_ψ is satisfied, whereas degenerate quadrilaterals never define a facet of R_f.

2 Facets of R_f and Facets of $R_f(r^1, \ldots, r^k)$

In this section, we give a relation between facets of R_f and facets of $R_f(r^1, \ldots, r^k)$. We first show that the degenerate cases can be ignored when dealing with $R_f(r^1, \ldots, r^k)$.

Theorem 3. *Let $r^1, \ldots, r^k$ be a set of $k \geq 1$ rays. Every nontrivial facet of $R_f(r^1, \ldots, r^k)$ can be obtained from a nondegenerate minimal valid function ψ for R_f.*

We will see in Theorem 6 that split inequalities always define facets of R_f. The situation for $R_f(r^1, \ldots, r^k)$ is a little bit more complicated, as the next theorem shows. See also Andersen et al. [1].

Theorem 4. *Let ψ be valid and minimal for R_f, with B_ψ unbounded and f in its interior. Let $r^1, \ldots, r^k$ be a set of $k \geq 1$ rays. Then ψ defines a split inequality $\sum_{i=1}^{k} \psi(r^i)s_{r^i} \geq 1$ for $R_f(r^1, \ldots, r^k)$. This inequality can also be obtained as a quadrilateral inequality if $\psi(r^i) > 0$ for $i = 1, \ldots, k$, and it defines a facet of $R_f(r^1, \ldots, r^k)$ otherwise.*

Nondegenerate minimal valid inequalities that are not split inequalities are generated by a function ψ with B_ψ bounded and f in the interior of B_ψ. Let $x^1, \ldots, x^k$ be the vertices of B_ψ. We always assume that these vertices are topologically ordered so that the edges of the boundary of B_ψ are convex combinations of x^i and x^{i+1} with indices taken modulo k. We define the *corner rays* of B_ψ to be the rays $\{r^1, \ldots, r^k\}$ joining f to the vertices of B_ψ, with $r^i = x^i - f$ for $i = 1, \ldots, k$.

The next theorem shows that if $r^1, \ldots, r^k$ are the corner rays of B_ψ then ψ is facet defining for R_f if and only if it is facet defining for $R_f(r^1, \ldots, r^k)$.

Theorem 5. *Assume that B_ψ is a polytope with f in its interior. Let ψ be valid and minimal for R_f and let $r^1, \ldots, r^k$ be the corner rays of B_ψ. Then ψ is facet defining for $R_f(r^1, \ldots, r^k)$ if and only if ψ is facet defining for R_f.*

The assumption of Theorem 5 that the rays are the corner rays of B_ψ can be relaxed slightly, keeping the proof almost identical:

Corollary 1. *Assume that B_ψ is a polytope with f in its interior. Let ψ be valid and minimal for R_f and let $r^1, \ldots, r^\ell$ be a set of rays including the corner rays of B_ψ. Then ψ is facet defining for $R_f(r^1, \ldots, r^\ell)$ if and only if ψ is facet defining for R_f.*

In [1], Andersen, Louveaux, Weismantel and Wolsey study $R_f(r^1, \ldots, r^k)$ and they prove that, when nonnegative combinations of $r^1, \ldots, r^k$ span $\mathbb{R}^2$, all the nontrivial facets of $R_f(r^1, \ldots, r^k)$ are split inequalities or are triangle or quadrilateral inequalities where the vertices of B_ψ are on the rays $f + \lambda r^i$, $\lambda > 0$, for $i = 1, \ldots, k$. They do not, however, describe precisely which triangles and quadrilaterals generate facets. Theorems 3, 4 and Corollary 1 show that the characterization obtained in this paper of the triangles and quadrilaterals that generate facets for R_f (Theorems 8 and 9) gives a complete characterization of the nontrivial facets of $R_f(r^1, \ldots, r^k)$.

3 Split Inequalities

3.1 Nondegenerate Case

Consider a direction $r^0 \in \mathbb{Q}^2 \setminus \{0\}$ such that the line $L_0 := \{x = f + \alpha r^0,\ \alpha \in \mathbb{R}\}$ contains no integral point. Let L_1 and L_2 be parallel lines to L_0, each containing integral points, such that the set of points between L_1 and L_2 contains no integral point in its interior and contains L_0. (See Figure 5.) Define $\psi(r_0) = \psi(-r_0) = 0$, $\psi(x - f) = 1$ for any $x \in L_1 \cup L_2$. Since ψ is homogeneous, this defines $\psi(r)$ for all $r \in \mathbb{Q}^2$. The valid inequality $\sum \psi(r) s_r \geq 1$ is the well known *split inequality* [4]. These inequalities are equivalent to Gomory's mixed integer inequalities [6].

Theorem 6. *Split inequalities define facets of R_f.*

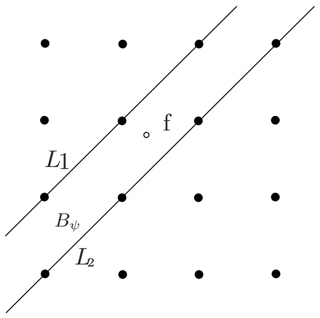


Fig. 5. Illustration for Theorem 6

3.2 Degenerate Case

Consider a direction $r^0 \in \mathbb{Q}^2 \setminus \{0\}$ such that the line $L_0 := \{x = f + \alpha r^0, \alpha \in \mathbb{R}\}$ contains integral points. Let L_1 be a line parallel to L_0 that contains integral points, such that the set of points between L_0 and L_1 contains no integral point in its interior. Let y^1 and y^2 be the first integral points encountered on the half-lines $f + \alpha r^0$, $\alpha \geq 0$, and $f - \alpha r^0$, $\alpha \geq 0$ respectively. Define $\psi(y^1 - f) = \psi(y^2 - f) = 1$ and $\psi(x - f) = 1$ for any $x \in L_1$. Since ψ is homogeneous, this defines $\psi(r)$ for all $r \in \mathbb{Q}^2$ in the closed half-space limited by L_0 and containing L_1. For all other $r \in \mathbb{Q}^2 \setminus \{0\}$, define $\psi(r) = +\infty$. The inequality $\sum \psi(r) s_r \geq 1$, a *degenerate split inequality*, is valid for R_f.

Theorem 7. *Degenerate split inequalities define facets of R_f.*

4 Triangle and Quadrilateral Inequalities

In this section, we assume that ψ is valid and minimal for R_f with f in the interior of B_ψ and that B_ψ is a polytope. Then by Theorem 1 and Theorem 2, B_ψ is either a triangle or a quadrilateral such that each of its boundary edges contains an integral point in its interior.

Let $x^1, \ldots, x^k$ be the vertices of B_ψ and $r^1, \ldots, r^k$ be the corner rays of B_ψ and let y^i be an integral point that can be obtained as a nontrivial convex combination of x^i and x^{i+1} for $i = 1, \ldots, k$ (indices are always implicitly taken modulo k).

Define M as the $2 \times k$ matrix whose column i is the vector y^i for $i = 1, \ldots, k$ (Recall that $k = 3$ or 4). Define X as the $2 \times k$ matrix whose column i is the vector x^i for $i = 1, \ldots, k$. Let S be the $k \times k$ matrix whose column i is the vector corresponding to the coefficients in the convex combination of x^i and x^{i+1} giving y^i for $i = 1, \ldots, k$.

We then have

$$M = X \cdot S \tag{5}$$

with

$$S = \begin{pmatrix} \alpha & 0 & 1-\gamma \\ 1-\alpha & \beta & 0 \\ 0 & 1-\beta & \gamma \end{pmatrix} \quad \text{or} \quad S = \begin{pmatrix} \alpha & 0 & 0 & 1-\delta \\ 1-\alpha & \beta & 0 & 0 \\ 0 & 1-\beta & \gamma & 0 \\ 0 & 0 & 1-\gamma & \delta \end{pmatrix}$$

where α, β, γ and δ are all strictly between 0 and 1.

Since we are interested in the dimension of faces of polyhedra, which requires checking affine independence of points, we add a third row full of 1s to the matrices M (resp. X) to obtain matrix $\bar{M}$ (resp., $\bar{X}$). Due to the specific form of the matrix S, we still have

$$\bar{M} = \bar{X} \cdot S \ . \tag{6}$$

Let A be an $m \times n$ matrix. The *nullspace* of A is $\mathcal{N}(A) = \{x \in \mathbb{R}^n \mid Ax = 0\}$ and the *columnspace* of A is $\mathcal{C}(A) = \{z \in \mathbb{R}^m \mid z = Ax \text{ for some } x \in \mathbb{R}^n\}$.

The following three results are classical results of linear algebra [10]:

Lemma 2. *Let A be an $m \times n$ matrix and B be an $n \times p$ matrix. Then*

$$rank(A \cdot B) = rank(B) - dim(\mathcal{N}(A) \cap \mathcal{C}(B)) \ .$$

Corollary 2. *Let A be an $m \times n$ matrix and B be an $n \times p$ matrix. If $rank(A) = n$, then*

$$rank(A \cdot B) = rank(B) \ .$$

Proof. If $rank(A) = n$, then $\mathcal{N}(A) = \{0\}$ and has dimension 0. Applying Lemma 2 yields the result.

Corollary 3. *Let A be an $m \times n$ matrix and B be an $n \times p$ matrix. Then*

$$rank(A \cdot B) \leq \min\{rank(A), rank(B)\} \ .$$

Proof. Apply Lemma 2 to $A \cdot B$ and its transpose.

4.1 Triangle Inequalities

Theorem 8. *Triangle inequalities define facets of R_f and of $R_f(r^1, r^2, r^3)$ where r^1, r^2 and r^3 are the corner rays of the maximal lattice-free triangle.*

Proof. Since $k = 3$ and B_ψ is a triangle, both $\bar{M}$ and $\bar{X}$ have rank 3. By Corollary 2, S has rank 3 too. It implies that the columns of S are affinely independent. Since they all satisfy with equality the inequality $\sum_{i=1}^{3} \psi(r^i) s_i \geq 1$, this inequality defines a facet of $R_f(r^1, r^2, r^3)$. By Theorem 5, ψ defines a facet of R_f.

4.2 Quadrilateral Inequalities

When $k = 4$, both $\bar{M}$ and $\bar{X}$ have rank 3. By Lemma 2, we have

$$3 = rank(\bar{M}) = rank(\bar{X} \cdot S) = rank(S) - dim(\mathcal{N}(\bar{X}) \cap \mathcal{C}(S)) .$$

Since $rank(\bar{X}) = 3$, we have that $\mathcal{N}(\bar{X})$ is a one-dimensional linear space. Hence $dim(\mathcal{N}(\bar{X}) \cap \mathcal{C}(S)) \leq 1$ and $rank(S) = 4$ if and only if $\mathcal{N}(\bar{X}) \subseteq \mathcal{C}(S)$.

Theorem 9. *Consider a maximal lattice-free quadrilateral with vertices x^i, integral point y^i on edge $x^i x^{i+1}$ (indices taken modulo 4) and corner rays r^i, $i = 1, \dots, 4$. The corresponding quadrilateral inequality defines a facet of $R_f(r^1, r^2, r^3, r^4)$ (and therefore of R_f) if and only if there is no $t \in \mathbb{R}_+$ such that the point y^i divides the edge joining x^i to x^{i+1} in a ratio t for odd i and in a ratio $1/t$ for even i, i.e.*

$$\frac{||y^i - x^i||}{||y^i - x^{i+1}||} = \begin{cases} t & \text{for } i = 1, 3 \\ \frac{1}{t} & \text{for } i = 2, 4 \end{cases} .$$

Proof. Let F be the face of $R_f(r^1, \dots, r^4)$ defined by $\sum_{i=1}^4 \psi(r^i) s_i = 1$. As $f + r^i = x^i$ is on the boundary of B_ψ, we have $\psi_i(r^i) = 1$ for $i = 1, \dots, 4$. Hence, if $s \in F$ then $\sum_{i=1}^4 s_i = 1$. Recall that $R_f(r^1, \dots, r^k)$ is the convex hull of vectors in the set $H := \{s \in \mathbb{R}^4_+ \mid f + \sum_{i=1}^4 r^i s_i \text{ is integral}\}$. Thus, if F is a facet, then there exists four affinely independent vectors s^j, for $j = 1, \dots, 4$, in H with

$$\sum_{i=1}^4 s_i^j = 1 \qquad \text{and} \qquad z^j = f + \sum_{i=1}^4 r_i s_i^j = \sum_{i=1}^4 (f + r_i) s_i^j \quad \text{integer} .$$

This implies that z^j is in the convex hull of $x^1, \dots, x^4$, for $j = 1, \dots, 4$. Theorem 2 shows that the only integral points in B_ψ are the points $y^1, \dots, y^4$. Moreover, for each $j = 1, \dots, 4$, there is a unique convex combination of $x^1, \dots, x^4$ that produces y^j, namely column j of matrix S. In other words, F is a facet if and only if the columns of S are affinely independent. Observe that the columns of S are affinely independent if and only if they are linearly independent since the sum of the entries in any column of S is 1. It follows that F is a facet if and only if $rank(S) = 4$.

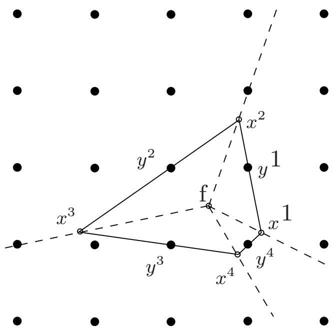
Let $u = (1, -1, 1, -1)^T$. By Theorem 2 iv), the points $y^1, \dots, y^4$ are the vertices of a parallelogram. This implies that $\bar{M} \cdot u = 0$. Then (6) gives $\bar{X} \cdot S \cdot u = 0$. We now have two cases:

i) $S \cdot u = 0$. Then $rank(S) \leq 3$ and Corollary 3 shows that $rank(S) = 3$. Solving the linear system $S \cdot u = 0$ gives $\alpha = 1 - \beta = \gamma = 1 - \delta$. This is equivalent to the ratio condition of the statement.

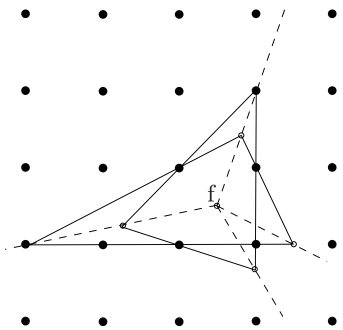
ii) $S \cdot u \neq 0$. Then for $v = S \cdot u$ we have $\bar{X} \cdot v = 0$, and as $v \neq 0$, we have that $\mathcal{N}(\bar{X})$ is the linear space spanned by v. Since v is obtained as a linear combination of the columns of S, we have $\mathcal{N}(\bar{X}) \subseteq \mathcal{C}(S)$ and by Lemma 2 we get $rank(S) = 4$. Since all the columns of S satisfy with equality the inequality $\sum_{i=1}^4 \psi(r^i) s_i \geq 1$, this inequality defines a facet of $R_f(r^1, r^2, r^3, r^4)$. By Theorem 5, ψ defines a facet of R_f.

We illustrate the condition in Theorem 9 by a couple of examples. This condition implies that the quadrilateral inequality generated from the square whose edges contain the integral points $\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix}$ in their middle (usually called *octahedron* inequality) does not define a facet. However, if one tilts just one edge of the square around its (integral) middle point, the resulting trapezoid has three distinct ratios $\frac{||y^i - x^i||}{||y^i - x^{i+1}||}$. Therefore Theorem 9 states that the resulting quadrilateral inequality defines a facet of R_f.

We give another more complicated example, see Figure 6. Let $f = \begin{pmatrix}\frac{1}{2}\\\frac{1}{2}\end{pmatrix}$ and Q the quadrilateral with vertices $x^1 = \begin{pmatrix}\frac{7}{6}\\\frac{1}{6}\end{pmatrix}$, $x^2 = \begin{pmatrix}\frac{7}{8}\\\frac{13}{8}\end{pmatrix}$, $x^3 = \begin{pmatrix}-\frac{7}{6}\\\frac{1}{6}\end{pmatrix}$, $x^4 = \begin{pmatrix}\frac{7}{8}\\-\frac{1}{8}\end{pmatrix}$.



(a) The quadrilateral Q.



(b) The two triangles T_1 and T_2.

Fig. 6. Illustration for the second example

Edge x^1x^2 contains integral point $y^1 = \begin{pmatrix}1\\1\end{pmatrix}$ with ratio $\frac{||y^1-x^1||}{||y^1-x^2||} = \frac{4}{3}$.

Edge x^2x^3 contains integral point $y^2 = \begin{pmatrix}0\\1\end{pmatrix}$ with ratio $\frac{||y^2-x^2||}{||y^2-x^3||} = \frac{3}{4}$.

Edge x^3x^4 contains integral point $y^3 = \begin{pmatrix}0\\0\end{pmatrix}$ with ratio $\frac{||y^3-x^3||}{||y^3-x^4||} = \frac{4}{3}$.

Edge x^4x^1 contains integral point $y^4 = \begin{pmatrix}1\\0\end{pmatrix}$ with ratio $\frac{||y^4-x^4||}{||y^4-x^1||} = \frac{3}{4}$.

Theorem 9 states that the quadrilateral inequality obtained from Q is not a facet.

Indeed, it can be obtained as a convex combination of two triangle inequalities, each with a multiplier $\frac{1}{2}$. The first triangle T_1 has vertices $\begin{pmatrix} \frac{3}{2} \\ 0 \end{pmatrix}$, $\begin{pmatrix} \frac{4}{5} \\ \frac{7}{5} \end{pmatrix}$, $\begin{pmatrix} -2 \\ 0 \end{pmatrix}$. The second triangle has vertices $\begin{pmatrix} 1 \\ -\frac{1}{3} \end{pmatrix}$, $\begin{pmatrix} 1 \\ 2 \end{pmatrix}$, $\begin{pmatrix} -\frac{3}{4} \\ \frac{1}{4} \end{pmatrix}$. Both triangles have all four points y^1, y^2, y^3, y^4 on their boundaries. The corner rays of Q are $r^1 = \begin{pmatrix} \frac{2}{3} \\ -\frac{1}{3} \end{pmatrix}$, $r^2 = \begin{pmatrix} \frac{3}{8} \\ \frac{9}{8} \end{pmatrix}$, $r^3 = \begin{pmatrix} -\frac{5}{3} \\ -\frac{1}{3} \end{pmatrix}$, $r^4 = \begin{pmatrix} \frac{3}{8} \\ -\frac{5}{8} \end{pmatrix}$. Triangle T_1 has corner rays positive multiples of r^1, r^2 and r^3. Triangle T_2 has corner rays positive multiples of r^2, r^3 and r^4. If ψ, ψ_1 and ψ_2 denote the functions defined by Q, T_1 and T_2 respectively, it is easy to verify that $\psi = \frac{1}{2}\psi_1 + \frac{1}{2}\psi_2$ in each of the cones $r^i r^{i+1}$ (indices defined modulo 4). Indeed, each of these functions is linear in each of the cones. So it is sufficient to verify the equality $\psi(r) = \frac{1}{2}\psi_1(r) + \frac{1}{2}\psi_2(r)$ in each of the directions r^i, $i = 1, \ldots, 4$. In direction r^1 we have $\psi_1 \begin{pmatrix} 1 \\ -\frac{1}{2} \end{pmatrix} = 1$ and $\psi_2 \begin{pmatrix} \frac{1}{2} \\ -\frac{1}{4} \end{pmatrix} = 1$. This implies $\psi_1(r^1) = \frac{2}{3}$ and $\psi_2(r^1) = \frac{4}{3}$. Therefore $\psi(r^1) = \frac{1}{2}\psi_1(r^1) + \frac{1}{2}\psi_2(r^1)$ as required. Similarly, for the other rays, we find $\psi_1(r^2) = \frac{5}{4}$ and $\psi_2(r^2) = \frac{3}{4}$; $\psi_1(r^3) = \frac{4}{3}$ and $\psi_2(r^3) = \frac{2}{3}$; $\psi_1(r^4) = \frac{3}{4}$ and $\psi_2(r^4) = \frac{5}{4}$.

5 Degenerate Triangle and Quadrilateral Inequalities

Theorem 10. *A vertex-degenerate triangle inequality defines a facet of R_f if and only if the edge of T opposite f contains at least two integral points in its interior.*

Theorem 11. *An edge-degenerate triangle inequality defines a facet of R_f if and only if at least one of the two edges not containing f contains two integral points y with $\psi(y - f) = 1$.*

Theorem 12. *Vertex-degenerate and edge-degenerate quadrilateral inequalities never define facets of R_f.*

References

1. Andersen, K., Louveaux, Q., Weismantel, R., Wolsey, L.: Cutting Planes from Two Rows of a Simplex Tableau. In: Proceedings of IPCO XII, Ithaca, New York, June 2007, pp. 1–15. (2007)
2. Balas, E.: Intersection Cuts - A New Type of Cutting Planes for Integer Programming. Operations Research 19, 19–39 (1971)
3. Borozan, V., Cornuejols, G.: Minimal Valid Inequalities for Integer Constraints, technical report (July 2007)
4. Cook, W., Kannan, R., Schrijver, A.: Chvátal Closures for Mixed Integer Programming Problems. In: Mathematical Programming, vol. 47, pp. 155–174 (1990)

5. Dey, S.S., Richard, J.-P.P., Miller, L.A., Li, Y.: Extreme Inequalities for Infinite Group Problems, technical report (2006)
6. Gomory, R.E.: An Algorithm for Integer Solutions to Linear Programs. In: Graves, R.L., Wolfe, P. (eds.) Recent Advances in Mathematical Programming, pp. 269–302. McGraw-Hill, New York (1963)
7. Gomory, R.E.: Thoughts about Integer Programming. In: 50th Anniversary Symposium of OR, Corner Polyhedra and Two-Equation Cutting Planes, George Nemhauser Symposium, University of Montreal, Atlanta (January 2007) (July 2007)
8. Gomory, R.E., Johnson, E.L.: Some Continuous Functions Related to Corner Polyhedra, In: Mathematical Programming, Part I. vol. 3, pp. 23–85 (1972)
9. Lovász, L.: Geometry of Numbers and Integer Programming. In: Iri, M., Tanabe, K. (eds.) Mathematical Programming: Recent Developments and Applications, pp. 177–210. Kluwer, Dordrecht (1989)
10. Meyer, C.D.: Matrix Analysis and Applied Linear Algebra. SIAM, Philadelphia (2000)
11. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley, Chichester (1988)

A Polyhedral Investigation of the LCS Problem and a Repetition-Free Variant

Cristina G. Fernandes, Carlos E. Ferreira, Christian Tjandraatmadja, and Yoshiko Wakabayashi

Universidade de São Paulo, Brazil
{cris,cef,christj,yw}@ime.usp.br

Abstract. We consider the longest common subsequence problem (LCS) and a variant of it where each symbol may occur at most once in the common subsequence. The LCS is a well-known problem that can be solved in polynomial time by a dynamic programming algorithm. We provide a complete description of a polytope we associate with the LCS. The integrality of this polytope can be derived by showing that it is in fact the clique polytope of a perfect graph. The repetition-free version of the problem is known to be difficult. We also associate a polytope with this version and investigate its facial structure. We present some valid and facet-defining inequalities for this polytope and discuss separation procedures. Finally, we present some computational results of a branch and cut algorithm we have implemented for this problem.

1 Introduction

Given two finite sequences s and t over an alphabet, the longest common subsequence problem (LCS) consists in finding a longest common subsequence of s and t. It is well known that this problem can be solved in polynomial time by a dynamic programming algorithm that runs in $O(|s||t|)$ time (see [4]). The LCS has important applications in Bioinformatics and Computational Biology, where the sequences are genomes and a common subsequence may be interpreted as a similarity measure between the genomes. It is also present in the core of the unix `diff` command.

We associate a polytope with the LCS and study its facial structure. For that, given an instance of the problem, we represent the feasible solutions of this instance as vectors in $\mathbb{R}^n$ (for some dimension n) and consider the polytope defined as the convex hull of these vectors. We give a complete description of this polytope, exhibiting all of its facets (faces of maximal dimension). Since the LCS is polynomially solvable, it is expected that such a complete description can be provided (see [6,7] for details). We discuss the description we provide in this paper and relate it with other known results.

We also study a variation of the LCS in which each symbol of the alphabet is allowed to occur at most once in the sought common subsequence. We refer to this version as the **Repetition-Free LCS** problem (RFLCS). It has been proved that this problem is NP-hard. Indeed, Adi et al. [1] proved that it is APX-hard even if each symbol appears at most twice in each of the given sequences. Another variation considered in the literature is the so-called *exemplar model*. In this problem, as in the RFLCS, each symbol may appear at most once in the common subsequence. Besides, some symbols are mandatory and must appear in the subsequence. This model has been studied by Bonizzoni et al. [2] who proved that the problem is also APX-hard. Yet another related variation was proposed by Sankoff [9] in the studies of gene families. All these variations may be useful in the comparisons of genomes [2,9] and may prove to be useful in other contexts as well.

This paper is organized as follows. In Section 2 we give a complete description of the polytope we associate with the LCS and show how to solve the separation problem for it in polynomial time. Moreover we show the relationship between the LCS polytope and the clique polytope of a class of perfect graphs. In Section 3 we provide a formulation for the RFLCS and present some results on valid and facet-defining inequalities for the corresponding polytope. Some computational results of a branch and cut algorithm we developed for the RFLCS are shown in Section 4. Finally, in Section 5 we present some conclusions.

2 A Polyhedral Study of the LCS Polytope

Consider an instance of the LCS consisting of two sequences $s = s_1, \dots, s_n$ and $t = t_1, \dots, t_m$ over some alphabet. For each symbol a of the alphabet, let $s(a)$ be the set of indices i of s such that $s_i = a$, and let $t(a)$ be defined analogously. Let $E \subseteq \{1, \dots, n\} \times \{1, \dots, m\}$ be the set of all pairs (i, j) in $s(a) \times t(a)$ where a ranges over all symbols of the alphabet. That is, $E = \cup_a \big(s(a) \times t(a)\big)$.

Note that any common subsequence w of s and t can be represented by a vector z in $\{0,1\}^E$ as follows. If $w = s_{i_1}, \dots, s_{i_p} = t_{j_1}, \dots, t_{j_p}$, where $i_1 < \cdots < i_p$ and $j_1 < \cdots < j_p$, then for each pair ij in E we have that $z_{ij} = 1$ if and only if there exists an index ℓ in $\{1, \dots, p\}$ such that $i = i_\ell$ and $j = j_\ell$. Thus, the feasible solutions of this instance of the LCS are the vertices of the following polytope (defined as the convex hull of these solutions).

$$P_{\text{LCS}} := \operatorname{conv}\{z \in \{0,1\}^E \mid z \text{ represents a common subsequence of } s \text{ and } t\}.$$

Obviously, finding a vector z^* in P_{LCS} with maximum number of nonzero components is equivalent to finding a longest common subsequence of s and t.

Proposition 1. *The polytope P_{LCS} is full-dimensional.*

Proof. It is sufficient to observe that the zero vector and the unit vectors e^{ij}, for all (i, j) in E (that is, the vector such that $e^{ij}_{ij} = 1$ and $e^{ij}_{k\ell} = 0$ for all $(k, \ell) \neq (i, j)$), are in P_{LCS} and are affinely independent. Thus the polytope P_{LCS} has dimension $|E|$. $\square$

We say that two distinct pairs (i,j) and (k,ℓ) in E **cross** if ($i \leq k$ and $j \geq \ell$) or ($k \leq i$ and $\ell \geq j$). A simple integer programming formulation for the LCS follows.

$$\begin{array}{lrcll} \text{maximize} & \sum_{(i,j)\in E} z_{ij} & & & \\ \text{subject to} & z_{ij} + z_{k\ell} & \leq & 1 & \text{for all } (i,j) \text{ and } (k,\ell) \text{ in } E \text{ that cross,} \\ & z_{ij} & \in & \{0,1\} & \text{for all } (i,j) \text{ in } E. \end{array} \tag{1}$$

Observe that z is a feasible solution of (1) if and only if it corresponds to a common subsequence of s and t.

Proposition 2. *For every (i,j) in E, the inequality $z_{ij} \geq 0$ defines a facet of P_{LCS}.*

Proof. Fix an (i,j) in E. It is easy to see that the zero vector and the unit vectors $e^{k\ell}$ for all $(k,\ell) \neq (i,j)$ are vertices of P_{LCS} that satisfy $z_{ij} = 0$ and they are affinely independent. □

We note that the inequalities in (1) do not always induce facets of P_{LCS}. As we will see in what follows, some of them may possibly be facet-defining. To simplify the notation, if z is a vector and S is a subset of E, then we denote by $z(S)$ the sum $\sum_{(i,j)\in S} z_{ij}$.

We say that a set $S \subseteq E$ is a **star** if any two distinct pairs in S cross. If S is a star then the inequality $z(S) \leq 1$ is called a **star inequality**. Note that the inequalities in (1) are star inequalities defined by stars of cardinality 2.

Lemma 1. *Consider a star $S \subseteq E$. Then the star inequality $z(S) \leq 1$ is valid for P_{LCS}, and it defines a facet if and only if S is maximal.*

Proof. It is immediate that the star inequality $z(S) \leq 1$ is valid for P_{LCS}. Now, let us prove that it defines a facet when S is maximal.

Consider a facet-defining inequality $az \leq \alpha$ and suppose

$$\{z \in \mathbb{R}^E \mid z(S) = 1\} \;\subseteq\; F := \{z \in \mathbb{R}^E \mid az = \alpha\}.$$

Note that, for each pair (p,q) in $E \setminus S$, there exists a pair (i,j) in S such that (i,j) and (p,q) do not cross (since S is maximal). Thus, e^{ij} and $e^{ij} + e^{pq}$ are incidence vectors of common subsequences of the sequences s and t and are in F. So, $a_{ij} = ae^{ij} = a(e^{ij} + e^{pq}) = a_{ij} + a_{pq}$, and therefore $a_{pq} = 0$ for every (p,q) in $E \setminus S$. Now observe that, for every (i,j) and (k,ℓ) in S, we have that e^{ij} and $e^{k\ell}$ are in F. Then, $a_{ij} = ae^{ij} = ae^{k\ell} = a_{k\ell} = \alpha$, for some constant α. This proves that $az \leq \alpha$ can be rewritten as $\alpha z(S) \leq \alpha$, and therefore the star inequality $z(S) \leq 1$ is facet-defining.

Conversely, suppose that the star inequality $z(S) \leq 1$ defines a facet of P_{LCS}. If S is not maximal, there exists a pair (p,q) in $E \setminus S$ such that (p,q) crosses all pairs in S. Thus, $S' := S \cup \{(p,q)\}$ is a star and the inequality $z(S') \leq 1$ is valid. In this case, the star inequality $z(S) \leq 1$ can be written as the sum of the inequality $z(S') \leq 1$ and the inequality $z_{pq} \geq 0$, a contradiction. □

A nice result is that the inequalities given in the lemma above define completely the polytope P_{LCS}. More precisely, any facet-defining inequality of P_{LCS} is either a nonnegativity inequality or a maximal star inequality. This result is proved in the next theorem. First, consider the following order relation on the pairs on E. Given (i,j) and (k,ℓ) in E, we write $(i,j) \preceq (k,\ell)$ if $i < k$ or ($i = k$ and $j \leq \ell$). Now, we can define a lexicographical order on the subsets of E as follows. Consider two subsets S_1 and S_2 of E. For $k = 1, 2$, let $((i_1^k, j_1^k), \ldots, (i_{\ell_k}^k, j_{\ell_k}^k))$ be the sequence of the pairs in S_k sorted according to this (total) order and refer to it as the sorted S_k. We say that S_1 is **lexicographically smaller than or equal to** S_2, and write $S_1 \preceq S_2$, if either the sorted S_1 is a prefix of the sorted S_2 or $(i_p^1, j_p^1) \preceq (i_p^2, j_p^2)$ for the first index p such that $(i_p^1, j_p^1) \neq (i_p^2, j_p^2)$ (if it exists).

Theorem 1

$$\begin{aligned} P_{\text{LCS}} = \{z \in \mathbb{R}^E \mid z_{ij} \geq 0 \quad & \textit{for all } (i,j) \in E \textit{ and} \\ z(S) \leq 1 \; & \textit{for all maximal star } S \subseteq E\}. \end{aligned}$$

Proof. Let $az \leq \alpha$ be an inequality that induces a facet, say F, of P_{LCS}. First we prove that, if this inequality has negative coefficients, it must be a nonnegativity constraint for some (i,j) in E. Say $a_{ij} < 0$ and suppose that the inequality $az \leq \alpha$ is not a multiple of $-z_{ij} \leq 0$. Then, there must be a vector z in P_{LCS} such that $az = \alpha$ and $z_{ij} = 1$ (otherwise, F would be contained in the hyperplane $z_{ij} = 0$). But then $z - e^{ij}$ is also a vector in P_{LCS}, and $a(z - e^{ij}) = az - a_{ij} > \alpha$, a contradiction.

So, we may assume that $a_{ij} \geq 0$ for all (i,j) in E. Consider now the support T of the inequality $az \leq \alpha$ and some (i,j) in $E \backslash T$. It is easy to see that there must exist some (k,ℓ) in T that does not cross (i,j). Otherwise F would be contained in the hyperplane $z_{ij} = 0$. It remains to be proved that T is a star.

Let $S_1 = \{(i_1, j_1), (i_2, j_2), \ldots, (i_t, j_t)\}$ be the first maximal star in T in the lexicographical order defined above. If $az \leq \alpha$ is not a maximal star inequality, there must exist some z in $\mathbb{R}^E$ such that $az = \alpha$ and $z(S_1) = 0$. Thus, there must exist two distinct pairs (i,j) and (k,ℓ) in $T \setminus S_1$ with $z_{ij} = z_{k\ell} = 1$ such that every pair in S_1 either crosses (i,j) or (k,ℓ). Assume without loss of generality that $(i,j) \preceq (k,\ell)$. As S_1 is maximal, (i,j) cannot cross all pairs in S_1. Since S_1 is the first maximal star in the lexicographical order, there is an index p in $\{1, \ldots, t\}$ such that (i,j) does not cross (i_p, j_p) and $(i_p, j_p) \preceq (i,j)$. Thus, $j \geq j_p$. But then, (k,ℓ) must cross (i_p, j_p), and therefore $\ell \leq j_p$. This implies that $i \leq k$ and $j \geq j_p \geq \ell$, and hence (i,j) and (k,ℓ) cross, a contradiction. □

2.1 The Separation Problem for the Star Inequalities

A polynomial-time algorithm to solve the separation problem for the star inequalities is the following. Let s and t be the two sequences for which we want to find a longest common subsequence, and let $z \in \mathbb{R}^E$ be such that $0 \leq z_{ij} \leq 1$ for all (i,j) in E. Consider the problem of finding a maximum weight common subsequence of the reversal of s and the sequence t, where the weight of aligning

s_i with t_j is z_{ij}. This problem can be solved in polynomial time using a dynamic programming algorithm (see [4]). Observe that such a common subsequence of the reversal of s and t corresponds to a maximal star (since the weights are non-negative). So, if such a maximum weight star, say S, has value greater than 1, the corresponding star inequality $z(S) \leq 1$ is violated by z. Conversely, if this value is less than or equal to 1, no star inequality is violated by the current solution z.

2.2 The Integrality of the LCS Polytope

Another way of proving that the polytope described in Theorem 1 is integral is by means of the following result shown by Fulkerson [5], Lovász [8] and Chvátal [3]. Given an undirected graph $G = (V, E)$, the **clique polytope** of G is defined as the convex hull of the incidence vectors of the cliques in G. We say a set $S \subseteq V$ is a **stable** set of G if S is a set of pairwise non-adjacent vertices. Consider the polytope:

$$\begin{aligned} P_C(G) = \{x \in \mathbb{R}^V \mid x_v \geq 0 & \quad \text{for all } v \in V \text{ and} \\ x(S) \leq 1 & \text{ for all stable set } S \subseteq V\}. \end{aligned}$$

Chvátal [3] proved that the description above is indeed the description of the clique polytope of G (and therefore integral) if and only if G is a perfect graph. (A graph is perfect if for every induced subgraph H of G the maximum size of a clique in H is equal to the minimum coloring of H).

Now we observe that the LCS polytope defined in Theorem 1 is the clique polytope of a perfect graph. Let G_{st} be the graph defined as follows: its vertex set consists of all pairs (i, j) in E (as we defined in the beginning of Section 2), and two vertices are adjacent if and only if the corresponding pairs do not cross.

Thus, finding a longest common subsequence of s and t is equivalent to finding a largest clique in the graph G_{st}. Note that the maximal star inequalities $z(S) \leq 1$ that are facets of P_{LCS} are in fact inequalities of the form $x(S) \leq 1$ of the clique polytope $P_C(G_{st})$, where S is a maximal stable set of G_{st}. Thus, $P_{\text{LCS}} = P_C(G_{st})$, that is, P_{LCS} is precisely the clique polytope of the graph G_{st}. Hence, P_{LCS} is integral if and only if G_{st} is perfect.

Now it remains to show that the graph G_{st} is perfect. To this end, we show that its complement $\bar{G}_{st}$ is perfect (it is known that the complement of a perfect graph is also a perfect graph [8]). We claim that $\bar{G}_{st}$ is a comparability graph, and therefore a perfect graph. We recall that a **comparability graph** is a graph that has a transitive orientation, that is, an orientation such that the resulting directed graph satisfies a transitive law: whenever there exist directed edges (a, b) and (b, c), there must exist a directed edge (a, c).

It is not difficult to prove that $\bar{G}_{st}$ is a comparability graph. It suffices to note that the following orientation $\mathcal{G}_{st}$ of $\bar{G}_{st}$ is transitive: if $e = \{u, v\}$ is an edge of $\bar{G}_{st}$, where $u = (i, j)$ and $v = (k, \ell)$ (note that the pairs (i, j) and (k, ℓ) cross), then orient e from u to v if $i \leq k$ and $j \geq \ell$ (see Figure 1). It is immediate that if $u = (i, j)$, $v = (k, \ell)$ and $w = (p, q)$ are vertices of $\mathcal{G}_{st}$, and in this graph there are arcs going from u to v and from v to w, then this graph has an arc going from u to w. This follows because we have $i \leq k$ and $j \geq \ell$ (as there is an arc from u to v) and we have $k \leq p$ and $\ell \geq q$ (as there is an arc from v to w) and therefore we have $i \leq k$ and $j \geq q$. Thus the orientation we defined is transitive.

Although the LCS polytope can be described as the clique polytope of the graph G_{st}, the proof of the complete description of P_{LCS} we have presented (by means of star inequalities) is interesting, as it is short and self-contained (it does not rely on the result concerning the clique polytope of perfect graphs).

3 Formulation for the RFLCS

Given two sequences s and t, the problem of finding a repetition-free LCS of s and t can be formulated as the following integer program.

$$\begin{array}{lrcll} \text{maximize} & \sum_{(i,j)\in E} z_{ij} & & & \\ \text{subject to} & z(E_a) & \leq & 1 & \text{for all symbol } a \text{ of the alphabet,} \\ & z_{ij} + z_{kl} & \leq & 1 & \text{for all } (i,j) \text{ and } (k,l) \text{ in } E \text{ that cross,} \\ & z_{ij} & \in & \{0,1\} & \text{for all } (i,j) \text{ in } E. \end{array} \tag{2}$$

As in the case of LCS, we can associate with RFLCS the following polytope:

$$P_{\text{RFLCS}} := \text{conv}\{z \in \{0,1\}^E \mid z \text{ represents a repetition-free common subsequence of } s \text{ and } t\}.$$

It is easy to see that the maximal star inequalities are valid for P_{RFLCS}. However, these inequalities are not facet-defining.

For a set $S \subseteq E$, let $S_a = S \cap (s(a) \times t(a))$, where a is a symbol of the alphabet. We say that a set $S \subseteq E$ is an **extended star** if, for every two distinct symbols a and b, each pair in S_a crosses all pairs in S_b. We prove in the next theorem that if S is a maximal extended star then the corresponding star inequality $z(S) \leq 1$ is facet-defining for P_{RFLCS}.

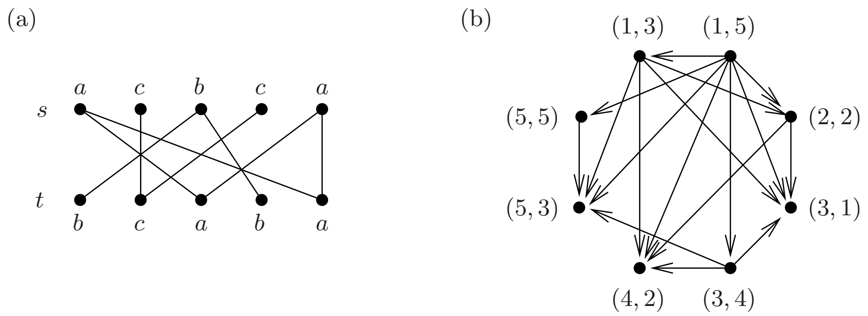


Fig. 1. (a) A graph indicating all pairs in E with respect to $s = acbca$ and $t = bcaba$. (b) The graph $\mathcal{G}_{st}$ for s and t.

Theorem 2. *Let S be a maximal extended star. The inequality $z(S) \leq 1$ defines a facet of P_{RFLCS}.*

Proof. It is immediate that the inequality $z(S) \leq 1$ is valid for P_{RFLCS}. In order to prove that it defines a facet of P_{RFLCS}, consider an inequality $az \leq \alpha$ that defines a facet F of P_{RFLCS} and suppose

$$\{z \in \mathbb{R}^E \mid z(S) = 1\} \subseteq F := \{z \in R^E \mid az = \alpha\}.$$

Consider a pair (i,j) in $E_a \setminus S$ for some symbol a. Since S is a maximal extended star, there is a pair (k,ℓ) in $S \cap E_b$, with $b \neq a$, that does not cross (i,j). Then, $e^{k\ell}$ and $e^{k\ell} + e^{ij}$ are both in F, and this implies that $a_{ij} = 0$ for every (i,j) in $E_a \setminus S$. Since this holds for every symbol a, we conclude that $a_{ij} = 0$ for every (i,j) in $E \setminus S$.

Now observe that for every (i,j) in S we have that $e^{ij} \in F$. Thus, $a_{ij} = \alpha$ for every (i,j) in S. Hence, the inequality $az \leq \alpha$ can be rewritten as $\alpha z(S) \leq \alpha$ and we conclude that the extended star inequality $z(S) \leq 1$ is facet-defining. $\square$

3.1 The Separation Problem for the Extended Star Inequalities

We show now that a polynomial-time algorithm using a dynamic programming algorithm can be used to solve the separation problem for the extended star inequalities. The idea of the algorithm is similar to the separation algorithm shown in the previous section. Let z be a fractional point we intend to separate. Every extended star (see Figure 2(a)) corresponds to an alignment as the one depicted in Figure 2(b), between the sequence s reversed, which we denote by s^r, and the sequence t. In the alignment shown in Figure 2(b) each star corresponds to some particular symbol (that is, only pairs corresponding to the same symbol can cross).

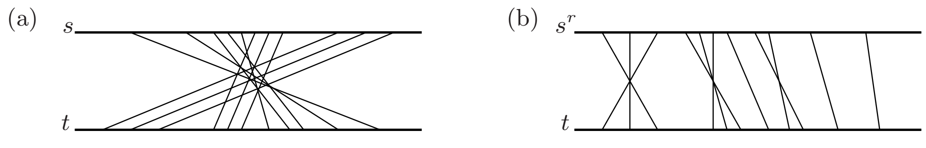


Fig. 2. (a) An extended star between s and t. (b) The corresponding alignment between s^r and t.

The separation algorithm has to find an extended star whose corresponding inequality is violated. For that, it looks for a maximum weight alignment, as the one in Figure 2(b), between s^r and t. The weights for the alignments are given by z. Such an alignment allows crossings only among pairs corresponding to a common symbol. One can find such an alignment by a dynamic programming algorithm.

Indeed, let $mw(n,m)$ be the weight of such a maximum weight alignment between the sequences $s^r[1\,..\,n]$ and $t[1\,..\,m]$. The following recurrence holds for $mw(n,m)$:

$$mw(n,m) = \begin{cases} 0 & \text{if } n=0 \text{ or } m=0, \\ \max\{mw(n-1,m), mw(n,m-1)\} & \text{if } s^r[n] \neq t[m], \\ \max_{1 \leq i,j \leq n}\{mw(i-1,j-1) + b_a(i,n,j,m)\} & \text{if } s^r[n] = t[m] = a, \end{cases}$$

where $b_a(i,n,j,m) = \sum\{z_{kl} : i \leq k \leq n,\ j \leq l \leq m,\ (k,l) \in E_a\}$.

It is not hard to see that $mw(n,m)$ can be computed by an $O(n^2m^2)$ algorithm. It is also not hard to derive an algorithm that finds an alignment between s^r and t of weight $mw(n,m)$. If $mw(n,m) < 1$, then such an alignment corresponds to an extended star that is violated. If $mw(n,m) \geq 1$, then no extended star is violated.

4 Computational Results

We implemented a branch and cut algorithm for the RFLCS using the exact separation procedure for the extended star inequalities shown in the previous section. We considered instances consisting of two sequences of equal length $n = 512$, and with alphabet sizes $\frac{1}{8}n, \frac{2}{8}n, \ldots, \frac{7}{8}n$. The instances are generated randomly with uniform probability. To solve the linear programming relaxation we use the GLPK (Gnu Linear Programming Kit), an ANSI C set of routines organized in the form of a callable library.

In the first experiment we tested our branch and cut algorithm on 10 random instances, each one with the above mentioned alphabet size. We limited the running time for each instance to one hour. The results obtained are summarized in Table 1. In each line the results shown correspond to 10 instances with the given alphabet size. The first column shows the *alphabet size*. In the second column we indicate the *average computational time* for the 10 instances; and we also indicate in parenthesis the *minimum and maximum running time*. The next column shows the *average number of cuts* added to the linear program. Then, we indicate how many of the instances could be solved to optimality within one hour. The next two columns show the *average lower and upper bound* achieved within the fixed time limit. Finally, the last two columns exhibit the *average number of active nodes at the moment the program was interrupted* and the *average number of nodes visited in the branch and bound tree* during the execution of the program.

In Table 1, we note that it becomes more difficult to solve instances in which the alphabet size is small compared to the length of the sequences (up to $\frac{3}{8}n$). This difficulty might be explained by the fact that when the alphabet is small then the number of repetitions in the sequences is large, and this implies that such instances may have many feasible solutions. It is interesting to note that we could solve all instances to optimality when the size of the alphabet is at least $\frac{1}{2}n$. The average number of nodes in the branch and cut tree (last column)

Table 1. Computational results with extended star inequalities

alph. size	time (min/max)	cuts	exact	lb	ub	active	nodes
64	3604.5 (3600/3609)	1162.5	0	44.4	63.9	1.0	0.0
128	3601.6 (3600/3604)	1927.7	0	56.8	75.2	1.0	0.0
192	2761.8 (762/3600)	2532.1	6	58.8	59.7	1.7	10.7
256	604.4 (224/1582)	1474.0	10	54.2	54.2	0.0	5.6
320	245.9 (93/498)	1077.1	10	46.5	46.5	0.0	1.8
384	113.0 (72/200)	797.0	10	43.0	43.0	0.0	1.0
448	76.0 (52/108)	660.6	10	40.7	40.7	0.0	1.4

indicate that most of the work was done in the root node. For the instances with small alphabet we achieve an average gap under 50% to the value of the best solution found in one hour.

In order to investigate the strength of the extended star inequalities we ran GLPK with the integer programming formulation (2). We set the parameters of the solver to use clique and Gomory cuts (these are the best parameters we could found). The results for the easier instances (with alphabet size at least $\frac{1}{2}n$) are summarized in Table 2.

Table 2. Branch and bound with formulation (2)

alph. size	time (min/max)	cuts	exact	lb	ub	active	nodes
256	3669.3 (3642/3724)	300.0	0	13.3	56.0	29.0	3.0
320	2739.3 (771/3631)	282.6	4	45.7	47.3	45.6	91.7
384	1159.9 (534/2886)	271.9	10	43.0	43.0	0.0	87.2
448	746.1 (254/2537)	236.5	10	40.7	40.7	0.0	117.0

As shown in Table 2, we could solve all 10 instances only for alphabet sizes 384 and 448. It is interesting to observe that in these cases the average time to solve the instances is around 10 times larger than the time we needed using the facet-defining inequalities and our separation procedure. In the other cases, we could solve only 4 instances to optimality, while using the extended star inequalities we could solve all instances within one hour.

5 Conclusion

The LCS is a well-known problem that has many nice applications. Perhaps because of the fact that a polynomial-time algorithm for this problem is known, a polyhedral approach to this problem has not been considered in the literature. We think the polyhedral results we have shown for the LCS in this paper have many interesting aspects: we give a complete and irredundant description of the polytope P_{LCS} we have associated to it and show that the separation problem for this polytope is polynomial. Furthermore, we have given an alternative proof of the integrality of this polytope by showing its relation with the clique polytope of the graph G_{st}, which we show to be perfect. The repetition-free version of LCS, an NP-hard problem, also has applications in the study of genomes. This variant has been less investigated. The polyhedral approach and the computational results we presented for the RFLCS show that this approach is an improvement to pure branch and bound strategies or simple heuristics. Further facets of this polytope, as well as some heuristics, may be incorporated to the present code, leading to improvements that can be useful to solve some larger instances of the problem.

Acknowledgements

We would like to thank José Coelho de Pina for some valuable comments related to the contents of Section 2.2. We also thank the financial support from CNPq (Proc. 490333/04, 307011/03-8, 308138/04-0, 301919/04-6), ProNEx - Fapesp/CNPq Proc. No. 2003/09925-5 and Fapesp (Proc. 07/54282-6), Brazil.

References

1. Adi, S.S., Braga, M., Fernandes, C.G., Ferreira, C.E., Martinez, F.H.V., Sagot, M.-F., Stefanes, M.A., Tjandraatmadja, C., Wakabayashi, Y.: Repetition-free longest common subsequence. In: Proc. IV Latin-American Algorithms, Graphs and Optimization Symposium (to appear) (2007)
2. Bonizzoni, P., Della Vedova, G., Dondi, R., Fertin, G., Vialette, S.: Exemplar longest common subsequence. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3992, pp. 622–629. Springer, Heidelberg (2006)
3. Chvátal, V.: On certain polytopes associated with graphs. J. Combinatorial Theory Ser. B 18, 138–154 (1975)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
5. Fulkerson, D.R.: Anti-blocking polyhedra. J. Combinatorial Theory Ser. B 12, 50–71 (1972)
6. Grötschel, M., Lovász, L., Schrijver, A.: The ellipsoid method and its consequences in combinatorial optimization. Combinatorica 1, 169–197 (1981)
7. Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization. Springer, Heidelberg (1988)
8. Lovász, L.: Normal hypergraphs and the perfect graph conjecture. Discrete Math. 2(3), 253–267 (1972)
9. Sankoff, D.: Genome rearrangement with gene families. Bioinformatics 15(11), 909–917 (1999)

Competitive Cost Sharing with Economies of Scale

Martin Hoefer[*]

Department of Computer Science, RWTH Aachen, Germany
mhoefer@cs.rwth-aachen.de

Abstract. We consider a general class of non-cooperative *buy-at-bulk cost sharing games*, in which k players must contribute to purchase a number of resources. The resources have costs and must be paid for to be available to players. Each player can specify payments and has a certain constraint on the number and types of resources that she needs to have available. She strives to fulfill this constraint with the smallest investment possible. Our model includes a natural economy of scale: for a subset of players, capacity must be installed at the resources. The cost increase for larger sets of players is composed of a fixed price $c(r)$ for each resource r and a global concave capacity function g. This cost can be shared arbitrarily between players. We consider the quality and existence of pure-strategy exact and approximate Nash equilibria. In general, prices of anarchy and stability depend heavily on the economy of scale and are $\Theta(k/g(k))$. For non-linear functions g pure Nash equilibria might not exist and deciding their existence is NP-hard. For subclasses of games corresponding to covering problems, primal-dual methods can be applied to derive cheap and stable approximate Nash equilibria in polynomial time. In addition, for singleton games optimal Nash equilibria exist. In this case expensive exact as well as cheap approximate Nash equilibria can be computed in polynomial time. Some of our results can be extended to games based on facility location problems.

1 Introduction

In this paper we consider a general class of non-cooperative *buy-at-bulk cost sharing games*, which can for instance be used to model crucial competitive cost sharing aspects of networks like the Internet, e.g. service installation, facility location or various network design problems. The formulation captures a realistic aspect of networks by including costs with economies of scale. In particular, we consider a game for k players that strive to obtain a number of resources with minimum investment. There is a set of resources, and each resource has a certain cost. Each player picks as a strategy a function that specifies their offer to each resource. If the sum of offers made by a set of players exceeds the resource cost, it is considered available for these players. For each player there is a constraint on the number and types of resources that must be available for her. She strives to fulfill this constraint with minimum total investment in her strategy. A resource becomes more expensive when it shall be available to a larger

[*] Supported by DFG through German excellence cluster UMIC at RWTH Aachen.

set of players. In particular, if resource r is available to a set of i players, the cost is $c(r, i) = c(r)g(i)$, in which $c(r)$ is a fixed cost and g is a non-decreasing and concave function, which is used for every resource r. A variety of problems, e.g. buy-at-bulk variants of set cover, facility location, and network design, can be turned into a game with the help of this model.

We first study our games with respect to the existence of pure-strategy exact Nash equilibria. We characterize prices of anarchy [17] and stability [2], which measure the cost of the worst and best Nash equilibria in terms of the cost of a socially optimum solution, respectively. We also consider a situation, in which a central institution with some means to influence agent behavior tries to induce a state that is as cheap and stable as possible. This poses a two-parameter optimization problem captured by the notion of (relative) (α, β)-approximate Nash equilibria. These are states, in which the equilibrium condition is relaxed by a factor of α and that represent a β-approximation to the socially optimum cost. We refer to α as the *stability ratio* and β as the *approximation ratio*. In accordance with previous work we consider properties of games, in which player constraints are equivalent to well-known covering and facility location problems. Our interest is to investigate the influence of the function g on the efficiency and computational complexity of exact and approximate Nash equilibria.

Related Work. There are a number of related game-theoretic models. Cooperative games have been studied quite intensively in the past (see [9, 11] and the references therein). In [9] the authors prove that the core of cooperative games based on covering and packing integer programs is non-empty if and only if the integrality gap is 1. They also show results on polynomial time computability of core solutions in a number of special cases. In [11] similar results are shown for class of cooperative facility location games. Some of these games have also been analyzed with respect to mechanism design. In addition, cost sharing mechanisms have been considered for games based on set cover and facility location. The authors in [10] presented strategyproof mechanisms for set cover and facility location games. For set cover games this work was extended [20, 18] to different fairness aspects and formulations with items or sets being agents, for facility location games computing cross-monotonic cost sharing schemes was considered in [19], and in [16] lower bounds on their budget-balance were provided. In contrast, our approach is an extension of non-cooperative games, which were first studied in [3] in a Steiner forest network design context. Recent work [5, 15, 14] presented extended results for exact and approximate Nash equilibria in covering and facility location games. Prices of anarchy and stability in these games are generally as large as $\Theta(k)$. For singleton games, in which each player is interested only in a single element, however, optimal Nash equilibria exist. In [5, 14] we proved the applicability of primal-dual methods to derive cheap and stable approximate Nash equilibria. None of these previous models, however, considers the influence of different economies of scale.

Starting with [4] network design problems with economies of scale became a vivid area of research. Typically, there are a number of source-sink pairs with demands that must be routed by an unsplittable flow. Edge and/or vertex costs increase with the demand routed over them. Recently, polylogarithmic approximation algorithms [6, 7] and logarithmic hardness results for general resource costs [1] were derived. For special cases, e.g. single-souce or rent-or-buy problems [12] there exist constant-factor approximation algorithms. This is also the case for unit-demand metric facility location [13].

Our Contribution. Buy-at-bulk investment games studied in this paper are a new general model to consider cost sharing in optimization problems with economies of scale. In addition, as an extension they address a frequent criticism to previous cost sharing games [3,15,14,5], which in the following we will call *regular cost sharing games*. In regular games, only a fixed cost for each resource must be paid for to make it available to *every* player, *no matter* whether she contributes or not. Hence, the game inherently allows *free riders* who can obtain a resource for free. This problem has been addressed e.g. in [2,8] by fixing a Shapley-value cost sharing. In contrast, our model allows smaller groups of players to obtain the resource at cheaper costs. This creates a force on every player to contribute for availability. The severeness of this force depends on the number of players that request a resource and is dynamically adjusted by g. Some undesirable properties of the game like a high price of anarchy are directly affected by this, the price of anarchy is exactly $\frac{k}{g(k)}$. Other properties are independent of this adjustment, e.g. for any non-linear g there are games without Nash equilibria. The price of stability is as large as $\Theta\left(\frac{k}{g(k)}\right)$, and it is NP-hard to decide the existence of Nash equilibria. Interestingly, some upper bounds on approximate Nash equilibria for regular games can be extended to hold for buy-at-bulk games. There are (f, f)-approximate Nash equilibria for set cover games, where f is the maximum frequency of any element in the sets. If each player wants to cover exactly one element, optimal Nash equilibria exist, and $(1+\epsilon, \beta)$-approximate Nash equilibria can be obtained in polynomial time by a local search from any β-approximate starting state. In addition, we provide a procedure to find an exact Nash equilibrium in polynomial time, which was not known before even for regular singleton games. A number of these results directly translate to a class of buy-at-bulk investment games for facility location. Due to space limitations some of the proofs are shortened or omitted.

2 Model and Basic Properties

In a buy-at-bulk cost sharing game there is a set $[k]$ of k non-cooperative players and a set R of resources. Each resource $r \in R$ has a *fixed cost* $c(r) \geq 0$. In addition, there is a function $g : \mathbb{N} \to \mathbb{R}_+^0$, which is non-negative, non-decreasing, concave, and has $g(0) = 0$ and $g(1) > 0$. We normalize the function to obey $g(1) = 1$. For convenience, we use $\mu(i) = g(i) - g(i-1)$, which is non-increasing and non-negative for all $i \geq 1$. The *bundle cost* of resource r is $c(r, i) = c(r)\, g(i)$. A strategy s_p of a player p is a function $s_p : R \to \mathbb{R}_+^0$ to specify her non-negative payment to each resource. A *state* is a vector $s = (s_1, \ldots, s_k)$ with a strategy for each player. We denote by s_{-p} the same vector without s_p. A resource r is *available* to a player p if there is a subset $p \in Q \subset [k]$ of players such that they purchase the corresponding bundle cost, i.e. $\sum_{q \in Q} s_q(r) \geq c(r, |Q|)$. For a player p we use $\rho_p(s)$ to denote the set of her available resources, and we drop the argument whenever context allows. Each player p has a player-specific *constraint* on ρ_p, which has a covering aspect in the sense that it can never be violated by having *additional* resources available to the ones required. If ρ_p in the current state s does not fulfill the constraint, we assume that the player is penalized with a prohibitively large cost, i.e. for her *individual cost* $c_p(s) = +\infty$. Otherwise, if her constraint is satisfied, the individual cost is her total investment $c_p(s_p, s_{-p}) = \sum_{r \in R} s_p(r)$. A player wants to minimize her individual cost, so she strives to fulfill her constraint with ρ_p at the least possible investment. A *Nash equilibrium* (denoted NE) is a state, in which no player can reduce her individual cost by changing her strategy. We restrict our attention to pure states in this paper and leave a deeper study of mixed NE for future work. As *social cost* of a state s of the game we use the sum of individual costs $c(s) = \sum_{p \in [k]} c_p(s)$. A (α, β)-approximate Nash equilibrium (denoted (α, β)-NE) is a state, in which no player can reduce her individual cost by a factor of more than α, and for which the social cost is a β-approximation to the minimum social cost over all states of the game. A social optimum state minimizing social cost will be denoted s^* throughout.

In a NE and in a social optimum state s^* the available resources for each player satisfy her constraints. Also, in NE and s^* due to concavity of g, there is a unique maximal set of players (denoted Q_r), for which the resource is available. This set includes as subsets all other sets of players, for which the resource is available. In NE no subset of i players will contribute more than $c(r, i)$ to any resource r. The strategies exactly purchase the bundle cost $c(r, |Q|)$ of every resource. Thus, a NE s represents a cost sharing of the set of resources. This property can be assumed for s^* as well, because here the cost distribution is irrelevant. Finding s^* is equivalent to finding a solution to the underlying buy-at-bulk minimization problem given by satisfying all player constraints at minimum total cost. In this problem, a feasible solution is a vector that indicates for each player, which resources are available to him, such that all constraints are satisfied.

Finally, the function $g(i) \in [1, i]$ for all $i \geq 1$. Previously considered regular cost sharing games were buy-at-bulk games with $g(i) = 1$ for all $i \geq 1$ [5, 14, 15, 3]. When referring to games in this paper - e.g. vertex cover games - we generally mean the buy-at-bulk version. It is explicitly mentioned when regular games are under consideration.

2.1 Covering and Facility Location

The definition allows a variety of games to be defined in this framework. A simple class is a (buy-at-bulk) vertex cover game on an undirected graph $G = (V, E)$. The resources $R = V$, and each player corresponds to a subset of edges $E_p \subset E$. Her constraint is satisfied, if for each edge there is at least one incident vertex available to her. In this way we generalize to set multi-cover games. There is a set of elements E, and the resources are given by $R = \mathcal{M} \subseteq 2^E$ of subsets $M \in \mathcal{M}$, such that $M \subseteq E$. Each player corresponds to a subset $E_p \subseteq E$ of elements, and there is a number $b(e) > 0$ for each $e \in E$. Player p is satisfied if for each $e \in E_p$ there are at least $b(e)$ sets available to her that include e.

Facility location games can be obtained as follows. We are given two sets T of terminals and F of facilities. The resources are facilities and connections, i.e., $R = F \cup (T \times F)$. A player p corresponds to a subset of terminals $T_p \subseteq T$. She strives to connect her terminals to facilities. As both the connections and the facilities are resources, they both generate a cost. We will refer to them as connection and opening costs, respectively. The constraint of a player p is satisfied if for each of her terminals

$t \in T_p$ at least one connection (t, f) and the corresponding facility $f \in F$ are available to her. In *metric* games the connection costs satisfy the triangle inequality.

3 Cost and Complexity of Nash Equilibria

In this section we consider the behavior of prices of anarchy and stability in the game and the hardness of finding NE. Our first result concerns the price of anarchy.

Theorem 1. *The price of anarchy in the buy-at-bulk cost sharing game is exactly* $k/g(k)$.

Proof. First, we prove the lower bound. Consider a vertex cover game on a star network, in which every player owns a single edge and each vertex v has fixed cost $c(v) = 1$. If every player contributes exactly the cost of the leaf node incident to her edge, a NE of cost k evolves. The optimum solution, however, consists of the center vertex v and has bundle cost $c(v, k) = g(k)$. This proves that the price of anarchy is at least $k/g(k)$.

For the upper bound consider any NE s of any buy-at-bulk cost sharing game with strategies s_p. In addition, let ρ_p^- be a set of resources for player p, which has minimum total fixed cost. Now consider a social optimum state s^*. Denote by ρ_p^* a subset of minimum cost of the available resources of player p in s^*, which suffices to satisfy her constraint. It is obvious that for the fixed cost

$$\sum_{r \in \rho_p^-} c(r) \leq \sum_{r \in \rho_p^*} c(r). \tag{1}$$

The concavity of g ensures that with increasing demands for resources in ρ_p^-, the cost to be paid for by player p can only decrease. Hence, it becomes ever more attractive for p to deviate to a strategy, which contributes only to ρ_p^-. However, as s is a NE, the fixed cost of ρ_p^- is an upper bound on current total contribution of p in s, i.e. $\sum_{r \in R} s_p(r) \leq \sum_{r \in \rho_p^-} c(r)$. Since s is a NE, the cost of the purchased resources must be fully paid for. Using the bound from (1) we get

$$\sum_{p \in [k]} \sum_{r \in R} s_p(r) \leq \sum_{p \in [k]} \sum_{r \in \rho_p^-} c(r) \leq \sum_{p \in [k]} \sum_{r \in \rho_p^*} c(r). \tag{2}$$

Consider the following procedure of constructing a lower bound on the cost of the social optimum solution. Iteratively add players and the cost of their available resources ρ_p^* to the solution. The presence of the i-th player on ρ_i^* adds at least a cost $\mu(i) \sum_{r \in \rho_i^*} c(r)$ to the cost of s^*. As μ is monotonic decreasing, we can lower bound $c(s^*)$ by

$$\sum_{i=1}^{k} \mu(i) \sum_{r \in \rho_i^*} c(r) \leq c(s^*). \tag{3}$$

Note that the cost of the resources is determined by the final set Q_r, and this is independent of the ordering in which players are considered. Hence, the value of this lower bound is the same for any ordering of the players chosen. By making $k - 1$ cyclic rotations of an initial ordering of players, we ensure that each player appears at each position i exactly once. Adding all resulting inequalities (3) we get $\sum_{p\in[k]}\sum_{i=1}^{k}\mu(i)\sum_{r\in\rho_p^*}c(r) = g(k)\sum_{p\in[k]}\sum_{r\in\rho_p^*}c(r) \le kc(s^*)$, and together with (2) this proves the theorem:

$$c(s) = \sum_{p\in[k]}\sum_{r\in R} s_p(r) \le \sum_{p\in[k]}\sum_{r\in\rho_p^*} c(r) \le \frac{k}{g(k)}\, c(s^*). \qquad \square$$

In fact, our proof bounds the price of anarchy for both, pure and mixed NE. If for a game $g(k) = k$, the game exhibits a decomposition property that allows for optimal NE. The previous theorem states that every NE is a social optimum. The reverse is also true, i.e. in this case there is always an optimum NE. However, once g is sublinear, then for a vertex cover game with sufficiently large number of players, there is no NE.

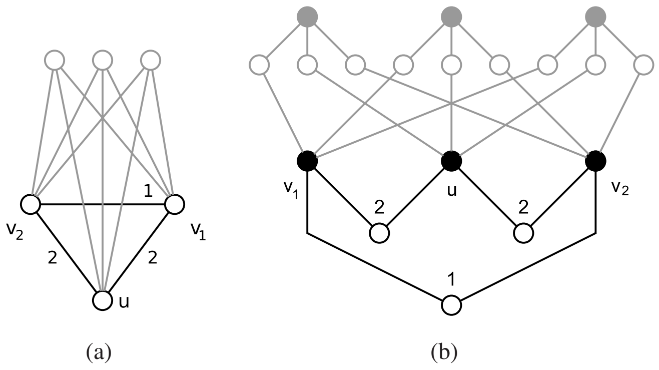


Fig. 1. (a) Vertex cover game without a NE. Edge labels indicate player ownership. Grey parts are introduced when considering auxiliary players to deal with arbitrary values of k_0. (b) Transformation into a facility location game. Filled vertices are facilities, empty vertices are terminals. Labels of terminals indicate player ownership.

Lemma 1. *If $g(i) = i$ for $i \le k_0$ and $g(i) < i$ for $i > k_0$, then for any $k > k_0$ there is a vertex cover game with k players without a Nash equilibrium.*

Consider the game in Figure 1(a) and $k_0 = 1$. Intuitively, whenever player 1 contributes the fixed cost to some vertex v_1 or v_2, player 2 is motivated to contribute to bundles of v_1 and v_2. In particular, she will purchase the fixed cost of the other vertex. This gives player 1 an incentive to remove payments, which gives player 2 an incentive to purchase vertex u. While this is not a formal argument, it can be verified that for each possible feasible solution no stable cost-sharing can be obtained. The transition to arbitrary k_0 uses additional $k_0 - 1$ *auxiliary players*. These players own a star with an expensive center and u, v_1, v_2 as leaves. They never contribute to the center and simply serve to "boost" the dynamics on the original game into the region, where the drop in function g occurs. Hence, as soon as players can profit from the investment of other players, they might not be able to agree upon a set of resources to purchase. Based on this observation we can show that given any fixed, non-linear function g, there is a class of games with sufficiently many players, in which determining existence of a NE is NP-hard. In addition, the price of stability can be as high as $\Theta\left(\frac{k}{g(k)}\right)$.

Theorem 2. *Given any non-linear function g, for which $g(i) = i$ for $i \leq k_0$ and $g(i) < i$ for $k > k_0$, then for each $k > k_0$ there is a class of vertex cover games with g and k players, for which it is* NP*-hard to determine the existence of a Nash equilibrium.*

Theorem 3. *For vertex cover games the price of stability is in $\Theta\left(\frac{k}{g(k)}\right)$.*

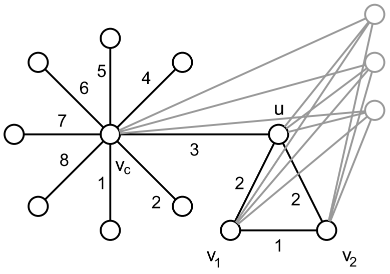


Fig. 2. Vertex cover game, in which the price of stability is $\Theta\left(\frac{k}{g(k)}\right)$. Edge labels indicate player ownership. Grey parts are introduced when considering auxiliary players to deal with arbitrary values of k_0.

Proof. Consider the game in Figure 2. Suppose every leaf vertex of the star and the star center v_c have constant fixed cost of at most $1 + \mu(k_0 + 1)$. The fixed cost of v_1 and v_2 are 1, for u it is $2 > c(u) > 1 + \mu(k_0 + 1)$. There are $k_0 - 1$ auxiliary players $k-k_0+2, \ldots, k$, and each has a star centered at an additional vertex w_p. The cost $c(w_p)$ is prohibitively high, so these players will boost the game to a range where g becomes sublinear. Now suppose there is at least one of the players $1, 2, 4, \ldots, k - k_0 + 1$, who strives to make v_c available to her. Then there are at least k_0 players, who pay a cost of $c(v_c, k_0)$ for v_c. Player 3 will contribute at most $c(v_c)\mu(k_0 + 1) = (1 + \mu(k_0 + 1))\mu(k_0 + 1) < c(u)\mu(k_0 + 1)$ to u. Thus, player 2 has to invest at least $c(u)$ to make vertex u available. As previously noted there can be no NE in this case. Thus, none of the players $1, 2, 4, \ldots, k_0 + 1$ shall make star center v_c available to her. Then player 3 can contribute $c(v_c)$ to a bundle cost of vertex u. Player 2 can add less than $1 + \mu(k_0 + 1)$ to u, and together with the auxiliary players this purchases the bundle cost of $c(u, k_0 + 1)$. Note that player 1 sticks to purchasing one of v_1 and v_2, and the remaining edges of the star can be covered by purchasing the leaf vertices. A NE of cost at least $(1 + \mu(k_0 + 1))k + 1 + (\mu(k_0 + 1))^2 + \mu(k_0 + 1) + 3(k_0 - 1)$ evolves. In the social optimum, however, all players $1, \ldots, k_0 + 1$ contribute to v_c yielding a state of cost at most $(1 + \mu(k_0 + 1))g(k) + 2 + \mu(k_0 + 1) + 4(k_0 - 1)$. For fixed g, parameter k_0 is a constant, and the ratio grows with $k/g(k)$. □

Note that any vertex cover game can be translated easily to a metric facility location game, which is equivalent in terms of the structure of NE. We replace each edge $e = (u, v)$ by a terminal t_e and two connections (t_e, u) and (t_e, v) of connection cost $c_{max} = \max_{v \in V} c(v)$. This creates the set of terminals. The former set of vertices becomes the set of facilities. For the remaining connections between facilities and terminals we assume a cost given by the shortest path metric, i.e. they are at least $3c_{max}$ (see Figure 1(b) for an example). Observe that a NE for the facility location game provides a NE for the corresponding vertex cover game and vice versa.

Corollary 1. *If $g(i) = i$ for $i \leq k_0$ and $g(i) < i$ for $i > k_0$, then for any $k > k_0$ there is a class of metric facility location games with g and k players, for which it is* NP*-hard to determine the existence of a Nash equilibrium.*

4 Approximate Nash Equilibria

In this section we consider set cover games with $b(e) = 1$ for all elements $e \in E$. While the lower bounds shown for vertex cover games extend to this case, it is possible to obtain (f, f)-NE in polynomial time, in which $f = \max_{e \in E} |\{M \in \mathcal{M}, e \in M\}|$ denotes the maximum *frequency* of any element in the sets.

Algorithm 1. (f, f)-NE for set cover games

```
1 $s_p(M) \leftarrow 0$ for all players $p$ and sets $M$
2 $\gamma_p(e) \leftarrow 0$ for all players $p$ and elements $e$
3 for every player $p = 1, \ldots, k$ do
4     Set $c^p(M) = \min_Q \{c(M, |Q| + 1) - \sum_{q \in Q} s_q(M)\}$ for $Q \subseteq [p - 1]$ and all $M$
5     while there is an uncovered element $e \in E_p$ do
6         Let $\gamma_p(e) \leftarrow \min_{e \in M} c^p(M)$
7         Increase payments: $s_p(M) \leftarrow s_p(M) + \gamma_p(e)$ for all $M$ with $e \in M$
8         Add all purchased sets to the cover
9         Reduce set costs: $c^p(M) \leftarrow c^p(M) - \gamma_p(e)$ for all $M$ with $e \in M$
```

Theorem 4. *Algorithm 1 returns a (f, f)-approximate Nash equilibrium for set cover games in polynomial time.*

Proof. The algorithm can be implemented to run in polynomial time. In line 4 we take all previous contributions into account and determine a set of players $Q \cup p$, for which the missing contribution to the bundle cost is minimal. The set Q is a subset of $[p-1] = 1, \ldots, p - 1$, because for all other players all contributions are still 0. We start with $Q = \emptyset$ and add players $q < p$ in non-increasing order of the contributions $s_q(M)$. This yields the desired set Q.

Our algorithm represents an adjustment of the primal-dual algorithm for minimum set cover (see for instance [21, chapter 15]). An approximation guarantee of f for the buy-at-bulk set cover problem has most likely been observed before, so a proof is omitted. For the stability ratio, we consider the p-th player after the execution of the algorithm and her best move taking into account the payments of all other players $q \neq p$. For that purpose, we consider for each set M the cost $c'(M) = \min_{Q \subset [k], p \neq Q} c(M, |Q| + 1) - \sum_{q \in Q} s_q(M)$. We have to show that the sum of the payments of player p is not greater than f times the cost of the cheapest set cover of E_p with respect to the costs c'. From the algorithm and the fact that bundle costs are concave we know that $s_p(M) \leq c'(M)$. Also from the algorithm, we know that for any set M that includes one or more elements of E_p, we have $s_p(M) = \sum_{e \in M \cap E_p} \gamma_p(e)$, so for any such M we have $\sum_{e \in M \cap E_p} \gamma_p(e) \leq c'(M)$. Now let us consider a minimum cost set cover $\mathcal{R}_p^*$ of E_p with respect to c'. We have: $\sum_{M \in \mathcal{R}_p^*} \sum_{e \in M \cap E_p} \gamma_p(e) \leq \sum_{M \in \mathcal{R}_p^*} c'(M) = c'(\mathcal{R}_p^*)$. Since $\mathcal{R}_p^*$ is a set cover of E_p, the charge $\gamma_p(e)$ of each element e in E_p is counted at least once in the left-hand side above. Hence $\sum_{e \in E_p} \gamma_p(e) \leq \sum_{M \in \mathcal{R}_p^*} \sum_{e \in M \cap E_p} \gamma_p(e) \leq c'(\mathcal{R}_p^*)$. Now we can conclude $\sum_{M \in \mathcal{M}} s_p(M) = f \sum_{e \in E_p} \gamma_p(e) \leq f c'(\mathcal{R}_p^*)$, which proves the theorem. □

In the special case of vertex cover ratios of $f = 2$ is tight even for regular vertex cover games [5]. The analysis cannot be strengthened to a ratio depending on g, because stability and approximation ratio coincide for single player games. For linear g the greedy algorithm achieves logarithmic stability and approximation ratio, but for regular set cover games this algorithm has an unbounded stability ratio [14]. It is an interesting open problem to obtain a procedure with improved bounds for intermediate functions g.

5 Single Element Players

In the previous section we showed that vertex cover games, in which each player owns at most two edges, might have no NE. Now we consider singleton set multi-cover games, in which each player has only a single element. For these games a NE always exists and can be found in polynomial time.

Theorem 5. *Algorithm 2 returns an exact Nash equilibrium for singleton set multi-cover games in polynomial time.*

Algorithm 2. Exact NE for singleton set multi-cover games

```
1 $d_M \leftarrow 1$ for all sets $M$
2 Construct $G_s = (\mathcal{M}, A)$ with $(M_1, M_2) \in A$ iff $M_1 \cap M_2 \neq \emptyset$ and
  $\mu(M_1, d_{M_1}) < \mu(M_2, d_{M_2})$
3 while there are remaining players do
4     for every remaining player $p$ do
5         if element $e$ of $p$ is included in exactly $b(e)$ sets $\mathcal{M}_e$ then
6             Assign $p$ to contribute $s_p(M) = c(M, d_M + 1)$ to all these sets $M \in \mathcal{M}_e$
7             Increase $d_M \leftarrow d_M + 1$ and drop $p$ from consideration
8             Adjust the arc set of $G_s$ for the new values of $d_M$
9     Find a sink in $G_s$ and drop the corresponding set from consideration
```

Proof. Clearly, Algorithm 2 can be implemented to run in polynomial time. A set M_1 *dominates* a set M_2 iff there is a player who prefers M_1 over M_2 with the bundle costs given with d_{M_1} and d_{M_2}. The algorithm constructs and maintains a directed acyclic graph G_s, which contains a directed edge between sets M_1 and M_2 iff M_1 dominates M_2. A set M that is dropped from consideration represents a sink in G_s. Then for each remaining player with $e \in M$ it is dominated by all remaining sets that contain her element. None of these players will contibute to M, as they have a cheaper alternative to cover their element. As no contribution will be assigned to M after it has been dropped, no player wants to contribute to sets that were dropped before she was dropped. When player p gets dropped, she is left with the set $\mathcal{M}_e$ of exactly $b(e)$ sets to cover e. The previous arguments show that she cannot profit from contributing to any other sets that contain her element. This is also true for the sets in $\mathcal{M}_e$. Consider another player q, who is assigned to contribute to $M \in \mathcal{M}_e$ after p has been dropped. q will only pay a cost representing the concave increase in bundle cost with p already counted towards d_M. Hence, there is no subset of players whose payments allow p to lower her contribution to $\mathcal{M}_e$. Thus, each player plays a best repsonse. This proves the theorem. □

Unfortunately, the proposed algorithm can compute worst-case NE, whose cost is a factor arbitrarily close to $\frac{k}{g(k)}$ worse than $c(s^*)$. In contrast, there are social optimal NE in every singleton set multi-cover game. Computing them is NP-hard, but with a local search procedure we can obtain near-stable and near-optimal approximate NE. The arguments can transfered to buy-at-bulk versions of *connection-restricted facility location* (CRFL) games [14]. Proofs are omitted due to space limitations.

Theorem 6. *For singleton set multi-cover and singleton CRFL games the price of stability is 1. For any constant $\epsilon > 0$, a $(1+\epsilon, \beta)$-approximate Nash equilibrium can be obtained in polynomial time from any state representing a β-approximation to the optimum social cost.*

References

1. Andrews, M.: Hardness of buy-at-bulk network design. In: Proc. 45th FOCS, pp. 115–124 (2004)
2. Anshelevich, E., Dasgupta, A., Kleinberg, J., Roughgarden, T., Tardos, É., Wexler, T.: The price of stability for network design with fair cost allocation. In: Proc. 45th FOCS, pp. 295–304 (2004)
3. Anshelevich, E., Dasgupta, A., Tardos, É., Wexler, T.: Near-optimal network design with selfish agents. In: Proc. 35th STOC, pp. 511–520 (2003)
4. Awerbuch, B., Azar, Y.: Buy-at-bulk network design. In: Proc. 38th FOCS, pp. 542–547 (1997)
5. Cardinal, J., Hoefer, M.: Selfish Service Installation in Networks. In: Spirakis, P.G., Mavronicolas, M., Kontogiannis, S.C. (eds.) WINE 2006. LNCS, vol. 4286, pp. 174–185. Springer, Heidelberg (2006)
6. Chekuri, C., Hajiaghayi, M.T., Kortarz, G., Salavatipour, M.: Approximation algorithms for non-uniform buy-at-bulk network design. In: Proc. 47th FOCS, pp. 677–686 (2006)
7. Chekuri, C., Hajiaghayi, M.T., Kortsarz, G., Salavatipour, M.: Approximation algorithms for node-weighted buy-at-bulk networks. In: Proc. 18th SODA (2007)
8. Chen, H.-L., Roughgarden, T.: Network design with weighted players. In: Proc. 18th SPAA, pp. 29–38 (2006)

9. Deng, X., Ibaraki, T., Nagamochi, H.: Combinatorial optimization games. In: Proc 8th SODA, pp. 720–729 (1997)
10. Devanur, N., Mihail, M., Vazirani, V.: Strategyproof cost-sharing mechanisms for set cover and facility location problems. Decision Support Systems 39(1), 11–22 (2005)
11. Goemans, M., Skutella, M.: Cooperative facility location games. J Algorithms 50(2), 194–214 (2004)
12. Gupta, A., Kumar, A., Pál, M., Roughgarden, T.: Approximation via cost sharing: Simpler and better approximation algorithms for network design. J ACM 54(3), 11 (2007)
13. Hajiaghayi, M.T., Mahdian, M., Mirrokni, V.: The facility location problem with general cost functions. Networks 42(1), 42–47 (2003)
14. Hoefer, M.: Non-cooperative facility location and covering games. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 369–378. Springer, Heidelberg (2006)
15. Hoefer, M.: Non-cooperative tree creation. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 517–527. Springer, Heidelberg (2006)
16. Immorlica, N., Mahdian, M., Mirrokni, V.: Limitations of cross-monotonic cost sharing schemes. In: Proc. 16th SODA, pp. 602–611 (2005)
17. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
18. Li, X., Sun, Z., Wang, W.: Cost sharing and strategyproof mechanisms for set cover games. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 218–230. Springer, Heidelberg (2005)
19. Pál, M., Tardos, É.: Group strategyproof mechanisms via primal-dual algorithms. In: Proc. 44th FOCS, pp. 584–593 (2003)
20. Sun, Z., Li, X., Wang, W., Chu, X.: Mechanism design for set cover games when elements are agents. In: Megiddo, N., Xu, Y., Zhu, B. (eds.) AAIM 2005. LNCS, vol. 3521, pp. 360–369. Springer, Heidelberg (2005)
21. Vazirani, V.: Approximation Algorithms. Springer, Berlin (2000)

Emergency Connectivity in Ad-Hoc Networks with Selfish Nodes

George Karakostas[1,2,⋆] and Euripides Markou[2,3,⋆⋆]

[1] Department of Computing & Software.
[2] School of Computational Engineering & Science.
McMaster University, 1280 Main Street West, Hamilton, Ontario L8S 4K1, Canada
[3] Department of Computer Science, University of Ioannina, Greece
karakos@mcmaster.ca, emarkou@cs.uoi.gr

Abstract. Inspired by the CONFIDANT protocol [1], we define and study a basic reputation-based protocol in multihop wireless networks with selfish nodes. Its reputation mechanism is implemented through the ability of any node to define a threshold of tolerance for any of its neighbors, and to cut the connection to any of these neighbors that refuse to forward an amount of flow above that threshold. The main question we would like to address is whether one can set the initial conditions so that the system reaches an equilibrium state where a non-zero amount of *every* commodity is routed. This is important in emergency situations, where all nodes need to be able to communicate even with a small bandwidth. Following a standard approach, we model this protocol as a game, and we give necessary and sufficient conditions for the existence of non-trivial Nash equilibria. Then we enhance these conditions with extra conditions that give a set of necessary and sufficient conditions for the existence of connected Nash equilibria. We note that it is not always necessary for all the flow originating at a node to reach its destination at equilibrium. For example, a node may be using unsuccessful flow in order to effect changes in a distant part of the network that will prove quite beneficial to it. We show that we can decide in polynomial time whether there exists a (connected) equilibrium without unsuccessful flows. In that case we calculate (in polynomial time) initial values that impose such an equilibrium on the network. On the negative side, we prove that it is NP-hard to decide whether a connected equilibrium exists in general (i.e., with some nodes using unsuccessful flows at equilibrium).

1 Introduction

In recent years there has been a great effort in designing robust and efficient wireless networks of devices that take upon themselves certain network responsibilities that used to be the responsibilities of a central network designer in traditional network design. For example, in ad-hoc networks the topology of the

⋆ Research supported by an NSERC Discovery Grant and MITACS.
⋆⋆ Research supported by MITACS.

network is the result of cooperation amongst the nodes themselves: in a multi-hop wireless network, a successful transmission between a pair of nodes requires the cooperation of intermediate nodes in order for the transmitted packets to reach their destination. While this may be guaranteed in networks with a central authority forcing the nodes to cooperate, in the absence of such an authority cooperation may not be guaranteed. This is due to the *selfishness* of each node, i.e., the effort by the node to maximize its own utility without caring about the results of its actions on the overall network-wide outcome. For example, if battery life is a valuable resource for a node, forwarding packages between two other nodes consumes energy that doesn't result in any kind of pay-off for this node, and as a result it may decide to stop cooperating in forwarding packages for others. If this behavior prevails throughout the whole network, it may eventually result in zero throughput for everybody, a phenomenon better known as the "Tragedy of the Commons" [5]. To cope with this problem one can offer *incentives* to nodes such as rewards for their cooperation or punishment for non-cooperation.

The two most commonly proposed forms of incentives are micro-payments, and reputation-based mechanisms. One of the main motivation for developing them is the desire of the network designer to not permanently punish a misbehaving node, but 're-socialize' it if it changes its uncooperative behavior.

Micro-payment schemes are based on the concept of distribution of credit to nodes, so that nodes are compensated for their cooperation by (virtual) credit payments, that they can then use to pay intermediate nodes for forwarding their own traffic. Hence if a node is consistently uncooperative, it will run out of credit and will have to stop transmitting. Usually, the distribution and/or the expenditure of credit is controlled by a central authority. Examples of such protocols are [2,10,11,3].

Reputation-based systems are based on lists that the nodes keep on the *reputation* of their neighbors, i.e., the fraction of packets forwarded by them. They use this information in order to decide how much traffic they should forward towards their neighbors. This may be decided in a *Tit-for-Tat* fashion, i.e., when a node has to relay a packet on behalf of a neighbor, it does so with the same probability with which this neighbor forwards its own packets (see [8,9] for examples of such mechanisms). Or, the amount to forward can be decided according to (centralized or local) *ratings tables*, that give the nodes an indication of the behavior of other nodes; if a node's rating of another node falls below a certain threshold, then the latter cannot be trusted to forward traffic, and therefore nothing is forwarded to it by the former, i.e., the edge connecting the two nodes is *cut* by the first node. An example of such a mechanism that actually distributes the reputation information so that each node can form its own ratings table is the CONFIDANT protocol [1]. More recent protocols [6,7] limit the distribution of reputation information only to one-hop neighbors.

Our results: In this work we address the connectivity issues arising in such reputation-based systems. More specifically, we would like to study whether it is possible in such a selfish environment to lead all nodes towards an equilibrium with good connectivity properties. In fact, we are very ambitious: we are looking for driving them towards an equilibrium that permits a non-zero quantity of *every* traffic demand to be satisfied. The reason for such a strict requirement is the fact that in an emergency situation police, firemen, emergency medical personnel, etc. should be able to communicate with each other even if the achieved bandwidth is very small (but still enough for emergency signals to be able to travel through the network). From the above, it is not at all obvious whether such a goal can be achieved, given the fact that each network node is autonomously playing a protocol game, after it's been set in its initial condition. Given the game-theoretic nature of such protocols, it is only natural to study them in terms of their (Nash) equilibrium states. Under this light, and given the rules of the game, i.e., the protocol, the most appropriate (indeed, in some cases the only) time a network designer can intervene in order to control the outcome is during the setting of the initial conditions, or, equivalently, by 'rebooting' the protocol with new initial values. This can be achieved by a separate broadcasting channel that all nodes are listening ('snooping') in, and whose packets are of the highest priority. Obviously, this is a very intruding method, and it would defy the purpose of selfishness if it were to be applied very frequently. But one does not (hopefully) expect catastrophic emergency situations to arise that frequently. Therefore broadcasting will not be used often.

Inspired by the CONFIDANT mechanism, we study a basic reputation-based system. The strategy of every node consists of the amount of traffic flow it sends to its various receivers, the routing of this flow, the amount of flow it forwards for every commodity in which it doesn't participate as a sender or a receiver, and a non-negative *threshold* value for each outgoing edge. The latter set of values is an abstraction of the reputation mechanism: if the amount of flow that is forwarded by node x to node y (including flow that originates at x), but is cut by y is more than the threshold value x has for y, then x disconnects edge (x, y). Later on, y may end up cutting flow that is less than the current threshold value of x for (x, y), in which case (x, y) reappears. The utility for every node increases with the flow originating at or destined for this node and reaches its destination, while decreases with the flow sent out or forwarded by this node (because, for example, the node has to spend battery energy to transmit).

The main drawback of this protocol is the assumption that every node has to make its strategy known to every other node. But at the same time, this complete knowledge of the game state gives great potential power to each node to affect parts of the network that are very far away, even in counter-intuitive ways, e.g., by sending flow whose sole purpose is to affect the current topology and discourage the flow of other nodes. Hence, this assumption may make our demand for complete connectivity even harder to achieve, and it may mean that things can be easier in a more restricted setting. As a first step towards achieving this goal, we are able to characterize the complexity of computing initial values that lead to a connected Nash equilibrium in our protocol. We do that, by giving necessary and sufficient conditions for the existence of *non-trivial* Nash equilibria. Then we enhance these conditions with extra conditions that give a set of necessary and sufficient conditions for the existence of *connected* Nash equilibria. Note that it is not always necessary for all the flow originating at a node to reach its destination at equilibrium. As mentioned above, a node may be using such *unsuccessful* flow in order to effect changes in a distant part of the network that will prove quite beneficial to it. We show that in case there is a connected Nash equilibrium *without unsuccessful flows*, we can calculate (in polynomial time) initial values that impose such an equilibrium on the network using linear programming. On the other hand, if the connected Nash equilibrium(-ia) exist, but nodes are allowed to use unsuccessful flows, then it is NP-hard even to decide whether an equilibrium exists.

Our results are derived using game-theoretic concepts, which is the standard approach for analyzing such protocols, modeled as games. But we emphasize that, other than the assumptions mentioned above, we don't impose any restrictions on the network topology, or any statistical distribution on the nodes' decisions.[1]

2 Model and Terminology

In this section we describe our model for the network and the protocol the nodes follow. The set of connections that can be realized is given by a directed graph $G(V, E)$. We emphasize that, depending on the current state of the game, not all these edges may be present. For every origin-destination pair (commodity) (u, v), $u, v \in V$ there is a demand $d_{(u,v)}$ that u wants to send to v. The flow is splittable, and u decides how to split and route this flow. Again, the current state of the game may not allow u to send all of $d_{(u,v)}$, so the latter serves more as an upper bound on the flow actually sent. We denote by $\mathcal{P}_i$ the set of paths connecting the i-th origin-destination pair in G, and let $\mathcal{P} := \cup_i \mathcal{P}_i$.

The current state of the network, together with the nodes' strategies are described by the following set of variables:

- $F^y_{(u,e,e',v)}$ **with** $e, e' \in E, e = (x, y), e' = (y, z), u, v, x, y, z \in V$ **and** $y \neq u, v$: This is the flow of commodity (u, v) that y receives through e, and forwards further through e'.
- $f^y_{(u,e,e',v)}$ **with** $e, e' \in E, e = (x, y), e' = (y, z), u, v, x, y, z \in V$ **and** $y \neq u, v$: This is the *decision* variable of y that sets an upper bound on the amount of flow $\sum_{g=(w,x)} F^x_{(u,g,e,v)}$ *routed through* e' that y actually forwards through e', i.e., $F^y_{(u,e,e',v)} = \min\{f^y_{(u,e,e',v)}, \sum_{g=(w,x)} F^x_{(u,g,e,v)}$ *routed through* $e'\}$ (notice that edge e' can be disconnected; in that case, what is being forwarded by y through e' is simply lost). We emphasize that $f^y_{(u,e,e',v)}$ is just the y's decision

[1] We don't assume any kind of synchronization amongst the nodes, but we do assume that the decision variables changes are instantaneous. Note that the game modeling the protocol is *not* a repeated game, and there isn't any notion of *rounds*.

variable that determines what y will do *if* there is flow from u to v which has been forwarded from x to y and needs to be forwarded through $e' = (y, z)$, while $\sum_{g=(w,x)} F^x_{(u,g,e,v)}$ is the actual flow that comes to y from x through e. So y maintains such a variable $f^y_{(u,e,e',v)}$, for every incoming edge $e = (x, y)$ and every outgoing edge $e' = (y, z)$, and every commodity (u, v).
- $O^y_{(u,e,v)}$ **with** $e \in E, e = (x, y), u, v, x, y \in V$ **and** $y \neq u, v$: This is an auxiliary variable, defined as $O^y_{(u,e,v)} = \sum_{e'=(y,z)} F^y_{(u,e,e',v)}$. It is simply the total flow of commodity (u, v) coming to y through edge e, and being forwarded by y through all its outgoing edges $e' = (y, z)$.
- $I^y_{(u,e',v)}$ **with** $e' \in E, e' = (y, z), u, v, y, z \in V$ **and** $y \neq v$: This is also an auxiliary variable, defined as $I^y_{(u,e',v)} = \sum_{e=(x,y)} F^y_{(u,e,e',v)}$. It is simply the total flow of commodity (u, v) coming to y through all its incoming edges $e = (x, y)$, and being forwarded by y through edge e'. Note that $I^y_{(y,e',v)}$ is the flow originated at y and routed through e' with destination v.
- ϵ^y_x: This auxiliary variable is defined as $\epsilon^y_x = \sum_{com.(u,v),v\neq y} (I^x_{(u,e,v)} - O^y_{(u,e,v)})$, i.e., as the part of the total flow that comes to y through e and is being blocked by y.
- $s^u_{(u,P,v)}$: This is the *decision* variable of u that determines how much flow of commodity (u, v) node u routes through path P (whether this flow amount eventually reaches v or not).
- $THR_x(y)$: This is the *decision* variable of node x that defines an upper bound on the flow forwarded by x and cut by y that x can tolerate before it cuts edge (x, y). We consider edge (x, y) disconnected when $\epsilon^y_x > 0$ AND $THR_x(y) \leq \epsilon^y_x$. Hence edge (x, y) exists in the network provided $\epsilon^y_x = 0$ OR $THR_x(y) > \epsilon^y_x$.

The following definition will be used repeatedly throughout this paper:

Definition 1. *An edge* (x, y) *is* connected *if* $\epsilon^y_x = 0$ *OR* $THR_x(y) > \epsilon^y_x$*, and* disconnected *otherwise.*

Therefore, the *strategy* of a node x is determined by the vector $(\boldsymbol{s^x}, \boldsymbol{THR_x}, \boldsymbol{f^x})$. Note that the routing of the flow x sends out is incorporated in the values for $\boldsymbol{s^x}$. Therefore x decides the following:

- Threshold $THR_x(y) \geq 0$, and hence decides whether edge (x, y) is connected or not.
- Variables ϵ^x_w, by deciding $f^x_{(u,e,e',v)}$ which, in turn, change the flows $F^x_{(u,e,e',v)}$. As a result, x decides whether edge $e = (w, x)$ is connected or not.
- The routing of the flow originating at x and its quantity, by deciding $s^x_{(x,P,y)}$ for any path P connecting x to y. But always $\sum_P s^x_{(x,P,y)} \leq d_{(x,y)}$.

We repeat that every node sees all decision variables of all other nodes, we don't assume any kind of synchronization amongst the nodes, but we do assume that the decision variables changes are instantaneous.

Definition of the utility function: Every node plays in a selfish way, i.e., so that its utility (defined below) is maximized. At any time t, we denote by $C_y^-, C_y^+, D_y^-, D_y^+$ the sets of connected incoming, connected outgoing, disconnected incoming and disconnected outgoing edges respectively, adjacent to node y. Then, for every node y its utility function is defined as follows:

$$util_t(y) = \begin{array}{c}\text{flow sent by } y\\ \text{and reached its}\\ \text{destination}\end{array} + \begin{array}{c}\text{flow received}\\ \text{by } y\end{array} - \begin{array}{c}\text{flow forwarded}\\ \text{by } y\end{array} - \begin{array}{c}\text{flow sent by } y\\ \text{and didn't reach}\\ \text{its destination}\end{array}.$$

More specifically,

$$util_t(y) = \sum_{e \in C_y^+} S_e^y + \sum_{e \in C_y^-} R_e^y - \sum_{e' \in C_y^+ \cup D_y^+} \sum_{u \neq y, v \in G} I_{(u,e',v)}^y - \left(\sum_{e' \in C_y^+ \cup D_y^+} \sum_{v \in G} I_{(y,e',v)}^y - \sum_{e \in C_y^+} S_e^y \right) \quad (1)$$

where

- S_e^y is the flow which has been sent by y (i.e. originated at y) through edge e and has reached its destination,
- R_e^y is the flow which has been received by y through edge e,
- $I_{(u,e',v)}^y$ is the flow of commodity (u, v) with $y \neq v$, and node y attempts to forward (or sent, if $u = y$) through edge e' (note that e' may be disconnected).

The intuition behind this definition of utility (which is very similar to the definition used in [1]), is that a node exchanges resource units (e.g., battery energy) for information units (i.e., packets received or sent successfully). Our assumption is that the correspondence is one for one. Different weighting of resources and information is a generalization left for future work.

Throughout this work, we use the standard definition of Nash equilibria, i.e., at equilibrium, no node gains an increase of its utility by changing its decision variables (strategy), while the other nodes maintain their own strategies. We will focus on *non-trivial* equilibria.

Definition 2. *A* trivial *equilibrium is any equilibrium with* $\boldsymbol{f^x} = \mathbf{0}, \forall x$, *and with* $s^u_{(u,(u,v),v)} = d_{(u,v)}$, $\forall$ *commodities* (u, v) *s.t.* $(u, v) \in E$ *and* $s^u_{(u,P,v)} = 0$ *otherwise.*

So from now on, whenever we write 'equilibrium' we mean 'non-trivial equlibrium', unless otherwise stated. We also assume that there is always at least one demand between non-adjacent nodes in G, since otherwise a trivial equilibrium is a connected one, and this case is not very interesting.

Definition 3. *An amount of flow with origin a node u and destination a node v routed through a path P is* successful *if it reaches node v, otherwise it is* unsuccessful*.*

3 Characterization of Nash Equilibria

In this section we give necessary and sufficient conditions for the existence of an equilibrium. Our hope will be that these conditions (probably together with additional ones) will simplify the study of connected equilibria.

Definition 4. *An unsuccessful flow Φ which has been routed through edge e is* responsible *for disconnecting edge e if e would be connected without Φ.*

We group the (non-disconnected) incoming and outgoing edges for a node x as follows:

- group 1: these edges transfer only successful flows,
- group 2: these edges transfer successful and unsuccessful flows,
- group 3: these edges transfer only unsuccessful flows.

The proof of the following Theorem appears in the full version:

Theorem 1. *The game is at an equilibrium if and only if for any node x the following conditions hold:*

1. *$\epsilon_x^y = 0$, where $g = (x, y) \in C_x^+$ (i.e., node y does not cut any flow forwarded by x through the connected edge g),*
2. *if there is a successful flow between nodes u,$v \neq y$ routed through edge $g = (x, y)$, then $THR_x(y) = 0$,*
3. *if there is no unsuccessful flow going through an edge $e = (t, x)$, then $R_e^x \geq \sum_{u \neq x,v} \sum_{g=(x,y)} F_{(u,e,g,v)}^x$ (i.e., the flow that node x receives through edge $e = (t, x)$ is not less than the total flow which is coming through e and x has to forward, if all this latter flow is successful),*
4. *for any disconnected edge $g' = (x, y') \in D_x^+$ it holds that $THR_x(y') = \epsilon_x^{y'} > 0$, node x does not send any flow through g', and the (unsuccessful) flows which are responsible for disconnecting g' are being sent by at least two nodes, other than x,*
5. *let $e = (t, x)$ be an incoming connected edge to x such that all unsuccessful flows which pass through e, have been routed through outgoing disconnected edges $g' = (x, y_i') \in D_x^+$ of x; then:*
   - *$THR_t(x) = 0$,*
   - *$R_e^x \geq \sum_{u \neq x,v} \sum_{g' \in D_x^+} F_{(u,e,g',v)}^x + \sum_{u \neq x,v} \sum_{g \in C_x^+} F_{(u,e,g,v)}^x$,*
6. *the flow that node x sends successfully through all of its (connected) outgoing edges $\Phi(x)$ is maximized over all possible routings $\boldsymbol{s^x}$,*

7. *any combination of the following possible actions taken by x cannot increase its utility:*
   (a) *disconnecting a number of edges of group 2,*
   (b) *decreasing the unsuccessful flow that x lets go through edges of group 3,*
   (c) *connecting edges $e' = (t', x) \in D_x^-$,*
   (d) *sending successful and unsuccessful flow through the outgoing edges of x,*
   (e) *increasing thresholds*

Theorem 1 is essentially a codification of all the conditions that happen simultaneously at equilibrium. But showing that such a (non-trivial) equilibrium exists (or, even more, compute it) is non-trivial. In fact, we will show that deciding the existence of an equilibrium is NP-hard. But it turns out it is much easier to check whether there is a non-trivial equilibrium with only successful flows; this can be reduced to the solution of a simple LP.

For every edge $e = (u, v)$, we set $d(e)$ equal to $d_{(u,v)}$ if commodity (u, v) exists, and 0 otherwise. Let $D := \sum_{e \in E} d(e)$. We will use the following notation:

- $e \in^* P$, when edge $e \in P$ is *not* the last edge of P,
- $e \in^0 P$, when edge $e \in P$ is the last edge of P.

In the following LP, variables $x(P)$ represent the amount of flow sent along path P:

$$\max \sum_{P \in \mathcal{P}} x(P) \quad \text{s.t.} \tag{LP-S}$$

$$\begin{aligned} \sum_{P: e \in^* P} x(P) - \sum_{P: e \in^0 P} x(P) &\leq 0 && \forall e \in E \\ \sum_{P \in \mathcal{P}_i} x(P) &\leq d_{(u_i, v_i)} && \forall i \\ x(P) &\geq 0 && \forall P \in \mathcal{P} \end{aligned}$$

Theorem 2. *A non-trivial equilibrium with only successful flows exists if and only if (LP-S) has a solution $x(P)$ with $\sum_{P \in \mathcal{P}} x(P) > D$.*

Proof: The proof appears in the full version. □

The solution of (LP-S) by standard techniques [4] implies the following

Corollary 1. *We can compute in polynomial time user strategies that are at equilibrium with only successful flows, if such an equilibrium exists.*

4 Connected Equilibria

In this section we study the following question: given an underlying network topology along with a set of demands between nodes, is it possible to assign values to the decision variables, so that the game converges to a connected equilibrium, when such an equilibrium exists?

Recall that we call the network *connected* iff a non-zero amount of every commodity reaches its destination. Therefore, if, in addition to being at equilibrium, we want the network to be connected, we have to add to Theorem 1 the condition that for every commodity (u, v), there is a successful non zero flow sent from u to v through a path P in the network. This translates to the following condition for every edge $e = (x, y)$ in path P: $THR_x(y) \geq \epsilon_x^y = 0$ AND $I_{(u,e,v)}^x > 0$ (especially when $y \neq v$, it must hold $THR_x(y) = \epsilon_x^y = 0$, as follows from condition 2 of Theorem 1).

Theorem 3. *A network is at a* connected *equilibrium if and only if in addition to the Theorem 1 conditions, for every commodity (u, v), either edge (u, v) is connected or there is a path connecting u, v, so that for every edge $e = (x, y)$ in the path it holds that $I_{(u,e,v)}^x > 0$ AND $\epsilon_x^y = 0$.*

It is easy to see that there are cases in which it is impossible for a game to converge to a connected equilibrium. For example, suppose that there is an edge $e = (x, y)$ in the network such that node x is neither a source nor a sink, and there is a commodity (u, v) such that all paths between u and v pass through e. Then it is easy to see that, in any equilibrium, there will be no flow from u to v. Indeed, suppose that there is a connected equilibrium. Hence there should be an edge $e = (t, x)$ in the network which carries some successful flow. If e carries *only* successful flow then the condition 3 of Theorem 1 would be violated. On the other hand if e carries successful *and* unsuccessful flow condition $7(a)$ would be violated since x would have a profit to disconnect edge e and gain in its utility.

As mentioned in the Introduction, the proof of existence, and the computation of strategies that lead to connected equilibria is, in general, very difficult, since we will prove in the next section that it is an NP-hard problem. But, building on the results of the previous section, we can prove the existence (or not) of a connected equilibrium with only successful flows in polynomial time, and compute strategies that achieve it. Using the characterization of such equilibria by Theorem 3, we can reduce this computation to the solution of the following extension of (LP-S):

$$\begin{aligned}
&\max \quad w && \text{s.t.} && \text{(LP-C)}\\
&\sum_{P: e \in^* P} x(P) - \sum_{P: e \in^0 P} x(P) \leq 0 && \forall e \in E\\
&\sum_{P \in \mathcal{P}_i} x(P) \leq d_{(u_i, v_i)} && \forall i\\
&\sum_{P \in \mathcal{P}_i} x(P) \geq w && \forall i\\
&x(P) \geq 0 && \forall P \in \mathcal{P}\\
&w \geq 0
\end{aligned}$$

Similarly to Theorem 2, we can prove the following

Theorem 4. *A* connected *equilibrium with only successful flows exists if and only if (LP-C) has a solution $x(P), w$ with $w > 0$.*

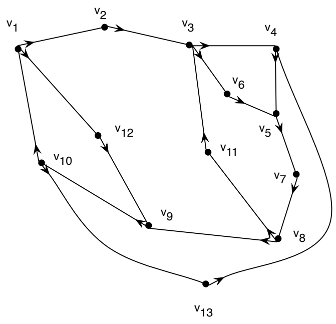


Fig. 1. A variable-subgraph

Again, the solution of (LP-C) by standard techniques [4] implies the following

Corollary 2. *We can compute in polynomial time user strategies that induce a* connected *equilibrium with only successful flows, if such an equilibrium exists.*

5 NP-hardness of Existence of a Connected Nash Equilibrium

Suppose a network is given together with a set of demands. In this section we prove that it is NP-hard to decide whether there exist values for the decision variables of the nodes so that the game converges to a connected equilibrium (that possibly uses successful and unsuccessful flows). We prove this by reduction from the satisfiability problem (SAT):

Sketch of the reduction: Given an instance I of the SAT problem we construct (in polynomial time on the number of the boolean variables) a network and a set of demands between nodes. The basic element of the construction is the *variable-subgraph* (Figure 1) which corresponds to a boolean variable of I and it is constructed in such a way, so that in any connected Nash equilibrium, in exactly one of its edges there is no successful flow at all. We then show that there is a truth assignment which satisfies an instance I of SAT if and only if there is a Nash equilibrium in the constructed network with the network being connected (i.e., for any demand there is a flow being delivered). We prove this by giving explicitly values to decision variables of the nodes so that the network is connected at a Nash equilibrium. We show that if a boolean variable A has value $FALSE$ in the truth assignment and appears as $\neg A$ in a literal of I (negative literal) then the corresponding subgraph (Figure 2a) is connected at a Nash equilibrium with only successful flows. If variable A has value $TRUE$ in the truth assignment and appears as A in a literal of I (positive literal) then the corresponding subgraph (Figure 2b) is connected at a Nash equilibrium with successful and unsuccessful flows.

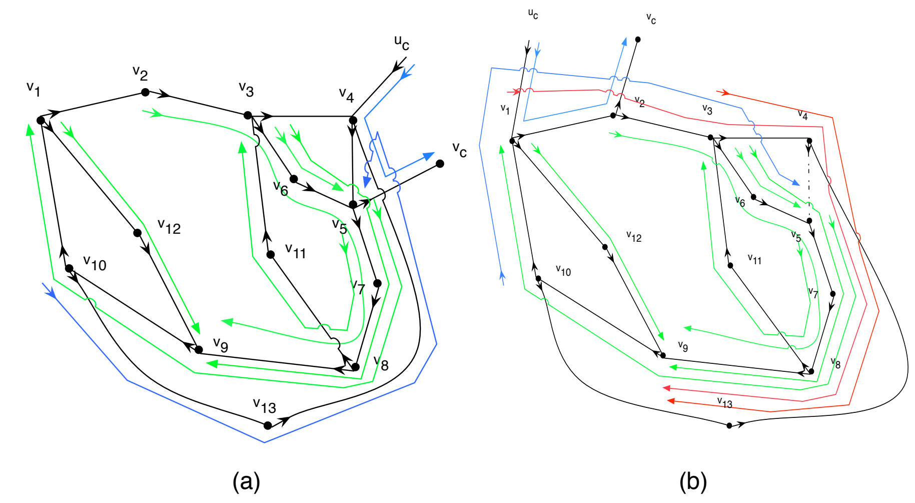


Fig. 2. (a) A negative-literal-subgraph which corresponds to a variable with value *FALSE*, with routed flow-paths. (b) A positive-literal-subgraph which corresponds to a variable with value *TRUE*, with routed flow-paths.

Theorem 5. *Given a network and a set of demands between nodes, it is NP-hard to decide whether there exist values for the decision variables of the nodes so that the game converges to a connected Nash equilibrium.*

The details and proofs appear in the full version of the paper.

6 Conclusion

The question of inducing Nash equilibria with specific attributes is a very general one, and applies to any protocol. In this work we study the property of connectivity, but other natural goals are the maximization of total utility, the maximization of the minimum demand satisfied (similar to concurrent multicommodity flow problems), the maximization of total bandwidth etc. We focused on a basic reputation-based model for ad-hoc networks, but the achievement of most of these goals remains open for this model as well. On the other hand, we were able to characterize the Nash equilibria for it in a way that allowed us to study connectivity properties in a very general setting, i.e., for general topologies and multiple commodities. We would like to combine these properties with additional ones, e.g., maximization of the minimum demand. This would involve network design decisions at the level of setting-up the topology, since there are simple examples with throughput (i.e. the minimum (over all commodities) fraction of satisfied demand) equal to $\frac{d_{min}}{(k-1)d_{max}}$, where d_{min}, d_{max} are the minimum, maximum demands respectively, and k is the number of commodities. Hence, a natural extension of our results would be to study these extra network design decisions when the installation of every new edge incurs a cost. Another natural extension would be the study of a minimal subset of nodes whose setting of initial values induces an equilibrium with the desired properties. Note that in our results we set the initial values for all nodes, thus inducing an equilibrium 'in one shot'.

References

1. Buchegger, S., Le Boudec, J.-Y.: Performance Analysis of the CONFIDANT Protocol: Cooperation Of Nodes Fairness In Dynamic Ad-hoc NeTworks. In: Proceedings of MOBIHOC 2002, pp. 226–236 (2002)
2. Buttyan, L., Hubaux, J.-P.: Stimulating Cooperation in Self-Organizing Mobile Ad Hoc Networks. In: Proceedings of ACM/Kluwer Mobile Networks and Applications, vol. 8(5), pp. 579–592 (2003)
3. Eidenbenz, S., Resta, G., Santi, P.: COMMIT: a sender-centric truthful and energy-efficient routing protocol for ad hoc networks with selfish nodes. In: Proceedings of IEEE Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN) (2005)
4. Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization. Springer, Berlin (1993)
5. Hardin, G.: The Tragedy of the Commons. Science 162(3859), 1243–1248 (1968)
6. He, Q., Wu, D., Khosla, P.: SORI: A Secure and Objective Reputation-based Incentive Scheme for Ad-hoc Networks. In: Proceedings of IEEE Wireless Communications and Networking Conference (WCNC2004), pp. 825–830 (2004)
7. Mahajan, R., Rodrig, M., Wetherall, D., Zahorjan, J.: Sustaining Cooperation in Multihop Wireless Networks. In: Proceedings of Second USENIX Symposium on Networked System Design and Implementation (NSDI 2005) (2005)
8. Milan, F., Jaramillo, J.J., Srikant, R.: Achieving cooperation in multihop wireless networks of selfish nodes. In: Proceedings of the 2006 workshop on Game theory for communications and networks (GameNets) (2006)
9. Srinivasan, V., Nuggehalli, P., Chiasserini, C.-F., Rao, R.: Cooperation in wireless ad-hoc networks. In: Proceedings of IEEE INFOCOM 2003, pp. 808–817 (2003)
10. Zhong, S., Chen, J., Yang, Y.R.: Sprite: A simple, Cheat-proof, Credit-based System for Mobile Ad-hoc Networks. In: Proceedings of IEEE INFOCOM 2003, pp. 1987–1997 (2003)
11. Zhong, S., Li, L.E., Liu, Y.G., Yang, Y.R.: On designing incentive-compatible routing and forwarding protocols in wireless ad-hoc networks: an integrated approach using game theoretic and cryptographic techniques. Wireless Networks 13(6), 799–816 (2007)

Fully-Compressed Suffix Trees

Luís M. S. Russo[1,⋆], Gonzalo Navarro[2,⋆⋆], and Arlindo L. Oliveira[1]

[1] INESC-ID, R. Alves Redol 9, 1000 LISBOA, Portugal
{lsr,aml}@algos.inesc-id.pt
[2] Dept. of Computer Science, University of Chile
gnavarro@dcc.uchile.cl

Abstract. Suffix trees are by far the most important data structure in stringology, with myriads of applications in fields like bioinformatics and information retrieval. Classical representations of suffix trees require $O(n \log n)$ bits of space, for a string of size n. This is considerably more than the $n \log_2 \sigma$ bits needed for the string itself, where σ is the alphabet size. The size of suffix trees has been a barrier to their wider adoption in practice. Recent compressed suffix tree representations require just the space of the compressed string plus $\Theta(n)$ extra bits. This is already spectacular, but still unsatisfactory when σ is small as in DNA sequences.

In this paper we introduce the first compressed suffix tree representation that breaks this linear-space barrier. Our representation requires sublinear extra space and supports a large set of navigational operations in logarithmic time. An essential ingredient of our representation is the lowest common ancestor (LCA) query. We reveal important connections between LCA queries and suffix tree navigation.

1 Introduction and Related Work

Suffix trees are extremely important for a large number of string processing problems. Their many virtues have been described by Apostolico [1] and Gusfield [2]. The combinatorial properties of suffix trees have a profound impact in the *bioinformatics* field, which needs to analyze large strings of DNA and proteins with no predefined boundaries. This partnership has produced several important results, but it has also exposed the main shortcoming of suffix trees. Their large space requirements, together with their need to operate in main memory to be useful in practice, renders them inapplicable in the cases where they would be most useful, that is, on large texts.

The space problem is so important that it has originated a plethora of research results, ranging from space-engineered implementations [3] to novel data structures to simulate it, most notably suffix arrays [4]. Some of those space-reduced variants give away some functionality in exchange. For example suffix arrays

⋆ Supported by the Portuguese Science and Technology Foundation by grants SFRH/BD/12101/2003, SFRH/BPD/34373/2006 and projects DBYeast, POSI/EIA/57398/2004 and ARN, PTDC/EIA/67722/2006.

⋆⋆ Partially funded by Millennium Institute for Cell Dynamics and Biotechnology, Grant ICM P05-001-F, Mideplan, Chile.

miss the important suffix link navigational operation. Yet, all these classical approaches require $O(n \log n)$ bits, while the indexed string requires only $n \log \sigma$ bits[1], n being the size of the string and σ the size of the alphabet. For example the human genome requires 700 Megabytes, while even a space-efficient suffix tree on it requires at least 40 Gigabytes [5], and the reduced-functionality suffix array requires more than 10 Gigabytes. This problem is particularly evident in DNA because $\log \sigma = 2$ is much smaller than $\log n$.

These representations are also much larger than the size of the *compressed* string. Recent approaches [6] combining data compression and succinct data structures have achieved spectacular results for the pattern search problem. For example Ferragina *et al.* [7] presented a compressed suffix array that requires $nH_k + o(n \log \sigma)$ bits and computes *occ* in time $O(m(1 + (\log_\sigma \log n)^{-1}))$. Here nH_k denotes the k-th order empirical entropy of the string [8], a lower bound on the space achieved by any compressor using k-th order modeling.

It turns out that it is possible to use this kind of data structures, that we will call *compressed suffix arrays*[2], and, by adding a few extra structures, support all the operations provided by suffix trees. Sadakane was the first to present such a result [5], adding $6n$ bits to the size of the compressed suffix array.

In this paper we break the $\Theta(n)$ extra-bits space barrier. We build a new suffix tree representation on top of a compressed suffix array, so that we can support all the navigational operations and our extra space fits within the sublinear $o(n \log \sigma)$ extra bits of the compressed suffix array. Our central tools are a particular sampling of suffix tree nodes, its connection with the suffix link and the lowest common ancestor (LCA) query, and the interplay with the compressed suffix array. We exploit the relationship between these actors and uncover some relationships between them that might be of independent interest.

A comparison between Sadakane's representation and ours is shown in Table 1. The result for the time complexities is mixed. Our representation is faster for the important Child operation, when $\log \sigma = o(\log \log n)$, yet Sadakane's is usually faster on the rest. On the other hand, our representation requires much less space. For DNA, assuming realistically that $H_k \approx 2$, Sadakane's approach requires $8n + o(n)$ bits, whereas our approach requires only $2n + o(n)$ bits. We choose a compressed suffix array that has the best Letter time, for $nH_k + o(n \log \sigma)$ bits. Only when $\sigma = \omega(\text{polylog}(n))$ and there are $O(nH_0) + o(n \log \sigma)$ bits available is Sadakane's compressed suffix array [9] faster at computing the Letter operation. In that case, using his compressed suffix array, Sadakane's suffix tree would work faster, while ours does not benefit from that. As such, Table 1 shows *the time complexities that can be obtained for suffix trees using the best asymptotic space achieved for compressed suffix arrays alone.* This space is optimal in the sense that no k-th order compressor can achieve asymptotically less space to represent T.

There exists a previous description [10] of a technique based on interval representation and sampling of suffix tree. However it is extremely brief and no theoretical bounds on the result are given.

[1] In this paper log stands for $\log_2$.

[2] These are also called compact suffix arrays, FM-indexes, etc., see [6].

Table 1. Comparing compressed suffix tree representations. The operations are defined along Section 2. Time complexities, but not space, are big-O expressions. Notice that $\textsc{Letter}(v, i)$ can also be computed in $O(i\psi)$ time. Also $\textsc{Child}$ can, alternatively, be computed using $\textsc{FChild}$ and at most σ times $\textsc{NSib}$. We give the generalized performance and an instantiation using $\delta = (\log_\sigma \log n) \log n$, assuming $\sigma = O(\text{polylog}(n))$, and using the FM-Index of Ferragina *et al.* [7] as the compressed suffix array (CSA).

	Sadakane's		Ours	
Space in bits	$\lvert CSA\rvert + \mathbf{6n} + o(n)$	$= nH_k + \mathbf{6n} + o(n \log \sigma)$	$\lvert CSA\rvert + O((n/\delta) \log n)$	$= nH_k + o(n \log \sigma)$
SDep/ Locate	Φ	$= (\log_\sigma \log n) \log n$	$\Psi\delta$	$= (\log_\sigma \log n) \log n$
Count/ Ancestor	1	$= 1$	1	$= 1$
Parent/ FChild/ NSib	1	$= 1$	$(\Psi + t)\delta$	$= (\log_\sigma \log n) \log n$
SLink	Ψ	$= 1$	$(\Psi + t)\delta$	$= (\log_\sigma \log n) \log n$
$\textsc{SLink}^i$	Φ	$= (\log_\sigma \log n) \log n$	$\Phi + (\Psi + t)\delta$	$= (\log_\sigma \log n) \log n$
$\textsc{Letter}(v, i)$	Φ	$= (\log_\sigma \log n) \log n$	Φ	$= (\log_\sigma \log n) \log n$
LCA	1	$= 1$	$(\Psi + t)\delta$	$= (\log_\sigma \log n) \log n$
Child	$\Phi \log \sigma$	$= (\log \log n) \log n$	$\log \sigma + \Phi \log \delta + (\Psi + t)\delta$	$= (\log \log n)^2 \log_\sigma n$
TDep	1	$= 1$	$(\Psi + t)\delta^2$	$= ((\log_\sigma \log n) \log n)^2$
LAQt	1	$= 1$	$\log n + (\Psi + t)\delta^2$	$= ((\log_\sigma \log n) \log n)^2$
LAQs	**Not Supported**		$\log n + (\Psi + t)\delta$	$= (\log_\sigma \log n) \log n$
WeinerLink	t	$= 1$	t	$= 1$

2 Basic Concepts

Figure 1 shows an example that illustrates the concepts in this section. We denote by T a **string**; by Σ the **alphabet** of size σ; by $T[i]$ the symbol at position $(i \bmod n)$; by $T.T'$ **concatenation**; by $T = T[..i{-}1].T[i..j].T[j{+}1..]$ respectively a **prefix**, a **susbtring** and a **suffix**; by $\textsc{Parent}(v)$ the parent node of node v; by $\textsc{TDep}(v)$ its tree-depth; by $\textsc{FChild}(v)$ its first child; by $\textsc{NSib}(v)$ the next child of the same parent; by $\textsc{LAQt}(v, d)$ its **level-d ancestor**; by $\textsc{Ancestor}(v, v')$ whether v is an ancestor of v'; by $\textsc{LCA}(v, v')$ the **lowest common ancestor**.

The **path-label** of a node v in a labeled tree is the concatenation of the edge-labels from the root down to v. We refer indifferently to nodes and to their path-labels, also denoted by v. The i-th letter of the path-label is denoted as $\textsc{Letter}(v, i) = v[i]$. The **string-depth** of a node v, denoted by $\textsc{SDep}(v)$, is the length of its path-label. $\textsc{LAQs}(v, d)$ is the highest ancestor of node v with $\textsc{SDep} \geq d$. $\textsc{Child}(v, X)$ is the node that results of descending from v by the edge whose label starts with symbol X, if it exists. The **suffix tree** of T is the deterministic compact labeled tree for which the path-labels of the leaves are the suffixes of $T\$$, where $\$$ is a terminator symbol not belonging to Σ. We will assume n is the length of $T\$$. For a detailed explanation see Gusfield's book [2]. The **suffix-link** of a node $v \neq \textsc{Root}$ of a suffix tree, denoted

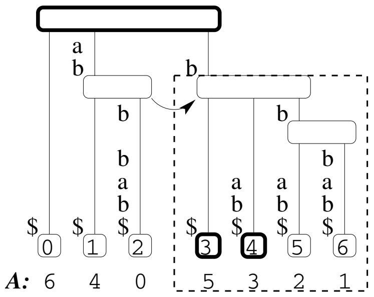


Fig. 1. Suffix tree $\mathcal{T}$ of string *abbbab*, with the leaves numbered. The arrow shows the SLink between node *ab* and *b*. Below it we show the suffix array. The portion of the tree corresponding to node *b* and respective leaves interval is highlighted with a dashed rectangle. The sampled nodes have bold outlines.

```
                   1             2
i: 01 234 56 7890 12 345 67 8901
   ((0)((1)(2))((3)(4)((5)(6))))

   (               (3)(4)        )
i: 0               1 23 4        5
```

Fig. 2. Parentheses representations of trees. The parentheses on top represent the suffix tree and those on the bottom represent the sampled tree. The numbers are not part of the representation; they are shown for clarity. The rows labeled `i:` give the index of the parentheses.

SLink(v), is a pointer to node $v[1..]$. Note that SDep(v) of a leaf v identifies the suffix of $T\$$ starting at position $n - \text{SDep}(v) = \text{Locate}(v)$. For example $T[\text{Locate}(ab\$)..] = T[7-3..] = T[4..] = ab\$$. The **suffix array** $A[0, n-1]$ stores the Locate values of the leaves in lexicographicall order. The *suffix tree nodes can be identified with suffix array intervals*: each node corresponds to the *range* of leaves that descend from v. The node b corresponds to the interval $[3, 6]$. Hence the node v will be represented by the interval $[v_l, v_r]$. Leaves are also represented by their left-to-right index (starting at 0). For example by $v_l - 1$ we refer to the leaf immediately before v_l, *i.e.* $[v_l - 1, v_l - 1]$. With this representation we can Count in constant time the number of leaves that descend from v. The number of leaves below b is $4 = 6 - 3 + 1$. This is precisely the number of times that the string b occurs in the indexed string T. We can also compute Ancestor in $O(1)$ time: $\text{Ancestor}(v, v') \;\Leftrightarrow\; v_l \leq v'_l \leq v'_r \leq v_r$.

3 Using Compressed Suffix Arrays

We are interested in compressed suffix arrays because they have very compact representations and support partial suffix tree functionality (being usually more powerful than the classical suffix arrays [6]). Apart from the basic functionality of retrieving $A[i] = \text{Locate}(i)$, state-of-the-art compressed suffix arrays support operation SLink(v) *for leaves* v. This is called $\psi(v)$ in the literature: $A[\psi(v)] = A[v] + 1$, and thus $\text{SLink}(v) = \psi(v)$, let its time complexity be $O(\Psi)$. The iterated version of ψ, denoted as ψ^i, can usually be computed faster than $O(i\Psi)$ with compressed indexes [6]. This is achieved with the A and A^{-1}, let its time complexity be $O(\Phi)$. It also supports the WeinerLink(v, a) operation [11] for nodes v: WeinerLink(v, X) gives the suffix tree node with path-label $X.v[0..]$. This is called the LF mapping in compressed suffix arrays, and is a kind of inverse of ψ, let its time complexity be $O(t)$. Consider the interval $[3, 6]$ that represents the leaves whose path-labels start by b. In this case we have that $\mathrm{LF}(a, [3, 6]) = [1, 2]$, *i.e.* by using the LF mapping with a we obtain the interval of leaves whose path-labels start by ab. We use an extension of LF to strings, $\mathrm{LF}(X.Y, v) = \mathrm{LF}(X, \mathrm{LF}(Y, v))$.

Finally, compressed suffix arrays are usually self-indexes, meaning that they replace the text: it is possible to extract any substring, of size ℓ, of the indexed text in $O(\Phi + \ell\Psi)$ time. A particularly easy case that is solved in constant time is to extract $T[A[v]]$ for a suffix array cell v, that is, the first letter of a given suffix[3]. This corresponds to $v[0]$, the first letter of the path-label of leaf v.

As anticipated, our compressed suffix tree representation will consist of a sampling of the suffix tree plus a compressed suffix array representation. A well-known compressed suffix array is Sadakane's CSA [9], which requires $\frac{1}{\epsilon} nH_0 + O(n \log\log\sigma)$ bits of space and has times $\Psi = O(1)$, $\Phi = O(\log^{\epsilon} n)$, and $t = O(\log n)$, for any $\epsilon > 0$. For our results we favor a second compressed suffix array, called the FM-index [7], which requires $nH_k + o(n \log\sigma)$ bits, for any $k \le \alpha \log_\sigma n$ and constant $0 < \alpha < 1$. Its complexities are $\Psi = t = O(1 + (\log_\sigma \log n)^{-1})$ and $\Phi = O((\log_\sigma \log n)\log n)$.[4] The instantiation in Table 1 is computed for the FM-index, but the reader can easily compute the result of using Sadakane's CSA. In that case the comparison would favor more Sadakane's compressed suffix tree, yet the space would be considerably higher.

4 The Sampled Suffix Tree

A pointer based implementation of suffix trees requires $O(n \log n)$ bits to represent a suffix tree of (at most) $2n$ nodes. As this is too much, we will store only a few *sampled* nodes. We denote our sampling factor by δ, so that in total we sample $O(n/\delta)$ nodes. Hence, provided $\delta = \omega(\log_\sigma n)$, the sampled tree can be represented using $o(n \log\sigma)$ bits. To fix ideas we can assume $\delta = \lceil(\log_\sigma \log n)\log n\rceil$. In our running example we use $\delta = 4$.

To understand the structure of the sampled tree notice that every tree with $2n$ nodes can be represented in $4n$ bits as a sequence of parentheses (see Figures 1 and 2). The representation of the sampled tree can be obtained by deleting the parentheses of the non-sampled nodes, as in Figure 2. For the sampled tree to be representative of the suffix tree it is necessary that every node is, in some sense, *close enough* to a sampled node.

Definition 1. *A* δ-sampled tree S *of a suffix tree* $\mathcal{T}$ *with* $\Theta(n)$ *nodes is formed by choosing* $O(n/\delta)$ *nodes of* $\mathcal{T}$ *so that for each node* v *of* $\mathcal{T}$ *there is an* $i < \delta$ *such that node* $\textsc{SLink}^i(v)$ *is sampled.*

[3] This is computed in $O(1)$ as the $c \in \Sigma$ satisfying $C[c] \le i < C[c+1]$, see [6].

[4] $\psi(i)$ can be computed as $select_{T[A[i]]}(T^{bwt}, T[A[i]])$ using the multiary wavelet tree [12]. The cost for Φ is obtained using a sampling step of $(\log_\sigma \log n)\log n$, so that $o(n \log\sigma)$ stands for $O((n \log\sigma)/\log\log n)$ as our other structures.

This means that if we start at v and follow suffix links successively, *i.e.* v, $\textsc{SLink}(v)$, $\textsc{SLink}(\textsc{SLink}(v))$, $\ldots$, we will find a sampled node in at most δ steps. Note that this property implies that the $\textsc{Root}$ must be sampled, since $\textsc{SLink}(\textsc{Root})$ is undefined. We sample the nodes v for which $\textsc{SDep}(v) \equiv_{\delta/2} 0$ and there is a another node v' and a string $|T'| \geq \delta/2$ such that $v' = LF(T', v)$. Notice that this guarantees that from the nodes $LF(T'[..i], v)$, for $-1 \leq i \leq |T'|$, only one is sampled. To be precise this guarantees that we sample at most $\lfloor 4n/\delta \rfloor$ nodes from a suffix tree with $2n$ nodes.

In addition to the pointers, that structure the sampled tree, we store in the sampled nodes their interval representation; their $\textsc{SDep}$ and $\textsc{TDep}$; the information to answer LCA queries in S, LCA_S, in constant time [13,14]; and the information to LAQ queries in S, LAQ_S, in constant time [15,16]; some further data is introduced later. All this requires $O((n/\delta)\log n)$ bits of space.

In order to make effective use of the sampled tree, we need a way to map any node v to its *lowest sampled ancestor*, $\text{LSA}(v)$. Another important operation is the *lowest common sampled ancestor* $\text{LCSA}(v, v')$, *i.e.* lowest common ancestor in the sampled tree S. For example $\text{LCSA}(3, 4)$ is the $\textsc{Root}$, whereas $\text{LCA}(3, 4)$ is $[3, 6]$, *i.e.* the node labeled b. Note that $\text{LCSA}(v, v') = \text{LCA}_S(\text{LSA}(v), \text{LSA}(v')) = \text{LSA}(\text{LCA}(v, v'))$. Next we show how LSA is supported for leaves in constant time and $O((n/\delta)\log n)$ extra bits. With that we also have LCSA in constant time (for leaves; later, we extend this to any node).

4.1 Computing LSA for Leaves

LSA is computed by using an operation $\textsc{Reduce}(v)$, that receives the numeric representation of leaf v and returns the position, in the parentheses representation of the sampled tree, where that leaf should be. Consider for example the leaf numbered by 5 in Figure 2. This leaf is not sampled, but in the original tree it appears somewhere between leaf 4 and the end of the tree, more specifically between parenthesis ')' of 4 and parenthesis ')' of the $\textsc{Root}$. We assume $\textsc{Reduce}$ returns the first parenthesis, *i.e.* $\textsc{Reduce}(5) = 4$. In this case since the parenthesis we obtain is a ')' we know that LSA should be the parent of that node. Hence we compute LSA as follows:

$$\text{LSA}(v) = \begin{cases} \textsc{Reduce}(v) & \text{, if there is a '(' at } \textsc{Reduce}(v) \\ \textsc{Parent}(\textsc{Reduce}(v)) & \text{, otherwise} \end{cases}$$

To compute $\textsc{Reduce}$ we use a bitmap $RedB$ and an array $RedA$. The bitmap $RedB$ is initialized with zeros. For every sampled node v represented as $[v_l, v_r]$ we set bits $RedB[v_l]$ and $RedB[v_r + 1]$ to 1. In our example $RedB$ is 1001110. This bitmap indicates the leaves for which we must store partial solutions to $\textsc{Reduce}$. In our example the leaves are $0, 3, 4, 5$. These partial solutions are stored in array $RedA$ (in case of a collision $v_r + 1 = v'_l$, the data for v'_l is stored). In our example these partial results are respectively $0, 1, 3, 4$. Therefore $\textsc{Reduce}(v) = RedA[\textsc{Rank}_1(RedB, v) - 1]$, where v is a leaf number and $\textsc{Rank}_1$ counts the number of 1's in $RedB$ up to and including position v.

First we show that $\textsc{Reduce}$ can be computed in $O(1)$ time with $O((n/\delta)\log n)$ bits. The bitmap $RedB$ cannot be stored in uncompressed form because it would require n bits. We store *RedB* with the representation of Raman *et al.* [17] that needs only $m \log \frac{n}{m} + o(n)$ bits, where $m = O(n/\delta)$ is the number of 1's in the bitmap (as every sampled node inserts at most two 1's in *RedB*). Hence *RedB* needs $O((n/\delta) \log \delta) = O((n/\delta) \log n)$ bits, and supports Rank_1 in $O(1)$ time. On the other hand, since there are also $O(n/\delta)$ integers in *RedA*, we can store them explicitly to have constant access time in $O((n/\delta) \log n)$ bits. Therefore Reduce can be computed within the assumed bounds. According to our previous explanation, so can LSA and LCSA, for leaves.

5 The Kernel Operations

We have described the two basic components of our compressed suffix tree representation. Most of our functionality builds on the LCA operation, which is hence fundamental to us. In this section we present an entangled mechanism that supports operations LCA and SLink depending on each other.

5.1 Two Fundamental Observations

We point out that SLink's and LCA's commute on suffix trees.

Lemma 1. *For any nodes* v, v' *such that* $\text{LCA}(v, v') \neq \text{Root}$ *we have that* $\text{SLink}(\text{LCA}(v, v')) = \text{LCA}(\text{SLink}(v), \text{SLink}(v'))$.

Proof. Assume that the path-labels of v and v' are respectively $X.\alpha.Y.\beta$ and $X.\alpha.Z.\beta'$, where $Y \neq Z$. According to the definitions of LCA and SLink, we have that $\text{LCA}(v, v') = X.\alpha$ and $\text{SLink}(\text{LCA}(v, v')) = \alpha$. On the other hand the path-labels of $\text{SLink}(v)$ and $\text{SLink}(v')$ are respectively $\alpha.Y.\beta$ and $\alpha.Z.\beta'$. Therefore the path-label of $\text{LCA}(\text{SLink}(v), \text{SLink}(v'))$ is also α. Hence this node must be the same as $\text{SLink}(\text{LCA}(v, v'))$. □

Figure 3 illustrates this lemma; ignore the nodes associated with ψ. The condition $\text{LCA}(v, v') \neq \text{Root}$ is easy to verify, in a suffix tree, by comparing the first letters of the path-label of v and v', *i.e.* $\text{LCA}(v, v') \neq \text{Root}$ iff $v[0] = v'[0]$.

The next Lemma shows a fundamental property for the kernel operations.

Lemma 2. *Let* v, v' *be nodes such that* $\text{SLink}^r(\text{LCA}(v, v')) = \text{Root}$, *and let* $d = \min(\delta, r + 1)$. *Then:*

$$\text{SDep}(\text{LCA}(v, v')) = \max_{0 \leq i < d}\{i + \text{SDep}(\text{LCSA}(\text{SLink}^i(v), \text{SLink}^i(v')))\}$$

Proof. The following reasoning holds for any valid i:

$$\begin{aligned} \text{SDep}(\text{LCA}(v, v')) &= i + \text{SDep}(\text{SLink}^i(\text{LCA}(v, v'))) && (1)\\ &= i + \text{SDep}(\text{LCA}(\text{SLink}^i(v), \text{SLink}^i(v'))) && (2)\\ &\geq i + \text{SDep}(\text{LCSA}(\text{SLink}^i(v), \text{SLink}^i(v'))) && (3) \end{aligned}$$

Equation (1) holds by iterating the fact that $\text{SDep}(v'') = 1 + \text{SDep}(\text{SLink}(v''))$ for any node v'' for which $\text{SLink}(v'')$ is defined. Equation (2) results from applying Lemma 1 repeatedly. Inequality (3) comes from the definition of LCSA and

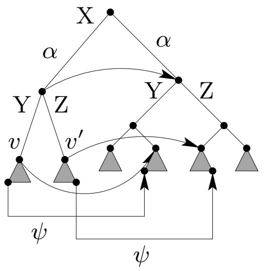


Fig. 3. Schematic representation of the relation between LCA and SLINK, see Lemma 1. Curved arrows represent SLINK and straight arrows the ψ function.

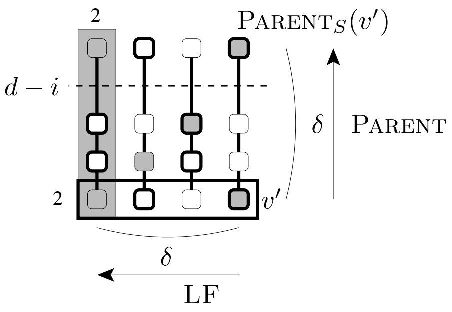


Fig. 4. Schematic representation of the $v_{i,j}$ nodes of the LAQS operation. The nodes sampled because of definition 1 are in bold and the nodes sampled because of the condition of TDEP are filled.

the fact that if node v''' is an ancestor of node v'' then $\text{SDEP}(v'') \geq \text{SDEP}(v''')$. Therefore $\text{SDEP}(\text{LCA}(v, v')) \geq \max_{0 \leq i < d}\{\ldots\}$. On the other hand, from Definition 1 we know that for some $i < \delta$ the node $\text{SLINK}^i(\text{LCA}(v, v'))$ is sampled. The formula goes only up to d, but $d < \delta$ only if $\text{SLINK}^d(\text{LCA}(v, v')) = \text{ROOT}$, which is also sampled. According to the definition of LCSA inequality (3) becomes an equality for that node. Hence $\text{SDEP}(\text{LCA}(v, v')) \leq \max_{0 \leq i < d}\{\ldots\}$. □

5.2 Entangled Operations

To apply Lemma 2 we need to support operations LCSA, SDEP, and SLINK. Operation LCSA is supported in constant time, but only for leaves (Section 4.1). Since SDEP is applied only to sampled nodes, we have it readily stored in the sampled tree. Sadakane [5] showed that $\text{SLINK}(v) = \text{LCA}(\psi(v_l), \psi(v_r))$, whenever $v \neq \text{ROOT}$. This is not necessarily equal to the $[\psi(v_l), \psi(v_r)]$ interval, see node $X.\alpha$ in Figure 3. In general $\text{SLINK}^i(v) = \text{LCA}(\psi^i(v_l), \psi^i(v_r))$.

Hence all we need is to support LCA. However this depends on Lemma 2.

Lemma 3. $\text{LCA}(v, v') = \text{LF}(v[0..i-1], \text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))$ *for any nodes* v, v'*, where* i *is given by Lemma 2.*

Proof. This is a direct consequence of Lemma 2. Let i be the index of the maximum of the set in Lemma 2, *i.e.* $\text{SLINK}^i(\text{LCA}(v, v'))$ is a sampled node and hence it is the same as $\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v'))$. Note that from the definition of LF mapping we have that $\text{LF}(v''[0], \text{SLINK}(v'')) = v''$. Applying this iteratively to $\text{SLINK}^i(\text{LCA}(v, v'))$ we obtain the equality in the lemma. □

To use this lemma we must know which is the correct i. This is easily determined if we first compute $\text{SDEP}(\text{LCA}(v, v'))$. Accessing the letters to apply LF is not a problem, as we have always to obtain the first letter of a path-label, $\text{SLINK}^i(v)[0] = \text{SLINK}^i(v')[0]$.

5.3 Breaking the Cycle

To get out of this dependency we need a new idea. We will handle all the computation over leaves, for which we can compute $\text{SL}\textsc{ink}(v) = \psi(v)$ and $\text{LCSA}(v, v')$.

Lemma 4. $\text{LCA}(v, v') = \text{LCA}(\min\{v_l, v'_l\}, \max\{v_r, v'_r\})$ *for any nodes* v, v'.

Proof. Let v'' and v''' be respectively the nodes on the left and on the right of the equality. Assume that they are represented as $[v''_l, v''_r]$ and $[v'''_l, v'''_r]$ respectively. Hence $v''_l \leq v_l, v'_l$ and $v''_r \geq v_r, v'_r$ since v'' is an ancestor of v and v'. This means that $v''_l \leq \min\{v_l, v'_l\} \leq \max\{v_r, v'_r\} \leq v''_r$, *i.e.* v'' is also an ancestor of $\min\{v_l, v'_l\}$ and $\max\{v_r, v'_r\}$. Since v''' is by definition the lowest common ancestor of these nodes we have that $v''_l \leq v'''_l \leq v'''_r \leq v''_r$. Using a similar reasoning for v''' we conclude that $v'''_l \leq v''_l \leq v''_r \leq v'''_r$ and hence $v'' = v'''$. □

Observe this property in Figure 3; ignore $\text{SL}\textsc{ink}$, ψ and the rest of the tree. Using this property and ψ the equation in Lemma 2 reduces to:

$$\begin{aligned}
&\text{SD}\textsc{ep}(\text{LCA}(v, v')) = \text{SD}\textsc{ep}(\text{LCA}(\min\{v_l, v'_l\}, \max\{v_r, v'_r\}))\\
&\quad = \max_{0 \leq i < d}\{i + \text{SD}\textsc{ep}(\text{LCSA}(\text{SL}\textsc{ink}^i(\min\{v_l, v'_l\}), \text{SL}\textsc{ink}^i(\max\{v_r, v'_r\})))\}\\
&\quad = \max_{0 \leq i < d}\{i + \text{SD}\textsc{ep}(\text{LCSA}(\psi^i(\min\{v_l, v'_l\}), \psi^i(\max\{v_r, v'_r\})))\}
\end{aligned}$$

Operationally, this corresponds to iteratively taking the ψ function, δ times or until the $\textsc{Root}$ is reached. At each step we find the LCSA of the two current leaves and retrieve its stored $\text{SD}\textsc{ep}$. The overall process takes $O(\Psi\delta)$ time. Likewise $\text{SD}\textsc{ep}$ and LCA simplify to:

$$\begin{aligned}
&\text{SD}\textsc{ep}(v) = \text{SD}\textsc{ep}(\text{LCA}(v, v)) = \max_{0 \leq i < d}\{i + \text{SD}\textsc{ep}(\text{LCSA}(\psi^i(v_l), \psi^i(v_r)))\}\\
&\text{LCA}(v, v') = \text{LF}(v[0..i-1], \text{LCSA}(\psi^i(\min\{v_l, v'_l\}), \psi^i(\max\{v_r, v'_r\})))
\end{aligned}$$

Now it is finally clear that we do not need $\text{SL}\textsc{ink}$ to compute LCA. The time to compute LCA is thus $O((\Psi + t)\delta)$. Using LCA we compute $\text{SL}\textsc{ink}$ in $O((\Psi + t)\delta)$ and $\text{SL}\textsc{ink}^i$ in $O(\Phi + (\Psi + t)\delta)$ time. Note that the arguments to LCSA do not correspond necessarily to nodes. Note also that using Lemma 4 we can extend LSA for a general node v as $\text{LSA}(v) = \text{LSA}(\text{LCA}(v, v)) = \text{LSA}(\text{LCA}(v_l, v_r)) = \text{LCSA}(v_l, v_r)$.

6 Further Operations

We now show how other operations can be computed on top of the kernel ones.

Computing $\textsc{Letter}$: Since $\textsc{Letter}(v, i) = \text{SL}\textsc{ink}^i(v)[0] = \psi^i(v_l)[0]$, we can solve it in time $O(\min(\Phi, i\Psi))$.

Computing $\textsc{Parent}$: For any node v represented as $[v_l, v_r]$ we have that $\textsc{Parent}(v)$ is either $\text{LCA}(v_l - 1, v_l)$ or $\text{LCA}(v_r, v_r + 1)$, whichever is lowest. This computation is correct because suffix trees are compact. Notice that if one of these nodes is undefined, either because $v_l = 0$ or $v_r = n$, then the parent is the other node. If both nodes are undefined the node in question is the $\textsc{Root}$ which has no $\textsc{Parent}$ node.

Computing Child: Suppose for a moment that every sampled node stores a list of its children and the corresponding first letters of the edges. In our example the Root would store the list $\{(\$, [0,0]), (a, [1,2]), (b, [3,6])\}$, which can be reduced to $\{(\$, 0), (a, 1), (b, 3)\}$. Hence, for sampled nodes, it would be possible to compute $\text{Child}(v, X)$ in $O(\log \sigma)$ time by binary searching its child list. To compute Child on non-sampled nodes we could use a process similar to Lemma 3: determine which $\text{SLink}^i(v)$, with $i < \delta$, is sampled; compute $\text{Child}(\text{SLink}^i(v), X)$; and use the LF mapping to obtain the answer, *i.e.* $\text{Child}(v, X) = \text{LF}(v[0..i-1], \text{Child}(\text{SLink}^i(v), X))$. This process requires $O(\log \sigma + (\Psi + t)\delta)$ time. Still, it requires too much space since it may need to store $O(\sigma n/\delta)$ integers.

To avoid exceeding our space bounds we *mark* one leaf out of δ, *i.e.* mark leaf v if $v \equiv_\delta 0$. Do not confuse this concept with sampling, they are orthogonal. In Figure 1 we mark leaves 0 and 4. For every sampled node, instead of storing a list with all the children, we consider only the children that contain marked leaves. In the case of the Root this means excluding the child $[1,2]$, hence the resulting list is $\{(\$, 0), (b, 3)\}$. A binary search on this list no longer returns only one child. Instead, it returns a range of, at most, δ children. Therefore it is necessary to do a couple of binary searches, inside that range, to delimit the interval of the correct child. This requires $O(\Phi \log \delta)$ time because now we must use Letter to drive the binary searches. Overall, we can compute $\text{Child}(v, X)$ in $O(\log \sigma + \Phi \log \delta + (\Psi + t)\delta)$ time. Let us now consider space. Ignoring unary paths in the sampled tree, whose space is dominated by the number of sampled nodes, the total number of integers stored amortizes to $O(n/\delta)$, the number of marked leaves. Hence this approach requires at most $O((n/\delta) \log n)$ bits.

Computing TDep: To compute $\text{TDep}(v)$ we need to add other $O(n/\delta)$ nodes to the sampled tree S, so as to guarantee that, for any suffix tree node v, $\text{Parent}^j(v)$ is sampled for some $0 \leq j < \delta$. Recall that the $\text{TDep}(v)$ values are stored in S. Notice that $\text{TDep}(v) = \text{TDep}(\text{LSA}(v)) + j$ where $\text{LSA}(v) = \text{Parent}^j(v)$, hence, computing $\text{TDep}(v)$ consists in reading $\text{TDep}(\text{LSA}(v))$ and adding the number of nodes between v and $\text{LSA}(v)$. The sampling guarantees that $j < \delta$. Hence to determine j we iterate Parent until reaching $\text{LSA}(v)$. The total cost is $O((\Psi + t)\delta^2)$.

Computing LAQt: We extend the $\text{Parent}_S(v)$ notation to represent $\text{LSA}(v)$ when v is a non-sampled node. Recall that the sampled tree supports constant-time level ancestor queries. Hence we have any $\text{Parent}^i_S(v)$ in constant time for any node v and any i. We binary search $\text{Parent}^i_S(v)$ to find the node v' with $\text{TDep}(v') \geq d > \text{TDep}(\text{Parent}_S(v'))$. Notice that this can be computed evaluating only the second inequality. Now we iterate the Parent operation, from v', exactly $\text{TDep}(v') - d$ times. We need the additional sampling introduced for TDep to guarantee $\text{TDep}(v') - d < \delta$. Hence the total time is $O(\log n + (\Psi + t)\delta^2)$.

Computing LAQs: We start by binary searching $\text{Parent}^i_S(\text{SLink}^{\delta-1}(v))$ to find a node v' for which $\text{SDep}(v') \geq d - (\delta - 1) > \text{SDep}(\text{Parent}_S(v'))$. Now we scan all the sampled nodes $v_{i,j} = \text{Parent}_S^j(\text{LSA}(\text{LF}(v[i..\delta-1], v')))$ with $\text{SDep}(v_{i,j}) \geq d - i$ and $i, j < \delta$. This means that we start at node v', follow LF, reduce every node found to the sampled tree S and use Parent_S until the SDep of the node drops below $d - i$. Our aim is to find the $v_{i,j}$ that minimizes $\text{SDep}(v_{i,j}) - (d - i) \geq 0$, and then apply the LF mapping to it. The answer is necessarily among the nodes considered.

The time to perform this operation depends on the number of existing $v_{i,j}$ nodes. For this operation the sampling must satisfy Definition 1 and the condition of TDep. Each condition contributes with at most two sampled nodes for every δ nodes. Therefore, there are at most 4δ nodes $v_{i,j}$ (see Figure 4). Unfortunately, the same trick does not work for TDep and LAQt, because we cannot know which is the "right" node without bringing all of them back with LF.

Computing FChild: To find the first child of $v = [v_l, v_r]$, where $v_l \neq v_r$, we simply ask for $\text{LAQs}(v_l, \text{SDep}(v) + 1)$. Likewise if we use v_r we obtain the last child. By $\text{TDep}_S(v) = i$ we mean that $\text{Parent}_S^i(v) = \text{Root}$. This is also defined when v is not sampled. It is possible to skip the binary search step by choosing $v' = \text{Parent}_S^i(v_l)$, for $i = \text{TDep}_S(v_l) - \text{TDep}_S(\text{LSA}(v)) - 1$.

Computing NSib: The next sibling of $v = [v_l, v_r]$ is $\text{LAQs}(v_r + 1, \text{SDep}(\text{Parent}(v)) + 1)$ for any $v \neq \text{Root}$. Likewise we can obtain the previous sibling with $v_l - 1$. We must check that the answer has the same parent as v, to cover the case where there is no previous/next sibling. We can also skip the binary search.

We are ready to state our summarizing theorem.

Theorem 1. *Using a compressed suffix array (CSA) that supports ψ, ψ^i, $T[A[v]]$ and* LF *in times $O(\Psi)$, $O(\Phi)$, $O(1)$, and $O(t)$, respectively, it is possible to represent a suffix tree with the properties given in Table 1.*

7 Conclusions and Future Work

We presented a fully-compressed representation of suffix trees, which breaks the linear-bits space barrier of previous representations at a reasonable (and in some cases no) time complexity penalty. Our structure efficiently supports common and not-so-common operations, including very powerful ones such as lowest common ancestor (LCA) and level ancestor (LAQ) queries. In fact our representation is largely based on the LCA operation. Suffix trees have been used in combination with LCA's for a long time, but our results show new ways to explore this partnership.

With respect to practical considerations, we believe that the structure can be implemented without large space costs associated to the sublinear term $o(n \log \sigma)$. In fact, by using parentheses representations of the sampled tree and compressed bitmaps, it seems possible to implement the tree with $\log n + O(\log \delta)$ bits per sampled node. Our structure has the potential of using much less space than alternative suffix tree representations. On the other hand, we can tune the space/time tradeoff parameter δ to fit the real space needs of the application. Even though some DNA sequences require 700 Megabytes, that is not always the case. Hence it is reasonable to use larger representations of the suffix tree to obtain faster operations, as long as the structure fits in main memory.

References

1. Apostolico, A.: Combinatorial Algorithms on Words. In: The myriad virtues of subword trees. NATO ISI Series, pp. 85–96. Springer, Heidelberg (1985)
2. Gusfield, D.: Algorithms on Strings, Trees and Sequences. Cambridge University Press, Cambridge (1997)
3. Giegerich, R., Kurtz, S., Stoye, J.: Efficient implementation of lazy suffix trees. Softw., Pract. Exper. 33(11), 1035–1049 (2003)
4. Manber, U., Myers, E.W.: Suffix arrays: A new method for on-line string searches. SIAM J. Comput. 22(5), 935–948 (1993)
5. Sadakane, K.: Compressed Suffix Trees with Full Functionality. Theo. Comp. Sys. (2007)
6. Navarro, G., Mäkinen, V.: Compressed full-text indexes. ACM Comp. Surv. 39(1) (2007) (article 2)
7. Ferragina, P., Manzini, G., Mäkinen, V., Navarro, G.: Compressed representations of sequences and full-text indexes. ACM Trans. Algor. 3(2) (2007) (article 20)
8. Manzini, G.: An analysis of the Burrows-Wheeler transform. J. ACM 48(3), 407–430 (2001)
9. Sadakane, K.: New text indexing functionalities of the compressed suffix arrays. J. of Algorithms 48(2), 294–313 (2003)
10. Foschini, L., Grossi, R., Gupta, A., Vitter, J.: When indexing equals compression: Experiments with compressing suffix arrays and applications. ACM Trans. Algor. 2(4), 611–639 (2006)
11. Weiner, P.: Linear pattern matching algorithms. In: IEEE Symp. on Switching and Automata Theory, pp. 1–11 (1973)
12. Lee, S., Park, K.: Dynamic rank-select structures with applications to run-length encoded texts. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 96–106. Springer, Heidelberg (2007)
13. Bender, M., Farach-Colton, M.: The LCA problem revisited. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000)
14. Fischer, J., Heun, V.: A new succinct representation of RMQ-information and improvements in the enhanced suffix array. In: Chen, B., Paterson, M., Zhang, G. (eds.) ESCAPE 2007. LNCS, vol. 4614, pp. 459–470. Springer, Heidelberg (2007)
15. Bender, M., Farach-Colton, M.: The level ancestor problem simplified. Theor. Comp. Sci. 321(1), 5–12 (2004)
16. Geary, R., Raman, R., Raman, V.: Succinct ordinal trees with level-ancestor queries. In: Munro, J.I. (ed.) SODA, pp. 1–10. SIAM, Philadelphia (2004)
17. Raman, R., Raman, V., Rao, S.S.: Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In: SODA, pp. 233–242 (2002)

Improved Dynamic Rank-Select Entropy-Bound Structures[⋆]

Rodrigo González and Gonzalo Navarro

Center for Web Research, Dept. of Computer Science, University of Chile
{rgonzale,gnavarro}@dcc.uchile.cl

Abstract. Operations *rank* and *select* over a sequence of symbols have many applications to the design of succinct and compressed data structures to manage text collections, structured text, binary relations, trees, graphs, and so on. We are interested in the case where the collections can be updated via insertions and deletions of symbols. Two current solutions stand out as the best in the tradeoff of space versus time (considering all the operations). One solution, by Mäkinen and Navarro, achieves compressed space (i.e., $nH_0 + o(n \log \sigma)$ bits) and $O(\log n \log \sigma)$ worst-case time for all the operations, where n is the sequence length, σ is the alphabet size, and H_0 is the zero-order entropy of the sequence. The other solution, by Lee and Park, achieves $O(\log n(1 + \frac{\log \sigma}{\log \log n}))$ amortized time and uncompressed space, i.e. $n \log \sigma + O(n) + o(n \log \sigma)$ bits. In this paper we show that the best of both worlds can be achieved. We combine the solutions to obtain $nH_0 + o(n \log \sigma)$ bits of space and $O(\log n(1 + \frac{\log \sigma}{\log \log n}))$ worst-case time for all the operations. Apart from the best current solution to the problem, we obtain several byproducts of independent interest applicable to partial sums, text indexes, suffix arrays, the Burrows-Wheeler transform, and others.

1 Introduction and Related Work

Compressed data structures aims at representing classical data structures such as sequences, trees, graphs, etc., in little space while keeping the functionality of the structure. That is, compressed data structures should operate without the need to decompress them. This is a very active area of research stimulated by today's steep memory hierarchies and large available data sizes. See e.g. [15].

One of the most useful structures are the bit vectors with *rank* and *select* operations: $rank(B, i)$ gives the number of 1-bits in $B[1, i]$ and $select(B, i)$ gives the position of the i-th 1 in B. This generalizes to sequences $T[1, n]$ over an alphabet Σ of size σ, where one aims at a (hopefully compressed) representation efficiently supporting the following operations: $access(T, i)$ returns the symbol $T[i]$; $rank_c(T, i)$ returns the number of times symbol c appears in the prefix $T[1, i]$; and $select_c(T, i)$ returns the position of the i-th c in T.

⋆ Supported in part by Millennium Nucleus Center for Web Research, Grant P04-067-F, Mideplan, and Fondecyt Grant 1-050493, Chile.

Improvements in *rank*/*select* operations on sequences have a great impact on many other succinct data structures, especially on those aimed at text indexing, but also labeled trees, structured texts, binary relations, graphs, and others [15].

The first structure providing support for *rank*/*select* on a sequence of symbols was the *wavelet tree* [7,5]. Wavelet trees are perfectly balanced static trees of height $\log \sigma$ (logarithms are in base 2 by default). They answer the three queries in $O(\log \sigma)$ time, by working $O(1)$ per tree level. They store a bitmap of length n per level, which is preprocessed for constant-time binary *rank*/*select* queries. Their total space requirement is $n \log \sigma + o(n \log \sigma)$, where the extra sublinear term is the space needed by the binary *rank*/*select* structures [14]. By representing those bitmaps in compressed form [16] the constant-time *rank*/*select* queries are retained and the space becomes $nH_0(T) + o(n \log \sigma)$, where $H_0(T)$ is the zero-order empirical entropy of T (that is, $\sum_{c \in \Sigma} \frac{n_c}{n} \log \frac{n}{n_c}$, where c occurs n_c times in T). Since the wavelet tree gives $access(T, i)$ to any symbol $T[i]$, it can be used to *replace* T.

A stronger version of wavelet trees are *multiary wavelet trees* [3], which achieve the same space but improve the query times to $O(1 + \frac{\log \sigma}{\log \log n})$. The trick is to make the tree ρ-ary for some $\rho = O(\log^\alpha n)$ and constant $0 < \alpha < 1$, so that its height is reduced. Now the tree does not store a bitmap per level, but rather a sequence over an alphabet of size ρ. They show how to do *rank*/*select* on those sequences in constant time for such a small ρ.

In [10] they add dynamism to the sequences, by adding operations $insert_c(T, i)$ inserts symbol c between $T[i]$ and $T[i+1]$; and $delete(T, i)$ deletes $T[i]$ from T.

They represent dynamic bitmaps B using $nH_0(B) + o(n)$ bits of space and solve all operations in $O(\log n)$ time. This is done with a binary tree that stores $\Theta(\log^2 n)$ bits at the leaves, and at internal nodes stores summary *rank*/*select* information on the subtrees. For larger alphabets, a wavelet tree using dynamic bitmaps yields a dynamic sequence representation that takes $nH_0(T) + o(n \log \sigma)$ bits and solves all the operations in time $O(\log n \log \sigma)$.

Very recently, in [9] they manage to improve the time complexities of this solution. They show that the $O(\log n)$ time complexities can be achieved for alphabets of size up to $\sigma = O(\log n)$. They combine this tool with a multiary wavelet tree to achieve $O(\log n (1 + \frac{\log \sigma}{\log \log n}))$ time.

The key to the success of [9] is a clever detachment of two roles of tree leaves that are entangled in [10]: In the latter, the leaves are the memory allocation unit (that is, whole leaves are allocated or freed), and also the information summarization unit (that is, the tree maintains information up to leaf granularity, and the rest has to be collected by sequentially scanning a leaf). In [9] leaves are the information summarization unit, but handle an internal linked list with smaller memory allocation units. This permits moving symbols to accommodate the space upon insertions/deletions within a leaf, without having to update summarization information for the data moved. This was the main bottleneck that prevented the use of larger alphabets in $O(\log n)$ time in [10].

However, compared to [10], the work in [9] has several weaknesses: (1) it is not compressed, but rather takes $n \log \sigma + O(n) + o(n \log \sigma)$ bits of space; (2) in addition to not compressing T, the extra space includes an $O(n)$ term, as shown; (3) times are amortized, not worst-case.

In this paper we show that it is possible to obtain the best from both worlds. We combine the works in [10,9] to obtain a structure that (1) takes $nH_0(T) + o(n \log \sigma)$ bits of space, and (2) performs all the operations in $O(\log n(1 + \frac{\log \sigma}{\log \log n}))$ worst-case time. (This is achieved even for the case where $\lceil \log n \rceil$ changes and so does the length of the structure pointers in order to maintain the promised space bounds.) The result becomes the most efficient dynamic representation of sequences, both in time and space, and we show immediate applications to other succinct data structures such as compressed text indexes.

This combination is by no means simple. Some parts are not hard to merge, such as the role detachment for leaves [9] with the compressed representation of sequences [3] and multi-ary wavelet trees, plus the memory management techniques to support changes of $\lceil \log n \rceil$ within the same worst-case time bounds and no extra space [10]. However, others require new algorithmic ideas. In [9] they spend $O(n)$ extra bits in bitmaps that maintain leaf-granularity information on *rank*/*select*. We show that this can be replaced by dynamic partial sums, which use sublinear space. However, we need σ partial sums and cannot afford to update them individually upon a leaf insertion/deletion. Hence we create a new structure where a collection of σ sequences are maintained in synchronization. The second problem was that leaf splitting/merging in [9] triggered too many updates to summarization data, which could not be handled in $O(\log n)$ worst-case time, only in $O(\log n)$ amortized time. To get rid of this problem we redefined the leaf fill ratio invariants, preferring a weaker condition that still ensures that leaves are sufficiently full and can be maintained within the $O(\log n)$-worst-case-time bound. Both ideas can be of independent interest.

As for the model of computation, our results (and all the mentioned ones) assume a RAM model with word size $w = \Omega(\log n)$, so that operations on $O(\log n)$ contiguous bits can be carried out in constant time. For the dynamic structures, we always allocate $\omega(\log n)$-bit chunks of the same size (or a finite set of sizes), which can be handled in constant time and asymptotically no extra space [17].

2 Collection of Searchable Partial Sums with Indels

The *Searchable Partial Sums with Indels (SPSI)* problem [8] consists in maintaining a sequence S of nonnegative integers $s_1, \ldots, s_n$, each one of $k = O(\log n)$ bits, supporting the following operations: $sum(S, i)$ is $\sum_{l=1}^{i} s_l$; $search(S, y)$ is the smallest i' such that $sum(S, i') \geq y$; $update(S, i, x)$ updates s_i to $s_i + x$ (x can be negative as long as the result is not); $insert(S, i, x)$ inserts a new integer x between s_{i-1} and s_i; and $delete(S, i)$ deletes s_i from the sequence.

It is possible to handle all these operations using $kn + o(kn)$ bits of space and $O(\log n)$ time per operation [10]. We now define an extension of this problem, that we call *Collection of Searchable Partial Sums with Indels (CSPSI)*. This problem consists in maintaining a collection of σ sequences $C = \{S^1, \ldots, S^\sigma\}$ of nonnegative integers $\{s_i^j\}_{1 \leq j \leq \sigma, 1 \leq i \leq n}$, each one of $k = O(\log n)$ bits. We support the following operations: $sum(C,j,i)$ is $\sum_{l=1}^{i} s_l^j$; $search(C,j,y)$ is the smallest i' such that $sum(C,j,i') \geq y$; $update(C,j,i,x)$ updates s_i^j to $s_i^j + x$; $insert(C,i)$ inserts 0 between s_{i-1}^j and s_i^j for all $1 \leq j \leq \sigma$.; $delete(C,i)$ deletes s_i^j from the sequence S^j for all $1 \leq j \leq \sigma$; To perform $delete(C,i)$ it must hold $s_i^j = 0$ for all $1 \leq j \leq \sigma$. Next we show how to carry out all of these queries/operations in $O(\sigma + \log n)$ time, using $O(\sigma k n)$ bits of space.

Data structure. We construct a red-black tree over C, where the size of each leaf goes from $\frac{1}{2}\log^2 n$ to $2\log^2 n$ bits (they are allocated to hold $2\log^2 n$ bits[1]). The leftmost leaf contains $s_1^1 \cdots s_{b_1}^1 s_1^2 \cdots s_{b_1}^2 \cdots\ s_1^\sigma \cdots s_{b_1}^\sigma$, the second leftmost leaf contains $s_{b_1+1}^1 \cdots s_{b_2}^1 s_{b_1+1}^2 \cdots s_{b_2}^2 \cdots s_{b_1+1}^\sigma \cdots s_{b_2}^\sigma$, and so on. The size of the leftmost leaf is $\sigma k b_1$ bits, the size of the second leftmost leaf is $\sigma k(b_2 - b_1)$ bits, and so on. The size of the leaves is variable and bounded, so $b_1, b_2, \ldots$ are such that $\frac{1}{2}\log^2 n \leq \sigma k b_1, \sigma k(b_2 - b_1), \ldots \leq 2\log^2 n$ hold[2]. Each internal node v stores counters $\{r^j(v)\}_{1 \leq j \leq \sigma}$ and $p(v)$, where $r^j(v)$ is the sum of the integers in the left subtree for sequence S^j and $p(v)$ is the number of positions stored in the left subtree (for any sequence).

Computing $sum(C,j,i)$. We traverse the tree to find the leaf containing the i-th position. We start with $sum \leftarrow 0$ and $v \leftarrow root$. If $p(v) \geq i$ we enter the left subtree, otherwise we enter the right subtree with $i \leftarrow i - p(v)$ and $sum \leftarrow sum + r^j(v)$. We reach the leaf that contains the i-th position in $O(\log n)$ time. Then we scan the leaf, summing up from where the sequence S^j begins, in chunks of size $\frac{1}{2}\log n$ bits using a universal precomputed table Y, until we reach position i. Table Y receives any possible sequence of dk bits, for $d = \lfloor \frac{\frac{1}{2}\log n}{k} \rfloor$, and gives the sum of the d k-bit numbers encoded. The last (at most $d-1$) integers must be added individually. (Note that if $k > \frac{1}{2}\log n$ we can just add each number individually within the time bounds.) The *sum* query takes in total $O(\log n)$ time, and table Y adds only $O(\sqrt{n}\ \mathrm{polylog}(n))$ bits of space.

Computing $search(C,j,y)$. We enter the tree to find the smallest i' such that $sum(C,j,i') \geq y$. We start with $pos \leftarrow 0$ and $v \leftarrow root$. If $r^j(v) \geq y$ we enter the left subtree, otherwise we enter the right subtree with $y \leftarrow y - r^j(v)$ and $pos \leftarrow pos + p(v)$. We reach the leaf that contains the i'-th position in $O(\log n)$ time. Then we scan the leaf, summing up from where the sequence S^j begins, in chunks of size $\frac{1}{2}\log n$ bits using table Y, until this sum is greater than y after adding up i' integers; the answer is then $pos + i'$. (Once an application of the table exceeds y, we must reprocess the last chunk integer-wise.) The *search* query takes in total $O(\log n)$ time.

Operation $update(C,j,i,x)$. We proceed similarly to *sum*, updating $r^j(v)$ as we traverse the tree. That is, we update $r^j(v)$ to $r^j(v) + x$ each time we go left from v. When we reach the leaf we directly update s_i^j to $s_i^j + x$. The *update* operation takes in total $O(\log n)$ time.

[1] In most cases we ignore floors and ceilings for simplicity.

[2] If $\sigma k > 2\log^2 n$, we just store σk bits per leaf. All the algorithms in the sequel are simplified and the complexities are maintained.

For the next operations, we note that a leaf has at most $m = \lfloor \frac{2\log^2 n}{\sigma k} \rfloor$ integers from any sequence. Then a subsequence of a given sequence has at most mk bits. So if we copy a subsequence in chunks of $\frac{1}{2}\log n$ bits, the subsequence will be copied in $1 + \frac{2mk}{\log n} = O(1 + \frac{\log n}{\sigma})$ time in the RAM model (this requires shifting bits, which in case it is not supported by the model, can be handled using small universal tables of the kind of Y). As we have σ sequences, we can copy a given subsequence of them all in $O(\sigma + \log n)$ time. The next operations are solved by a constant number applications of these copying operations.

Operation $insert(C, i)$. We traverse the tree similarly to *sum*, updating $p(v)$ as we traverse the tree. That is, we increase $p(v)$ by 1 each time we go left from v. Then we copy the leaf arrived at to a new memory area, adding a 0 between s^j_{i-1} and s^j_i for all j. This is done by first copying the subsequences $\ldots s^j_{i-1}$ for all j, then adding 0 to each sequence, and finally copying the subsequences $s^j_i \ldots$ for all j. As we have just explained, this can be done in $O(\sigma + \log n)$ time.

If the new leaf uses more than $2\log^2 n$ bits, the leaf is split in two. An overflowed leaf has $m = \lfloor 2\log^2 n/(\sigma k) \rfloor + 1$ integers in each sequence. So we store in the left leaf the first $\lfloor m/2 \rfloor$ integers of each sequence and in the right leaf we store the rest. These two copies can be done again in $O(\sigma + \log n)$ time. These new leaves are made children of a new node μ. We compute each $r^j(\mu)$ by scanning and summing on the left leaf. This summing can be done in $O(\sigma + \log n)$ time using table Y. We also set $p(\mu) = \lfloor m/2 \rfloor$. Finally, we check if we need to rebalance the tree. If needed, the read-black tree is rebalanced with just one rotation and $O(\log n)$ red-black tag updates. After a rotation we need to update $r^j(\cdot)$ and $p(\cdot)$ only for three nodes, which is easily done in $O(\sigma)$ time. The *insert* operation takes in total $O(\sigma + \log n)$ time.

Operation $delete(C, i)$. We traverse the tree similarly to *sum*, updating $p(v)$ while we traverse the tree. That is, we decrease $p(v)$ by 1 each time we go left from v. Then we copy the leaf to a new memory area, deleting s^j_i for all j, similarly to *insert*, in $O(\sigma + \log n)$ time.

There are three possibilities after this deletion: (i) The new leaf uses more than $\frac{1}{2}\log^2 n$ bits, in which case we are done. (ii) The new leaf uses less than $\frac{1}{2}\log^2 n$ and its sibling is also a leaf, in which case we merge it with its sibling, again in $O(\sigma + \log n)$ time. Note that this merging removes the leaf's parent but does not require any recomputation of $r^j(\cdot)$ or $p(\cdot)$. (iii) The new leaf uses less than $\frac{1}{2}\log^2 n$ and its sibling is an internal node μ, in which case by the red-black tree properties we have that μ must have two leaf children. In this case we merge our new leaf with the closest child of μ, updating the counters of μ in $O(\sigma)$ time, and letting μ replace the parent of our original leaf.

In cases (ii) and (iii), the merged leaf might use more than $2\log^2 n$ bits. In this case we split it again into two halves, just as we do in *insert* (and including the recomputation of $r^j(\cdot)$ and $p(\cdot)$). The tree might have to be rebalanced as well. The *delete* operation takes in total $O(\sigma + \log n)$ time.

The space requirement is at most $4\sigma k n$ bits for all the leaves. For each internal node we have two pointers, a counter $p(\cdot)$, and σ counters $r^j(\cdot) \leq 2^k \cdot n$, totalizing

$O(\log n) + \sigma(k + \log n) = O(\sigma \log n)$ bits per node. So, all the internal nodes use $O(\frac{\sigma kn}{\log^2 n}\sigma \log n) = O(\frac{\sigma^2 kn}{\log n})$ bits. We have proved our claim.

Theorem 1. *The Collection of Searchable Partial Sums with Indels problem with σ sequences of n numbers of k bits can be solved, in a RAM machine of $w = \Omega(\log n)$ bits, using $O(\sigma kn(1 + \frac{\sigma}{\log n}))$ bits of space, supporting all the operations in $O(\sigma + \log n)$ worst-case time. Note that, if $\sigma = O(\log n)$ the space is $O(\sigma kn)$ and the time is $O(\log n)$.*

We note that we have actually assumed that $w = \Theta(\log n)$ in our space computation (as we have used w-bit system pointers). The general case $w = \Omega(\log n)$ can be addressed using exactly the same techniques developed in [10], using a more refined memory management with pointers of $(\log n) \pm 1$ bits, and splitting the sequence into three in a way that retains the worst-case complexities.

We also note that the space can be improved to $\sigma kn(1 + O(\frac{\sigma}{\log n}))$ by using a finer memory allocation policy for the leaves, just as done in the next sections for sequences. The simpler result suffices for this paper.

3 Uncompressed Dynamic Rank-Select Structures for a Small Alphabet

For a small alphabet of size $\sigma = O(\log n)$, we construct a red-black tree over $T[1, n]$ where each leaf contains a non-empty *superblock* of size up to $2 \log^2 n$ bits. We will introduce invariants that guarantee that there are at most $1 + \frac{2n \log \sigma}{\log^2 n}$ superblocks. Each internal node v stores counters $r(v)$ and $p(v)$, where $r(v)$ is the number of superblocks in the left subtree and $p(v)$ is the number of symbols stored in the left subtree. For each superblock i, we maintain s_i^j, the number of occurrences of symbol j in superblock i. We store all these sequences of numbers using a *Collection of Searchable Partial Sums with Indels*, C. The length of each sequence will be at most $\frac{2n \log \sigma}{\log^2 n}$ integers, $\sigma = O(\log n)$ and $k = O(\log \log n)$. So the partial sums operate in $O(\log n)$ worst-case time.

Each superblock is further divided into *blocks* of $\sqrt{\log n} \log n$ bits, so each superblock has up to $2\sqrt{\log n}$ blocks. We maintain these blocks using a linked list. Only the last block could be not fully used.

A superblock storing less than $\log^2 n$ bits is called *sparse*. Operations *insert* and *delete* will maintain the invariant that no two consecutive sparse superblocks may exist. This ensures that every consecutive pair of superblocks holds at least $\log^2 n$ bits from T, and thus that there are at most $1 + \frac{2n \log \sigma}{\log^2 n}$ superblocks.

The space usage of our structure is $n \log \sigma + O(\frac{n \log \sigma}{\sqrt{\log n}})$, as $\sigma = O(\log n)$: The text itself uses $n \log \sigma$ bits of space. The *CSPSI* uses $O(\sigma \log \log n \frac{n \log \sigma}{\log^2 n}) = O(\frac{n \log \log n \log \sigma}{\log n})$ bits of space. Each pointer of the linked list of blocks uses $O(\log n)$ bits and we have $O(\frac{n \log \sigma}{\sqrt{\log n} \log n})$ blocks, totalizing $O(\frac{n \log \sigma}{\sqrt{\log n}})$ bits. The last block in each superblock is not necessarily fully used. We have at most $1 + \frac{2n \log \sigma}{\log^2 n}$ superblocks, each of which can waste a full block of size $\sqrt{\log n} \log n$

bits, totalizing $O(\frac{n \log \sigma}{\sqrt{\log n}})$ bits. Inside each block, we can lose at most $\log \sigma$ bits due to symbol misalignment, totalizing $O(\frac{n \log^2 \sigma}{\sqrt{\log n} \log n}) = O(\frac{n \log \log n \log \sigma}{\sqrt{\log n} \log n})$ bits. The tree pointers and counters use $O(\frac{n \log \sigma}{\log^2 n} \cdot \log n) = O(\frac{n \log \sigma}{\log n})$ bits.

Now we show how to carry out all the queries/operations in $O(\log n)$ time. First, it is important to notice that each block can be scanned or shifted in $\sqrt{\log n}$ time, using tables that process chunks of $\frac{1}{2} \log n$ bits (again, if $\sigma > \frac{1}{2} \log n$ we can process each symbol individually within the time bounds). Given that there are at most $O(\sqrt{\log n})$ blocks in a superblock, we can scan or shift elements within a superblock in $O(\log n)$ time, even considering block boundaries.

Computing access(T, i). We traverse the tree to find the leaf containing the i-th position. We start with $sb \leftarrow 1$ and $pos \leftarrow i$. if $p(v) \geq pos$ we enter the left subtree, otherwise we enter the right subtree with $sb \leftarrow sb + r(v)$ and $pos \leftarrow pos - p(v)$. We reach the leaf that contains the i-th position in $O(\log n)$ time. Then we directly access the *pos*-th symbol of *sb*. Note that, within the same $O(\log n)$ time, we can extract any $O(\log n)$-bit long sequence of symbols from T.

Computing $rank_c(T, i)$. We find the leaf containing the i-th position, just as for *access*. Then we scan superblock *sb* from the first block summing up the occurrences of c up to the position *pos*, using a table Z to sum the c's. We add to this quantity $sum(C, c, sb - 1)$, the number of times that c appears before superblock *sb*. The *rank* query takes in total $O(\log n)$ time. Table Z is of the same spirit of Y and requires $O(\sigma\sqrt{n}\ \mathrm{polylog}(n)) = O(\sqrt{n}\ \mathrm{polylog}(n))$ bits.

Computing $select_c(T, i)$. We calculate $j = search(C, c, i)$; this way we know that the i-th c belongs to superblock j and it is the i'-th appearance of c within superblock j, for $i' = i - sum(C, c, j - 1)$. Then we traverse the tree to find the leaf representing superblock j. We start with $sb \leftarrow j$ and $pos \leftarrow 0$. if $r(v) \geq sb$ we enter the left subtree, otherwise we enter the right subtree with $sb \leftarrow sb - r(v)$ and $pos \leftarrow pos + p(v)$. We reach the correct leaf in $O(\log n)$ time. Then we scan superblock j from the first block, searching for the position of the i'-th appearance of symbol c within superblock j, using table Z. To this position we add *pos* to obtain the final result. The *select* query takes in total $O(\log n)$ time.

Operation $insert_c(T, i)$. We obtain *sb* and *pos* just like in the *access* query, except that we start with $pos \leftarrow i - 1$, so as to insert right after position $i - 1$. Then, if superblock *sb* contains room for one more symbol, we insert c right after the *pos*-th position of *sb*, by shifting the symbols through the blocks as explained. We also carry out $update(C, c, sb, 1)$ and retraverse the path from the root to *sb* adding 1 to $p(v)$ each time we go left from v.

If this insertion causes an overflow in the last block of *sb*, we simply add a new block at the end of the linked list to hold the trailing symbol (which is usually not the same symbol inserted). In this case we finish in $O(\log n)$ time.

If, instead, the superblock is full, we cannot carry out the insertion yet. We first move one symbol to the previous superblock (creating a new one if this is not possible). We first $delete_d(T, \cdot)$ the first symbol d from block *sb*, which cannot cause an underflow of *sb*. Now, we check how many symbols does superblock

$sb-1$ have: we traverse the tree searching for it, and deduce its size from the $r(v)$ counters in the tree. If it can hold one more symbol, we $insert_d(T,\cdot)$ the removed symbol d at the end of superblock $sb-1$. This recursive invocation to *insert* will not overflow leaf $sb-1$.

If superblock $sb-1$ is also full or does not exist, then we are entitled to create a sparse superblock between $sb-1$ and sb, without breaking the invariant on sparse superblocks. We create such an empty superblock and insert symbol d into it, using the following procedure: We retraverse the path from the root to sb, updating $r(v)$ to $r(v)+1$ each time we go left from v. When we arrive again at leaf sb we create a new node μ with $r(\mu)=1$ and $p(\mu)=1$. Its left child is the new empty superblock, where the single symbol d is inserted, and its right child is sb. We also execute $insert(C,sb)$ and $update(C,sb,d,1)$.

Finally, we check if we need to rebalance the tree. If it is needed, it can be done with one rotation and $O(\log n)$ red-black tag updates, given that we use a red-black tree. After a rotation we need to update $r(\cdot)$ and $p(\cdot)$ only for three nodes. These updates can be done in $O(1)$ time.

Now that we have finally made room to carry out the original insertion, we rerun $insert_c(T,i)$ and it will not overflow again. The whole *insert* operation takes $O(\log n)$ time.

Operation $delete(T,i)$*.* We obtain sb and pos just as in the *access* query, updating $p(v)$ to $p(v)-1$ each time we go left from v. Then we delete the pos-th position (let c be the symbol deleted) of the sb-th superblock, by shifting the symbols back through the blocks. If this deletion empties the last block, we free it. In any case we call $update(C,c,sb,-1)$ on the partial sums.

There are three possibilities after this deletion: (i) superblock sb is not sparse after the deletion, in which case we are done; (ii) sb was already sparse before the deletion, in which case we have only to check that it has not become empty; (iii) sb turned to sparse due to the deletion, in which case we have to care about the invariant on sparse superblocks.

If superblock sb becomes empty, we retraverse the path from the root to it, updating $r(v)$ to $r(v)-1$ each time we go left from v, in $O(\log n)$ time. When we arrive at leaf sb again, we delete it and do operation $delete(C,sb)$. Finally, we check if we need to rebalance the tree, in which case one rotation and $O(\log n)$ red-black tag updates suffice, just as for insertion. After a rotation we also need to update $r(\cdot)$ and $p(\cdot)$ only for three nodes. These updates take $O(1)$ time.

If, instead, superblock sb turned to sparse, we make sure that neither superblocks $sb-1$ or $sb+1$ are also sparse. If they are not, then superblock sb can become sparse and hence we finish without further intervention.

If superblock $sb-1$ is sparse, we $delete(T,\cdot)$ its last symbol d, and $insert_d(T,\cdot)$ it in the beginning of superblock sb. This recursive call brings no problems because $sb-1$ is already sparse, and we restore the non-sparse status of sb. If superblock $sb-1$ becomes empty, we delete it just as explained for the case of superblock sb. The action is symmetric if $sb-1$ is not sparse but $sb+1$ is.

The *delete* operation takes in total $O(\log n)$ time.

Theorem 2. *Given a text T of length n over a small alphabet of size $\sigma = O(\log n)$, the Dynamic Sequence with Indels problem under RAM model with word size $w = \Omega(\log n)$ can be solved using $n \log \sigma + O(\frac{n \log \sigma}{\sqrt{\log n}})$ bits of space, supporting all the operations in $O(\log n)$ worst-case time.*

We note again that we have actually assumed that $w = \Theta(\log n)$ in our space computation, and that the general case $w = \Omega(\log n)$ can be obtained using exactly the same techniques developed in [10, Sec. 4.5, 4.6, and 6.4].

4 Compressed Dynamic Rank-Select Structures

Thm. 2 can be extended to use a compressed sequence representation, by just changing the way we store/manage the blocks. The key idea is to detach the *representational* and the *physical* (i.e., compressed) sizes of the storage units at different levels.

We use the same red-black tree over $T[1, n]$, where each leaf contains a non-empty superblock *representing* up to $2 \log^2 n$ bits *of the original text* T (they will actually store more or less bits depending on how compressible is the portion of T they represent). The same superblock splitting/merging policy related to sparse superblocks is used. Each internal node has the same counters and they are managed in the same way. So all the queries/operations are exactly the same up to the superblock level. Compression is encapsulated within superblocks.

In *physical* terms, a superblock is divided into blocks just as before, and they are still of the same *physical* size, $\sqrt{\log n} \log n$ bits. Depending on compressibility, blocks will represent more or less symbols of the original text, as their physical size is fixed.

In *logical* terms, a superblock is be divided into *subblocks* representing $\frac{1}{2} \log n$ bits (that is, $\frac{1}{2} \log_\sigma n$ symbols[3]) from T. We represent each subblock using the (c, o)-pair encoding of [3]: The c part is of fixed width and tells how many occurrences of each alphabet symbol are there in the subblock; whereas the o part is of variable width and gives the identifier of the subblock among those sharing the same c component. Each c component uses at most $\sigma \log \log n$ bits; while the o components use at most $\frac{1}{2} \log n$ bits each, and overall add up to $nH_0(T) + O(n \log \sigma / \log n)$ bits [3, Sec. 3.1].

In a block of $\sqrt{\log n} \log n$ bits, we store as many subblocks as they fit, wasting at most $\sigma \log \log n + \frac{1}{2} \log n$ unused bits at the end. The universal tables (like Y) used to sequentially process the blocks in chunks of $\Theta(\log n)$ bits must now be modified to process the compressed sequence of (c, o) pairs. This is complex because an insertion in a subblock introduces a displacement that propagates over all the subblocks of the block, which must be completely recomputed and rewritten (the physical size of the whole superblock can even double!). Fortunately all those tedious details have been already sorted out in [10, Sec. 5.2, 6.1, and 6.2], where their superblocks play the role of our blocks, and their tree rearrangements are not necessary for us because we are within a leaf now. Their

[3] Or just one symbol if $\frac{1}{2} \log_\sigma n < 1$.

"partial blocks" mechanism is also not useful for us, because we can tolerate those propagations to extend over all the blocks of our superblocks. Hence only the last block of our superblocks is not as full as misalignments permit.

The time achieved in [10] is $O(1)$ per $\Theta(\log n)$ physical bits. Even in the worst case (where compression does not work at all in the superblock), the number of physical bits will be $\frac{2\log^2 n}{\frac{1}{2}\log n}(\sigma \log\log n + \frac{1}{2}\log n) = O(\log^2 n + \sigma \log n \log\log n)$, and thus the time to solve any query or carry out any update on a superblock will be $O(\log n + \sigma \log\log n)$.

We compute the space usage of these new structures, where it differs from the uncompressed version: The text itself uses $nH_0(T) + O(\frac{\sigma n \log\log n}{\log_\sigma n})$ bits. Inside each block, we can lose at most $O(\sigma \log\log n + \log n)$ bits due to misalignments, totalizing $O(\frac{n\log\sigma(\sigma\log\log n+\log n)}{\sqrt{\log n}\log n})$ bits. The extra space to operate the (c, o) encoding is $O(\sqrt{n}\ \sigma \text{ polylog}(n))$ bits.

The time and space complexities depend sharply on σ. Thus the solution is indeed of interest only for rather small $\sigma = o(\log n / \log\log n)$. For such a small alphabet we have the following theorem. Again, all the issues of varying $\lceil \log n \rceil$ and the case $w = \omega(\log n)$ are handled just as in [10, Sec. 4.5, 4.6, and 6.4].

Theorem 3. *Given a text T of length n over a small alphabet of size $\sigma = o(\log n/\log\log n)$ and zero-order entropy $H_0(T)$, the Dynamic Sequence with Indels problem under RAM model with word size $w = \Omega(\log n)$ can be solved using $nH_0(T) + O(\frac{n\log\sigma}{\sqrt{\log n}})$ bits of space, supporting all operations in $O(\log n)$ worst-case time.*

To extend our results for a large alphabet of size $\sigma = \Omega(\log n/\log\log n)$, we use a generalized ρ-ary wavelet tree [3] over T, where $\rho = \Theta(\sqrt{\log n})$. Essentially, this generalized wavelet tree makes a sequence with the first $\log\rho$ bits of the symbols at the first level, the next $\log\rho$ bits at the second level (where the symbols with the same first $\log\rho$ bits are grouped in the same child of the root), and so on. The tree has $O(\log_\rho \sigma) = O(\frac{\log\sigma}{\log\log n})$ levels. We store on each level a sequence over an alphabet of size ρ, which can be handled using the solution of Thm. 3, for which ρ is small enough. Hence each operation takes $O(\log n)$ time per level, adding up $O(\log n \frac{\log\sigma}{\log\log n})$ worst-case time.

As shown in [3], the sum of the zero-order-entropy representations of the sequences at each level adds up to the zero-order entropy of T. In addition, the generalized ρ-ary wavelet tree handles changes in $\lceil \log n \rceil$ automatically, as this is encapsulated within each level. We thus obtain our main theorem, where we have included the case of small σ as well. We recall that, within the same time, *access* can retrieve $O(\log_\sigma n)$ consecutive symbols from T.

Theorem 4. *Given a text T of length n over an alphabet of size σ and zero-order entropy $H_0(T)$, the Dynamic Sequence with Indels problem under RAM model with word size $w = \Omega(\log n)$ can be solved using $nH_0(T) + O(\frac{n\log\sigma}{\sqrt{\log n}})$ bits of space, supporting all the operations in $O(\log n(1 + \frac{\log\sigma}{\log\log n}))$ worst-case time.*

5 Conclusions

We have shown that the best two existing solutions to the *Dynamic Sequence with Indels* problem [10,9] can be merged so as to obtain the best from both. This merging is not trivial and involves some byproducts that can be of independent interest. We show now a couple of immediate consequences of our result.

In [10,11] it is shown that a wavelet tree built over the Burrows-Wheeler Transform T^{bwt} of a text T [1], and compressed using the (c, o) pair technique, achieves high-order entropy space, namely $nH_h(T) + o(n \log \sigma)$ for any $h + 1 \leq \alpha \log_\sigma n$ and constant $0 < \alpha < 1$, where $H_h(T)$ is the h-th order empirical entropy of T [13]. This is used in [10] to obtain a dynamic text index that handles a collection $\mathcal{C}$ of texts and permits searching for patterns, extracting text snippets, and inserting/deleting texts in/from the collection. Using the definitions of [10, Sec. 7] and using the same sampling step, we can state a stronger version of those theorems:

Theorem 5. *The Dynamic Text Collection problem can be solved with a data structure of size* $nH_h(\mathcal{C}) + o(n \log \sigma) + O(\sigma^{h+1} \log n + m \log n + w)$ *bits, simultaneously for all* h*. Here* n *is the length of the concatenation of* m *texts,* $\mathcal{C} = 0\ T_1 0\ T_2 \cdots 0\ T_m$*, and we assume that* $\sigma = o(n)$ *is the alphabet size and* $w = \Omega(\log n)$ *is the machine word size under the RAM model. The structure supports counting of the occurrences of a pattern* P *in* $O(|P| \log n(1 + \frac{\log \sigma}{\log \log n}))$ *time, and inserting and deleting a text* T *in* $O(|T| \log n(1 + \frac{\log \sigma}{\log \log n}))$ *time. After counting, any occurrence can be located in time* $O(\log n + \log_\sigma n \log \log n)$*. Any substring of length* ℓ *from any* T *in the collection can be displayed in time* $O(\log n + \log_\sigma n \log \log n + \ell(1 + \frac{\log \sigma}{\log \log n}))$*. For* $h \leq (\alpha \log_\sigma n) - 1$*, for any constant* $0 < \alpha < 1$*, the space complexity simplifies to* $nH_h(\mathcal{C}) + o(n \log \sigma) + O(m \log n + w)$*.*

Another important application that derives from this one is the compressed construction of text indexes. For example, a variant of the FM-index [3] requires h-th entropy space once built, but in order to build it we need $O(n \log n)$ bits of space. The previous theorem can be used in order to build the FM-index of a text by starting with an empty collection and inserting the text T of interest. Our new results make this process faster.

Theorem 6. *The Alphabet-Friendly FM-index of a text* $T[1, n]$ *over an alphabet of size* σ *can be built using* $nH_h(T) + o(n \log \sigma)$ *bits, simultaneously for all* $h \leq (\alpha \log_\sigma n) - 1$ *and any constant* $0 < \alpha < 1$*, in time* $O(n \log n(1 + \frac{\log \sigma}{\log \log n}))$*.*

We note that this is the same asymptotic space required for the final, static, FM-index [3]. This result has some obvious consequences on building suffix arrays [12] and computing the *Burrows-Wheeler Transform* [1] of T in little space, which we omit for lack of space.

In [2] they show that the *Dynamic Bit-Sequence with Indels* problem can be solved in $O(\frac{\log n}{\log \log n})$ time for all operations, using $O(n)$ bits of space. Combining with wavelet trees one achieves $O(n \log \sigma)$ bits of space and $O(\frac{\log n \log \sigma}{(\log \log n)^2})$ time for *Dynamic Sequences with Indels* . This raises the challenge of achieving that time within $nH_0 + o(n \log \sigma)$ bits of space.

Alternatively, one would like to improve the space to high-order entropy (not only for the *Dynamic Text Collection* problem, but for the *Dynamic Sequence with Indels* problem). This has not been achieved even if we limit the operations to *access*, *insert*, and *delete*. The dynamic support for the existing nH_k-space solutions to *access* is currently null or very rudimentary [18,6,4].

References

1. Burrows, M., Wheeler, D.: A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation (1994)
2. Chan, H., Hon, W., Lam, T., Sadakane, K.: Compressed indexes for dynamic text collections. ACM TALG 3(2), 21 (2007)
3. Ferragina, P., Manzini, G., Mäkinen, V., Navarro, G.: Compressed representations of sequences and full-text indexes. ACM TALG 3(2) (2007) (article 20)
4. Ferragina, P., Venturini, R.: A simple storage scheme for strings achieving entropy bounds. Theoretical Computer Science 372(1), 115–121 (2007)
5. Foschini, L., Grossi, R., Gupta, A., Vitter, J.: When indexing equals compression: Experiments with compressing suffix arrays and applications. ACM TALG 2(4), 611–639 (2006)
6. González, R., Navarro, G.: Statistical encoding of succinct data structures. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 295–306. Springer, Heidelberg (2006)
7. Grossi, R., Gupta, A., Vitter, J.: High-order entropy-compressed text indexes. In: Proc. 14th SODA, pp. 841–850 (2003)
8. Hon, W.-K., Sadakane, K., Sung, W.-K.: Succinct data structures for searchable partials sums. In: Ibaraki, T., Katoh, N., Ono, H. (eds.) ISAAC 2003. LNCS, vol. 2906, pp. 505–516. Springer, Heidelberg (2003)
9. Lee, S., Park, K.: Dynamic rank-select structures with applications to run-length encoded texts. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 95–106. Springer, Heidelberg (2007)
10. Mäkinen, V., Navarro, G.: Dynamic entropy-compressed sequences and full-text indexes. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 307–318. Springer, Heidelberg (2006), ftp://ftp.dcc.uchile.cl/pub/users/gnavarro/dynamic.ps.gz
11. Mäkinen, V., Navarro, G.: Implicit compression boosting with applications to self-indexing. In: Ziviani, N., Baeza-Yates, R. (eds.) SPIRE 2007. LNCS, vol. 4726, pp. 214–226. Springer, Heidelberg (2007)
12. Manber, U., Myers, G.: Suffix arrays: A new method for on-line string searches. SIAM Journal of Computing 22, 935–948 (1993)
13. Manzini, G.: An analysis of the Burrows-Wheeler transform. Journal of the ACM 48(3), 407–430 (2001)
14. Munro, I.: Tables. In: Chandru, V., Vinay, V. (eds.) FSTTCS 1996. LNCS, vol. 1180, pp. 37–42. Springer, Heidelberg (1996)
15. Navarro, G., Mäkinen, V.: Compressed full-text indexes. ACM Computing Surveys 39(1) (2007) (article 2)

16. Raman, R., Raman, V., Rao, S.: Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In: Proc. 13th SODA, pp. 233–242 (2002)
17. Raman, R., Rao, S.S.: Succinct dynamic dictionaries and trees. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 357–368. Springer, Heidelberg (2003)
18. Sadakane, K., Grossi, R.: Squeezing succinct data structures into entropy bounds. In: Proc. 17th SODA, pp. 1230–1239 (2006)

An Improved Algorithm Finding Nearest Neighbor Using Kd-trees

Rina Panigrahy

Microsoft Research, Mountain View CA, USA
rina@microsoft.com

Abstract. We suggest a simple modification to the Kd-tree search algorithm for nearest neighbor search resulting in an improved performance. The Kd-tree data structure seems to work well in finding nearest neighbors in low dimensions but its performance degrades even if the number of dimensions increases to more than two. Since the exact nearest neighbor search problem suffers from the curse of dimensionality we focus on approximate solutions; a c-approximate nearest neighbor is any neighbor within distance at most c times the distance to the nearest neighbor. We show that for a randomly constructed database of points if the query point is chosen close to one of the points in the data base, the traditional Kd-tree search algorithm has a very low probability of finding an approximate nearest neighbor; the probability of success drops exponentially in the number of dimensions d as $e^{-\Omega(d/c)}$. However, a simple change to the search algorithm results in a much higher chance of success. Instead of searching for the query point in the Kd-tree we search for a random set of points in the neighborhood of the query point. It turns out that searching for $e^{\Omega(d/c)}$ such points can find the c-approximate nearest neighbor with a much higher chance of success.

1 Introduction

In this paper we study the problem of finding the nearest neighbor of a query point in a high dimensional (at least three) space focusing mainly on the Euclidean space: given a database of n points in a d dimensional space, find the nearest neighbor of a query point. This fundamental problem arises in several applications including data mining, information retrieval, and image search where distinctive features of the objects are represented as points in $\mathbb{R}^d$ [28,30,6,7,12,11,27,10].

One of the earliest data structures proposed for this problem that is still the most commonly used is the Kd-tree [3] that is essentially a hierarchical decomposition of space along different dimensions. For low dimensions this structure can be used for answering nearest neighbor queries in logarithmic time and linear space. However the performance seems to degrade as a number of dimensions becomes larger than two. For high dimensions, the exact problem of nearest neighbor search seems to suffer from the curse of dimensionality; that is, either the running time or the space requirement grows exponentially in d. For instance Clarkson [5] makes use of $O(n^{\lceil d/2 \rceil (1+\delta)})$ space and achieves $O(2^{O(d \log d)} \log n)$ time. Meiser [24] obtains a query time of $O(d^5 \log n)$ but with $O(n^{d+\delta})$ space.

The situation is much of a better for finding an approximate solution whose distance from the query point is at most $1 + \epsilon$ times its distance from the nearest neighbor [2, 21, 18, 22]. Arya et. al. [2] use a variant of Kd-trees that they call BDD-trees (Balanced Box-Decomposition trees) that performs $(1+\epsilon)$-approximate nearest neighbor queries in time $O(d\lceil 1 + 6d/\epsilon \rceil^d \log n)$ and linear space. For arbitrarily high dimensions, Kushilevitz et. al. [22] provide an algorithm for finding an $(1 + \epsilon)$-approximate nearest neighbor of a query point in time $\tilde{O}(d \log n)$ using a data structure of size $(nd)^{O(1/\epsilon^2)}$. Since the exponent of the space requirement grows as $1/\epsilon^2$, in practice this may be prohibitively expensive for small ϵ. Indeed, since even a space complexity of $(nd)^2$ may be too large, perhaps it makes more sense to interpret these results as efficient, practical algorithms for c-approximate nearest neighbor where c is a constant greater than one. Note that if the gap between the distance to the nearest neighbor and to any other point is more than a factor of c then the c-approximate nearest neighbor is same as the nearest neighbor. So in such cases – which may very well hold in practice – these algorithms can indeed find the nearest neighbor.

Indyk and Motwani [18] provide results similar to those in [22] but use hashing to perform approximate nearest neighbor search. They provide an algorithm for finding the c-approximate nearest neighbor in time $\tilde{O}(d + n^{1/c})$ using an index of size $\tilde{O}(n^{1+1/c})$ (while their paper states a query time of $\tilde{O}(dn^{1/c})$, if d is large this can easily be converted to $\tilde{O}(d + n^{1/c})$ by dimension reduction). In their formulation, they use a locality sensitive hash function that maps points in the space to a discrete space where nearby points out likely to get hashed to the same value and far off points out likely to get hashed to different values. Precisely, given a parameter m that denotes the probability that two points at most r apart hash to the same bucket and g the probability that two points more than cr apart hash to the same bucket, they show that such a family of hash functions can find a c-approximate nearest neighbor in $\tilde{O}(d + n^\rho)$ time using a data structure of size $\tilde{O}(n^{1+\rho})$ where $\rho = log(1/m)/log(1/g)$. Recently, Andoni and Indyk [1] obtained and improved locality sensitive hash function for the Euclidean norm resulting in a ρ value of $O(1/c^2)$ matching the lower bounds for locality sensitive hashing method from [25]. An information theoretic formulation of locality sensitive hashing was studied in [26] resulting in a data structure of linear size; the idea there was to perturb the query point before searching for it in the hash table and do this for a few iterations.

However, to the best of our knowledge the most commonly used method in practice is the old Kd-tree. We show that a simple modification to the search algorithm on a Kd-tree can be used to find the nearest neighbor in high dimensions more efficiently. The modification consists of simply perturbing the query point before traversing the tree, and repeating this for a few iterations. This is essentially the same idea from [26] on locality sensitive hash functions applied to Kd-trees. For a certain database of random points if we choose a query point close to one of the points in the database, we show that the traditional Kd-tree search algorithm has a very low probability of finding the nearest neighbor – $e^{-\Omega(d/c)}$ where c is a parameter that denotes how much closer the query point is to the nearest neighbor than to other points in the database. Essentially c is the inverse ratio of the distance of the nearest query point to the nearest and the second nearest neighbor; so one can think of the nearest neighbor as a c-approximate nearest neighbor. Next we show that the modified algorithm significantly improves the probability of finding the c-approximate nearest neighbor by performing $e^{O(d/c)}$ iterations. One may be tempted to think that if the traditional algortithm has a success probability of $e^{-\Omega(d/c)}$, perhaps we could simply repeat it $e^{O(d/c)}$ times to boost the probability to a constant. However, this doesn't work in our situation since repeating the same query in a tree will always give the same result. One can use a different (randomly constructed) tree each time but this will blow up the space requirement to that of $e^{O(d/c)}$ trees. Our result essentially shows how to boost the probability while using one tree but by perturbing the query point each time. The intuition behind this approach is that when we perturb the query point and then search the tree, we end up looking not only at one leaf region but also the neighboring leaf regions that are close to the query point thus increasing the probability of success. This is similar in spirit to some of the variants of Kd-trees such as [2] that maintain explicit pointers from each leaf to near by leaves; our method on the other hand performs this implicity without maintaining pointers which keep the data structure simple. We also provide empirical evidence through simulations to show that the simple modification results in high probability of success in finding nearest neighbor search in high dimensions.

We apply the search algorithms on a planted instance of the nearest neighbor problem on a database of n points chosen randomly in a unit d-dimensional cube. We then plant a query point close of one of the database points p (chosen randomly) and then ask the search algorithm for the nearest neighbor of our planted query point. The query point is chosen randomly on a ball of a certain radius around p so that it is much closer to p than to any other point in the database – by a factor of c. We measure the efficacy of an algorithm by looking at the probability that it returns the nearest neighbor p on query q. It is for this distribution that we will show a low success probability for the traditional Kd-tree search algorithm and a high probability of success for our modified algorithm.

In our experiments we observed that our modified algorithm indeed boosts the probability of success. For instance in a database of million points in 3 dimensions we plant a query point close to a random database point; the query point is closer to its nearest neighbor than any other point by a factor of $c = 2$. For this instance we find that the Kd-tree search algorithm succeeds with probability 74%; whereas, this can be boosted to about 90% byrunning our modified algorithm with only 5 iterations. The success probability increases with more iterations.

2 Preliminaries

2.1 Problem Statement

Given a set S of n points in d-dimensions, our objective is to construct a data structure that given a query finds the nearest (or a c-approximate) neighbor. A c-approximate near neighbor is a point at distance at most c times the distance to the nearest neighbor. Alternatively it can be viewed as finding the exact nearest neighbor when the second nearest neighbor is more than c times the distance to the nearest neighbor.

We also work with the following decision version of the c-approximate nearest neighbor problem: given a query point and a parameter r indicating the distance to its nearest neighbor, find any neighbor of the query point that is that distance at most cr. We will refer to this decision version as the (r, cr)-nearest neighbor problem and a solution to this as a (r, cr)-nearest neighbor. It is well known that the reduction to the decision version adds only a logarithmic factor in the time and space complexity [18,13]. We will be working with the euclidian norm in $\mathbb{R}^d$ space.

2.2 Kd-trees

Although many different flavors of Kd-trees have been devised, their essential strategy is to hierarchically decompose space into a relatively small number of cells such that no cell contains too many input objects. This provides a fast way to access any input object by position. We traverse down the hierarchy until we find the cell containing the object. Typical algorithms construct Kd-trees by partitioning point sets recursively along with different dimensions. Each node in the tree is defined by a plane through one of the dimensions that partitions the set of points into left/right (or up/down) sets, each with half the points of the parent node. These children are again partitioned into equal halves, using planes through a different dimension. Partitioning stops after $\log n$ levels, with each point in its own leaf cell. The partitioning loops through the different dimensions for the different levels of the tree, using the median point for the partition. Kd-trees are known to work well in low dimensions but seem to fail as the number of dimensions increase beyond three.

Notations:

– $B(p, r)$: Let $B(p, r)$ denote the sphere of radius r centered at p a point in $\mathbb{R}^d$; that is the set of points at distance r from p.
– $I(X)$: For a discrete random variable X, let $I(X)$ denote its information-entropy. For example if X takes N possible values with probabilities $w_1, w_2, ..., w_N$ then $I(X) = I(w_1, w_2, .., w_N) = \sum I(w_i) = \sum -w_i \log w_i$. For a probability value p, we will use the overloaded notation $I(p, 1-p)$ to denote $-p \log p - (1-p) \log(1-p)$
– $N(\mu, r), \eta(x)$: Let $N(\mu, r)$ denote the normal distribution with mean μ and variance r^2 with probability density function given by $\frac{1}{r\sqrt{2\pi}} e^{-(x-\mu)^2/(2r^2)}$. Let $\eta(x)$ denote the function $\frac{1}{\sqrt{2\pi}} e^{-x^2/2}$.

- $N^d(p,r)$: For the d-dimensional Euclidean space, for a point $p = (p_1, p_2, ..., p_d) \in \mathbb{R}^d$ let $N^d(p,r)$ denote the normal distribution in $\mathbb{R}^d$ around the point p where the ith coordinate is randomly chosen from the normal distribution $N(p_i, r/\sqrt{d})$ with mean p_i and variance r^2/d. It is well known that this distribution is spherically symmetric around p. A point from this distribution is expected to be at root-mean squared distance r from p; in fact, for large d its distance from p is close to r with high probability (see for example lemma 6 in [18])
- $erf(x), \Phi(x)$: The well-known error function $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-x^2}\, dx$, is equal to the probability that a random variable from $N(0, 1/\sqrt{2})$ lies between $-x$ and x. Let $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-x^2/2}\, dx = \frac{1-erf(x/\sqrt{2})}{2}$. For $x \geq 0$, $\Phi(x)$ is the probability that a random variable from the distribution $N(0,1)$ is greater than x.
- $Pinv(k,r)$: Let $Pinv(k,r)$ denote the distribution of the time it takes to see k events in a poisson process of rate r. $Pinv(1,r)$ is the exponential distribution with rate r.

3 An Improved Search Algorithm on Kd-trees

We will study the nearest neighbor search problem on the following planted instance in a random database. Consider a database of points chosen randomly and uniformly from the unit cube $[0,1]^d$. We will then choose a random database point p and implant query point q that is about c times closer to p than its distance to any other database point. Success of an algorithm is measured by the probability of finding the nearest neighbor. Let r denote the distance of a random point in $[0,1]^d$ to the nearest database point. We will show that if the query point is chosen to be a random point at distance about r/c around p (precisely, q is chosen from $N^d(p, r/c)$) then the probability that a Kd-tree search algorithm reports the nearest neighbor p is about $e^{-\Omega(d/c)}$.

We then propose a minor modification to the search algorithm so as to boost this probability to a large value. The modification is simple: Instead of searching for the query point q in the Kd-tree, perturb it randomly by distance r/c (precisely, the perturbed point is chosen from $N^d(q, r/c)$) and search for this perturbed point in the Kd-tree using the regular Kd-tree search algorithm. Repeat this for $e^{O(d/c)}$ random perturbations and report the nearest neighbor found using the Kd tree. We will show that this simple modification results in an a much higher chance of finding the nearest neighbor p.

4 Traditional Kd-tree Algorithms Fails with High Probability

In this section we show that the traditional Kd-tree algorithm fails in solving the planted instance of the nearest neighbor problem with high probability; the success probability is at most $e^{-\Omega(d/c)}$. The following lemma estimates r, the distance from a random point in the unit cube to the nearest database point.

Lemma 1. *With high probability of $1 - O(1/2^d)$, the distance r of a random point in the unit cube to the nearest point in the database is $\Theta(\sqrt{d}/n^{1/d})$*

Proof. The volume of a sphere of radius r in d dimensions is $\frac{2\pi^{d/2}r^d}{d\Gamma(d/2)} = \Theta(1)\frac{1}{d^{3/2}}(\frac{2e\pi}{d})^{d/2}r^d$. Since there are n random points in the database, the expected number of points in a ball of radius r is $\Theta(1)n\frac{1}{d^{3/2}}(\frac{2e\pi}{d})^{d/2}r^d$. So the radius for which the expected number of points in this ball is one is given by $r = \Theta(\frac{\sqrt{d}}{n^{1/d}})$ (We have used the Stirlings approximation for the Γ function which is valid only for integral arguments. Although, $d/2$ may not be integral it lies between two consecutive integers and this proves the Θ bound on r. Note that even if the point is near the boundary of the cube, at least $1/2^d$ fraction of the sphere is inside the cube which does not change the value of r by more than a constant factor). The value of r is sharply concentrated around this value as the volume of the sphere in high dimensions changes dramatically with a change in the radius; by a factor of 2^d with a factor 2 change in the radius. High concentration bounds can be used to show that there must be at least one point for larger radii and almost no point for smaller radii. For this value of r, the probability that there is a database point at distance $r/2$ is at most $n1/(2^dn) = 1/2^d$. And the probability that there is no point within distance $2r$ is atmost $(1 - 2^d/n)^n = exp(-2^d)$.

Lemma 2. *If we pick a random query point q at distance r/c from a random point in the database then the probability that a Kd-tree search returns the nearest neighbor is at most $e^{-\Omega(d/c)}$.*

Proof. Let us focus on a leaf cell containing the point p. We need to compute the probability that a random query point q chosen from $N(p, r/c)$ lies within the same cell.

If we project the cell along any one dimension we get an interval. Since the Kd-tree has depth $\log n$, the number of branches along any one dimension is $\frac{\log n}{d}$. And since we are picking a random cell, the expected value of the length l of this interval containing p is $E[l] = 1/2^{\frac{\log n}{d}} = 1/n^{1/d}$. So with probability at least $1/2$, $l < 2/n^{1/d}$. Conditioned on the event that $l < 2/n^{1/d}$, we will argue that the probability that the query point q lies outside the interval is $\Omega(1/c)$. To see this note that if x denotes the distance of p from one of the end points of the interval then x is distributed uniformly in the range $[0, l]$ since p is a random point in its cell. Since along one dimension q and p are separated by a distance of $N(0, \frac{r}{c\sqrt{d}}) = N(0, \frac{1}{cn^{1/d}})$ (ignoring the Θ in the expression for r for simpler notation), the probability that q does not line the interval is $\Phi(xcn^{1/d})$. By a standard change of variable to $z = xcn^{1/d}$ this amounts to the expected value of $\Phi(z)$ where z is uniformly distributed in the range $[0, 2c]$. Looking at integral values of z, observe that $\Phi(z) = e^{-\Omega(z^2)}$ drops at least geometrically with each increment of z. So the expected value of $\Phi(z)$ is $\int_0^{2c} \frac{1}{2c}e^{-\Omega(z^2)}\,dz = \Omega(1/c)$.

This implies that along any one dimension the probability that q lies within the projection along that dimension of the cell containing p is at most $1 - \Omega(1/c)$. Since values along all dimensions are independent, the probability that q lies within the cell of p is at most $(1 - \Omega(1/c))^d = e^{-\Omega(d/c)}$.

5 Modified Algorithm has a Higher Probability of Success

We will show that the modified algorithm has a much higher probability of success. Although our theoretical guarantee only provides a success probability of $\Omega(c/d)$, we believe this is simply an artifact of our analysis. Our experimental results show that the success probability is very high.

Theorem 1. *With probability at least $\Omega(c/d)$, the new search algorithm finds the nearest neighbor in $e^{O(d/c)}$ iterations for the random database and the planted query point.*

The proof of this theorem follows from the following two lemmas.

Guessing the value of a random variable: We first present a lemma that states the number of times one needs to guess a random variable with a given distribution so as to guess its correct value. If a random variable takes one of N discrete values with equal probability then a simple coupon collection based argument shows that if we guess N random values at least one of them should hit the correct value with constant probability. The following lemma taken from [26] states the required number of samples for arbitrary random variables so as to 'hit' a given random value of the variable.

Lemma 3. *[26] Given an random instance x of a discrete random variable with a certain distribution D with entropy I, if $O(2^I)$ random samples are chosen from this distribution at least one of them is equal to x with probability at least $\Omega(1/I)$.*

Proof. Assume that the distribution D takes N values with probabilites $w_1, w_2, ..., w_N$. x is equal to the ith value with probability w_i. If s samples are randomly chosen from the distribution, the probability that at least one of them takes the ith value is $1 - (1 - w_i)^s$. After $s = 4.(2^I + 1)$ samples the probability that x is chosen is $\sum_i w_i[1 - (1 - w_i)^s]$. If $w_i \geq 1/s$ then the term in the summation is at least $w_i(1 - 1/e)$. Divide the probability values $w_1, w_2, ..., w_N$ into two parts – those at least $1/s$ and the others less than $1/s$. So if all the $w_i's$ that are at least $1/s$ add up to at least $1/I$ then the above sum is at least $\Omega(1/I)$. Otherwise we have a collection of $w_i's$ each of which is at most $1/s$ and they together add up to more than $1 - 1/I$.

But then by paying attention to these probabilities we see that the entropy $I = \sum_i w_i \log(1/w_i) \geq \sum_i w_i \log s \geq (1 - 1/I) \log s \geq (1 - 1/I)(I + 2) = I + 1 - 2/I$. For $I \geq 4$, this is strictly greater than I, which is a contradiction. If $I < 4$ then the largest w_i must be at least $1/16$ as otherwise a similar argument shows that

$I = \sum_i w_i \log(1/w_i) > w_i \log 16 = 4$, a contradiction; so in this case even one sample guesses x with constant probability.

Distribution of Kd-tree partition sizes: Let us now estimate the distribution of the Kd-tree partition sizes as we walk down the tree. Consider a random point p in the database and the Kd-tree traversal while searching p. As we walk down the tree the cell containing p shrinks from the entire unit cube to the single leaf cell containing p. The leaf cell of p is specified by $\log n$ choices of left or right while traversing the Kd-tree. Let us track the length of the cell along a dimension as we walk down the tree. The number of decisions along each dimension is $\frac{\log n}{d}$. Focusing on the first dimension (say x-axis), look at the interval along the x-axis containing p. The interval gets shorter as we partition along the x-axis. Let l_i denote the length of this interval after the i^{th} branch along the x-axis; note that two successive branches along the x-axis are separated by $d-1$ branches along the other dimensions. The initial cell length $l_0 = 1$. $E[l_i] = 1/2^i$; let us look at the distribution of l_i. The database points projected along the x-axis gives us n points randomly distributed in $[0,1]$. For large n, any two successive values are separated by an exponential distribution with rate $1/n$, and the distance between k consecutive points is given by $Pinv(k, 1/n)$, the inverse poisson distribution at rate $1/n$ for k arrivals. The median point is the first partition point dividing the interval into two parts containing $n/2$ points each. p lies in one of these intervals of length l_1 distributed as $Pinv(n/2, 1/n)$.

To find the distribution of l_2, note that there are $d-1$ branches along other dimensions between the first and the second partition along the x-axis, and each branch eliminates 1/2 the points from p's cell. Since coordinate values along different dimensions are independent, this corresponds to randomly sampling $1/2^{d-1}$ fraction of the points from the interval after the first branch. From the points that are left, we partition again along the median and take one of the two parts; each part contains $n/2^d$ points. Since the original n points are chosen randomly and we sample at rate $1/2^{d-1}$, after the second branch successive points are separated by an exponential distribution with rate $1/2^{d-1}$. So after the second branch along x-axis the side of the interval l_2 is distributed as $Pinv(n/2^d, 1/2^{d-1})$. Note that this is not entirely accurate since the points are not sampled independently but instead exactly $1/2^{d-1}$ fraction of the points are chosen in each part; however thinking of n and 2^d as large we allow ourselves this simplification.

Continuing in this fashion, we get that the interval corresponds to l_i contains $n/2^{di}$ points and the distance between successive points is distributed as the exponential distribution with rate $2^{(d-1)i}/n$. So l_i is distributed as $Pinv(n/2^{di}, 2^{(d-1)i}/n)$. Since p is a random point in its cell, the distance x_i between p and the dividing plane at the i^{th} level is a random value between 0 and l_i. At the leaf level $l_{\log n/d}$ is distributed as $Pinv(1, 2^{\frac{(d-1)\log n}{d}}/n)$ which is same as the exponential distribution with rate $2^{\frac{(d-1)\log n}{d}}/n = 1/n^{1/d}$. Extending this argument to other dimensions tells us that at the leaf level the length of the cell along any dimension is given by an exponential distribution with rate $O(1/n^{1/d})$.

Now p is a random database point and q is a random point with distribution $N^d(p, r/c)$. Let $L(p)$ denote the leaf of the Kd-tree (denoted by T) where p resides. For a fixed Kd-tree, look at the distribution of $L(p)$ given q. We will estimate $I[L(p)|q, T]$ – the information required to guess the cell containing p given the query point q and the tree structure T (the tree structure T includes positions of the different partition planes in the tree).

Lemma 4. $I[L(p)|q, T] = O(d/c)$

Proof. The cell of p is specified by $\log n$ choices of left or right while traversing the Kd-tree. The number of decisions along each dimension is $\frac{\log n}{d}$. Let b_{ij} ($i \in 1 \cdot\cdot \log n/d, j \in 0 \cdot\cdot d{-}1$) denote this choice in the i^{th} branch along dimension j. Let B_{ij} denote the set of all the previous branches on the path to the branch point b_{ij}. Then since the leaf cell $L(p)$ is completely specified by the branch choices b_{ij}, $I[L(p)|q, T] \leq \sum_{i,j} I[b_{ij}|B_{ij}, q, T]$. Let us now bound $I[b_{ij}|B_{ij}, q, T]$. Focusing on the first dimension for simplicity, for the i^{th} choice in the first dimension, the distance x_i between p and the partition plane is uniform in the range $[0, l_i]$ where l_i has mean $1/2^i$ (and distribution $Pinv(n/2^{di}, 2^{(d-1)i}/n)$). The distance between q and p along the dimension is given by $N(0, \frac{r}{c\sqrt{d}}) = N(0, \frac{1}{cn^{1/d}})$ (again ignoring the Θ in the expression for r for simpler notations). So it is easy to verify that the distribution of the distance y_i of q from the partition plane is close to uniform (up to constant factors) in the range $[0, 1/2^i]$ (essentially y_i is obtained by first picking a random value in the interval $[0, l_i]$ and then perturbing it by a small value drawn from $N(0, \frac{1}{cn^{1/d}})$).

If the distance y_i of q from the partition plane in the i-th branch is equal to y, the probability that p and q are on different sides is $\Phi(ycn^{1/d})$. Since y_i is completely specified by the path B_{i0}, q and the tree structure T, $I[b_{i0}|B_{i0}, q, T] \leq I[b_{i0}|y_i = y] = E_y[I(\Phi(ycn^{1/d}), 1 - \Phi(ycn^{1/d}))]$. So we need to bound the expected value of $I(\Phi(ycn^{1/d}), 1 - \Phi(ycn^{1/d}))$. Again by a change of variable to $z = ycn^{1/d}$ this is equal to $E_z[I(\Phi(z), 1 - \Phi(z))]$.

We will argue that this is $O(\frac{2^i}{cn^{1/d}})$. Looking at integral values of z, observe that $I(\Phi(z), 1 - \Phi(z)) = e^{-\Omega(z^2)}$ drops faster than geometrically with each increment of z. It is $o(\frac{2^i}{cn^{1/d}})$ for $z > \frac{cn^{1/d}}{2^i}$. Further since the distribution of y in the range $[0, 1/2^i]$ is uniform (up to constant factor) so is the distribution of z in the range $[0, \frac{cn^{1/d}}{2^i}]$. So the probability that z lies in any unit interval in this range is $O(\frac{2^i}{cn^{1/d}})$. So the expected value of $I(\Phi(z), 1 - \Phi(z))$ is $O(\frac{2^i}{cn^{1/d}})$. So $I[b_{i0}|B_{ij}, q, T] = O(\frac{2^i}{cn^{1/d}})$. Similarly bounding and summing over all dimensions, $I[L(p)|q, T] \leq \sum_{i,j} I[b_{ij}|B_{ij}, q, T] \leq d \sum_i O(\frac{2^i}{cn^{1/d}}) = O(d/c)$

We are now ready to complete the proof of Theorem 1.

Proof. **[of Theorem 1].** The proof follows essentially from lemmas 3 and 4. Lemma 4 states that $I[L(p)|q, T] = O(d/c)$. But $I[L(p)|q, T]$ is the expected value of $I[L(p)]$ for a random fixed choice of q and T. So by Markov's inequality for a random fixed choice of q and T, with probability at least $1/2$, $I[L(p)] \leq$

$2I[L(p)|q,T] = O(d/c)$. So again with probability at least 1/2 for a random instance of the problem, $I[L(p)] = O(d/c)$. Now lemma 3 states that $2^{O(c/d)}$ samples from the distribution of $L(p)$ given q must hit upon the correct cell containing p, completing the proof.

6 Experiments

We perform experiments on the planted instance of the problem with both the standard Kd-tree algorithm and our modified algorithm on a database of $n = 1$ million points chosen randomly in the unit cube in d dimensions. We then picked a random point p in the database and measured its distance r from the nearest other database point. We then planted a query point q with the distribution $N^d(p, r/c)$ so that it is at distance about r/c from p. We tried four different values for d : $3, 5, 10$ and 20; and three different values for c : $4/3, 2$ and 4. For each combination of values we performed $10,000$ search operations using both the traditional Kd-tree search algorithm and our modified algorithm. In the modified algorithm the query point was perturbed before the search and this was repeated for a few iterations; the number of iterations was varied from 5 to 30.

The success rates of finding the nearest neighbor are summarized in table 1. The third column shows the success rate of the traditional kd-tree search algortihm and the remaining columns state the success rate for the modified algorithm for different number of iterations. As can be seen, in 3 dimensions for $c = 4$, Kd-trees report the nearest neighbor 84% of the time whereas even with 5 iterations of the new algorithm this goes up to 96%. In 5 dimensions for $c = 4$ our algorithm boosts the success rate from 73% to 91% in 5 iterations and to 97.5% in 15

Table 1. Simulation Results: The entries indicate the percentage of times the nearest neighbor is found. As the number of iterations k is increased the success rate increases. The third column gives the success rate for the standard Kd-tree search algorithm. The latter columns report the performance of the modified algorithm for different number of iterations of searching for perturbed points.

d	c	Kd-tree	5 iter	15 iter	20 iter	25 iter	30 iter
3	4	84	96.1	98.8	99.3	99.3	99.8
3	2	73.9	89.5	97.4	98.4	99.0	98.7
3	4/3	73	88.5	96	96.6	98.7	98.7
5	4	73.6	91	97.5	98.1	98.5	99.3
5	2	54	78	92.1	94.9	94.4	96.2
5	4/3	50.7	71.3	87	91.2	92.3	94
10	4	60.7	80.5	94.8	96.6	96.7	96.8
10	2	36	56.4	77.6	84.3	86.6	88.4
10	4/3	25	43.7	61	70	73.4	75.6
20	4/3	13	25	28	41	42	46
20	2	22	42	67	68	70	72

iterations. In 10 dimensions for $c = 4$, 15 iterations raises the success rate from 60% to about 95%. In 20 dimensions for $c = 2$, Kd-trees succeed only 22% of the time, where as the new algorithm succeeds 67% of the time with 15 iterations.

References

1. Andoni, A., Indyk, P.: Near-Optimal Hashing Algorithms for Near Neighbor Problem in High Dimensions. In: Proceedings of the Symposium on Foundations of Computer Science (FOCS 2006) (2006)
2. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.: An optimal algorithm for approximate nearest neighbor searching. In: Proc. 5th ACM-SIAM Sympos. Discrete Algorithms, pp. 573–582 (1994)
3. Bentley, J.L., Friedman, J.H., Finkel, R.A.: An algorithm for finding best matches in logarithmic expected time. ACM Transactions on Mathematical Software 3(3), 209–226 (1977)
4. Borodin, A., Ostrovsky, R., Rabani, Y.: Lower bounds for high dimensional nearest neighbor search and related problems. In: Proceedings of the 31st ACM Symposium on Theory of Computing, pp. 312–321 (1999)
5. Clarkson, K.L.: Nearest neighbor queries in metric spaces. In: Proceedings of the 29th Annual ACM Symposium on Theory of Computing, May 1997, pp. 609–617 (1997)
6. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Trans. Information Theory IT-13, 21–27 (1967)
7. Deerwester, S., Dumais, S.T., Landauer, T.K., Furnas, G.W., Harshman, R.A.: Indexing by latent semantic analysis. Journal of the Society for Information Science 41(6), 391–407 (1990)
8. Dolev, D., Harari, Y., Parnas, M.: Finding the neighborhood of a query in a dictionary. In: Proc. 2nd Israel Symposium on Theory of Computing and Systems, pp. 33–42 (1993)
9. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.: Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. In: Proceedings of the Symposium on Computational Geometry (2004), Talk is available at: `http://theory.lcs.mit.edu/ indyk/brown.ps`
10. Devroye, L., Wagner, T.J.: Nearest neighbor methods in discrimination. In: Krishnaiah, P.R., Kanal, L.N. (eds.) Handbook of Statistics, vol. 2, North-Holland, Amsterdam (1982)
11. Fagin, R.: Fuzzy Queries in Multimedia Database Systems. In: Proc. ACM Symposium on Principles of Database Systems, pp. 1–10 (1998)
12. Flickner, M., Sawhney, H., Niblack, W., Ashley, J., Huang, Q., Dom, B., Gorkani, M., Hafner, J., Lee, D., Petkovic, D., Steele, D., Yanker, P.: Query by image and video content: The QBIC system. Computer 28, 23–32 (1995)
13. Har-Peled, S.: A replacement for voronoi diagrams of near linear size. In: Proceedings of the Symposium on Foundations of Computer Science (2001)
14. Indyk, P.: High-dimensional computational geometry, Dept. of Comput. Sci., Stanford Univ. (2001)
15. Indyk, P.: Approximate Nearest Neighbor under Frechet Distance via Product Metrics. In: ACM Symposium on Computational Geometry (2002)
16. Indyk, P.: Nearest neighbors in high-dimensional spaces. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, ch. 39, 2nd edn., CRC Press, Boca Raton (2004)

17. Indyk, P., Motwani, R., Raghavan, P., Vempala, S.: Locality-preserving hashing in multidimensional spaces. In: Proceedings of the 29th ACM Symposium on Theory of Computing, pp. 618–625 (1997)
18. Indyk, P., Motwani, R.: Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In: Proc. 30th Symposium on Theory of Computing, pp. 604–613 (1998)
19. Indyk, P., Thaper, N.: Embedding Earth-Mover Distance into the Euclidean space (manuscript, 2001)
20. Jayram, T.S., Khot, S., Kumar, R., Rabani, Y.: Cell-probe lower bounds for the partial match problem. In: Proc. 35th Annu. ACM Symp. Theory Comput., pp. 667–672 (2003)
21. Kleinberg, J.: Two algorithms for nearest-neighbor search in high dimension. In: Proc. 29th Annu. ACM Sympos. Theory Comput., pp. 599–608 (1997)
22. Kushilevitz, E., Ostrovsky, R., Rabani, Y.: Efficient search for approximate nearest neighbor in high dimensional spaces. In: Proc. of 30th STOC, pp. 614–623 (1998)
23. Linial, N., Sasson, O.: Non-Expansive Hashing. In: Proc. 28th STOC, pp. 509–517 (1996)
24. Meiser, S.: Point location in arrangements of hyperplanes. Information and Computation 106(2), 286–303 (1993)
25. Motwani, R., Naor, A., Panigrahy, R.: Lower Bounds on Locality Sensitive Hashing. In: Proceedings of the 22nd Annual ACM Symposium on Computational Geometry (2006)
26. Panigrahy, R.: Entropy based nearest neighbor search in high dimensions. In: SODA 2006: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, Miami, FL, pp. 1186–1195. ACM Press, New York (2006)
27. Pentland, A., Picard, R.W., Sclaroff, S.: Photobook: Tools for content-based manipulation of image databases. In: Proceedings of the SPIE Conference On Storage and Retrieval of Video and Image Databases, February 1994, vol. 2185, pp. 34–47 (1994)
28. van Rijsbergen, C.J.: Information Retrieval, Butterworths, London, United Kingdom (1990)
29. Vapnik, V.N., Chervonenkis, A.Y.: On the uniform convergence of relative frequencies of events to their probabilities. Theory Probab. Appl. 16, 264–280 (1971)
30. Salton, G.: Automatic Text Processing. Addison-Wesley, Reading (1989)

List Update with Locality of Reference

Spyros Angelopoulos[1,2], Reza Dorrigiv[1], and Alejandro López-Ortiz[1]

[1] Cheriton School of Computer Science, University of Waterloo
Waterloo, Ont., N2L 3G1, Canada
[2] Max-Planck-Institut für Informatic, Saarbrücken, Germany
{sangelop,rdorrigiv,alopez-o}@uwaterloo.ca

Abstract. It is known that in practice, request sequences for the list update problem exhibit a certain degree of locality of reference. Motivated by this observation we apply the locality of reference model for the paging problem due to Albers et al. [STOC 2002/JCSS 2005] in conjunction with bijective analysis [SODA 2007] to list update. Using this framework, we prove that Move-to-Front (MTF) is the unique optimal algorithm for list update. This addresses the open question of defining an appropriate model for capturing locality of reference in the context of list update [Hester and Hirschberg ACM Comp. Surv. 1985]. Our results hold both for the standard cost function of Sleator and Tarjan [CACM 1985] and the improved cost function proposed independently by Martínez and Roura [TCS 2000] and Munro [ESA 2000]. This result resolves an open problem of Martínez and Roura, namely proposing a measure which can successfully separate MTF from all other list-update algorithms.

1 Introduction

List update is a fundamental problem in the context of on-line computation. Consider an unsorted list of l items. The input to the algorithm is a sequence of n requests that must be served in an on-line manner. Let $\mathcal{A}$ be an arbitrary on-line list update algorithm. To serve a request to an item x, $\mathcal{A}$ linearly searches the list until it finds x. If x is the i^{th} item in the list, $\mathcal{A}$ incurs a cost i to access x. Immediately after this access, $\mathcal{A}$ can move x to any position closer to the front of the list at no extra cost. This is called a *free exchange*. Also $\mathcal{A}$ can exchange any two consecutive items at a cost of 1. These are called *paid exchanges*. An efficient algorithm can thus use free and paid exchanges to minimize the overall cost of serving a sequence. This is called the *standard cost model* [5].

The competitive ratio, first introduced formally by Sleator and Tarjan [22], has served as a practical measure for the study and classification of on-line algorithms in general and list-update algorithms in particular. An algorithm is said to be α-competitive (assuming a cost-minimization problem) if the cost of serving any specific request sequence never exceeds α times the optimal cost (up to some additive constant) of an *off-line* algorithm which knows the entire request sequence. List update algorithms were among the first algorithms studied using competitive analysis. Three well-known deterministic on-line algorithms are

Move-To-Front (MTF), *Transpose*, and *Frequency-Count* (FC). MTF moves the requested item to the front of the list whereas Transpose exchanges the requested item with the item that immediately precedes it. FC maintains an access count for each item ensuring that the list always contains items in non-increasing order of frequency count. Sleator and Tarjan showed that MTF is 2-competitive, while Transpose and FC do not have constant competitive ratios [22]. Since then, several other deterministic and randomized on-line algorithms have been studied using competitive analysis. (See [16,1,4,14] for some representative results.)

Notwithstanding its wide applicability, competitive analysis has some drawbacks. For certain problems, it gives unrealistically pessimistic performance ratios and fails to distinguish between algorithms that have vastly differing performance in practice. Such anomalies have led to the introduction of many alternatives to competitive analysis of on-line algorithms (see [13] for a comprehensive survey). While list update algorithms with better competitive ratio tend to have better performance in practice the validity of the cost model has been debated. More precisely, Martínez and Roura [18] and Munro [19], independently addressed the drawbacks of the standard cost model. Let $(a_1, a_2, \ldots, a_l)$ be the list currently maintained by an algorithm $\mathcal{A}$. Martínez and Roura argued that in a realistic setting a complete rearrangement of all items in the list which precede item a_i would in practice require time proportional to i, while this has cost proportional to i^2 in the standard cost model. Munro provided the example of accessing the last item of the list and then reversing the entire list. The real cost of this operation in an array or a linear link list should be $O(l)$, while it costs about $l^2/2$ in the standard cost model. As a consequence, their main objection to the standard model is that it prevents on-line algorithms from using their true power. They instead proposed a new model in which the cost of accessing the i^{th} item of the list plus the cost of reorganizing the first i items is linear in i. We will refer to this model as the *modified cost model*. Surprisingly, it turns out that the off-line optimum benefits substantially more from this realistic adjustment than the on-line algorithms do. Indeed, under this model, every on-line algorithm has amortized cost of $\Theta(l)$ per access for some arbitrary long sequences, while an optimal off-line algorithm incurs a cost of $\Theta(\log l)$ on every sequence and hence all on-line list update algorithm have a constant competitive ratio of $\Omega(l/\log l)$. One may be tempted to argue that this is proof that the new model makes the off-line optimum too powerful and hence this power should be removed, however this is not correct as in real life on-line algorithms can rearrange items at the cost indicated. Observe that the ineffectiveness of this power for improving the worst case competitive ratio does not preclude the possibility that under certain realistic input distributions (or other similar assumptions on the input) this power might be of use. Martínez and Roura observed this and posed the question: "an important open question is whether there exist alternative ways to define competitiveness such that MTF and other good online algorithms for the list update problem would be competitive, even for the [modified] cost model".

As well, a common objection to competitive analysis is that it relies on an optimal off-line algorithm, OPT, as a baseline for comparing on-line algorithms.

While this may be convenient, it is rather indirect: one could argue that in comparing two online algorithms $\mathcal{A}$ and $\mathcal{B}$ all the information we should need is the cost incurred by the algorithms on each request sequence. For example, for some problems OPT is too powerful, causing all on-line algorithms to seem equally bad. Certain alternative measures allow direct comparison of on-line algorithms, for example the *Max-Max Ratio* [9], *Relative Worst Order Ratio* [11], Bijective Analysis and Average Analysis [6]. These measures have been applied mostly to the paging problem as well as some other on-line problems. We are not aware of any result in the literature that applies the above measures to on-line list update algorithms.

Another issue in the analysis of on-line algorithms is that "real-life" sequences usually exhibit *locality of reference*. Informally, this property suggests that the currently requested item is likely to be requested again in the near future. For the paging problem, several models for capturing locality of reference have been proposed [23,2,8]. Input sequences of list update algorithms in practice show locality of reference [15,21,10] and on-line list update algorithms try to take advantage of this property [15,20]. Hester and Hirschberg [15] posed the question of providing a satisfactory formal definition of locality of reference for the list update problem as an open problem. However, to the best of our knowledge, locality of reference for list update algorithms has not been formally studied. In addition, it has been commonly assumed, based on intuition and experimental evidence, that MTF is the best algorithm on sequences with high locality of reference, e.g., Hester and Hirschberg [15] claim: "move-to-front performs best when the list has a high degree of locality" (see also [3], page 327).

To this end, we introduce a natural measure of locality of reference. Perhaps not surprisingly, this measure seems to parallel MTF's behaviour as the latter has been tailored to benefit from locality of reference. This should not be construed as a drawback of the measure, but rather as evidence of the fact that the design of the MTF algorithm *optimally* incorporates the presence of locality of reference into its choices. Our theoretical proof of the optimality of MTF in this context is then perhaps not surprising, yet this fact had eluded proof until now.

Our Results. We begin by showing that all on-line list update algorithms are equivalent according to Bijective Analysis under the modified cost model. We then extend a model for locality of reference, proposed by Albers et al. [2] in the context of the paging problem to the list update problem. The validity of the extended model is supported by experimental results obtained on the Calgary Corpus, which is frequently used as a standard benchmark for evaluating the performance of compression algorithms (and by extension list update algorithms, e.g. [7]). Thus, we resolve the open problem posed by Hester and Hirschberg [15]. Our main result proves that under both the standard and the modified cost functions MTF is never outperformed in our model, while it always outperforms *any other on-line list update algorithm* in at least one instance. As mentioned earlier, Martínez and Roura [18] posed the open problem of finding an alternative measure that shows the superiority of MTF in the modified cost model and suggested that this can be done by adding some restrictions over the sequences of requests. Our analysis technique allows us to resolve this problem as well. As noted above the model used for this proof builds upon the work of Albers et al. and Angelopoulos et al. for paging with locality of reference. As such, the results in this paper also provide evidence of the applicability of these models to problems other than paging.

2 Bijective Analysis

In this section, we first provide the formal definitions of Bijective Analysis and Average Analysis and then we show equivalence of all list update algorithms under the modified model according to these measures. We choose to employ these measures since they reflect certain desirable characteristics for comparing online algorithms: they allow for direct comparison of two algorithms without appealing to the concept of the "optimal" cost (see [6] for a more detailed discussion), and they do not evaluate the performance of the algorithm on a single "worst-case" request, but instead use the cost that the algorithm incurs on each and all request sequences. These two measures have already been successfully applied in the context of the paging problem [6].

For the sake of simplicity, in this paper we only consider the *static* list update problem. This means that we only have accesses to list items and do not have any insert or delete operations. In particular, we have a set $S = \{a_1, a_2, \dots, a_l\}$ of l items initially organized as a list $\mathcal{L}_0 = (a_1, a_2, \dots, a_l)$. The results in this paper can easily be extended to the dynamic version of the problem. For an on-line algorithm $\mathcal{A}$ and a sequence σ, we denote by $\mathcal{A}(\sigma)$ the cost that $\mathcal{A}$ incurs to serve σ. We denote by $\mathcal{I}_n$ the set of all request sequences of length n, and by $\mathcal{I}_{k+1}(\sigma)$ where $|\sigma| = k$, the set of sequences in $\mathcal{I}_{k+1}$ which have σ as their prefix.

Informally, Bijective Analysis aims to pair input sequences for two algorithms $\mathcal{A}$ and $\mathcal{B}$ using a bijection in such a way that the cost of $\mathcal{A}$ on input σ is no more than the cost of $\mathcal{B}$ on the image of σ, for all request sequences σ of the same length. In this case, intuitively, $\mathcal{A}$ is no worse than $\mathcal{B}$. On the other hand, Average Analysis compares the average cost of the two algorithms over all request sequences of the same length.

Definition 1. *[6] We say that an on-line algorithm $\mathcal{A}$ is* no worse *than an on-line algorithm $\mathcal{B}$ according to Bijective Analysis if there exists an integer $n_0 \geq 1$ so that for each $n \geq n_0$, there is a bijection $b : \mathcal{I}_n \leftrightarrow \mathcal{I}_n$ satisfying $\mathcal{A}(\sigma) \leq \mathcal{B}(b(\sigma))$ for each $\sigma \in \mathcal{I}_n$. We denote this by $\mathcal{A} \preceq_b \mathcal{B}$. Otherwise we denote the situation by $\mathcal{A} \not\preceq_b \mathcal{B}$. Similarly, we say that $\mathcal{A}$ and $\mathcal{B}$ are* the same *according to Bijective Analysis if $\mathcal{A} \preceq_b \mathcal{B}$ and $B \preceq_b A$. This is denoted by $\mathcal{A} \equiv_b \mathcal{B}$. Lastly we say $\mathcal{A}$ is* better *than $\mathcal{B}$ according to Bijective Analysis if $\mathcal{A} \preceq_b \mathcal{B}$ and $\mathcal{B} \not\preceq_b \mathcal{A}$. We denote this by $\mathcal{A} \prec_b \mathcal{B}$.*

Definition 2. *[6] We say that an on-line algorithm $\mathcal{A}$ is* no worse *than an on-line algorithm $\mathcal{B}$ according to Average Analysis if there exists an integer $n_0 \geq 1$ so that for each $n \geq n_0$, $\sum_{I \in \mathcal{I}_n} \mathcal{A}(I) \leq \sum_{I \in \mathcal{I}_n} \mathcal{B}(I)$. We denote this by $\mathcal{A} \preceq_a \mathcal{B}$. Otherwise we denote the situation by $\mathcal{A} \not\preceq_a \mathcal{B}$. $\mathcal{A} \equiv_a \mathcal{B}$, and $\mathcal{A} \prec_a \mathcal{B}$ are defined as for Bijective Analysis.*

Observation 1. *[6] If $\mathcal{A} \not\preceq_a \mathcal{B}$, then $\mathcal{A} \not\preceq_b \mathcal{B}$. In addition, if $\mathcal{A} \preceq_b \mathcal{B}$, then $\mathcal{A} \preceq_a \mathcal{B}$ and similar statements hold for $\mathcal{A} \equiv_b \mathcal{B}$ and $\mathcal{A} \prec_b \mathcal{B}$.*

Suitability of the Measure. Note that rather than considering a worst case sequence, these measures take into account all sequences of the same length. To be precise, bijective analysis compares the performance of two algorithms over pairs of different inputs of the same size. A natural question is if this is a reasonable comparison. To answer this, it is necessary to briefly review standard worst case analysis. Worst case analysis of an algorithm A considers the running time of A over all possible inputs of a given size n and selects as representative for this set the maximum or worst case time observed in that class. Let $I_{A,n}$ denote this worst case input of size n for A. Now when the worst case performance of A is compared to that of algorithm B, worst case analysis compares the timing of A on $I_{A,n}$ with that of B on $I_{B,n}$. Observe that in general $I_{A,n} \neq I_{B,n}$ and hence bijective analysis is no different than worst case analysis in terms of pairing different inputs of the same size. The main difference is that bijective analysis studies the performance of both algorithms across the entire spectrum on inputs of size n as opposed to the worst case. This is similar to average case analysis which also measures performance across all inputs of a given size.

The following theorem proves that under the modified cost model all list update algorithms are equivalent. This result parallels the equivalence of all lazy paging algorithms under Bijective Analysis as shown in [6].

Theorem 1. *Let $\mathcal{A}$ and $\mathcal{B}$ be two arbitrary on-line list update algorithms. Under the modified cost model, we have $\mathcal{A} \equiv_b \mathcal{B}$.*

Proof. We prove that for every $n \geq 1$ there is a bijection $b^n : \mathcal{I}_n \leftrightarrow \mathcal{I}_n$ so that $\mathcal{A}(\sigma) \leq \mathcal{B}(b^n(\sigma))$ for each $\sigma \in I_n$. We show this by induction on n, the length of sequences. Since $\mathcal{A}$ and $\mathcal{B}$ start with the same initial list, they incur the same cost on each sequence of length 1. Therefore the statement trivially holds for $n = 1$. Assume that it is true for $n = k$. Thus there is a bijection $b^k : \mathcal{I}_k \leftrightarrow \mathcal{I}_k$ so that $\mathcal{A}(\sigma) \leq \mathcal{B}(b^k(\sigma))$ for each $\sigma \in I_k$. Let σ be an arbitrary sequence of length k and $\sigma' = b^k(\sigma)$. We map $\mathcal{I}_{k+1}(\sigma)$ to $\mathcal{I}_{k+1}(\sigma')$ as follows. Let $\mathcal{L}(\mathcal{A}, \sigma) = (a_1, a_2, \ldots, a_l)$ be the list maintained by $\mathcal{A}$ after serving σ and $\mathcal{L}(\mathcal{B}, \sigma') = (b_1, b_2, \ldots, b_l)$ be the list maintained by $\mathcal{B}$ after serving σ'. Consider an arbitrary sequence $\sigma_1 \in \mathcal{I}_{k+1}(\sigma)$ and let its last request be to item a_i. We map σ_1 to the sequence $\sigma_2 \in \mathcal{I}_{k+1}(\sigma')$ that has b_i as its last request. Since $\mathcal{A}(\sigma) \leq \mathcal{B}(\sigma')$ and $\mathcal{A}$'s cost on the last request of σ_1 is the same as $\mathcal{B}$'s cost on the last request of σ_2, we have $\mathcal{A}(\sigma_1) \leq \mathcal{B}(\sigma_2)$. Therefore we get the desired mapping from $\mathcal{I}_{k+1}(\sigma)$ to $\mathcal{I}_{k+1}(\sigma')$. We obtain a bijection $b^{k+1} : \mathcal{I}_{k+1} \leftrightarrow \mathcal{I}_{k+1}$ by considering the above mapping for each sequence $\sigma \in \mathcal{I}_k$. Thus our induction statement is true and we have $\mathcal{A} \preceq_b \mathcal{B}$. Using a similar argument, we can show $\mathcal{B} \preceq_b \mathcal{A}$. Therefore we have $\mathcal{A} \equiv_b \mathcal{B}$.

We will call a list update algorithm *economical* if it does not use paid exchanges. Since an economical list update algorithm does not incur any cost for reorganizing the list we can prove the following statement using an argument analogous to the proof of Theorem 1.

Corollary 1. *All economical on-line list update algorithms are equivalent according to Bijective Analysis under the standard cost model.*

These results show that so long as we consider all possible request sequences, all on-line list update algorithms are equivalent in a strong sense. However, as stated earlier, in practice request sequences tend to exhibit locality of reference. Therefore, the algorithm can focus on input sequences with this property. In the next section we show that we can use such an assumption to prove the superiority of MTF.

3 List Update with Locality of Reference

As stated in the Introduction, several models have been proposed for paging with locality of reference [23,2,8]. In this paper, we consider the model of Albers et al. [2], in which a request sequence has high locality of reference if the number of distinct requests in a window of size n is small. In Section 4 we will present experimental evidence which supports the validity of this model for the list update problem. Consider a function that represents the maximum number of distinct items in a window of size n, on a given request sequence. For the paging problem, extensive experiments with real data show that this function can be bounded by a concave function for most practical request sequences [2]. Let f be an increasing concave function. We say that a request sequence is *consistent* with f if the number of distinct requests in any window of size n is at most $f(n)$, for any $n \in \mathcal{N}$. In order to model locality, we restrict the request sequences to those consistent with a concave function f. Let $\mathcal{I}^f$ denote the set of such sequences. We can easily modify the definitions of Bijective Analysis and Average Analysis (Definition 1 and Definition 2) by replacing $\mathcal{I}$ with $\mathcal{I}^f$ throughout. We denote the corresponding relations by $\mathcal{A} \preceq_b^f \mathcal{B}$, $\mathcal{A} \preceq_a^f \mathcal{B}$, etc. Observe that the performance of list update algorithms are now evaluated within the subset of request sequences of a given length that are consistent with f, which we denote as $\mathcal{I}_n^f$, where n is the length of the requence sequences.

Note that the inductive argument used to prove that all on-line list update algorithms are equivalent according to Bijective Analysis (Theorem 1) does not necessarily carry through under concave analysis because the bijection of the proof may map a sequence in $\mathcal{I}^f$ to one not in $\mathcal{I}^f$.

Definition 3. *Let $\mathcal{A}$ and $\mathcal{B}$ be list update algorithms, and f be a concave function. $\mathcal{A}$ is said to* (m, f)-dominate *$\mathcal{B}$ for some integer m, if we have*

$$\sum_{\sigma \in \mathcal{I}_m^f} \mathcal{A}(\sigma) \leq \sum_{\sigma \in \mathcal{I}_m^f} \mathcal{B}(\sigma).$$

$\mathcal{A}$ is said to dominate *$\mathcal{B}$ if there exists an integer $m_0 \geq 1$ so that for each $m \geq m_0$ and every concave function f, $\mathcal{A}$ (m, f)-dominates $\mathcal{B}$.*

Observation 2. *$\mathcal{A} \preceq_a^f \mathcal{B}$ if and only if there exists an integer $m_0 \geq 1$ so that $\mathcal{A}$ (m, f)-dominates $\mathcal{B}$ for each $m \geq m_0$.*

Lemma 1. *For every on-line list update algorithm $\mathcal{A}$, MTF dominates $\mathcal{A}$.*

Proof. Let f be an arbitrary concave function and m be a positive integer. For any $1 \leq i \leq m$, let $\mathcal{F}_{i,m}(\mathcal{A})$ be the total cost $\mathcal{A}$ incurs on the i^{th} request of all sequences in $\mathcal{I}_m^f$. We will first show that $\mathcal{F}_{i,m}(MTF) \leq \mathcal{F}_{i,m}(\mathcal{A})$ for any $1 \leq i \leq m$. For $i = 1$, we have $\mathcal{F}_{1,m}(MTF) = \mathcal{F}_{1,m}(\mathcal{A})$, as all algorithms start with the same list. Now suppose that $i > 1$. Let σ be an arbitrary sequence of length $i-1$, T_σ denote the set of all sequences in $\mathcal{I}_m^f$ that have σ as their prefix, and $\mathcal{F}_{i,m}(\mathcal{A}\,|\,\sigma)$ be the total cost $\mathcal{A}$ incurs on the i^{th} request of all sequences in T_σ. Denote by $\mathcal{L}(MTF, \sigma) = (a_1, a_2, \ldots, a_l)$ and $\mathcal{L}(\mathcal{A}, \sigma) = (b_1, b_2, \ldots, b_l)$ the lists maintained by MTF and $\mathcal{A}$ after serving σ, respectively. Suppose that c_j (resp., d_j) sequences in T_σ have a_j (resp., b_j) as their i^{th} request, for $1 \leq j \leq l$. Note that $\sum_{1\leq j\leq l} c_j = \sum_{1\leq j\leq l} d_j = |T_\sigma|$ and $(d_1, d_2, \ldots, d_l)$ is a permutation of $(c_1, c_2, \ldots, c_l)$.

We first show that $c_{j+1} \leq c_j$ for $1 \leq j < l$. Let C_j and C_{j+1} denote the set of sequences in T_σ that have a_j and a_{j+1} as their i^{th} request. We provide a one-to-one mapping from C_{j+1} to C_j which proves that $|C_{j+1}| \leq |C_j|$. We map every sequence τ in C_{j+1} to a sequence τ' in C_j by replacing every a_j with a_{j+1} and every a_{j+1} by a_j, starting from position i. Since a_j occurs before a_{j+1} in MTF's list after serving σ, we know that the last request to a_j occurs after the last request to a_{j+1} in σ. Therefore if τ is consistent with f, so is τ'. Thus every sequence in C_{j+1} is mapped to a unique sequence in C_j and we have $c_{j+1} = |C_{j+1}| \leq |C_j| = c_j$.

Therefore $(c_1, c_2, \ldots, c_l)$ is a permutation of $(d_1, d_2, \ldots, d_l)$ in non-increasing order, and thus $\mathcal{F}_{i,m}(MTF\,|\,\sigma) = \sum_{1\leq j\leq l} j \times c_j \leq \sum_{1\leq j\leq l} j \times d_j = \mathcal{F}_{i,m}(\mathcal{A}\,|\,\sigma)$. Now since

$$\mathcal{F}_{i,m}(MTF) = \sum_{\sigma\in\mathcal{I}_{i-1}} \mathcal{F}_{i,m}(MTF\,|\,\sigma) \text{ and } \mathcal{F}_{i,m}(\mathcal{A}) = \sum_{\sigma\in\mathcal{I}_{i-1}} \mathcal{F}_{i,m}(\mathcal{A}\,|\,\sigma),$$

we get $\mathcal{F}_{i,m}(MTF) \leq F_{i,m}(\mathcal{A})$. We have

$$\sum_{\sigma\in\mathcal{I}_m^f} MTF(\sigma) = \sum_{1\leq i\leq m} \mathcal{F}_{i,m}(MTF) \leq \sum_{1\leq i\leq m} \mathcal{F}_{i,m}(\mathcal{A}) = \sum_{\sigma\in\mathcal{I}_m^f} \mathcal{A}(\sigma).$$

Thus MTF (m, f)-dominates $\mathcal{A}$ for every concave function f, and every integer $m \geq 1$. Hence MTF dominates $\mathcal{A}$.

Corollary 2. *For any concave function f and any on-line list update algorithm $\mathcal{A}$,*

$$MTF \preceq_a^f \mathcal{A}.$$

Therefore MTF is an optimal algorithm according to Average Analysis, when we classify the input sequences by locality of reference. A natural question is whether MTF is a unique optimum or not, i.e., is there an on-line list update algorithm $\mathcal{A}$ that dominates MTF?

Lemma 2. *No on-line list update algorithm (other than MTF itself) dominates MTF.*

Proof. Assume by way of contradiction that an on-line list update algorithm $\mathcal{A}$ dominates MTF and that $\mathcal{A}$ is different from MTF. According to the definition, there exists an integer $m_0 \geq 1$ so that for each $m \geq m_0$ and every concave function f, $\mathcal{A}$ (m,f)-dominates MTF, i.e.,

$$\sum_{\sigma \in \mathcal{I}_m^f} \mathcal{A}(\sigma) \leq \sum_{\sigma \in \mathcal{I}_m^f} MTF(\sigma).$$

Following the proof of Lemma 1, this holds only if $\mathcal{F}_{i,m}(\mathcal{A} \,|\, \sigma) = \mathcal{F}_{i,m}(MTF \,|\, \sigma)$ for every $m \geq m_0$, $2 \leq i \leq m$, and every sequence σ of length $i-1$. Let $\sigma \in \mathcal{I}_{i-1}^f$ be a sequence so that $\mathcal{L}(\mathcal{A}, \sigma)$ is different from $\mathcal{L}(MTF, \sigma)$, k be the largest index so that $y = a_k \neq b_k = x$ (for a_k and b_k defined as in Lemma 1, and p be the smallest index so that $\sigma[p..i-1]$ contains at most $k-1$ distinct items. Select the concave function f so that $\lfloor f(i-p) \rfloor = \lfloor f(i-p+1) \rfloor = k-1$. Since $y \in \sigma[p..i-1]$ and $x \notin \sigma[p..i-1]$, we have $c_k = 0 < d_k$ (the sequence of length $m > i$ obtained by repeating y in any position starting from i^{th} position is consistent with f). Therefore

$$\mathcal{F}_{i,m}(MTF \,|\, \sigma) = \sum_{1 \leq j \leq l} j \times c_j < \sum_{1 \leq j \leq l} j \times d_j = \mathcal{F}_{i,m}(\mathcal{A} \,|\, \sigma),$$

which is a contradiction.

Theorem 2. *Let $\mathcal{A}$ be an on-line list update algorithm other than MTF. Then $MTF \preceq_b^f \mathcal{A}$ and there exists at least one concave function f so that*

$$\mathcal{A} \not\preceq_a^f MTF, \qquad \textit{which implies} \qquad \mathcal{A} \not\preceq_b^f MTF.$$

We can prove separation with respect to Bijective Analysis between MTF and specific algorithms, e.g., Transpose, for a much larger family of concave functions.

Theorem 3. *For all concave functions f such that $f(l) < l$ (l is the size of list),*

$$Transpose \not\preceq_b^f MTF.$$

Proof. Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_l)$ be the initial list. Assume by way of contradiction that $Transpose \preceq_b^f MTF$. Therefore there is an integer $n_0 \geq 1$ so that for each $n \geq n_0$, there is a bijection $b : \mathcal{I}_n^f \leftrightarrow \mathcal{I}_n^f$ satisfying $Transpose(\sigma) \leq MTF(b(\sigma))$ for each $\sigma \in \mathcal{I}_n^f$. Now consider a sequence σ of length $m \geq n_0$ obtained by considering the prefix of the infinite sequence $a_l a_{l-1} a_l a_{l-1} \ldots$. Transpose incurs a cost of l on each request and we have $Transpose(\sigma) = m \times l$. Note that σ is consistent with f, because it has two distinct items.[1] Thus $\sigma \in \mathcal{I}_m^f$ and from the assumption there should exist some sequence $\sigma' \in \mathcal{I}_m^f$ so that $m \times l = Transpose(\sigma) \leq MTF(\sigma')$. Therefore MTF should incur a

[1] We can assume that $f(2) = 2$ because otherwise we are restricted to sequences that contain only one item.

cost of l on each request of σ'. Hence σ' should be a prefix of the sequence $a_l a_{l-1} a_{l-2} \ldots a_1 a_l a_{l-1} a_{l-2} \ldots a_1 \ldots$. Now any window of size l in σ' has l distinct items. Since we started with $f(l) < l$, σ' is not consistent with f and this contradicts the assumption that $\sigma' \in \mathcal{I}_m^f$.

4 Experimental Results and Analysis

In this section we test the validity of the locality of reference assumption as described in Section 3 against experimental data. For our experiments, we considered the fourteen files of the Calgary Compression Corpus [24] which are frequently used as a standard benchmark for file compression. Recall that list update algorithms can be used in a very direct way in file compression. For each file, we computed the maximum number of characters in windows of all possible sizes, up to the size of the whole file. Figures 1 and 2 show the resulting graphs. Note that since we observed that the maximum number of distinct items does not change much as we increase the size of window to values more than 3500, we only show the results for windows of size up to 3500.

As can be seen from these graphs, the curves have an overall concave shape. We should note that for some of the input files, the function we obtained is not concave for some intervals. However, this is not a major concern, since we

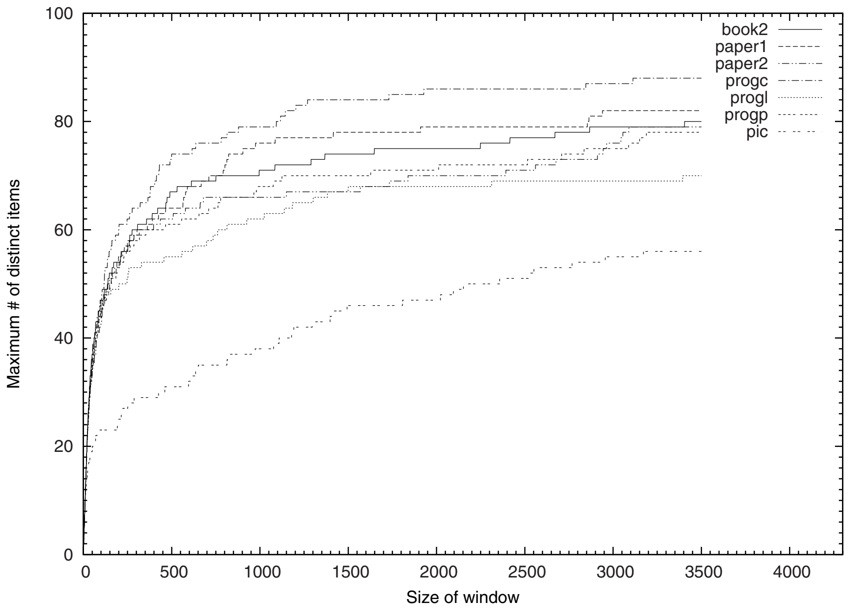


Fig. 1. Maximum number of distinct characters in windows of size up to 3500 for the files in Calgary Compression Corpus

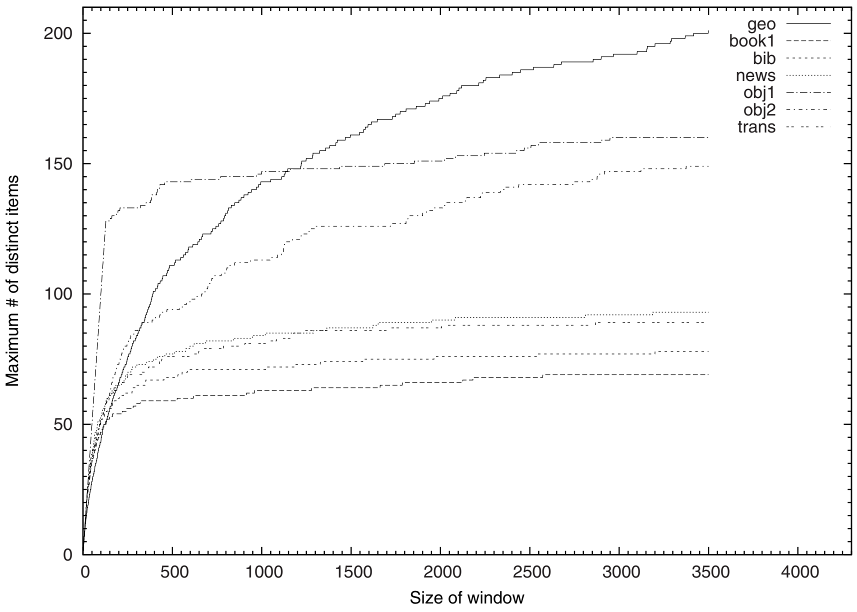


Fig. 2. Maximum number of distinct characters in windows of size up to 3500 for the files in Calgary Compression Corpus

can bound said function by any concave function f which is such that $f(i)$ is an upper bound on the maximum number of distinct items in windows of size i. For instance, we can take the upper convex hull of the data points. In fact, Albers et al. [2] observed that similar non-concavity (mostly localized within small intervals) was present in their experimental results concerning locality of reference in typical request sequences for the paging problem. Albers et al. put forth this argument to justify the fact that local small deviations from concavity do not impose a serious problem.

Albers and Mitzenmacher [3] compared the efficiency of MTF and Timestamp (TS) algorithms for compressing the files of the Calgary Compression Corpus. TS is a list update algorithm that is 2-competitive [1]. After accessing an item a, TS inserts a in front of the first item b that appears before a in the list and was requested at most once since the last request for a. If there is no such item b, or if this is the first access to a, TS does not reorganize the list. They compared MTF and TS in two settings: with or without Burrows-Wheeler transform (BWT). Informally, BWT transforms a string to one of its permutations that has more locality of reference, which is hence more readily compressible [12,17]. Their results show that although TS outperforms MTF on compression without BWT, MTF usually has better performance when we use BWT. This is consistent with our results as BWT is a transform designed with the goal of increasing the locality of reference in the representation of the string.

5 Conclusions

In this paper we addressed certain open questions concerning the well-studied list update problem. We first considered the issue of modeling locality of reference for typical request sequences for this problem. We provided experimental evidence which suggests that the concave-function model of Albers et al., originally devised for the context of paging algorithms, can satisfactorily model locality of reference within the domain of list update. We then combined this model with two recently proposed measures for comparing online algorithms, namely Bijective Analysis and Average Analysis. Our choice was based on the fact that these measures allow direct comparison of two online algorithms, by considering their relative performance on all requests sequences of the same length, rather than on some specific pathological sequences. These measures have been previously applied with success in separating several paging algorithms, a situation which has long been known but cannot be resolved by resorting solely to competitive analysis.

Using the above framework, we showed that while all list update algorithms are equivalent in the modified-cost model, when locality of reference is considered, MTF emerges as the sole optimum online algorithm for the problem. This resolves an open problem posed by Martínez and Roura. We believe that our techniques might well be applicable to other problems in which competitive analysis has failed to yield satisfactory results such as the online bin packing, but this remains the subject of future work.

The model proposed is, to our knowledge, the first that both incorporates locality of reference and achieves full separation of MTF. However locality of reference is a phenomenon which has only recently begun to be thoroughly understood. Thus we fully expect that future further refinements of the model by researchers in the field will reflect even more faithfully locality of reference as it is observed in practice.

References

1. Albers, S.: Improved randomized on-line algorithms for the list update problem. SIAM Journal on Computing 27(3), 682–693 (1998)
2. Albers, S., Favrholdt, L.M., Giel, O.: On paging with locality of reference. Journal of Computer and System Sciences 70(2), 145–175 (2005)
3. Albers, S., Mitzenmacher, M.: Average case analyses of list update algorithms, with applications to data compression. Algorithmica 21(3), 312–329 (1998)
4. Albers, S., von Stengel, B., Werchner, R.: A combined BIT and TIMESTAMP algorithm for the list update problem. Information Processing Letters 56, 135–139 (1995)
5. Albers, S., Westbrook, J.: Self-organizing data structures. In: Fiat, A. (ed.) Dagstuhl Seminar 1996. LNCS, vol. 1442, pp. 13–51. Springer, Heidelberg (1998)
6. Angelopoulos, S., Dorrigiv, R., López-Ortiz, A.: On the separation and equivalence of paging strategies. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), pp. 229–237 (2007)

7. Bachrach, R., El-Yaniv, R.: Online list accessing algorithms and their applications: Recent empirical evidence. In: Proc. 8th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA 1997), pp. 53–62 (1997)
8. Becchetti, L.: Modeling locality: A probabilistic analysis of LRU and FWF. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 98–109. Springer, Heidelberg (2004)
9. Ben-David, S., Borodin, A.: A new measure for the study of on-line algorithms. Algorithmica 11, 73–91 (1994)
10. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
11. Boyar, J., Favrholdt, L.M.: The relative worst order ratio for on-line algorithms. In: Proceedings of the 5th Italian Conference on Algorithms and Complexity (2003)
12. Burrows, M., Wheeler, D.J.: A block-sorting lossless data compression algorithm. Technical Report 124, DEC SRC (1994)
13. Dorrigiv, R., López-Ortiz, A.: A survey of performance measures for on-line algorithms. SIGACT News (ACM Special Interest Group on Automata and Computability Theory) 36(3), 67–81 (2005)
14. El-Yaniv, R.: There are infinitely many competitive-optimal online list accessing algorithms (manuscript, 1996)
15. Hester, J.H., Hirschberg, D.S.: Self-organizing linear search. ACM Computing Surveys 17(3), 295 (1985)
16. Irani, S.: Two results on the list update problem. Information Processing Letters 38, 301–306 (1991)
17. Kaplan, H., Landau, S., Verbin, E.: A simpler analysis of burrows-wheeler based compression. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 282–293. Springer, Heidelberg (2006)
18. Martínez, C., Roura, S.: On the competitiveness of the move-to-front rule. Theoretical Computer Science 242(1–2), 313–325 (2000)
19. Munro, J.I.: On the competitiveness of linear search. In: Paterson, M. (ed.) ESA 2000. LNCS, vol. 1879, pp. 338–345. Springer, Heidelberg (2000)
20. Reingold, N., Westbrook, J., Sleator, D.: Randomized competitive algorithms for the list update problem. Algorithmica 11, 15–32 (1994)
21. Schulz, F.: Two new families of list update algorithms. In: Chwa, K.-Y., H. Ibarra, O. (eds.) ISAAC 1998. LNCS, vol. 1533, pp. 99–108. Springer, Heidelberg (1998)
22. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM 28, 202–208 (1985)
23. Torng, E.: A unified analysis of paging and caching. Algorithmica 20(2), 175–200 (1998)
24. I. H. Witten and T. Bell. The Calgary/Canterbury text compression corpus. Anonymous ftp from: ftp.cpsc.ucalgary.ca/pub/text.compression/corpus/text.compression.corpus.tar.Z

Approximating Steiner Networks with Node Weights

Zeev Nutov

The Open University of Israel, Raanana, Israel
nutov@openu.ac.il

Abstract. The (undirected) Steiner Network problem is: given a graph $G = (V, E)$ with edge/node weights and edge-connectivity requirements $\{r(u, v) : u, v \in U \subseteq V\}$, find a minimum weight subgraph H of G containing U so that the uv-edge-connectivity in H is at least $r(u, v)$ for all $u, v \in U$. The seminal paper of Jain [12], and numerous papers preceding it, considered the Edge-Weighted Steiner Network problem, with weights on the edges only, and developed novel tools for approximating minimum weight edge-covers of several types of set functions and families. However, for the Node-Weighted Steiner Network (NWSN) problem, nontrivial approximation algorithms were known only for $0, 1$ requirements.

We make an attempt to change this situation, by giving the first nontrivial approximation algorithm for NWSN with arbitrary requirements. Our approximation ratio for NWSN is $r_{\max} \cdot O(\ln |U|)$, where $r_{\max} = \max_{u,v \in U} r(u, v)$. This generalizes the result of Klein and Ravi [14] for the case $r_{\max} = 1$. We also give an $O(\ln |U|)$-approximation algorithm for the node-connectivity variant of NWSN (when the paths are required to be internally-disjoint) for the case $r_{\max} = 2$. Our results are based on a much more general approximation algorithm for the problem of finding a minimum node-weighted edge-cover of an *uncrossable set-family*. We also give the first evidence that a polylogarithmic approximation ratio for NWSN might not exist even for $|U| = 2$ and unit weights.

1 Introduction

1.1 Motivation, Problem Definition, and Previous Work

Network design problems require finding a minimum weight (sub-)network that satisfies prescribed properties, often connectivity requirements. Classic examples with $0, 1$ connectivity requirements are: Shortest Path, Minimum Spanning Tree, Minimum Steiner Tree/Forest, and others. Examples of problems with high connectivity requirements are: Min-Cost k-Flow, k-Edge/Node-Connected Spanning Subgraph, Steiner Network, and others.

Two main types of weights are considered in the literature: the edge weights and the node weights. We consider the latter, which is usually more general than the former. For most undirected network design problems, a simple reduction transforms edge weights to node weights, but the inverse is usually not true. The study of network design problems with node weights is well motivated and established from both theoretical as well as practical considerations, c.f., [14,10,17,2,16]. For example, in telecommunication networks, expensive equipment such as routers/switches/transmitters is located at the nodes of the network, and thus it is natural to model these problems by assigning weights to the nodes and/or to the edges, rather than to the edges only.

In directed graphs, it is often possible to reduce node weights case to the edge weights case via an approximation ratio preserving reduction. However, this is usually not so for undirected graphs, and an attempt to transform an undirected problem into a directed one typically results in a problem which is significantly harder to approximate; e.g., in undirected graphs, for Steiner Forest a 2-approximation is known for edge-costs [1], an $O(\log n)$-approximation is known for node-costs and this is tight [14], while the *directed* variant does not admit a polylogarithmic ratio unless NP⊆Quasi(P) [3].

Let $\lambda_H(u,v)$ denote the maximum number of edge-disjoint uv-paths in a graph H. We consider the following fundamental problem on undirected graphs:

Node-Weighted Steiner Network (NWSN)
Instance: A graph $G=(V,E)$, node weights $\{w(v) : v \in V\}$, and edge-connectivity requirements $\{r(u,v) : u,v \in U \subseteq V\}$.
Objective: Find a minimum weight subgraph H of G containing U so that

$$\lambda_H(u,v) \geq r(u,v) \quad \forall u,v \in U \ . \tag{1}$$

Let $r_{\max} = \max_{u,v \in U} r(u,v)$. The Edge-Weighted Steiner Network problem was studied extensively, starting from the first 2-approximation algorithm of Agrawal, Klein, and Ravi [1] for $r_{\max}=1$ (see Goemans and Williamson [8] for more general algorithm and simpler proof) continuing with $2r_{\max}$-approximation of Williamson et. al [20] and $2\ln r_{\max}$-approximation of Goemans et. al [7], and ending with the seminal 2-approximation of Jain [12]. See surveys in [9,13,15] on approximation algorithms for various connectivity problems.

However, for the node-weighted version NWSN, nontrivial approximation algorithms were known only for $r_{\max}=1$. The first approximation algorithm for NWSN with $r_{\max}=1$ due to Klein and Ravi [14] appeared in 1995, at the same time as the 2-approximation of Klein, Agrawal, and Ravi [1], for the edge-weighted case with $r_{\max}=1$. The Klein-Ravi [14] algorithm uses a greedy approach. Based on "spider decomposition" of trees, they proved that iteratively adding spiders (subtrees with at most one node of degree ≥ 3) that minimize the ratio of the weight of the spider over the number of "minimal deficient sets" it connects minus 1, is a $2H(|U|)$-approximation algorithm, where $H(n)=\sum_{i=1}^{n} 1/i = O(\ln n)$ is the nth Harmonic number. The approximation ratio was improved by Guha and Khuller [10] to $(1.35+\varepsilon)H(|U|)$ using a slight generalization of spiders. These ratios are nearly tight, as the case $r_{\max}=1$ of NWSN generalizes the Set-Cover problem, and thus has an $(1-\varepsilon)\ln|U|$-approximation threshold [4]. However, unlike the case of edge weights, for node weights almost no progress has been made since the Klein-Ravi paper [14]: no approximation algorithm was known for NWSN with $r_{\max} > 1$, not even for the case $r_{\max} = 2$.

1.2 Our Results

We give the first non-trivial algorithm for NWSN with arbitrary requirements.

Theorem 1. NWSN *admits a* $3r_{\max} \cdot H(|U|)$*-approximation algorithm.*

The approximation ratio in Theorem 1 is tight (up to a constant factor) if $r_{\max}$ is "small" (usually, $r_{\max} \leq 3$ in practical networks), but may seem weak if $r_{\max}$ is large. We give the first evidence that a polylogarithmic approximation algorithm for NWSN may not exist even for very simple instances. Let the Node-Weighted k-Flow (NWk-F) problem be the restriction of NWSN to instances with $U = \{s,t\}$ and $r(s,t) = k$. We show a reduction from the following extensively studied problem to unit weight NWk-F. For an edge set E on V and $X \subseteq V$ let $E(X)$ denote the set of edges in E with both endpoints in X.

Densest ℓ-Subgraph (Dℓ-S)
Instance: A graph $G = (V,E)$ and an integer ℓ.
Objective: Find $X \subseteq V$ with $|X| \leq \ell$ and $|E(X)|$ maximum.

The best known approximation ratio for Dℓ-S due to Feige, Kortsarz, and Peleg [5] is $|V|^{-1/3+\delta}$, where $\delta \approx 1/60$. This is so even for the case of bipartite graphs, which is up to a constant factor is as hard to approximate as the general case. In spite of numerous attempts to improve it, this ratio holds for more than 10 years. We prove:

Theorem 2. *Suppose that* NWk-F *admits a* ρ*-approximation algorithm. Then:*
– *The* Hitting-Set *problem admits a* ρ*-approximation algorithm.*
– Dℓ-S *on bipartite graphs admits a* $1/(2\rho^2)$*-approximation algorithm.*

Remark 1. It was shown in [11] that *directed* NWk-F cannot be approximated within $O(2^{\log^{1-\varepsilon} n})$ for any fixed $\varepsilon > 0$, unless NP $\subseteq$ Quasi(P); for edge weights, this case is in P. On the other hand, the "augmentation" version of NWk-F that seeks to find a minimum node-weight augmenting edge set to increase the st-edge-connectivity by 1 is reducible to the shortest path problem, and thus is solvable in polynomial time, see Section 4. This implies a k-approximation algorithm for NWk-F. Also, NWk-F with node-disjoint paths is easily reducible to the Min-Cost k-Flow problem, and thus is solvable in polynomial time.

We also consider the node-connectivity version of NWSN, when the paths are required to be internally node-disjoint. The edge-weighted version with internally disjoint paths is usually referred to as the Survivable Network Design Problem (SNDP). SNDP does not admit a polylogarithmic approximation algorithm, unless NP$\subseteq$Quasi(P), even if the input graph G is complete with edge weights in $\{0,1\}$ [18]. However, the $\{0,1,2\}$-SNDP admits a 2-approximation algorithm [6]. We consider the node weighted version NWSNDP of SNDP, and specifically the $\{0,1,2\}$-NWSNDP, and prove:

Theorem 3. $\{0,1,2\}$-NWSNDP *admits an $O(\ln n)$-approximation algorithm.*

Theorems 1 and 3 are just applications of a more general approximation algorithm for finding a minimum "node-weighted" (edge-)cover of an extensively studied type of set-families. We need some definitions to present this result.

Definition 1. *Let $\mathcal{F} \subseteq 2^V$ be a set-family of subsets of a ground-set V.*

- *$\mathcal{F}$ is* uncrossable *if $X \cap Y, X \cup Y \in \mathcal{F}$ or $X - Y, Y - X \in \mathcal{F}$ for any intersecting $X, Y \in \mathcal{F}$.*
- *An edge set I on V* covers *$\mathcal{F}$ (or I is an $\mathcal{F}$-*cover*) if for every $X \in \mathcal{F}$ there is an edge in I with exactly one end-node in X.*

Definition 2. *For an edge set I on V let $V(I) = \bigcup_{uv \in I}\{u,v\}$ denote the set of end-nodes of the edges in I. Given node weights $\{w(v) : v \in V\}$, let $w(I) = w(V(I))$ be the* node-weight *of I.*

We consider the following general problem:

Node-Weighted Set-Family (Edge-)Cover (NWSFC)
Instance: A set-family $\mathcal{F}$, an edge set E on V, and node weights $\{w(v) : v \in V\}$.
Objective: Find a minimum node-weight $\mathcal{F}$-cover $I \subseteq E$.

We give a $3H(|V|)$-approximation algorithm for the problem of finding a minimum node-weight cover of an uncrossable family $\mathcal{F}$, but its polynomial implementation requires that certain queries related to $\mathcal{F}$ can be answered in polynomial time. Given an edge set I on V, the *residual family $\mathcal{F}_I$ of $\mathcal{F}$* (w.r.t. I) consists of all members of $\mathcal{F}$ that are uncovered by the edges of I. It is well known that if $\mathcal{F}$ is uncrossable, so is $\mathcal{F}_I$, for any I, c.f., [12].

Definition 3. *A set $C \in \mathcal{F}$ is an $\mathcal{F}$-*core*, or simply a* core *if $\mathcal{F}$ is understood, if C does not contain two disjoint members of $\mathcal{F}$. An inclusion minimal (maximal) $\mathcal{F}$-core is a* min-$\mathcal{F}$-core *(*max-$\mathcal{F}$-core*). Let $\mathcal{C}(\mathcal{F})$ denote the family of min-$\mathcal{F}$-cores. For $s \in V$ and $C \in \mathcal{C}(\mathcal{F})$ let $\mathcal{F}(s,C)$ be the family of cores containing C and not containing s.*

Clearly, the members of $\mathcal{C}(\mathcal{F})$ are pairwise disjoint if $\mathcal{F}$ is uncrossable. For any edge set I on V, make the following two assumptions:

Assumption 1
The family $\mathcal{C}(\mathcal{F}_I)$ of min-$\mathcal{F}_I$-cores can be found in polynomial time.

Assumption 2
A minimum node-weight $\mathcal{F}_I(s,C)$-cover can be found in polynomial time for any $s \in V$ and $C \in \mathcal{C}(\mathcal{F}_I)$.

Theorem 4. NWSFC *with uncrossable $\mathcal{F}$ admits a $3H(|\mathcal{C}(\mathcal{F})|)$-approximation algorithm under Assumptions 1 and 2.*

Remark 2. Theorem 4 is unlikely to extend to arbitrary weakly supermodular set-functions, due to our hardness result given in Theorem 2.

Section 2 presents our main tool – a novel decomposition of covers of uncrossable families. Theorems 4 and 1 are proved in Sections 3 and 4, respectively. For proofs of Theorems 2 and 3 see the full paper.

2 Decomposition of Covers of Uncrossable Families

2.1 Spider-Covers and Decompositions

The main tool used to prove Theorem 4 is a novel decomposition of covers of uncrossable families into *spider-covers*, generalizing the Klein-Ravi [14] decomposition of a forest into spiders. As uncrossable families and spiders arise in various network design problems, we believe that our decomposition can have further application. However, even extending properly the notions of "spider" and "spider-decomposition" to set-families is already a nontrivial task. We start by briefly describing the decomposition of [14] of a tree (or a forest) into spiders.

Definition 4. *A* spider *is a tree having at least two leaves and at most one node of degree* ≥ 3. *A* spider decomposition $\mathcal{D}$ *of a tree* T *is a collection of node disjoint spiders, each of them is a subtree of* T, *so that every leaf of* T *belongs to exactly one spider of* $\mathcal{D}$.

Lemma 1 ([14]). *Any tree* T *admits a spider decomposition.*

One possible proof of Lemma 1 is as follows. Let U be the set of leaves of T. Consider the set-family $\mathcal{F} = \{X \subseteq V : |X \cap U| = 1\}$. It is easy to see that $\mathcal{F}$ is uncrossable. It can be shown that any inclusion minimal $\mathcal{F}$-cover $F \subseteq T$ is a collection $\mathcal{D}$ of pairwise node-disjoint spiders; consequently, $\mathcal{D}$ is a spider decomposition of T. Note that a spider with leaf set U' and center s covers all $\mathcal{F}$-cores (in fact, all members of $\mathcal{F}$) that contain a node from U' and do not contain s. Motivated by the latter observation, we suggests the following analogue of spiders for covers of set families.

Definition 5. *Let* $\mathcal{F}$ *be a set-family on* V, *let* $s \in V$, *and let* $\mathcal{C} \subseteq \mathcal{C}(\mathcal{F})$. *An edge set* S *on* V *with* $s \in V(S)$ *is an* $\mathcal{F}(s,\mathcal{C})$-cover *with* center *if it is an* $\mathcal{F}(s,C)$-*cover for every* $C \in \mathcal{C}$, *and if* $\mathcal{C} = \{C\}$ *then* s *does not belong to any* $\mathcal{F}$-*core containing* C. *An* $(s,\mathcal{C})$-*cover* S *is a* spider-cover *(or an* $(s,\mathcal{C})$-spider-cover*) if it can be partitioned into* $\mathcal{F}(s,C)$-*covers* $\{S_C : C \in \mathcal{C}\}$ *such that the node sets* $\{V(S_C) - \{s\} : C \in \mathcal{C}\}$ *are pairwise disjoint.*

Equivalently, spider-cover is a union of some (s,C)-covers, $C \in \mathcal{C} \subseteq \mathcal{C}(\mathcal{F})$, so that only s can be a common endnode of two of them. Spider-covers are much more complex objects than [14] spiders, e.g., they are not even connected graphs. Our definition of "spider-cover decomposition" of covers of set-families is:

Definition 6. *A sub-partition* $S_1, \ldots, S_q$ *of an* $\mathcal{F}$-*cover* F *is a* spider-cover decomposition *of* F *if* $V(S_1), \ldots, V(S_q)$ *are pairwise disjoint, and there exists* $s_1, \ldots, s_q \in V$ *and a partition* $\{\mathcal{C}_1, \ldots, \mathcal{C}_q\}$ *of* $\mathcal{C}(\mathcal{F})$ *so that each* S_i *is a spider-cover of* $\mathcal{F}(s_i, \mathcal{C}_i)$.

The main result of this section is the following:

Theorem 5 (The Spider-Cover Decomposition Theorem)
Any cover F *of an uncrossable family* $\mathcal{F}$ *admits a spider-cover decomposition.*

Unlike [14], we cannot use graph properties in the proof of Theorem 5, but can rely only on properties of uncrossable families. In [19], a variant of Theorem 5 was proved for *directed* cover of an *intersecting* family, when $X, Y \in \mathcal{F}, X \cap Y \neq \emptyset$ implies $X \cap Y, X \cup Y \in \mathcal{F}$, and for every $X \in \mathcal{F}$ there should be an edge in F *entering* X. The definition of a spider-covers and decompositions in [19] was slightly different than the one here, e.g., it required disjointness of the tails. For this case, in [19] is proved that there exists a spider-cover decomposition that covers at least $2|\mathcal{C}(\mathcal{F})|/3$ min-cores (in the setting of [19], this bound is the best possible). The proof of this result is easier than that of Theorem 5: in the case of intersecting families, the max-cores are pairwise disjoint, and, because the edges are directed, every edge with head in some max-core can cover only cores contained in this core. Hence any such edge is assigned to a unique max-core. This enables to apply some arguments as in the proof of Lemma 1. However, for *undirected* covers of *uncrossable* families, the situation is more involved; the max-cores may not be disjoint, many edges may cover the same max-core M, and edges contained in M may cover cores contained in other max-cores.

2.2 Cores and Laminar Families

The following property of cores is immediate:

Lemma 2. *Let X, Y be cores of an uncrossable family $\mathcal{F}$. Then:*
- *$X \cap Y, X \cup Y \in \mathcal{F}$ if, and only if, X, Y contain the same min-core.*
- *$X - Y, Y - X \in \mathcal{F}$ if, and only if, X, Y contain distinct min-cores.*

Definition 7. *$X, Y \subseteq V$* cross *if each one of the sets $X \cap Y$, $X - Y$, $Y - X$ is nonempty. A set family $\mathcal{L}$ is* laminar *if its members are pairwise non-crossing, namely, if for any intersecting $X, Y \in \mathcal{L}$ either $X \subset Y$ or $Y \subset X$ holds.*

Definition 8. *Let F be an $\mathcal{F}$-cover and let $e \in F$. A set $W_e \in \mathcal{F}$ is a witness set for e (w.r.t. F) if e is the unique edge in F that covers W_e. A family $\mathcal{W} \subseteq \mathcal{F}$ is a* witness family for F *if every $e \in F$ has a unique witness set $W_e \in \mathcal{W}$.*

Clearly, any inclusion minimal cover F of a set-family $\mathcal{F}$ has a witness family. The following statement was implicitly proved in several papers, c.f., [1,20,12].

Proposition 1. *Let F be an inclusion minimal cover of a uncrossable family $\mathcal{F}$. Then there exists a laminar witness family $\mathcal{L} \subseteq \mathcal{F}$ for F.*

Let $\mathcal{L} \subseteq \mathcal{F}$ be a laminar witness family for a minimal $\mathcal{F}$-cover F. The following two simple reductions enable to simplify the exposition.

Reduction 1: We may assume that every member of $\mathcal{F}$ is an $\mathcal{F}$-core. This is since Definitions 5 and 6 consider covers of $\mathcal{F}$-cores only. Thus we may replace $\mathcal{F}$ by the family of $\mathcal{F}$-cores; the latter is uncrossable if $\mathcal{F}$ is, by Lemma 2. Note that in the Node-Weighted Steiner Tree problem, $\mathcal{F} = \{X \subseteq V : X \cap U, (V - X) \cap U \neq \emptyset\}$; the spider decomposition covers the family $\{X \subseteq V : |X \cap U| = 1\}$ of $\mathcal{F}$-cores, but may not cover the entire family $\mathcal{F}$.

Reduction 2: We may assume that the minimal members of $\mathcal{L}$ are the minimal $\mathcal{F}$-cores. Otherwise, apply the following transformation. For every $C \in \mathcal{C}(\mathcal{F})$ add to V two new nodes v_C, u_C, replace every $X \in \mathcal{F}$ containing C by $X \cup \{v_C, u_C\}$, add $\{v_C\}$ to $\mathcal{F}$, and add the edge $u_C v_C$ to F. The new family is uncrossable, F covers $\mathcal{F}$ if, and only if, $F \cup \{u_C v_C : C \in \mathcal{C}(\mathcal{F})\}$ covers the new family, and $\{v_C\}$ is the witness set for $u_C v_C$. Proving Theorem 5 for the modified family implies Theorem 5 for the original family. This transformation is an analogue of "moving terminals to leaves" used in [14] for the Node-Weighted Steiner Tree.

2.3 Proof of Theorem 5

Definition 9. *For every $C \in \mathcal{C}(\mathcal{F})$ define (see Fig. 1):*

- *L_C is the maximal set in $\mathcal{L}$ containing C (L_C exists and is a core, by Reductions 1,2).*
- *$e_C = s_C v_C$ is the unique edge in F covering L_C, where $v_C \in L_C$.*
- *S_C is the set of edges in F with both endpoints in L_C plus e_C.*

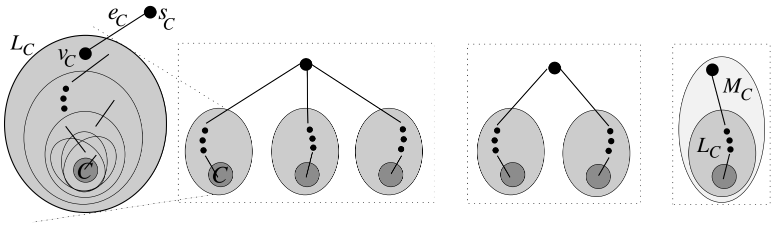


Fig. 1. Illustration to Definitions 9 and 10

Lemma 3

(i) *The sets $\{L_C : C \in \mathcal{C}(\mathcal{F})\}$ are pairwise disjoint.*
(ii) *For every $e = uv \in F$ there is a unique $C \in \mathcal{C}(\mathcal{F})$ so that $\{u, v\} \cap L_C \neq \emptyset$; thus $S_C = \{uv \in F : \{u, v\} \cap L_C \neq \emptyset\}$ and $\{S_C : C \in \mathcal{C}(\mathcal{F})\}$ partition F.*
(iii) *S_C covers all cores contained in L_C for every $C \in \mathcal{C}(\mathcal{F})$.*

Proof. (i) Part (i) follows from the laminarity of $\mathcal{L}$ and the maximality of L_C.
(ii) Let W_e be the witness set for $e = uv \in F$. By the laminarity of $\mathcal{L}$ and the maximality of the sets L_C, $W_e \subseteq L_C$ for some $C \in \mathcal{C}(\mathcal{F})$. Consequently, e has at least one end-node in L_C. Furthermore, e has exactly one end-node in L_C if, and only if, $e = e_C$; in this case, L_C is the witness set for e, and thus e cannot have an end-node in $L_{C'}$ for $C' \in \mathcal{C}(\mathcal{F}) - \{C\}$, since every edge in F has a unique witness set.
(iii) Part (iii) follows from part (ii) and the simple observation that if an edge e covers a set contained in L_C, then it has at least one end-node in L_C.

Corollary 1. *Any partition $\mathcal{C}_1, \ldots, \mathcal{C}_q$ of $\mathcal{C}(\mathcal{F})$ induces a partition $S_1, \ldots, S_q$ of F, where $S_i = \cup\{S_C : C \in \mathcal{C}_i\}$.*

We obtain a spider-cover decomposition of F as a decomposition induced by a certain partition of $\mathcal{C}(\mathcal{F})$. A natural partition of $\mathcal{C}(\mathcal{F})$ (see Fig. 1) is by the stars of $\{e_C : C \in \mathcal{C}(\mathcal{F})\}$. As we show later (see Corollary 2), every star with at least two edges indeed gives a spider-cover. However, this direct approach fails because for a star consisting of a single edge $e_C = s_C v_C$, the edge-set S_C is *not* a spider-cover, if there is a core M_C containing $L_C + s_C$, see Fig. 1. We will handle this difficulty by defining a partition of such "dangerous" cores, showing that every part of size at least 2 is a spider-cover, and joining every singleton part to a "non-dangerous" star. This motivates the following definition:

Definition 10. *A min-core C is* active *if there exists a core containing $L_C + s_C$; an active core is* dangerous *if* $\deg_F(s_C) = 1$. *Let $\mathcal{A}$ denote the set of active and $\mathcal{D}$ the set of dangerous min-cores (note that $\mathcal{D} \subseteq \mathcal{A}$). For $C \in \mathcal{A}$ let M_C be the (unique, by Lemma 2) minimal core among the cores containing $L_C + s_C$.*

Corollary 2

– *If $C \in \mathcal{C}(\mathcal{F}) - \mathcal{D}$ then S_C is an $\mathcal{F}(s, C)$-cover for any $s \in V - L_C$.*
– *If $C \in \mathcal{D}$ then S_C is an $\mathcal{F}(s, C)$-cover for any $s \in M_C - L_C$.*

Proof. We will show that if some $X \in \mathcal{F}(s, C)$ is not covered by S_C, then we must have $s_C \in X$. This immediately gives a contradiction to the case $C \in \mathcal{C}(\mathcal{F}) - \mathcal{D}$. In the case $C \in \mathcal{D}$ we obtain a contradiction to the minimality of M_C: by Lemma 2, $Y = (L_C \cup X) \cap M_C \in \mathcal{F}$, but $L_C + s_C \subseteq Y$ and $Y \subsetneq M_C - s$.

It remains to show that we must have $s_C \in X$. By Lemma 2, $L_C \cap X \in \mathcal{F}$. By Lemma 3 (iii), there is $e \in S_C$ that covers $L_C \cap X$, say $e = uv$ where $v \in L_C \cap X$. We have $u \notin L_C$, as otherwise e covers X. Hence e covers L_C, implying that $e = e_C$ and $u = s_C$. However, $u \in X$, as otherwise e covers X.

Lemma 4. *For every $C \in \mathcal{A}$ the following holds:*

(i) *$M_C \cap L_{C'} = \emptyset$ for any $C' \in \mathcal{C}(\mathcal{F}) - \{C\}$.*
(ii) *M_C is covered by some edge $e_{C'}$, $C' \in \mathcal{C}(\mathcal{F}) - \{C\}$.*
(iii) *If $M_C \cap M_{C'} \neq \emptyset$ for $C' \in \mathcal{A}$ then $s_C, s_{C'} \in M_C \cap M_{C'}$.*

Proof. (i) Assume to the contrary that $M_C \cap L_{C'} \neq \emptyset$ for some $C' \in \mathcal{C}(\mathcal{F}) - \{C\}$. By Lemma 2, $M_C - L_{C'} \in \mathcal{F}$. By Lemma 3(i), $L_C \subseteq M_C - L_{C'}$. If $s_C \in M_C - L_{C'}$, then $L_C + s_C \subseteq M_C - L_{C'}$, contradicting the minimality of M_C. Otherwise, $s_C \in M_C \cap L_{C'}$; but then e_C covers $L_{C'}$, contradicting that $L_{C'}$ is a witness set for $e_{C'}$.

(ii) Part (ii) follows from part (i) and Lemma 3(ii).

(iii) Assume to the contrary that $s_C \in M_C - M_{C'}$; the case $s_{C'} \in M_{C'} - M_C$ is identical. By Lemma 2, $M_C - M_{C'} \in \mathcal{F}$. By part (i), $L_C \subset M_C - M_{C'}$. Hence $L_C + s_C \subseteq M_C - M_{C'}$, contradicting the minimality of M_C.

Corollary 3. *$\mathcal{R} = \{(C, C') \in \mathcal{A} \times \mathcal{A} : M_C \cap M_{C'} \neq \emptyset\}$ is an equivalence relation.*

Proof. Clearly, $\mathcal{R}$ is symmetric and reflexive; transitivity is by Lemma 4(iii).

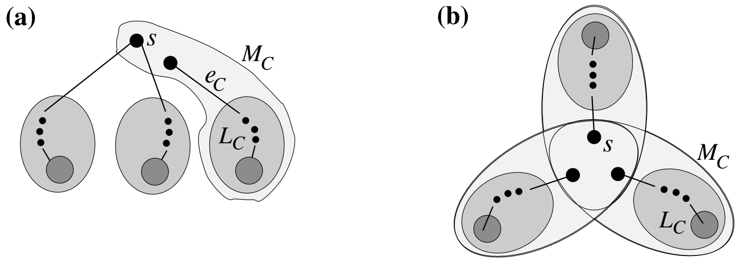


Fig. 2. (a) $\mathcal{C}$ is obtained from a star with center s by joining core $C \in \mathcal{D}$ with $s \in M_C$. (b) $\mathcal{C}$ is an equivalence class of the relation $\mathcal{R} = \{(C, C') \in \mathcal{D} \times \mathcal{D} : M_C \cap M_{C'} \neq \emptyset\}$.

We obtain a spider-cover decomposition of F as a decomposition induced by a partition $\mathcal{C}_1, \ldots, \mathcal{C}_q$ of $\mathcal{C}(\mathcal{F})$, which we define in two steps, as follows.

The first step defines a subpartiton of $\mathcal{F}(\mathcal{C})$ and the corresponding centers as follows. Partition $\mathcal{C}(\mathcal{F}) - \mathcal{D}$ according to stars (equivalence classes of the relation $\{(C, C') : s_C = s_{C'}\}$) of $\{e_C : C \in \mathcal{C}(\mathcal{F}) - \mathcal{D}\}$. Add to this partition the sub-partition of $\mathcal{D}$ into equivalence classes of size at least 2 of the relation $\mathcal{R}$ in Corollary 3. Let $\mathcal{C}_1, \ldots, \mathcal{C}_q$ be a sub-partition of $\mathcal{C}(\mathcal{F})$ obtained; s_i is chosen arbitrarily from $\{s_C : C \in \mathcal{C}_i\}$. Note that if $\mathcal{C}_i$ is a part of $\mathcal{C}(\mathcal{F}) - \mathcal{D}$ then s_i is unique, while if $\mathcal{C}_i$ is a part of $\mathcal{D}$ then there are $|\mathcal{C}_i|$ distinct choices of s_i.

In the second step we join every singleton part $\{C\}$ of $\mathcal{D}$ to some part of $\mathcal{C}(\mathcal{F}) - \mathcal{D}$, as follows (see Fig. 2(a)). By Lemma 4(ii), there exists $C' \in \mathcal{C}(\mathcal{F}) - \{C\}$ so that $e_{C'}$ covers M_C (namely, so that $s_{C'} \in M_C$). Note that $C' \in \mathcal{C}(\mathcal{F}) - \mathcal{D}$, as otherwise C, C' would belong to the same class of $\mathcal{R}$. Hence there is a part of $\mathcal{C}(\mathcal{F}) - \mathcal{D}$ which contains a core C' so that $e_{C'}$ covers M_C; we join $\{C\}$ to one (arbitrarily chosen) such part of $\mathcal{C}(\mathcal{F}) - \mathcal{D}$.

Let $\mathcal{C}_1, \ldots, \mathcal{C}_q$ be the partition of $\mathcal{C}(\mathcal{F})$ obtained. We claim that the induced partition $S_1, \ldots, S_q$ of $\mathcal{C}_1, \ldots, \mathcal{C}_q$, where $S_i = \cup\{S_C : C \in \mathcal{C}_i\}$, with the corresponding centers $s_1, \ldots, s_q$ (chosen at the first step), is a spider-cover decomposition of F. From Lemma 3(ii) and the construction it follows that $V(S_1), \ldots, V(S_q)$ are pairwise disjoint. Thus it remains to show that S_i is an $(s_i, \mathcal{C}_i)$-spider-cover for every $i = 1, \ldots, q$.

Fix some part $\mathcal{C} = \mathcal{C}_i$, and let $S = S_i$ and $s = s_i$. We prove that S is an $(s, \mathcal{C})$-spider cover, where the corresponding partition of S is $\{S_C : C \in \mathcal{C}\}$. Note that if $\mathcal{C} = \{C\}$, then $C \notin \mathcal{D}$, hence no $X \in \mathcal{F}$ contains both C and s; otherwise, if there is such X, then $X \cup L_C \in \mathcal{F}$ by Lemma 2, contradicting that $C \notin \mathcal{D}$. Now recall that the pair $\mathcal{C}, s$ was obtained in one of the following two ways:

1. A subset of $\mathcal{C}(\mathcal{F}) - \mathcal{D}$ corresponding to a star with center s chosen at step 1, to which we added at step 2 some cores $C \in \mathcal{D}$ with $s \in M_C$ (see Fig. 2(a)).
2. An equivalence class $\mathcal{C}$ of size at least 2 of the relation $\mathcal{R}$ on $\mathcal{D}$, with s chosen arbitrarily from $\{s_C : C \in \mathcal{C}\}$ (see Fig. 2(b)).

In both cases, the node sets $\{V(S_C) - s : C \in \mathcal{C}\}$ are pairwise disjoint by Lemma 3(ii) and the construction, and S_C is an $\mathcal{F}(s, C)$-cover for every $C \in \mathcal{C}$ by Corollary 2. Consequently, S is an $(s, \mathcal{C})$-spider-cover, as claimed.

3 Covering Uncrossable Families (Proof of Theorem 4)

We use a *Greedy Algorithm* for the following type of problems:

Covering Problem
Instance: A ground-set E and integral function ν, w on 2^E, where $\nu(E) = 0$.
Objective: Find $I \subseteq E$ with $\nu(I) = 0$ and with $w(I)$ minimized.

In the Covering Problem, ν, w may be given by an evaluation oracle; ν is the *deficiency function* that measures how far is I from being a feasible solution, and w the *weight function*. Let $\rho > 1$ and let opt be the optimal solution value for the Covering Problem. The *ρ-Approximate Greedy Algorithm* starts with $I = \emptyset$ and as long as $\nu(I) \geq 1$ adds to I a set $S \subseteq E - I$ so that

$$\sigma_I(S) = \frac{w(S)}{\nu(I) - \nu(I+S)} \leq \rho \cdot \frac{\mathsf{opt}}{\nu(I)}. \tag{2}$$

The following statement is known (c.f., [14] for a slightly weaker statement).

Theorem 6. *For any Covering Problem so that ν is decreasing and w is increasing and sub-additive, the ρ-Approximate Greedy Algorithm computes a solution I so that $w(I) \leq \rho H(\nu(\emptyset)) \cdot \mathsf{opt}$.*

For $I \subseteq E$ define: $\nu(I) = |\mathcal{C}(\mathcal{F}_I)|$, $w(I) = w(V(I))$. Clearly, ν is decreasing, and w is increasing and sub-additive. Theorem 4 will be proved if we prove:

Lemma 5. *For $\nu(I) = |\mathcal{C}(\mathcal{F}_I)|$ and $w(I) = w(V(I))$, an edge set $S \subseteq E - I$ satisfying (2) with $\rho = 3$ can be found in polynomial time under Assumptions 1,2.*

For simplicity of exposition, let us revise our notation and use $\mathcal{F}$ instead of $\mathcal{F}_I$, and let $\nu = \nu(\emptyset)$. We assume that E is a feasible solution, thus $\nu(E) = 0$. Let $\Delta(S) = \nu - \nu(S)$. Then we need to show that under Assumptions 1 and 2 one can find in polynomial time an edge set $S \subseteq E$ so that:

$$\sigma_\emptyset(S) = \frac{w(S)}{\Delta(S)} \leq 3 \cdot \frac{\mathsf{opt}}{\nu}. \tag{3}$$

Lemma 6 (The Spider-Cover Lemma). *Let $\mathcal{F}$ be an uncrossable set-family, let S be an $(s, \mathcal{C})$-cover, and let $\Delta(S) = \nu - \nu(S)$. Then $\Delta(S) \geq \lceil (|\mathcal{C}| - 1)/2 \rceil$ and $\Delta(S) \geq 1$ if $|\mathcal{C}| = 1$.*

Proof. The minimal $\mathcal{F}_S$-cores are pairwise disjoint, and each of them contains some minimal $\mathcal{F}$-core. Let t be the number of $\mathcal{F}_S$-cores containing exactly one minimal $\mathcal{F}$-core. By the definition of an $(s, \mathcal{C})$-cover, any $\mathcal{F}_S$-core C' that contains some $\mathcal{F}$-core C, contains s or contains some other minimal $\mathcal{F}$-core distinct from C. Furthermore, if $\mathcal{C} = \{C\}$ only the latter can hold. Thus $t \leq |\mathcal{C}(\mathcal{F})| - (|\mathcal{C}| - 1)$ if $|\mathcal{C}| \geq 2$, and $t \leq |\mathcal{C}(\mathcal{F})| - 1$ if $|\mathcal{C}| = 1$. The statement follows.

Remark 3. The bound on $\Delta(S)$ given in Lemma 6 is tight, see the full paper. This is the reason why our ratio is $3H(n)$, and *not* $2H(n)$, as in [14]. One might think that a better definition of an $(s, \mathcal{C})$-spider-cover is: an edge-set that covers *all* members of $\mathcal{F}$ separating s and some $C \in \mathcal{C}$. However, then there are examples showing that an appropriate decomposition as in Theorem 5 does not exist.

Corollary 4. *There exists an $(s, \mathcal{C}')$-spider-cover S for which* (3) *holds.*

Proof. Note that if S is an $(s, \mathcal{C})$-cover, then $\Delta(S) \geq |\mathcal{C}|/3$, by Lemma 6; the worse case is when $|\mathcal{C}| = 3$, but in this case $\Delta(S) = 1$. Let $S_1, \ldots, S_q$ be a spider-cover decomposition of an optimal $\mathcal{F}$-cover F. Now the statement follows by a simple averaging argument (see the full paper for a complete proof).

Let us show that Corollary 4 implies Lemma 5. The following algorithm finds $S \subseteq E$ satisfying (3). For every $s \in V$ compute $S \subseteq E$ as follows. For every $C \in \mathcal{C}$ let $W(C)$ be the minimum weight of an $\mathcal{F}(s, C)$-cover, ignoring the weight of s; $W(C)$ can be computed in polynomial time by Assumption 2. Sort the members of $\mathcal{C}$ by increasing weight $W(C_1) \leq W(C_2) \leq \ldots \leq W(C_q)$. Let $W_j = w(s) + \sum_{i=1}^{j} W(C_i)$. Now set:

- $\sigma_1 = W_1$ if s is not in the maximal core containing C_1 and $\sigma_1 = 0$ otherwise;
- $\sigma_j = W_j / \lceil (j-1)/2 \rceil$, $j = 2, \ldots, q$.

We find the index j for which σ_j is maximum, which determines the edge set S. Among the edge sets $\{S : s \in V\}$ computed choose one with $\sigma_\emptyset(S)$ maximum. The time complexity is the time required to compute the family $\mathcal{C}(\mathcal{F})$ (polynomial by Assumption 1), plus $n|\mathcal{C}(\mathcal{F})|$ times the time required to find a minimum weight $\mathcal{F}(s, C)$-cover (polynomial by Assumption 2).

4 Algorithm for NWSN (Proof of Theorem 1)

The algorithm has $r_{\max}$ iterations. Iteration k starts with a partial solution H satisfying $\lambda_H(u, v) \geq \min\{r(u, v), k-1\}$ for all $u, v \in V$ and returns an augmenting edge set $F \subseteq E - E(H)$ of node-weight $w(F) \leq 3H(|U|) \cdot \mathsf{opt}$ so that $\lambda_{H+F}(u, v) \geq \min\{r(u, v), k\}$ for all $u, v \in V$. Hence after $r_{\max}$ iterations, a feasible solution of weight at most $3r_{\max} \cdot H(|U|) \cdot \mathsf{opt}$ is found.

By Menger's Theorem, computing such F is equivalent to finding an $\mathcal{F}$-cover of the family $\mathcal{F} = \{X \subset V : r(X) \geq k, \deg_H(X) = k-1\}$. This $\mathcal{F}$ is uncrossable, c.f., [20]. To apply Theorem 4, we need to show that Assumptions 1,2 hold for $\mathcal{F}$. For that, we show a polynomial time algorithm for the following "augmentation version" of NWk-F:

Node-Weighted k-Flow Augmentation (NWk-FA)
Instance: A graph $G = (V, E)$ with node weights $\{w(v) : v \in V\}$, $s, t \in V$, an integer k, and a subgraph $G_0 = (V, E_0)$ of G so that $\lambda_{G_0}(s, t) = k-1$.
Objective: Find $F \subseteq E - E_0$ so that $\lambda_{G_0+F}(s, t) = k$ and $w(V(F))$ is minimum.

Proposition 2. NWk-FA *can be solved using one shortest path computation.*

Proof. See the full version.

Any edge set I added at some previous step of iteration k is included in H. To show that Assumptions 1,2 hold for $\mathcal{F}$, we prove in the full paper that:

Corollary 5. *For $\mathcal{F} = \{X \subset V : r(X) \geq k, \deg_H(X) = k-1\}$:*

1. *The family $\mathcal{C}(\mathcal{F})$ can be found using $|U|(|U|-1)/2$ max-flow computations.*
2. *A min-weight $\mathcal{F}(s, C)$-cover can be found with one shortest path computation.*

References

1. Agrawal, A., Klein, P.N., Ravi, R.: When trees collide: An approximation algorithm for the generalized Steiner problem on networks. SIAM J. Comput. 24(3), 440–456 (1995)
2. Chekuri, C., Hajiaghayi, M.T., Kortsarz, G., Salavatipour, M.R.: Approximation algorithms for node-weighted buy-at-bulk network design. In: SODA, pp. 1265–1274 (2007)
3. Dodis, Y., Khanna, S.: Design networks with bounded pairwise distance. In: STOC, pp. 750–759 (1999)
4. Feige, U.: A threshold of $\ln n$ for approximating set cover. Journal of the ACM 45(4), 634–652 (1998)
5. Feige, U., Kortsarz, G., Peleg, D.: The dense k-subgraph problem. Algorithmica 29, 410–421 (2001)
6. Fleischer, L.K., Jain, K., Williamson, D.P.: An iterative rounding 2-approximation algorithm for the element connectivity problem. In: FOCS, pp. 339–347 (2001)
7. Goemans, M.X., Goldberg, A.V., Plotkin, S., Shmoys, D.B., Tardos, E., Williamson, D.P.: Improved approximation algorithms for network design problems. In: SODA, pp. 223–232 (1994)
8. Goemans, M.X., Williamson, D.P.: A general approximation technique for constrained forest problems. SIAM J. on Comput. 24, 296–317 (1995)
9. Goemans, M.X., Williamson, D.P.: The primal-dual method for approximation algorithm and its application to network design problems. In: Hochbaum, D.S. (ed.) Approximation Algorithms for NP-hard problems, ch. 4, pp. 144–191. PWS (1995)
10. Guha, S., Khuller, S.: Improved methods for approximating node weighted steiner trees and connected dominating sets. Inf. Comput. 150(1), 57–74 (1999)
11. Hajiaghayi, M.T., Kortsarz, G., Mirokni, V.S., Nutov, Z.: Power optimization for connectivity problems. Math. Programming 110(1), 195–208 (2007)
12. Jain, K.: A factor 2 approximation algorithm for the generalized Steiner network problem. Combinatorica 21(1), 39–60 (2001)
13. Khuller, S.: Approximation algorithms for for finding highly connected subgraphs. In: Hochbaum, D.S. (ed.) Approximation Algorithms for NP-hard problems, ch. 6, pp. 236–265. PWS (1995)
14. Klein, P.N., Ravi, R.: A nearly best-possible approximation algorithm for node-weighted steiner trees. Journal of Algorithms 19(1), 104–115 (1995)
15. Kortsarz, G., Nutov, Z.: Approximating minimum cost connectivity problems. In: Gonzales, T.F. (ed.) Approximation Algorithms and Metahueristics, ch. 58, PWS (2007)
16. Kortsarz, G., Nutov, Z.: Approximating some network design problems with node costs (manuscript, 2007)
17. Moss, A., Rabani, Y.: Approximation algorithms for constrained node weighted Steiner tree problems. In: STOC, pp. 373–382 (2001)
18. Nutov, Z.: Approximating connectivity augmentation problems. In: SODA, pp. 176–185 (2005)
19. Nutov, Z.: Approximating minimum power covers of intersecting families and directed connectivity problems. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX 2006 and RANDOM 2006. LNCS, vol. 4110, pp. 236–247. Springer, Heidelberg (2006)
20. Williamson, D.P., Goemans, M.X., Mihail, M., Vazirani, V.V.: A primal-dual approximation algorithm for generalized Steiner network problems. Combinatorica 15, 435–454 (1995)

Approximating Minimum-Power Degree and Connectivity Problems

Guy Kortsarz[1], Vahab S. Mirrokni[2], Zeev Nutov[3], and Elena Tsanko[4]

[1] Rutgers University, Camden
guyk@camden.rutgers.edu
[2] Microsoft Research
mirrokni@microsoft.com
[3] The Open University of Israel, Raanana
nutov@openu.ac.il
[4] IBM, Haifa
tsanko@il.ibm.com

Abstract. Power optimization is a central issue in wireless network design. Given a (possibly directed) graph with costs on the edges, the power of a node is the maximum cost of an edge leaving it, and the power of a graph is the sum of the powers of its nodes. Motivated by applications in wireless networks, we consider several fundamental undirected network design problems under the power minimization criteria. Given a graph $\mathcal{G} = (V, \mathcal{E})$ with edge costs $\{c_e : e \in \mathcal{E}\}$ and degree requirements $\{r(v) : v \in V\}$, the Minimum-Power Edge-Multi-Cover (MPEMC) problem is to find a minimum-power subgraph of $\mathcal{G}$ so that the degree of every node v is at least $r(v)$. We give an $O(\log n)$-approximation algorithms for MPEMC, improving the previous ratio $O(\log^4 n)$ of [11]. This is used to derive an $O(\log n + \alpha)$-approximation algorithm for the undirected Minimum-Power k-Connected Subgraph (MPk-CS) problem, where α is the best known ratio for the min-cost variant of the problem (currently, $\alpha = O(\ln k)$ for $n \geq 2k^2$ and $\alpha = O(\ln^2 k \cdot \min\{\frac{n}{n-k}, \frac{\sqrt{k}}{\log n}\})$ otherwise). Surprisingly, it shows that the min-power and the min-cost versions of the k-Connected Subgraph problem are *equivalent* with respect to approximation, unless the min-cost variant admits an $o(\log n)$-approximation, which seems to be out of reach at the moment. We also improve the best known approximation ratios for small requirements. Specifically, we give a 3/2-approximation algorithm for MPEMC with $r(v) \in \{0, 1\}$, improving over the 2-approximation by [11], and a $3\frac{2}{3}$-approximation for the minimum-power 2-Connected and 2-Edge-Connected Subgraph problems, improving the 4-approximation by [4]. Finally, we give a $4r_{\max}$-approximation algorithm for the undirected Minimum-Power Steiner Network (MPSN) problem: find a minimum-power subgraph that contains $r(u, v)$ pairwise edge-disjoint paths for every pair u, v of nodes.

1 Introduction

1.1 Motivation and Problems Considered

Wireless networks are studied extensively due to their wide applications. The power consumption of a station determines its transmission range, and thus also the stations it can send messages to; the power typically increases at least quadratically in the transmission range. Assigning power levels to the stations (nodes) determines the resulting communication network. Conversely, given a communication network, the cost required at v only depends on the furthest node that is reached directly by v. This is in contrast with wired networks, in which every pair of stations that need to communicate directly incurs a cost. We study the design of symmetric wireless networks that meet some prescribed degree or connectivity properties, and such that the total power is minimized. An important network property is fault-tolerance, which is often measured by minimum degree or node-connectivity of the network. Node-connectivity is much more central here than edge-connectivity, as it models stations failures. Such power minimization problems were vastly studied. See for example [1,3,11,18,19,4] for a small sample of papers in this area. The first problem we consider is finding a low power network with specified lower bounds on node degrees. This is the power variant of the fundamental b-Matching/Edge-Multicover problem, c.f., [7]. The second problem is the Min-Power k-Connected Subgraph problem which is the power variant of the classic Min-Cost k-Connected Subgraph problem. We devise approximation algorithms for these problems, improving significantly the previously best known ratios.

Definition 1. *Let $G = (V, E)$ be a graph with edge-costs $\{c(e) : e \in E\}$. For $v \in V$, the* power *$p(v) = p_G(v)$ of v in G (w.r.t. c) is the maximum cost of an edge in G leaving v, i.e., $p(v) = p_E(v) = \max_{vu \in E} c(vu)$. The power of the graph is the sum of the powers of its nodes.*

Unless stated otherwise, graphs are assumed to be undirected and simple. Let $G = (V, E)$ be a graph. For $X \subseteq V$, $\Gamma_G(X) = \{u \in V - X : v \in X, vu \in \mathcal{E}\}$ is the set of neighbors of X, and $d_E(X) = |\Gamma_E(X)|$ is the degree of X in G. Let $\mathcal{G} = (V, \mathcal{E}; c)$ be a *network*, that is, $(V, \mathcal{E})$ is a graph and c is a cost function on $\mathcal{E}$. Let $n = |V|$ and $m = |\mathcal{E}|$. Given a network $\mathcal{G} = (V, \mathcal{E}; c)$, we seek to find a low power *communication network*, that is, a low power subgraph $G = (V, E)$ of $\mathcal{G}$ that satisfies some property. Two such fundamental properties are: degree constraints and fault-tolerance/connectivity. In fact, these problems are related, and we use our algorithm for the former as a tool for approximating the latter.

Definition 2. *Given a requirement function r on V, we say that a graph $G = (V, E)$ (or that E) is an* r-edge cover *if $d_G(v) \geq r(v)$ for every $v \in V$, where $d_G(v) = d_E(v)$ is the degree of v in G.*

Finding a minimum-cost r-edge cover is a fundamental problem in combinatorial optimization, as this is essentially the b-Matching problem, c.f., [7]. The following problem is the power variant.

Minimum-Power Edge-Multi-Cover (MPEMC)
Instance: A network $\mathcal{G} = (V, \mathcal{E}; c)$ and degree requirements $\{r(v) : v \in V\}$.
Objective: Find a min-power subgraph G of $\mathcal{G}$ so that G is an r-edge cover.

We now define our connectivity problems. A graph is *k-connected* (*k-edge-connected*) if it contains k internally-disjoint (k edge-disjoint) uv-paths for all $u, v \in V$.

Minimum-Power k-Connected Subgraph (MPk-CS)
Instance: A network $\mathcal{G} = (V, \mathcal{E}; c)$, and an integer k.
Objective: Find a minimum-power k-connected spanning subgraph G of $\mathcal{G}$.

We also consider min-power variant of the min-cost Steiner Network problem.

Minimum-Power Steiner Network (MPSN)
Instance: A network $\mathcal{G} = (V, \mathcal{E}; c)$ and requirement $\{r(u, v) : u, v \in V\}$.
Objective: Find a minimum-power subgraph G of $\mathcal{G}$ so that G contains $r(u, v)$ pairwise edge-disjoint uv-paths for every $u, v \in V$.

We give improved approximation algorithms for these problems. As a tool for approximating MPEMC, we consider a special case of the following problem:

Budgeted Multi-coverage with Group Constraints (BMGC)
Instance: A bipartite graph $\mathcal{G} = (A + B, \mathcal{E})$, costs $\{c(a) : a \in A\}$, budget P, degree requirements $\{r(b) : b \in B\}$, and a partition $\mathcal{A}$ of A.
Objective: Find $S \subseteq A$ with $c(S) \leq P$ and $\mathsf{val}(S) = \sum_{b \in B} \min\{|\Gamma_{\mathcal{G}}(b) \cap S|, r(b)\}$ maximum, so that $|S \cap A^i| \leq 1$ for every $A^i \in \mathcal{A}$.

If $\mathcal{A}$ is not a partition, but just a collection of subsets of A (even of size 2), then BMGC includes the Independent Set problem even if $r(b) = 1$ for all $b \in B$. Hence assuming that $\mathcal{A}$ partitions A is essential. BMGC generalizes both the Budgeted Maximum Coverage problem (when $\mathcal{A}$ is a partition into singletons) which admits a $(1 - 1/e)$-approximation [14], and the Maximum Coverage with Group Constraints problem in which there is no global budget P and all the requirement are 1. For this special case, [5] gave a 1/2-approximation. We also mention that BMGC belongs to the class of problems that seek to maximize a non-decreasing submodular function under certain constraints. There exists a 1/2-approximation algorithm for matroid constrains [10], and there exist a $(1 - 1/e)$ approximation algorithm for knapsack constrains [21]. BMGC has *both* matroid *and* knapsack constrains, and we are not aware of a technique that handles both.

Studying the approximability of BMGC is beyond the scope of this paper. To get the $O(\log n)$ approximation for MPEMC, we give a $(1 - 1/e)$-approximation algorithm for the following special case.

Definition 3. *A* BMGC *instance has the* Star-Property *if every $A^i \in \mathcal{A}$ admits an ordering $a_1, a_2, \ldots$ by non-decreasing costs so that $\Gamma_{\mathcal{G}}(a_{j-1}) \subseteq \Gamma_{\mathcal{G}}(a_j)$. Let* BMGC* *be the restriction of* BMGC *to instances with the Star-Property.*

1.2 Related Work

Results on MPEMC*:* The Minimum-Cost Edge-Multicover problem is essentially the fundamental b-Matching problem, which is solvable in polynomial time, c.f.,

[7]. The previously best known approximation ratio for the min-power variant MPEMC was $\min\{r_{\max}+1, O(\log^4 n)\}$ due to [11]. The directed MPEMC generalizes the classic Minimum-Cost Set-Multicover problem; the latter is a special case when for every $v \in V$ all the edges leaving v have the same cost.

Results on connectivity problems: The simplest connectivity problem is when we require the network to be connected. In this case, the minimum-cost variant is just the Minimum-Cost Spanning Tree problem, while the minimum-power variant is APX-hard. A 5/3-approximation algorithm for the Minimum-Power Spanning Tree problem is given in [1]. Minimum-cost connectivity problems for arbitrary k were extensively studied, see surveys in [13] and [17]. The best known approximation ratios for the Minimum-Cost k-Connected Subgraph (MCk-CS) problem are $O(\ln^2 k \cdot \min\{\frac{n}{n-k}, \frac{\sqrt{k}}{\ln k}\})$ for both directed and undirected graphs [16], and $O(\ln k)$ for undirected graphs with $n \geq 2k^2$ [6]. It turns out that (for undirected graphs) approximating MPk-CS is closely related to approximating MCk-CS and MPEMC, as shows the following statement.

Theorem 1 ([11])
(i) *If there exists an α-approximation algorithm for* MCk-CS *and a β-approximation algorithm for* MPEMC *then there exists a $(2\alpha+\beta)$-approximation algorithm for* MPk-CS.
(ii) *If there exists a ρ-approximation algorithm for* MPk-CS *then there exists a $(2\rho+1)$-approximation for* MCk-CS.

One can combine various values of α, β with Theorem 1 to get approximation algorithms for MPk-CS. In [11] the bound $\beta = \min\{k+1, O(\log^4 n)\}$ was derived. The best known values for α are: $\alpha = \lceil (k+1)/2 \rceil$ for $2 \leq k \leq 7$ (see [2] for $k = 2, 3$, [8] for $k = 4, 5$, and [15] for $k = 6, 7$); $\alpha = k$ for $k = O(\log n)$ [15], $\alpha = 6H(k)$ for $n \geq k(2k-1)$ [6], and $\alpha = O(\ln k \cdot \min\{\sqrt{k}, \frac{n}{n-k} \ln k\})$ for $n < k(2k-1)$ [16]. Thus for undirected MPk-CS the following ratios follow: $3k$ for any k, $k + 2\lceil (k+1)/2 \rceil$ for $2 \leq k \leq 7$, and $O(\log^4 n)$ unless $k = n - o(n)$. Improvements over the above bounds are known only for $k \leq 2$. Calinescu and Wan [4] gave a 4-approximation algorithm for the case $k = 2$ of undirected MPk-CS. They also gave a $2k$-approximation algorithm for undirected MPk-ECS for arbitrary k. For further results on other minimum-power connectivity problems, among them problems on directed graphs see [3,11,19,18].

1.3 Our Results

The previous best approximation ratio for MPEMC was $\min\{r_{\max}+1, O(\log^4 n)\}$ [11]. We prove:

Theorem 2. *Undirected* MPEMC *admits an $O(\log n)$-approximation algorithm.*

This result uses the following statement:

Lemma 1. BMGC* *admits a $(1-1/e)$-approximation algorithm.*

The previously best known ratio for MPk-CS was $O(\alpha + \log^4 n)$ [11], where α is the best ratio for MCk-CS. From Theorems 2 and 1, and from [6], we get:

Theorem 3. MPk-CS *admits an* $O(\alpha + \log n)$*-approximation algorithm, where* α *is the best ratio for* MCk-CS*. In particular, for* $n \geq 2k^2$, MPk-CS *admits an* $O(\log n)$*-approximation algorithm.*

Theorem 3 implies that the min-cost and the min-power variants of the k-Connected Subgraph problem are equivalent with respect to approximation, unless the min-cost variant admits a better than $O(\log n)$-approximation; the latter seems to be out of reach at the moment, see [16,6]; the best known ratio for MCk-CS when $k = n - o(n)$ is $\tilde{O}(\sqrt{n})$ [16]. This equivalence can turn useful for establishing a lower bound for MCk-CS. In particular, if we can show an approximation threshold of $\Omega(\log^{1+\varepsilon} n)$ for MPk-CS, then the same threshold applies for MCk-CS; on the other hand, if MPk-CS admits a logarithmic ratio, then so does MCk-CS. Note that for $n \geq 2k^2$, our ratio for MPk-CS is $O(\log n)$, and this matches the best known ratio for MCk-CS with $n \geq 2k^2$ of [6].

We also consider the case of small requirements which often arises in practical networks. For $0,1$-MPEMC (namely, MPEMC with $0,1$-requirements) the previously best known ratio was 2 [11]. We prove:

Theorem 4. $0,1$-MPEMC *admits a* $3/2$*-approximation algorithm.*

Theorem 5. *Undirected* MPk-ECS *with* k *arbitrary and undirected* MPk-CS *with* $k \in \{2,3\}$ *admit a* $(2k - 1/3)$*-approximation algorithm.*

For $k = 2$, Theorem 5 improves the best previously known ratio of 4 [4] to $3\frac{2}{3}$. For $k = 3$ the improvement is from 7 to $5\frac{2}{3}$.

We also consider the MPSN problem. Williamson et. al [22] gave a $2r_{\max}$-approximation algorithm for the min-cost case. The currently best known ratio for the min-cost case is 2 [12]. We show that the algorithm of [22] for the minimum-cost case, has approximation ratio $4r_{\max}$ for the minimum-power variant MPSN.

Theorem 6. *Undirected* MPSN *admits a* $4r_{\max}$*-approximation algorithm.*

Theorems 2 and 4 are proved in Sections 2 and 3, respectively. For the proofs of Lemma 1 and Theorems 5 and 6, see the full paper.

1.4 Techniques

The technique used for approximating MPEMC is new and is not similar to the weaker approximation given in [11]. For MPk-CS we use the easy reduction from approximating MPk-CS to approximating MPEMC [11] and rely on our new MPEMC approximation. Thus, designing new approximation for MPEMC is the crux of the matter. Approximating MPEMC turned up to be a rather challenging task (some reasons for that are explained in Section 1.5). Intuitively, the difficulty is that adding an edge to the solution may cause the increase in power for both endpoints of the edge. Thus if we are given a budget and attempt to satisfy as much demand as possible within the budget, this turns out to be as hard as the dense k-subgraph problem (see [9]), as explained in Section 1.5. The algorithm of [11] is unsuited for deriving an $O(\log n)$ ratio for MPEMC, as it pays a $\log^2 n$ factor in the ratio, from the get-go. Hence, a completely new strategy is required. The ideas of our algorithm are summarized as follows:

1. **Reduction to bipartite graphs:** We reduce the problem to a bipartite graph $\mathcal{G}' = (A + B, \mathcal{E}')$, with each of A and B being a copy of V. Thus every node has two occurrence, one in A and one in B. The side of B has degree requirement while A is the "covering side" and has no demands. This reduction is simple, but it is crucial for technical reasons.
2. **Ignoring dangerous edges:** The algorithm works in iterations. At every iteration some edges are declared "dangerous", hence forbidden for use in the current iteration; *this is our main new technique*. Classifying edges as dangerous depends not only on their cost, but also on the *residual demand*; hence the set of dangerous edges changes from iteration to iteration. We prove that at any specific iteration, the contribution of dangerous edges to the cover cannot be too large, as they are too expensive to cover "too much" of the demand. Intuitively, ignoring dangerous edges is a trick that allows us to focus on minimizing the power of the nodes in A only; even if every $b \in B$ is touched by its most expensive non-dangerous edge, we are still able to appropriately bound the increase in the power of the nodes in B. We believe that this technique will have further applications.
3. **Reduction to the BMGC*:** At every iteration of the algorithm the goal is to pay $O(\mathsf{opt})$ in the power increase, and reduce the sum of the (residual) demands by a constant fraction. Hence, after $O(\log n)$ iterations, all the requirements are satisfied, and the $O(\log n)$ ratio follows. In every iteration, after the dangerous edges are ignored, we are able to cast the problem we need to solve as an instance of BMGC*.
4. **Approximating BMGC*:** We design a simple "local-replacement" $(1-1/e)$-approximation algorithm for BMGC*. The analysis, which is quite involved, generalizes the analysis of the algorithm of [14] for the Budgeted Maximum Coverage problem. The difference is that the [14] algorithm only adds elements, and hence it is not a local replacement algorithm.

Our approach for $0,1$-MPEMC is inspired by the decomposition method used by Prömel and Steger [20] for the Minimum-Cost Steiner Tree problem: decomposing solutions into small parts, and then reducing the problem to the minimum-cost case in 3-uniform hypergraphs, with loss of $5/3$ in the approximation ratio. A similar method was used in [1] for the Minimum-Power Spanning Tree problem. In our case, to prove Theorem 4, we use a reduction to the minimum-cost case in graphs, and the loss in the approximation ratio is $3/2$. This method works only for $\{0,1\}$ requirements.

For MPk-CS with $k = 2, 3$ and for MPk-ECS, we show a 2-approximation algorithm for the "augmentation problem" of increasing the connectivity by 1. Combining with the $5/3$-approximation algorithm of [1] for the Minimum-Power

Spanning Tree gives the ratio in Theorem 5. However, the 2-approximation for the augmentation problem is not straightforward, and uses new techniques. The augmentation version admits an easy 4-approximation by combining three facts:

(i) any minimal solution to the augmentation problems is a forest, c.f., [22];
(ii) the min-cost augmentation problem admits a 2-approximation [22,8];
(iii) $c(F) \le p(F) \le 2c(F)$ if F is a forest, see Proposition 1.

This is how we obtain our $4r_{\max}$-approximation for MPSN in Theorem 6. However, getting a ratio of 2 for the augmentation version of MPk-CS with $k = 2, 3$ and for MPk-ECS, is *not* straightforward. This is done by considering *directed* solutions to a related problem with so called k-inconnected graphs, and showing, by a careful analysis, that they have low maximum indegree.

1.5 Power Optimization vs. Cost Optimization: A Comparison

Theorem 3 implies that, unless MCk-CS admits a better than $O(\log n)$ approximation ratio, the minimum-power version MPk-CS and the minimum-cost version MCk-CS of the k-Connected Subgraph problem are *equivalent* with respect to approximation: one of the problems admits a polylogarithmic approximation if, and only if, the other does, and the same holds for superlogarithmic approximation thresholds. This near approximability equivalence of MPk-CS and MCk-CS is a rare and surprising example in power versus cost problems. Typically, problems behave completely differently in the minimum-power versus the minimum-cost models. Power problems are "threshold" type of problems, in the sense that, if many edges of the same (maximum) cost touch a node v, or just one such edge touches v, the power of v is the same.

We now compare in detail some additional aspects of power versus cost problems. Note that $p(G)$ differs from the ordinary cost $c(G) = \sum_{e \in E} c(e)$ of G even for unit costs; for unit costs, if G is undirected, then $c(G) = |E|$ and (if G has no isolated nodes) $p(G) = |V|$. For example, if E is a perfect matching on V then $p(G) = 2c(G)$. If G is a clique then $p(G)$ is roughly $c(G)/\sqrt{|E|/2}$. For directed graphs, the ratio of the cost over the power can be equal to the maximum outdegree, e.g., for stars with unit costs. The following statement (c.f., [11]) shows that these are the extremal cases for general edge costs.

Proposition 1. *$c(G)/\sqrt{|E|/2} \le p(G) \le 2c(G)$ for any undirected graph $G = (V, E)$, and if G is a forest then $c(G) \le p(G) \le 2c(G)$. For any directed graph G holds: $c(G)/\Delta(G) \le p(G) \le c(G)$, where $\Delta(G)$ is the maximum outdegree of a node in G.*

Minimum-power problems are usually harder than their minimum-cost versions. The Minimum-Power Spanning Tree problem is APX-hard. The problem of finding minimum-cost k pairwise edge-disjoint paths is in P (this is the Minimum-Cost k-Flow problem, c.f., [7]) while both directed and undirected minimum-power variants are unlikely to have even a polylogarithmic approximation [11,18]. Another example is finding an arborescence rooted at s, that is, a subgraph that contains an sv-path for every node v. The minimum-cost case is in P (c.f., [7]), while the minimum-power variant is at least as hard as the Set-Cover problem. For more examples see [1,3,18,19].

For min-cost problems, a standard reduction from the undirected variant to the directed one is replacing every undirected edge $e = uv$ by two opposite directed edges uv, vu of the same cost as e, finding a solution D to the directed variant and take the underlying graph G of D. However, this reduction does not work for *min-power* problems. The power of G can be much larger than that of D, e.g., if D is a star. In the power model, directed and undirected variants behave rather differently, as illustrated by the following example.

Example: Suppose that we are given an instance of MPEMC and a budget P and our goal is to solve the "budgeted coverage" version of MPEMC: to cover the maximum possible demand using power at most P. We will show that this problem is harder than the Densest k-Subgraph problem, which is defined as follows: given a graph $\mathcal{G} = (V, \mathcal{E})$ and an integer k, find a subgraph of $\mathcal{G}$ with k nodes that has the maximum number of edges. The best known approximation ratio for Densest k-Subgraph is roughly $n^{-1/3}$ [9], and in spite of numerous attempts to improve it, this ratio holds for over 11 years. We prove:

Proposition 2. *If there exists a ρ-approximation algorithm for the budgeted coverage version of* MPEMC *with unit costs, then there exists a ρ-approximation algorithm for* Densest k-Subgraph*.*

Proof. Given an instance $\mathcal{G} = (V, \mathcal{E}), k$ of Densest k-Subgraph, define an instance $(\mathcal{G}, r, P)$ of budgeted coverage version of MPEMC with unit costs as follows: $r(v) = k - 1$ for all $v \in V$ and $P = k$. Then the problem is to find a node subset $U \subseteq V$ with $|U| = k$ so that the number of edges in the subgraph induced by U in $\mathcal{G}$ is maximum. The later is the Densest k-Subgraph problem.

The most natural heuristic for approximating MPEMC is as follows. Guess opt (more precisely, using binary search, guess an almost tight lower bound on opt). Cover maximum amount of the demand within budget opt, and iterate. Proposition 2 shows that this strategy fails.

2 Approximating MPEMC (Proof of Theorem 2)

2.1 Reduction to Bipartite Graphs

We will show an $O(\log n)$-approximation algorithm for (undirected) *bipartite* MPEMC where $\mathcal{G} = (A + B, \mathcal{E})$ is a bipartite graph and $r(a) = 0$ for every $a \in A$. The following statement shows that getting an $O(\log n)$-approximation algorithm for the bipartite MPEMC is sufficient.

Lemma 2. *If there exists a ρ-approximation algorithm for bipartite* MPEMC *then there exists a 2ρ-approximation algorithm for general* MPEMC*.*

Proof. Given an instance $(\mathcal{G} = (V, \mathcal{E}), c, r)$ of MPEMC, construct an instance $(\mathcal{G}' = (V' = A + B, \mathcal{E}'), c', r')$ of bipartite MPEMC as follows. Let $A = \{a_v :$

$v \in V\}$ and $B = \{b_v : v \in V\}$ (so each of A, B is a copy of V) and for every $uv \in \mathcal{E}$ add two edges: $a_u a_v$ and $a_v a_u$ each with cost $c(uv)$. Also, set $r'(b_v) = r(v)$ for every $b_v \in B$ and $r'(a_v) = 0$ for every $a_v \in A$. Given $F' \subseteq \mathcal{E}'$ let $F = \{uv \in \mathcal{E} : a_u b_v \in F' \text{ or } a_v b_u \in F'\}$ be the edge set in $\mathcal{E}$ that corresponds to F'. Now compute an r'-edge cover E' in $\mathcal{G}'$ using the ρ-approximation algorithm and output the edge set $E \subseteq \mathcal{E}$ that corresponds to E', namely $E = \{uv \in \mathcal{E} : a_u b_v \in E' \text{ or } a_v b_u \in E'\}$. It is easy to see that if F' is an r'-edge cover then F is an r-edge cover. Furthermore, if for every edge in F correspond two edges in F' ($|F'| = 2|F|$), then F is an r-edge cover if, and only if, F' is an r'-edge cover. The later implies that $\mathsf{opt}' \leq 2\mathsf{opt}$, where opt and opt' is the optimal solution value to $\mathcal{G}, c, r$ and $\mathcal{G}', c', r'$, respectively. Consequently, E is an r-edge cover, and $p_E(V) \leq p_{E'}(V') \leq \rho\mathsf{opt}' \leq 2\rho\mathsf{opt}$.

2.2 Am $O(\log n)$-approximation for Bipartite MPEMC

We prove that bipartite MPEMC admits an $O(\log n)$-approximation algorithm. The *residual requirement* of $v \in V$ w.r.t. an edge set I is defined by $r_I(v) = \max\{r(v) - d_I(v), 0\}$. One of the main challenges is achieving the following reduction, which will be proved in the next section using our algorithm for BMGC.

Lemma 3. *For bipartite* MPEMC *there exists a polynomial time algorithm that given an integer τ and $\gamma > 1$ either establishes that $\tau <$ opt or returns an edge set $I \subseteq \mathcal{E}$ such that for $\beta = (1 - 1/e)(1 - 1/\gamma)$ the following holds:*

$$p_I(V) \leq (\gamma + 1)\tau \tag{1}$$

$$r_I(B) \leq (1 - \beta)r(B) \tag{2}$$

Note that if $\tau < \mathsf{opt}$ the algorithm may return a edge set I that satisfies (1) and (2); if the algorithm declares "$\tau < \mathsf{opt}$" then this is correct. An $O(\log n)$-approximation algorithm for the bipartite MPEMC easily follows from Lemma 3:

```
*While* $r(B) > 0$ do
        - Find the least integer $\tau$ so that the algorithm in Lemma 3
          returns an edge set $I$ so that (1) and (2) holds.
        - $E \leftarrow E + I$, $\mathcal{E} \leftarrow \mathcal{E} - I$, $r \leftarrow r_I$.
*End While*
```

We note that the least integer τ as in the main loop can be found in polynomial time using binary search. For any constant $\gamma > 1$, say $\gamma = 2$, the number of iterations is $O(\log r(B))$, and at every iteration an edge set of power at most $(1+\gamma)\mathsf{opt}$ is added. Thus the algorithm can be implemented to run in polynomial time, and has approximation ratio $O(\log r(B)) = O(\log(n^2)) = O(\log n)$.

2.3 Proof of Lemma 3

Let τ be an integer and let $R = r(B) = \sum_{b \in B} r(b)$. An edge $ab \in \mathcal{E}$, $b \in B$, is *dangerous* if $c(ab) \geq \gamma\tau \cdot r(b)/R$. Let $\mathcal{I}$ be the set of non-dangerous edges in $\mathcal{E}$.

Lemma 4. *Assume that $\tau \geq opt$. Let F be a set of dangerous edges with $p_F(B) \leq \tau$. Then $r_F(B) \geq R(1-1/\gamma)$. Thus $r_{\mathcal{I}}(B) \leq R/\gamma$.*

Proof. Let $D = \{b \in B : d_F(b) > 0\}$. We show that $r(D) \leq R/\gamma$, implying $r_F(V) \geq R - r(D) \geq R(1-1/\gamma)$. Since all the edges in F are dangerous, $p_F(b) \geq \gamma\tau \cdot r(b)/R$ for every $b \in D$. Thus

$$\tau \geq opt \geq \sum_{b \in D} p_F(b) \geq \sum_{b \in D} (\gamma\tau \cdot r(b)/R) = \frac{\gamma\tau}{R} \sum_{b \in D} r(b) = \frac{\gamma\tau}{R} r(D) \ .$$

For the second statement, note that there exists $E \subseteq \mathcal{E}$ with $p_E(V) \leq \tau$ so that $r_E(B) = 0$. Thus $r_I(B) \leq R/\gamma$ holds for the set I of non-dangerous edges in E. As $I \subseteq \mathcal{I}$, the statement follows.

Lemma 5. $p_{\mathcal{I}}(B) \leq \gamma\tau$.

Proof. Note that $p_{\mathcal{I}}(b) \leq \gamma\tau \cdot r(b)/R$ for every $b \in B$. Thus:

$$p_{\mathcal{I}}(B) = \sum_{b \in B} p_{\mathcal{I}}(b) \leq \sum_{b \in B} (\gamma\tau \cdot r(b)/R) = \frac{\gamma\tau}{R} \sum_{b \in B} r(b) = \gamma\tau \ .$$

Lemmas 4 and 5 imply that we may ignore the dangerous edges and still be able to cover a constant fraction of the total demand. Once dangerous edges are ignored, the algorithm does not need to take the power incurred in B into account, as the total power of B w.r.t. all the non-dangerous edges is $\gamma\tau = O(\mathsf{opt})$. Therefore, the problem we want to solve is similar to the bipartite MPEMC, except that we want to minimize the power of A only. Formally:

Instance: A bipartite graph $\mathcal{G} = (A + B, \mathcal{I})$, edge-costs $\{c(e) : e \in \mathcal{I}\}$, requirements $\{r(b) : b \in B\}$, and budget $\tau = P$.
Objective: Find $I \subseteq \mathcal{I}$ with $p_I(A) \leq P$ and maximum $\sum_{b \in B} \min\{d_I(b), r(b)\}$.

Lemma 6. *The above problem admits a* $(1-1/e)$*-approximation algorithm.*

Proof. We show that the problem above can be reduced, while preserving approximation ratio, to BMGC*. Given an instance of the above problem, construct an instance of BMGC* as follows. For every $a \in A$ do the following. Let $e_1, ..., e_k$ be the edges incident to a sorted by increasing costs. For every e_i add a node a_i of cost $c(a_i) = c(e_i)$ and for every edge ab of cost $\leq c(e_i)$ add an edge $a_i b$. The group corresponding to $a \in A$ is $A^a = \{a_1, \ldots, a_k\}$, so $\mathcal{A} = \{A^a : a \in A\}$. Clearly, the groups are disjoint, hence we obtain a BMGC instance. The Star-Property holds by the construction. Every node in A^a corresponds an edge incident to a and has the cost of this edge; thus choosing one node from $\mathcal{A}^a$ also determines the power level of a. Thus, keeping costs, to every solutions to the obtained BMGC instance, corresponds a unique solution to the problem defined above, and vice versa. The statement now follows from Lemma 1.

The algorithm for Lemma 3 is as follows:

1. With budget τ, compute $I \subseteq \mathcal{I}$ using the $(1-1/e)$-approximation algorithm from Lemma 6.
2. *If* $r_I(B) \leq (1-\beta)R$ (recall that $\beta = 1/2(1-1/\gamma)$) then output I;
   Else declare "$\tau < \mathsf{opt}$".

We show that if $\tau \geq \mathsf{opt}$ then the algorithm outputs an edge set I that satisfies (1) and (2). By Lemma 4, if the algorithm returns an edge set I then (1) holds for I, and if the algorithm declares "$\tau < \mathsf{opt}$" then this is correct. All the edges in I are not dangerous, thus $p_I(B) \leq \gamma\tau$ by Lemma 5. As we used budget τ, $p_I(A) \leq \tau$. Thus $p_I(V) = p_I(A) + p_I(B) \leq (1+\gamma)\tau$.

3 Approximating 0, 1-MPEMC (Proof of Theorem 4)

Given $S \subseteq V$ we say that an edge set F on V is an S-*cover*, if every node in S has an edge in F incident to it. Note that 0, 1-MPEMC is equivalent to the Minimum-Power S-Cover problem, where $S = \{v \in V : r(v) = 1\}$. We reduce Minimum-Power S-cover to Minimum-Cost S-Cover in graphs, where the problem is solvable in polynomial time, c.f., [7], with loss of 3/2 in the approximation ratio. That is, given an instance $(\mathcal{G}, S)$ of Minimum-Power S-Cover, we construct in polynomial time an instance $(\mathcal{G}', S)$ of minimum-Cost S-Cover such that $\mathsf{opt}(\mathcal{G}') \leq 3\mathsf{opt}(\mathcal{G})/2$ and such that for any feasible solution F' to $\mathcal{G}'$ corresponds a feasible solution F to $\mathcal{G}$ with $p(F) \leq c'(F')$.

Clearly, any minimal S-cover is a union of node disjoint stars. Let F be (an edge set of) a star with center v_0. A partition $\mathcal{F} = \{F_1, \ldots, F_{\ell+1}\}$ of F into stars is a t-*decomposition* of F if $|F_{\ell+1}| \leq t-1$ and any other part has at most t edges; $F_{\ell+1}$ covers all the end-nodes of its edges (in particular, it covers v_0) while each part in $\mathcal{F} - F_{\ell+1}$ covers the end-nodes of its edges except v_0 (so every node is covered exactly once). The power $p(\mathcal{F}) = \sum_{F_j \in \mathcal{F}} p(F_j)$ of $\mathcal{F}$ is the sum of the powers of its parts. For a collection of stars the definition is similar.

Lemma 7. *Any star F admits a t-decomposition $\mathcal{F}$ with $p(\mathcal{F}) \leq (1+1/t)p(F)$.*

Proof. Let v_0 be the center of F, let $\{v_1, \ldots, v_d\}$ be the leaves of F, and let $e_i = v_0v_i$ and $c_i = c(e_i)$, $i = 1, \ldots, d$. W.l.o.g., $c_1 \geq c_2 \geq \cdots \geq c_d \geq 1$. Define a t-decomposition $\mathcal{F}$ of F as follows. Let $\ell = \lfloor (d-1)/t \rfloor$, and set: $F_j = \{e_{(j-1)t+1}, \ldots, e_{jt}\}$ for $j = 1, \ldots \ell-1$ and $F_\ell = \{e_{(\ell-1)t+1}, \ldots, e_d\}$. Note that $p(F) = c(F) + c_1$ and $c_{(j-1)t+1} \leq c(F_j)/t$ for $j = 2, \ldots, \ell$; the later is since $e_{(j-1)t+1} \in F_j$, while every edge in F_{j-1} has cost larger than any edge in F_j. Therefore,

$$p(\mathcal{F}) = c(F) + c_1 + \sum_{j=2}^{\ell} c_{(j-1)t+1} \leq c(F) + c_1 + \sum_{j=2}^{\ell} c(F_{j-1})/t \leq (1+1/t)p(F) .$$

Given an instance $(\mathcal{G} = (V, \mathcal{E}; c), S)$ of MPEMC, construct an instance $(\mathcal{G}' = (S, \mathcal{E}'; c'), S)$ of min-cost edge-cover as follows. $\mathcal{G}'$ is a complete graph on S, and

$c'(uv) = p(F_{uv})$ for every $u, v \in S$, where F_{uv} is some min-power $\{u, v\}$-cover that consists of one edge or two adjacent edges. Clearly, we can construct $(\mathcal{G}', S)$ and compute a minimum-cost S-cover in $\mathcal{G}'$ in polynomial time. The following statement that follows from Lemma 7 with $t = 2$ finishes the proof of Theorem 4.

Corollary 1. *If F' is a minimum-cost S-cover in $\mathcal{G}'$ then $F = \cup\{F_{uv} : uv \in E'\}$ is an S-cover in $\mathcal{G}$ and $p(F) \leq c'(F') \leq 3\mathsf{opt}/2$.*

References

1. Althaus, E., Calinescu, G., Mandoiu, I., Prasad, S., Tchervenski, N., Zelikovsky, A.: Power efficient range assignment for symmetric connectivity in static ad-hoc wireless networks. Wireless Networks 12(3), 287–299 (2006)
2. Auletta, V., Dinitz, Y., Nutov, Z., Parente, D.: A 2-approximation algorithm for finding 3-vertex-connected spanning subgraph. J. of Algorithms 32, 21–30 (1999)
3. Calinescu, G., Kapoor, S., Olshevsky, A., Zelikovsky, A.: Network lifetime and power assignment in ad hoc wireless networks. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 114–126. Springer, Heidelberg (2003)
4. Calinescu, G., Wan, P.J.: Range assignment for biconnectivity and k-edge connectivity in wireless ad hoc networks. Mobile Networks and Applications 11(2), 121–128 (2006)
5. Chekuri, C., Kumar, A.: Maximum coverage problem with group budget constraints and applications. In: Jansen, K., Khanna, S., Rolim, J.D.P., Ron, D. (eds.) RANDOM 2004 and APPROX 2004. LNCS, vol. 3122, pp. 72–83. Springer, Heidelberg (2004)
6. Cheriyan, J., Vempala, S., Vetta, A.: An approximation algorithm for the minimum-cost k-vertex connected subgraph. SIAM J. on Computing 32(4), 1050–1055 (2003)
7. Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., Scrijver, A.: Combinatorial Optimization. Wiley, Chichester (1998)
8. Dinitz, Y., Nutov, Z.: A 3-approximation algorithm for finding 4,5-vertex-connected spanning subgraph. J. of Algorithms 32, 31–40 (1999)
9. Feige, U., Kortsarz, G., Peleg, D.: The dense k-subgraph problem. Algorithmica, 410–421 (2001)
10. Fisher, M.L., Nemhauser, G.L., Wolsey, L.A.: An analysis of approximations for maximizing submodular set functions–II. Math. Prog. Study 8, 73–87 (1978)
11. Hajiaghayi, M.T., Kortsarz, G., Mirrokni, V.S., Nutov, Z.: Power optimization for connectivity problems. Math. Program. 110(1), 195–208 (2007)
12. Jain, K.: A factor 2 approximation algorithm for the generalized Steiner network problem. Combinatorica 21(1), 39–60 (2001)
13. Khuller, S.: Approximation algorithms for for finding highly connected subgraphs. In: Hochbaum, D.S. (ed.) Approximation Algorithms for NP-hard problems, ch. 6, pp. 236–265. PWS (1995)
14. Khuller, S., Moss, A., Naor, J.: The budgeted maximum coverage problem. Information Processing Letters 70(1), 39–45 (1999)
15. Kortsarz, G., Nutov, Z.: Approximating node-connectivity problems via set covers. Algorithmica 37, 75–92 (2003)
16. Kortsarz, G., Nutov, Z.: Approximating k-node connected subgraphs via critical graphs. SIAM J. on Computing 35(1), 247–257 (2005)

17. Kortsarz, G., Nutov, Z.: Approximating min-cost connectivity problems. In: Gonzales, T. (ed.) Approximation algorithms and Metaheuristics, ch. 58 (2007)
18. Lando, Y., Nutov, Z.: On minimum power connectivity problems. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 87–98. Springer, Heidelberg (2007)
19. Nutov, Z.: Approximating minimum power covers of intersecting families and directed connectivity problems. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX 2006 and RANDOM 2006. LNCS, vol. 4110, pp. 236–247. Springer, Heidelberg (2006)
20. Prömel, H.J., Steger, A.: A new approximation algorithm for the Steiner tree problem with performance ratio 5/3. J. of Algorithms 36(1), 89–101 (2000)
21. Sviridenko, M.: A note on maximizing a submodular set function subject to a knapsack constraint. Oper. Res. Lett. 32(1), 41–43 (2004)
22. Williamson, D.P., Goemans, M.X., Mihail, M., Vazirani, V.V.: A primal-dual approximation algorithm for generalized steiner network problems. Combinatorica 15, 435–454 (1995)

Energy Efficient Monitoring in Sensor Networks[⋆]

Amol Deshpande, Samir Khuller, Azarakhsh Malekian, and Mohammed Toossi

Department of Computer Science.
University of Maryland, College Park, MD 20742, USA
{amol,samir,malekian,toossi}@cs.umd.edu

Abstract. In this paper we study a set of problems related to efficient energy management for monitoring applications in wireless sensor networks. We study several generalizations of a basic problem called Set k-Cover, which can be described as follows: we are given a set of sensors, and a set of regions to be monitored. Each region can be monitored by a subset of the sensors. To increase the lifetime of the sensor network, we would like to partition the sensors into k sets (or time-slots) and activate each partition in a different time-slot. The goal is to find the partitioning that maximizes the coverage of the regions. This problem is known to be NP-hard. We first develop improved approximation algorithms for this problem based on its similarities to the max k-cut problem. We then consider a variation, called Set (k, α)-cover, where each sensor is allowed to be active in α different time-slots. We develop a randomized routing algorithm for this problem. We then consider extensions where each sensor can monitor only a bounded number of regions in any time-slot. We develop the first approximation algorithms for this problem. An experimental evaluation of the algorithms we propose can be found in the full version of the paper.

1 Introduction

Efficient energy management in sensor networks is a primary challenge as we expect wireless devices to communicate and to continue to function effectively for long periods of time. In this paper we study a basic set of problems dealing with energy efficient monitoring in sensor networks. One particular question of this type was first formalized in a paper by Slijepcevic and Potkonjak [11], in which they asked for a collection of disjoint set covers.

Let $H = (S \cup R, E)$ denote a bipartite graph in which nodes in set S correspond to sensors, and nodes in set R correspond to regions. There is an edge between node s_i and node r_j if sensor s_i can monitor region r_j. By keeping all sensors activated all the time, clearly all the regions can be monitored continuously (assuming no nodes in R have zero degree). The problem with this solution is that the sensors may not last very long. One might now wonder if a better solution can be obtained. There are multiple ways in which we could formulate this problem. One approach is to partition the nodes in S into k sets $S_1, \ldots, S_k$, such that the sensors in set S_i cover all the regions in R. This is also referred to as the domatic set problem [6] for which a randomized approximation algorithm with factor $O(\log n)$ has been proposed. An alternative formulation (called

[⋆] Full version available at http://www.cs.umd.edu/∼samir/grant/latin-full.pdf

Set k-cover), studied by Abrams et al. [1], instead asks for a partitioning that maximizes the total regions covered. More formally, let $R_i \subseteq R$ denote the regions covered by the sensors in S_i; in other words, $R_i = \{r | (s, r) \in E \wedge s \in S_i\}$. The goal is to maximize $\sum_{i=1}^{k} |R_i|$.

Given such a partitioning, the idea then is to cycle through the k sets S_i in a round-robin fashion. When we activate all sensors in set S_i, we cover the regions in R_i. Thus the objective function tries to maximize the coverage of the regions in the different time slots. We will consume significantly less energy this way, as each sensor is activated in only one of the k sets. Moreover there is evidence to suggest that the battery life of sensors is increased significantly when batteries are used in short bursts, rather than being used continuously [3].

At the same time we will maximize the coverage over time. *Ideally*, each region will belong to R_i for each value of i. Notice that we have relaxed the requirement that each region is always monitored. If k is not very large it is entirely possible that we can actually monitor all regions at all times. Of course, the larger the value of k, the longer we extend the lifetime of the system, while paying a penalty of lowering the coverage level within each time-slot. It also depends on the redundancy of coverage, in other words, it also depends on how many sensors are monitoring each region.

In this paper, we identify and address several generalizations of this problem, many of which, to our best knowledge, have not been addressed before. One generalization we define is called the Set (k, α)-Cover problem. We would like to find k sets, $S_1, S_2, \ldots, S_k$. As before, $S_i \subset S$. We require that each vertex in S belong to at most α such sets. More formally, for all $s_j \in S$ we require that $|\{i | s_j \in S_i\}| \leq \alpha$. A solution for this problem can be mapped to a sensor schedule in which each sensor is active in α of the k time-slots. We have relaxed the requirement that the S_i sets are disjoint which corresponds to the case when $\alpha = 1$. Since this is a generalization of the set k-cover problem, it is also NP-hard.

Since each sensor now belongs to α sets, the battery life is extended by a factor of $\frac{k}{\alpha}$. Our main goal really is to maximize the battery life, subject to adequate coverage requirements. However, for the development of the algorithms, it is easier to fix the parameters k and α and to develop algorithms that work with these parameters. In practice one would consider a spectrum of solutions produced by our algorithms for different choices of k and α.

We then consider a generalization where sensors have *capacity constraints*. In many scenarios, even though a sensor may be able to cover multiple regions, at any given time, it may only be able to actually monitor one or a small number of them. For example, a pan-and-tilt camera can monitor only one region (or a few regions) at any time. We call this a capacity constraint, and denote by $c(s_i)$ the maximum number of regions sensor s_i can cover in one time slot. The goal is to solve the Set k-Cover or Set (k, α)-cover problems given such capacity constraints.

Finally, given a k or a (k, α) pair, a sensor cover problem asks for the best partition of the sensors (along with a designation of which regions to monitor for the capacity-constrained version) that optimizes some coverage property. The optimization goal itself could be one of the following three:

- **avg-coverage:** The average coverage over the k time slots. Formally this would be $\frac{\sum_{i=1}^{k}|R_i|}{k}$.
- **min-coverage(time):** The minimum value of the fractional coverage in any time slot. Formally, this would be $\min_{i=1}^{k} \frac{|R_i|}{|R|}$.
- **min-coverage(region):** The minimum over all regions, of the fraction of time a region is covered. This is important for application where high coverage of regions is required over time.

Different applications may demand support for different optimization goals. For example, for the min-coverage(time) version, we could fix a coverage requirement, by specifying that $|R_i| \geq \gamma|R|$ for some $1 \geq \gamma > 0$ and ask to minimize α.
Under this classification, Abrams et al. [1] study the $\{$Set k-cover, NC, avg-coverage$\}$ version of the problem (where NC denotes that there are no capacity constraints).

We now consider a slightly different view of the Set k-Cover problem. Given the bipartite graph H describing the sensor-region relationship, we construct the following hypergraph $G = (V, E)$. Each node in V corresponds to a sensor. For each region r, we create a hyperedge e that contains the set of sensors that cover the region. We assume that each region is covered by at most d sensors, i.e., the hyper edge has size d. The goal now is to color the nodes of the graph with k colors (this is simply a way to view the partitioning into k sets). The objective is to maximize the total *benefit* of all hyper edges. The benefit of a hyper edge is the number of different colors that the nodes in the hyper edge are colored with. If all nodes in this hyper edge have the same color, then the benefit is 1 since they are all in the same set. If the nodes have k different colors, then this region is always monitored and its benefit is k.

The practical problem is of interest for small d, so it is worth studying this case in more detail, since we do not expect too many sensors to cover the same region, otherwise this suggests that the density of sensors is too high. For $d = 2$ this problem is clearly related to the well studied max k-cut problem for which a SemiDefinite Programming (SDP) based algorithm does very well [7,8]. The max k-cut problem asks for a partitioning of the vertices of a graph into k groups so as to *maximize the number of edges across the cut* (edges connecting vertices in two different groups). While both problems ask for a partitioning of the vertices, the precise objective functions are different.

Outline of Contributions

Our first algorithm (see Section 3) shows how to "reduce" the set k-cover problem to the max k-cut problem and then apply known approximation methods for the latter. In fact, this approach gives rise to an extremely fast and practical method to solve the problem. We also prove some improved approximation factors for small d using this approach (see Section 3). For $d \leq 3$, this gives significantly improved worst case approximation factors compared to the randomized approach in [1]. However, the key point is that this approach gives almost optimal solutions in practice, even for larger values of d. We also present a worst case analysis of this method for large d.

In addition, we are also able to develop a *direct* SDP based algorithm for this problem (the same approach was used to develop algorithms for max k-cut). This gives rise to an algorithm that runs in polynomial time for constant d. The solutions produced are almost as good as the solutions produced by the max k-cut approach but the algorithm is slower compared to the direct reduction to max k-cut. However we believe that this approach will eventually give a better worst case approximation bound for the case when the hyper-edges are large.

In Section 5 we develop an LP based randomized rounding algorithm for approximating the lifetime of the network for the $\{$set (k, α)-cover, NC, min-coverage (time)$\}$ version of the problem (an extension to the min-coverage (region) case is straightforward). We fix a k, α pair. Assuming that a feasible integral solution exists, we are guaranteed to find a feasible fractional solution. Once we obtain the fractional solution, we round it. We use a scaling parameter s to do the rounding. This may increase the number of sets a node belongs to, with the expected number being $s\alpha$. We can prove that the probability that each region is covered is very high and at least $1 - \frac{1}{e^s}$.

For the case where each sensor can only cover one region (unit capacity) when it is active, we develop a polynomial time algorithm that computes an optimal solution for the capacitated set (k, α)-cover problem (see Section 6). For the case of *arbitrary* capacities we develop a polynomial time $(1 - \frac{1}{e})$ approximation since we can show that the problem is NP-hard even when the capacities are three.

2 Prior Work

Designing sleep schedules to maximize the network lifetime while guaranteeing coverage has been one of the most active research areas in wireless sensor networks. Cardei and Wu [5] survey the work in this area, and identify two types of coverage problems, *area* coverage (where the goal is to cover maximally cover the area the sensor network is deployed in), and *target* coverage (where the goal is to cover a set of targets).

The problem we address in this paper can be seen as a target coverage problem, and we briefly review the prior work on this problem. Slijepcevic and Potkonjak [11] pose the problem with full coverage requirement; given a sensor network, the goal is to identify mutually exclusive sets of sensor nodes such that the members of each set cover the monitored regions (targets) completely. They provide several heuristics for this problem. Abrams, Goel and Plotkin [1] develop approximation algorithms for the Set k-Cover problem, where k is provided, and the goal is to find a partitioning that maximizes the coverage. They present a simple randomized algorithm, where each sensor is assigned to one of the k sets, and they show that the resulting solution approximates the optimal solution within a factor of $1 - \frac{1}{e}$. In fact their bound is 0.75 when $k = 2, d = 2$ and approaches $(1 - \frac{1}{e})$ for large d and k. They also show that it is NP-hard to get a polynomial time approximation algorithm with a factor better than $\frac{15}{16} + \epsilon$ for any $\epsilon > 0$. This is shown by a direct reduction from the E4-SET SPLITTING problem for which a $\frac{7}{8} + \epsilon$ hardness has been shown by Hastad [9]. However, the gap between $1 - \frac{1}{e}$ and $\frac{15}{16}$ is significant and our goal is to try and consider other approaches that can be used to narrow this gap further. They also present a distributed greedy algorithm that is a $\frac{1}{2}$ approximation for the problem. Unlike the algorithms we develop in this paper, their randomized algorithms are oblivious to the actual graph structure. Cardei et al. [4] consider a version of the problem that is similar to the Set (k, α)-cover problem, where they allow sensors to belong to multiple sets, and allow the sets to be active for different durations. They present several heuristics for solving the problem. Another related paper is [10] in which the unit capacity case is considered.

3 Max k-cut Approach

We now discuss the Set k-Cover problem. We are given a hyper graph $G = (V, E)$, where each node in V corresponds to a sensor. For each region r we create a hyper edge e that contains the set of sensors that cover the region. The goal now is to color the nodes of the graph with k colors. The objective is to maximize the total *benefit* of all hyper edges. The benefit of a hyper edge is the number of different colors that the nodes in the hyper edge are colored with.

Our algorithm works as follows. We replace each hyper edge $e = \{a_1, \ldots, a_p\}$ by edges (a_i, a_j) for $i \neq j$, essentially replacing each hyper edge by a clique. We then apply the SDP based Max k-Cut approximation algorithm [7] that tries to maximize the number of edges across the cut after partitioning the vertices into k sets. We use the partitioning produced by this algorithm, even though our original objective function is different. Let α_k be the approximation ratio of the SDP based algorithm [7] for Max k-Cut. Frieze and Jerrum showed that α_k satisfies the following.

(i) $\alpha_k > 1 - \frac{1}{k}$
(ii) $\alpha_k - (1 - \frac{1}{k}) \sim 2\frac{\ln k}{k^2}$
(iii) $\alpha_2 \geq 0.878$, $\alpha_3 \geq 0.8$, $\alpha_4 \geq 0.85$, $\alpha_{10} \geq 0.926$, $\alpha_{100} \geq 0.99$.

In the next two subsections we present a worst case analysis of this method. We first give a simple analysis for the case $d = 2$ and this will convey some intuition about why this scheme works well. The analysis for general d is more complex, and we present it subsequently.

3.1 Analysis for $d = 2$

In the Max k-Cut problem the goal is to partition the vertices into k sets to maximize the number of edges whose end points are in different sets. By using an α_k approximation for Max k-Cut, we are able to obtain an approximation guarantee of $\frac{1}{2}(1 + \alpha_k)$.

We know that we will get at least $\alpha_k E^*$ edges across the cut once we run the Max k-Cut algorithm, where E^* is the number of edges across the cut in an optimal solution for Max k-Cut (which is derived from an optimal solution for set k-cover, each region that is monitored in both time slots is essentially an edge across the cut). The optimal solution has total benefit $(E - E^*) + 2E^* = E + E^*$. We get a benefit of at least $2\alpha_k E^* + (E - \alpha_k E^*) = E + \alpha_k E^*$. Taking the ratio, and using the fact that $E^* \leq E$ gives the desired bound. This is significantly better than the oblivious approach of randomly coloring the nodes, regardless of the structure of the graph. If we plug in $\alpha_2 = 0.878$ [8] then we obtain a bound of 0.939. If we plug in $\alpha_3 = 0.8$ [7], we get a bound of 0.9. (In contrast the randomized method of [1] gives a bound of $(1 - \frac{1}{2k})$, which is 0.75 and 0.83 for $k = 2, 3$. Note that α_k also improves as k increases [7].

3.2 Analysis for General d

Recall that we constructed a new multi-graph $G' = (V', E')$ based on the hyper graph $G = (V, E)$. $V' = V$ but instead of each hyper edge $e \in E$, we add edges between all pair of vertices belonging to hyper edge e. We then use the max k-cut algorithm due to Frieze and Jerrum [7] on the graph G'. The next theorem presents the approximation ratio obtained by the max k-cut approach.

The proof is more complicated for when all the hyper-edges have arbitrary sizes. We illustrate the proof for the simpler case when all hyper-edges have the same size d. We show the proof for the case $k \geq d$ since the proof is simpler. Some modifications are required for arbitrary k.

Theorem 1. *The benefit obtained from the method based on Max k-cut is at least $\frac{1}{d} + \frac{\alpha_k}{2}(1 - \frac{1}{d})$ fraction of the maximum benefit for the set k-cover problem.*

Before starting the analysis, we need to define some notation.

Definitions

E_i: set of hyper edges that have i different colors in the optimal solution.
E_i' is the set of hyper edges that have i different colors in the solution based on Max k-cut.
E represents the set of hyper edges in G.
C^* is the Max k-cut in G', with E^* the edges across the cut.
C_o is the cut in G' obtained by the optimal solution for set k-cover, with B_o the corresponding benefit function.
C' is the cut obtained by the approximation algorithm for finding a max k-cut in G' and $|C'|$ the benefit for set k-cover, and E' the corresponding edges across the cut in G'.

We first prove that the optimal benefit for Set k-Cover can be upper bounded. In fact, we show that the total benefit in the optimal solution is at most $|E| + \frac{2|E^*|}{d}$ (Lemma 1). We also show that the benefit obtained from the Max k-cut approach is at least $|E| + \frac{\alpha_k|E^*|}{d}$ (Lemma 2). By using these two observations, we can guarantee that the approximation ratio is least $\frac{1}{d} + \frac{\alpha_k}{2}(1 - \frac{1}{d})$.

Lemma 1. *The total benefit (B_o) in the optimal solution for the set k-cover problem is at most $|E| + 2\frac{|E^*|}{d}$.*

Proof. The benefit obtained from the optimal solution is simply $\sum_{i=1}^{d} i|E_i|$. Since $|E| = \sum_{i=1}^{d} |E_i|$ this can be rewritten as $B_o = \sum_{i=1}^{d} |E_i|(i-1) + |E|$. The solution for set k-cover partitions the vertices into k sets. Consider a hyper edge $e \in E$ that has i different colors on the end points in an optimal solution. The minimum number of edges that it can contribute to the cut obtained by the solution is $\binom{d}{2} - \binom{d-i+1}{2}$. This is when we put all $d-(i-1)$ nodes into one partition, and each of the remaining $(i-1)$ nodes into a separate partition. Hence we have: $|C_o| \geq \sum_{i=1}^{d}(\binom{d}{2} - \binom{d-i+1}{2})|E_i|$. Comparing the corresponding coefficients, it can be seen that the largest ratio is $\frac{2}{d}$. So $B_o = \sum_{i=1}^{d} |E_i|(i-1) + |E| \leq \frac{2|C_o|}{d} + |E| \leq \frac{2|E^*|}{d} + |E|$.

Let us briefly consider a diversion for the case $d = 3$. Lemma 1 shows that $B_o \leq |E| + \frac{2}{3}|E^*|$. Before we prove Lemma 2 in general, we prove a lower bound on the quality of the obtained solution. There are two main reasons for this. The first is that in fact we get a slightly better bound when $d = 3$ as follows (in contrast the bound obtained by [1] is 0.70). The second reason is that it provides some intuition for the case when d is arbitrary, but this proof is easier to understand.

Theorem 2. *For the case when* $d = 3$*, we obtain an approximation factor of* $\frac{1}{3} + \frac{1}{2}\alpha_k$.

Proof. Essentially each hyper edge of size three is reduced to a triangle. Note that when all three nodes belong to different time slots, the benefit function is 3, and we have 3 edges crossing the cut. When the nodes of this hyper edge belong to two time slots, we also have two edges crossing the cut. This is better than the bound obtained by randomized rounding given in [1] which is 0.703.

Let C' be the cut produced by the max k-cut algorithm, and $|E'|$ the number of edges across the cut. The benefit from the max k-cut approach is $\sum_{i=1}^{3} |E'_i|(i-1) + |E|$. The number of edges across the cut in G' is $|E'|$. Clearly $|E'| = 2|E'_2| + 3|E'_3| = 2(|E'_2| + \frac{3}{2}|E'_3|)$. Hence $\frac{|E'|}{2} = |E'_2| + \frac{3}{2}|E'_3|$. So we can conclude that $|C'| = |E| + |E'_2| + 2|E'_3| \geq |E| + \frac{|E'|}{2} \geq |E| + \frac{\alpha_k |E^*|}{2}$. We also know that $|E| \geq \frac{|E^*|}{3}$. Putting these equations together gives us a lower bound on $\frac{|C'|}{B_o}$. Using the bound from Lemma 1, the benefit from the solution based on the Max k-cut approach is at least $(\frac{1}{3} + \frac{1}{2}\alpha_k)$ of the maximum benefit in the optimal solution.

When $d = 3$ our bound depends on α_k and is at least 0.828 for large k. When $k = 3$, α_k is 0.8 and we get a bound slightly better than 0.733.

Lemma 2. *The benefit obtained from the Max k-cut approach is at least* $|E| + \frac{\alpha_k |E^*|}{d}$.

Proof. The benefit from the Max k-cut approach can be formulated as $|C'| = \sum_{i=1}^{d} |E'_i|(i-1) + |E|$. Also $|E'| \leq \sum_{j=2}^{d} \sum_{i=\lceil \frac{d}{j} \rceil}^{\lfloor \frac{d}{j-1} \rfloor} (\binom{d}{2} - d(j-1) + \binom{j}{2} i)|E'_i|$. This is obtained basically by spreading out all d nodes into i sets as evenly as possible, with each group having either $\lceil \frac{d}{i} \rceil$ nodes or $\lfloor \frac{d}{i} \rfloor$ nodes. The ratio of the corresponding coefficients of $|E'_i|$'s is at least $\frac{1}{d}$. So we can conclude that $|C'| \geq |E| + \frac{|E'|}{d} \geq |E| + \frac{\alpha_k |E^*|}{d}$.

Proof (Proof Of Theorem 1). Using the above two lemmas and considering the fact that $|E| \geq \frac{|E^*|}{\binom{d}{2}}$ it can be shown that the benefit from the solution based on the Max k-cut approach is at least $\frac{1}{d} + \frac{\alpha_k}{2}(1 - \frac{1}{d})$ fraction of the maximum benefit in the optimal solution.

4 SDP-Based Formulation

Our approach will be to formulate this as a Semi definite programming (SDP) problem. Consider the following SDP for constant d (and arbitrary k). This formulation is inspired by the Max k-Cut work by Frieze and Jerrum [7] (and the description in this section is taken from their paper). However in our direct SDP formulation (Subsection 4), as we will see shortly, the constraints are significantly more complex.

Let y_j be one of k vectors $a_1, \ldots, a_k$ defined as follows. Consider an equilateral simplex Σ_k in R^{k-1} with vertices $b_1, \ldots b_k$. Let $c = \frac{\sum_{i=1}^{k} b_i}{k}$ be the centroid of Σ_k and let $a_i = b_i - c$. Assume that Σ_k is scaled so that $|a_i| = 1$ for all i.

Frieze and Jerrum show that $a_i \cdot a_j = \frac{-1}{k-1}$, when $i \neq j$. If $i = j$, clearly $a_i \cdot a_j = 1$. (We would like $y_j = a_p$ if and only if $v_j \in S_p$.)

In this framework, let Y_{ij} represent the dot product of the two vectors y_i and y_j. The dot product of the two vectors is 1 if the vertices i and j belong to the same group. The dot product is $\frac{-1}{k-1}$ if they belong to different groups.

The Max k-Cut problem can be formulated as follows:

$$\max \ \frac{k-1}{k} \sum_{i<j} w_{i,j}(1 - y_i \cdot y_j) \ \text{ such that } \ y_j \in \{a_1, \ldots, a_k\}$$

Note that, $1 - y_i \cdot y_j = 0$ if $y_i = y_j$ and $= \frac{k}{k-1}$ otherwise.

In this formulation, the graph has weights on the edges specified by $w_{i,j}$. Since the graph we compute G' is a multi-graph, we define the weight of an edge to be the number of copies of the edge in the multi-graph.

To obtain the SDP relaxation, now replace y_i by v_i where v_i is any n-dimensional unit vector. We add the constraint $v_i \cdot v_j \geq -\frac{1}{k-1}$. Thus we obtain an SDP (semidefinite program) of the following form:

$$\max \ \frac{k-1}{k} \sum_{i<j} w_{i,j}(1 - v_i \cdot v_j) \ \text{ such that } v_j \in S_n, \ \ v_i \cdot v_j \geq -\frac{1}{k-1} \forall\, i \neq j$$

The (fractional) solution obtained from solving this SDP is now rounded by a simple randomized rounding algorithm to obtain a solution for the Max k Cut problem. Rather than using exactly the same rounding as given by Frieze and Jerrum [7] we developed another procedure that had excellent performance.

After finding the solution to the convex program, we perform the following procedure to find the partitioning into k sets. First we compute the $\mathcal{V} = v_1, \ldots v_n \in S_n$ from the decomposition of matrix Y. We choose k vectors at random from $\mathcal{V}$, call them $z_1, \ldots, z_k$. Each of the vectors in Z represents one of the k sets. Vector v_i (which represents sensor i) will be assigned to the set j if and only if z_j is the closest vector to v_i. In other words v_i is assigned to j iff $|v_i - z_j| \leq |v_i - z_{i'}|, \forall i' \in \{1 \ldots k\}$.

In this section we take a different route and explore a *direct* SDP formulation for the set k-cover problem without first reducing to max k-cut. Consider a hyper edge $e = \{v^e_{i_1}, v^e_{i_2}, \ldots, v^e_{i_{d_e}}\}$. For each hyper-edge e, we define a new variable α_e. We can formulate the problem as follows.

$$\max \sum_{e \in E} [(\frac{k-1}{k}((d_e - 1) - \alpha_e)) + 1]$$

such that for each hyper edge e we have the following set of constraints.

Consider all possible Hamilton paths P_e among the set of vertices in the hyper edge e. We will generate a constraint for each possible path. This formulation has an exponential number of constraints, but this is not a serious problem. In practice, each region is only covered by a small number of sensors (for constant d, it is polynomial in any case).

For each hyper edge e, we define a new variable α_e and the following set of constraints (one constraint for each path P_e).

$$\alpha_e \geq \sum_{(i_p,i_q)\in P_e} Y_{i_p i_q} \text{such that } y_j \in \{a_1,\ldots,a_k\}$$

If a hyper edge e has size d_e and intersects p_e groups, then we show that the contribution to the objective function is exactly p_e. This can be seen as follows: Consider the path $P' \in P_e$ with the following structure – the path visits all the nodes in a group before visiting the nodes in another group. Note that this path has exactly $p_e - 1$ edges (each dot product contributes $-\frac{1}{k-1}$ for these edges) that go across two groups, and $d_e - p_e$ local edges (each such dot product clearly contributes 1). For this path the rhs of the constraint is exactly $-\frac{p_e-1}{k-1} + (d_e - p_e)$ (and this is the maximum value of the rhs). Note that for every other path, if there fewer local edges, the sum is only smaller. Putting this value for α_e (the smallest valid choice) gives the correct objective function value of p_e. $(\frac{k-1}{k}((d_e-1) + \frac{p_e-1}{k-1} - d_e + p_e)) + 1 = p_e$.

We use the same rounding approach as described in the previous subsection. We do not report on the computational results as the solutions obtained in practice were slightly worse than the ones obtained by reducing to max k-cut, and the algorithm was much slower.

5 Set (k, α)-Cover Problem

We start with the following Integer Program (IP) formulation (for fixed k, α). We define a 0/1 variable x_{ij}, for $i = 1\ldots k$, $j = 1\ldots n$. When $x_{ij} = 1$ it means that sensor j is active in time-slot i. Constraints are as follows:

$$\forall j \in S \quad \sum_{i=1}^{k} x_{ij} \leq \alpha$$
$$\forall r \in R\ \forall i = 1\ldots k \sum_{(r,p)\in E} x_{ip} \geq 1$$

The first constraint simply states that each sensor may belong to at most α time-slots. The second constraint states that each region j is covered in each time slot.

Clearly there is no benefit to increasing a variable beyond 1. This gives us a linear program (LP) which can be solved efficiently.

We now use randomized rounding to obtain an integral solution in which with high probability each region is covered in all the time slots. This is a bicriteria approximation algorithm in a sense that the approximation factor increases when we look for solutions with higher probability for coverage. To do the randomized rounding, we first scale all the x_{ij} variable by a scale factor s and then we round up x_{ij} to 1 with probability equal to the scaled x_{ij}, call the new variable $x'_{ij} = \min(1, s \cdot x_{ij})$. Each sensor will be active in $s\alpha$ time-slots (in expectation); and each region will be covered in each time-slot with probability at least $(1-\frac{1}{e^s})$. For each sensor $j \in S$, the expected value of $E(\sum_{i=1}^{k} x'_{ij}) = \sum_{i=1}^{k} s x_{ij} = s \sum_{i=1}^{k} x_{ij} \leq s\alpha$. So the expected cost of the new integral solution is at most $s \cdot OPT$. Now we show that each region will be covered in all the time slots with high probability. Probability that a given region r in a given time slot i is not covered is equal to the probability that none of the x_{ip} variables $\forall(r,p) \in E$ is rounded up to one. In other words, $\Pr[r \text{ is not covered at time } i] = \prod_{(r,p)\in E}(1 - sx_{ip}) \leq \frac{1}{e^s}$. So each region in each time slot will be covered with probability at least $1 - \frac{1}{e^s}$.

One might wonder if there is a rounding that satisfies all the constraints with constant s, thus giving a constant approximation. However, it can be shown there is an non-constant integrality gap for this problem. Consider a matrix M of size $\binom{k}{k/2} \times k$. The rows contain all the binary strings of length k that have exactly $\frac{k}{2}$ 1's. Consider each column as a sensor and each row as a region. So if $M_{ij} = 1$, sensor j will cover region i and otherwise not. Suppose we have k time slots. One (fractional) solution is to activate each sensor fractionally for $\frac{2}{k}$ in each time slot so that each location is covered since we have $\frac{k}{2}$ sensors covering each region. This gives us $\alpha = 2$ for the fractional solution. At each time step, at least $(\frac{k}{2} + 1)$ sensors have to be active (otherwise a region is uncovered). Thus the total number of active sensors (summing over all times) is $\Omega(k^2)$. Hence at least one sensor is active in $\Omega(k)$ time slots, and has $\alpha = \Omega(k)$. This shows an integrality gap of $\Omega(k)$, and we can make k as large as $\Omega(\frac{\log n}{\log \log n})$, while still having a polynomial number of regions.

Extensions for the min-coverage(region) are straightforward.

6 Sensors with Capacity Constraints

In many applications, sensors have constraints as to how many locations they can monitor even when the sensor is active. For example, a camera sensor s_i may have the *capability* to monitor a set of regions $N(s_i)$, but in a time slot when s_i is active it can only monitor at most $c(s_i)$ regions. For a fixed camera sensor, in fact $c(s_i)$ may just be 1. For a moving sensor, it is possible that the sensor can cover multiple regions. In this case, we also have to come up with an assignment of each sensor to at most $c(s_i)$ regions in a time-slot when the sensor is active.

6.1 NP-Completeness

We examine several cases and either provide polynomial time algorithms, or approximation algorithms when the problem can be shown to be NP-complete.

We first show that even when the capacities are as low as 3, even the basic problem is NP-complete for $k = 2$ and each region being covered by exactly two sensors.

First recall that Max-Cut[1] is NP-complete for graphs with degree at most 3 [12]. The question is - is there a way to partition the nodes of a graph $G = (V,E)$ into two groups so that at least Δ edges cross the cut?

We give a sketch of the reduction. Let $S = \{s_i | v_i \in V\}$. Each edge $(v_i, v_j) \in E$ corresponds to a region in R that has neighbors s_i and s_j. Since each node has degree at most 3, in any case a sensor covers at most 3 regions so with a capacity of 3 each sensor can cover all regions adjacent to it. Set $k = 2$. There is a partition in which the total coverage is at least $|E| + \Delta$ if and only if there is a solution to Max Cut with at least Δ edges across the cut.

[1] This is the Max k-cut problem when $k = 2$.

6.2 General Capacity

In this case, we consider the set (k, α)-cover problem. Each sensor can be activated in at most α sets and the goal is to maximize the average coverage. Unlike the previous case, each sensor s_i can cover $c(s_i)$ regions in each time slot in which it is active. We develop a randomized $(1 - \frac{1}{e})$-approximation algorithm for this problem. As in the previous section we first construct a bipartite graph. Let $H_c = (S_c, R_c, E'_c)$. For each sensor s_i we create α vertices $s_i^1, \ldots, s_i^\alpha \in S_c$. We put an edge from each of the α copies of a sensor to a region node if that the sensor can cover that region. Next, we select a bounded degree subgraph of H_c with the maximum number of edges. The bound on the degree of each $s_i \in S_c$ is $c(s_i)$ and the bound on the degree of each $r \in R_c$ is k. As in the previous case, we can find the subgraph with the maximum number of edges in polynomial time using network flow.Call the subgraph H_c^*. It is easy to prove that the number of edges in H_c^* is an upper bound on OPT. To actually find the schedule we use the following randomized algorithm:
For each sensor, randomly, choose α of the k available time slots (without replacement).

Theorem 3. *The expected value of the coverage given by the randomized algorithm is at least $1 - \frac{1}{e}$ of the number of edges in H_c^*.*

Proof. The argument used here is similar to the one given in [1] with some adaptations to work for the new method. We first compute the probability that a region r is not covered in a given time slot t. We use N_r to denote the set of neighbors of r in H_c^*. Also N_{rs} refers to the copies of sensor s that belongs to N_r.

We first show that the probability that r is not covered in a specified time slot t is $(1 - \frac{1}{k})^{N_r}$. For each sensor s, the probability that none of the copies of s belonging to N_{rs}, have been covered it, is at most $(1 - \frac{|N_{rs}|}{k})$. For $k \geq 1$, $(1 - \frac{|N_{rs}|}{k}) \leq (1 - \frac{1}{k})^{|N_{rs}|}$. The probability that a specified region r is not covered at a given time slot t is the probability that it is not covered by any of its neighbors. Since for sensors i, j, the two events that N_{ri} is not covering r and N_{rj} is not covering r are independent events, the probability that r is not covered in t is the product of all these probabilities for all N_{rs} sets. So the probability that r is not covered is bounded by $(1 - \frac{1}{k})^{\sum_{s \in S_c} |N_{rs}|} = (1 - \frac{1}{k})^{|N_r|}$. Since we know that N_r is the union of all N_{rs} sets for $s \in S$. In other words, $N_r = \cup_{s \in S} N_{rs}$.

The probability that r is covered is at least in each time slot $1 - (1 - \frac{1}{k})^{N_r}$. Let l_r be the number of sensors covering region r, in our solution. We can see that $E(l_r) \geq k - k((1 - \frac{1}{k})^{|N_r|}$. The optimal solution can be bounded by the number of edges in the H_c^* which is $\sum_r |N_r|$. As shown in [1], for each region r, $\frac{E(l_r)}{|N_r|} \geq (1 - \frac{1}{e})$ which completes the proof.

Since $H_c^* \geq OPT$, we have a $(1 - \frac{1}{e})$ approximation.

6.3 Unit Capacity

We consider the set (k, α)-cover problem with the objective to maximize average coverage. We show that this problem can be solved optimally in polynomial time. Our goal is to maximize the number of regions that can be covered. Recall that we need to define k subsets S_i such that each sensor belongs to at most α subsets.

We first construct the following bipartite multi-graph. Let $H = (S, R, E')$. We create a vertex in S for each sensor and a vertex for each region in R. We put α parallel edges between each sensor and region pair if that the sensor can cover the region. We now select a maximum *bounded degree subgraph*[2] of H with the maximum number of edges. The degree bound on sensor nodes in S is exactly α and the degree bound on region nodes in R is k. This problem can be solved in polynomial time for bipartite graphs using network flows.Once we find a maximum subgraph (it is not necessarily unique) H^*, we then find an edge coloring [2] of H^* using at most k colors where k is the maximum degree (since $\alpha \leq k$). An edge coloring of a graph is an assignment of colors to the edges such that no pair of edges that are incident on a common vertex have the same color. Each color class forms a matching in the bipartite graph and corresponds to a time slot. The sensor nodes will be members of the α color classes corresponding to the colors of the edges incident on the sensor nodes.

Theorem 4. *The running time of the algorithm is constrained by the time taken to compute the bounded degree subgraph with the maximum number of edges. This takes* $O(n^3)$ *time in the worst case where* n *is the number of vertices in the graph.*

Acknowledgements. This research was supported by NSF grants CCF-0430650, CCF-0728839 and CNS-0509220. We thank David Kempe for useful discussions and Matt McCutchen for useful comments on the writeup.

References

1. Abrams, Z., Goel, A., Plotkin, S.: Set k-cover algorithms for energy efficient monitoring in wireless sensor networks. In: IPSN 2004: Proceedings of the third international symposium on Information processing in sensor networks, pp. 424–432 (2004)
2. Alon, N.: A simple algorithm for edge-coloring bipartite multigraphs. Inf. Process. Lett. 85(6), 301–302 (2003)
3. Benini, L., Castelli, G., Macii, A., Macii, E., Poncino, M., Scarsi, R.: A discrete-time battery model for high-level power estimation. In: DATE 2000: Proceedings of the conference on Design, automation and test in Europe, New York, NY, USA, pp. 35–41 (2000)
4. Cardei, M., Thai, M.T., Li, Y., Wu, W.: Energy-efficient target coverage in wireless sensor networks. In: IEEE Infocom (2005)
5. Cardei, M., Wu, J.: Energy-efficient coverage problems in wireless ad-hoc sensor networks. In: Computer Communications, pp. 413–420 (2006)
6. Feige, U., Halldorsson, M., Kortsarz, G., Srinivasan, A.: Approximating the domatic number. SIAM J. on Comput. 32, 172–195 (2002)
7. Frieze, A.M., Jerrum, M.: Improved approximation algorithms for max k-cut and max bisection. In: Proceedings of the 4th International IPCO Conference, pp. 1–13 (1995)
8. Goemans, M.X., Williamson, D.P.: 879-approximation algorithms for max cut and max 2sat. In: STOC 1994: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing, pp. 422–431. ACM Press, New York (1994)

[2] This problem can be viewed as a generalization of the problem of finding a matching in a graph. Each node v has a upper bound of $b(v)$ on the number of chosen edges in the subgraph. With this restriction we wish to compute a subgraph with the maximum number of edges.

9. Hastad, J.: Some optimal inapproximability results. J. ACM 48, 798–859 (2001)
10. Liu, H., Jia, X., Wan, P.: Maximizing lifetime of sensor surveillance systems. IEEE/ACM Trans. on Networking 15, 172–195 (2007)
11. Slijepcevic, S., Potkonjak, M.: Power efficient organization of wireless sensor networks. In: IEEE International Conference on Communications (ICC 2001) (2001)
12. Yannakakis, M.: Node-and edge-deletion NP-complete problems. In: STOC 1978: Proceedings of the tenth annual ACM symposium on Theory of computing, pp. 253–264. ACM Press, New York (1978)

Approximation Algorithms for k-Hurdle Problems

Brian C. Dean, Adam Griffis, and Adam Whitley

School of Computing, Clemson University
Clemson, SC, USA
{bcdean,abgriff,awhitle}@cs.clemson.edu

Abstract. The polynomial-time solvable k-hurdle problem is a natural generalization of the classical s-t minimum cut problem where we must select a minimum-cost subset S of the edges of a graph such that $|p \cap S| \geq k$ for every s-t path p. In this paper, we describe a set of approximation algorithms for "k-hurdle" variants of the NP-hard multiway cut and multicut problems. For the k-hurdle multiway cut problem with r terminals, we give two results, the first being a pseudo-approximation algorithm that outputs a $(k-1)$-hurdle solution whose cost is at most that of an optimal solution for k hurdles. Secondly, we provide two different $2(1-\frac{1}{r})$-approximation algorithms. The first is based on rounding the solution of a linear program that embeds our graph into a simplex, and although this same linear program yields stronger approximation guarantees for the traditional multiway cut problem, we show that its integrality gap increases to $2(1-\frac{1}{r})$ in the k-hurdle case. Our second approximation result is based on half-integrality, for which we provide a simple randomized half-integrality proof that works for both edge and vertex k-hurdle multiway cuts that generalizes the half-integrality results of Garg et al. for the vertex multiway cut problem. For the k-hurdle multicut problem in an n-vertex graph, we provide an algorithm that, for any constant $\varepsilon > 0$, outputs a $\lceil (1-\varepsilon)k \rceil$-hurdle solution of cost at most $O(\log n)$ times that of an optimal k-hurdle solution, and we obtain a 2-approximation algorithm for trees.

1 Introduction

Ever since the early work of Ford and Fulkerson [9], the minimum s-t cut problem and its dual, the maximum s-t flow problem, have together served as a cornerstone for the foundation of the field of combinatorial optimization. Numerous theoretical and practical applications are based on the duality between minimum s-t cuts and maximum s-t flows.

In this paper, we study a natural generalization of the minimum s-t cut problem known as the *k-hurdle problem*, whose objective is to choose a minimum-cost subset of the edges of a graph that cuts every s-t path at least k times. Letting $G = (V, E)$ be a graph with $n = |V|$ vertices and $m = |E|$ edges with costs $c : E \rightarrow \mathbf{R}^+$, we can write the k-hurdle problem as the following integer program,

$$\begin{aligned} OPT = \text{Minimize} & \sum_{e \in E} x(e)c(e) \\ \text{Subject to} & \sum_{e \in p} x(e) \geq k \ \ \forall p \in P_{st} \\ & x(e) \in \{0,1\} \ \forall e \in E, \end{aligned}$$

where P_{st} denotes the set of all s-t paths, each of which we assume has length at least k edges or else there is no feasible solution. Since the two-terminal k-hurdle problem has been well-studied in the literature and known to be solvable in polynomial time, we focus in this paper on "k-hurdle" generalizations of the NP-hard multiway cut and multicut problems.

The *k-hurdle multiway cut problem* takes as input a set of r terminals $t_1 \ldots t_r$ and asks us to compute a minimum-cost subset of edges that cuts every terminal-to-terminal path at least k times. We provide three approximation results for this problem. The first is a pseudo-approximation algorithm that outputs a multiway cut with at least $k-1$ hurdles whose cost is no larger than the optimal cost of a k-hurdle multiway cut, and other two are true approximation algorithms with guarantee $2(1-\frac{1}{r})$. One of these is based on rounding a "simplex embedding" linear program (LP) that, for the classical multiway cut problem ($k=1$) leads to much stronger approximation guarantees (the current champion being a 1.3438-approximation algorithm of [17]); however, we show that the integrality gap of this LP surprisingly increases to $2(1-\frac{1}{r})$ in the k-hurdle case, thereby matching our approximation bound. Our second approximation result is based on half-integrality, where we simplify and extend the work of Garg et al. [13] to the k-hurdle case.

The *k-hurdle multicut problem* takes as input r terminal pairs $(s_i, t_i) \ldots (s_r, t_r)$ and asks us to select a minimum-cost subset of edges that cuts each s_i-t_i path at least k_i times, where the hurdle count k_i can now vary by commodity. The k-hurdle multicut problem seems somewhat more difficult to approximate well if we wish to find a solution containing all of the required hurdles. For any constant $\varepsilon > 0$, we show how to compute a solution that provides a $1-\varepsilon$ fraction of the required hurdles (rounded up) for each commodity, whose cost is at most $O(\log n)$ times that of an optimal k-hurdle solution. If $k_i = O(1)$ for every commodity i, we therefore obtain an $O(\log n)$-approximation. For the special case of k-hurdle multicut in a tree, we obtain a 2-approximation algorithm.

1.1 Literature Review

Many authors [4,21,22,25,26] have studied the k-hurdle problem (also known as the minimum k-cut problem in the literature), and several polynomial-time solution algorithms for it are known. Its linear programming relaxation can be solved in polynomial time using the ellipsoid method (using a shortest path algorithm as a feasibility oracle), and it can also be restated using a polynomial number of constraints. Burch et al. [4] show that as a consequence of total unimodularity, one can always find an optimal integer-valued solution.

The k-hurdle problem and its relatives arise often in practice in the domain of *network interdiction*, also called *network inhibition* (see, e.g., [3,7,23,24,27]), where for example we might want to build multiple redundant layers of checkpoints for inspecting goods being shipped through a network, or we might want to disable multiple layers of edges in a network to inhibit the movement of a malicious adversary. The k-hurdle problem and its LP relaxation can also be viewed as special cases of "shortest path" network interdiction problems, where we can pay $c(e)$ per unit length to increase the length of edge e, with a goal of increasing the shortest s-t path length to at least k; see also [16].

The dual of the k-hurdle problem is known in the literature as the k-maximum flow problem [26], and since it can be expressed as a minimum cost circulation problem with unit costs, we can solve it (and hence also the k-hurdle problem) in $\tilde{O}(mn)$ time [14], where $\tilde{O}$ hides logarithmic factors. Linear programs of this flavor are often found in network upgrading and improvement applications, where we want to maximize the amount of additional flow one can send from s to t (typically subject to a budget constraint), where the capacity of certain edges can be upgraded at a price; see [10,18,20] for further details.

The classical ($k = 1$) multiway cut problem is APX-hard for $r \geq 3$ terminals [8], but can be approximated fairly well. There are several ways to obtain a $2(1-1/r)$-approximation bound, the first being a simple "isolating cut" heuristic due to Dahlhaus et al. [8]. A performance bound of 2 is also achievable via LP rounding, since the natural LP relaxation of the multiway cut problem is known to be 1/2-integral. Recently, Călinescu et al. [5] developed an elegant $(1.5 - \frac{1}{r})$-approximation algorithm that solves an LP to embed a graph into an r-simplex, then cuts the simplex using *side-parallel cuts* (hyperplanes parallel to the faces of the simplex) to induce a multiway cut in the graph. This same approach was improved by Karger et al. [17] to obtain a guarantee of 1.3438, and Karger et al. as well as Cheung et al. [6] independently obtained a guarantee of 12/11 for the special case of $r = 3$.

The classical multicut problem ($k_i = 1$ for every commodity $i = 1 \dots r$), is APX-hard for $r \geq 3$ and can be approximated to within an $O(\log r)$ factor using the prominent "region-growing" approach of Garg et al. [11]. In a tree, one can obtain a 2-approximation algorithm using the primal-dual algorithm of Garg et al. [12], or a more recent approach independently discovered by Golovin et al. [15] as well as Levin and Segev [19].

2 k-Hurdle Multiway Cut

Let $t_1 \dots t_r$ denote a set of terminals, and let P denote the set of all terminal-to-terminal paths. We can write the NP-hard k-hurdle multiway cut problem as the following integer program,

$$\begin{array}{lrl} & OPT = \text{Minimize} & \sum_{e \in E} x(e)c(e) \\ \text{(IP2)} & \text{Subject to} & x(p) \geq k \qquad \forall p \in P \\ & & x(e) \in \{0,1\} \; \forall e \in E, \end{array}$$

whose corresponding LP relaxation we denote by $LP2$.

2.1 Sacrificing One Hurdle

We begin our study of the k-hurdle multiway cut problem by showing the following pseudo-approximation result.

Theorem 1. *In polynomial time, one can obtain a $(k-1)$-hurdle solution of cost at most OPT, where OPT denotes the cost of an optimal solution with k hurdles.*

Let x be an optimal solution to $LP2$ of cost z_{LP2}, and let $d_x(u,v)$ denote the shortest path distance from u to v using edge lengths x. Choose α uniformly at random from $[0,1]$ and consider making concentric cuts around each terminal as follows. We define $E_{i,\rho}$ as the set of edges $uv \in E$ such that $\rho \in [d_x(t_i,u), d_x(t_i,v)]$ and let E_i denote the union of $E_{i,\rho}$ over all $\rho \in \{\alpha, 1+\alpha, 2+\alpha, \ldots\} \cap [0, \lfloor k/2 \rfloor]$. The "cutset" E_i contains the edges chosen by t_i for inclusion in our cut. Visually, we think of E_i as defined by a set of concentric rings emanating out from t_i at distances $\alpha, 1+\alpha$, and so on. Although these rings do not overlap, if k is even, an edge e straddling the frontier at mutual distance $k/2$ from two terminals t_i and t_j could be included in both E_i and E_j; otherwise, e will belong to at most one cutset E_i. We now show that $S = \cup_{i=1}^{r} E_i$ is a $(k-1)$-hurdle multiway cut whose cost is at most $z_{LP2} \le OPT$.

Lemma 1. $\mathbf{E}[c(S)] \le z_{LP2}$.

Proof. Note that $\mathbf{Pr}[e \in S] \le x_e$, since due to the triangle inequality, the range of values of α that result in $e \in S$ has size at most x_e. This is true even in the special case where e belongs to two different cutsets E_i and E_j. Therefore, $\mathbf{E}[c(S)] = \sum_e c(e)\mathbf{Pr}[e \in S] \le \sum_e c(e)x(e) = z_{LP2}$.

Lemma 2. *The set S is a $(k-1)$-hurdle multiway cut.*

Proof. Consider any path $p \in P$ connecting some terminal t_i to some other terminal t_j. Let $p_i = p \cap E_i$ and $p_j = p \cap E_j$. Suppose first that k is odd, in which case $|p_i| \ge (k-1)/2$, $|p_j| \ge (k-1)/2$, and $p_i \cap p_j = \emptyset$, from which it follows that $|p \cap S| \ge k-1$. On the other hand, if k is even, then $|p_i| \ge k/2$ and $|p_j| \ge k/2$ but one edge e in p might appear in $p_i \cap p_j$, so again $|p \cap S| \ge k-1$.

A slight variation on the argument above allows us to prove that $LP2$ is 1/2-integral, thereby giving an immediate 2-approximation algorithm for the k-hurdle multiway cut problem, and also (by appropriately generalizing the methods in [13]) a $(2-\frac{1}{r})$-approximation.

Theorem 2. *$LP2$ is 1/2-integral.*

Proof. Let x be an optimal fractional solution to $LP2$. Select $\alpha \in [0,1/2]$ uniformly at random. For each terminal i, construct k sets of edges $E_{i,\rho}$ as above for $\rho \in \{\alpha, 1-\alpha, 1+\alpha, 2-\alpha, 2+\alpha, 3-\alpha, \ldots\} \cap [0, k/2]$. Set $x^*(e)$ to half the total number of sets $E_{i,\rho}$ in which edge e appears. We claim that the 1/2-integral solution x^* is optimal for $LP2$. To show that x^* is feasible for

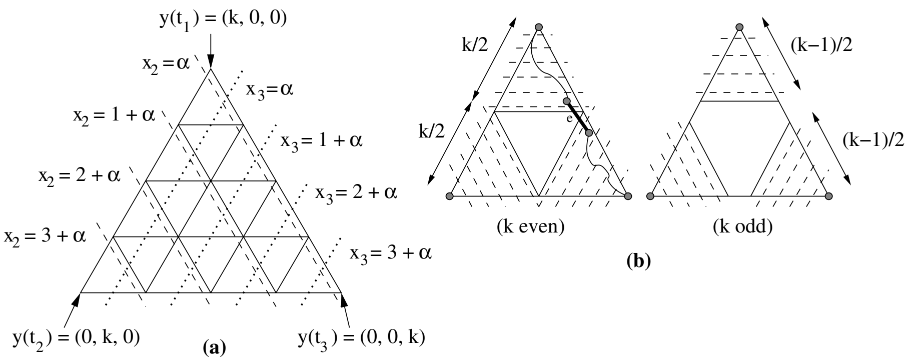


Fig. 1. Illustrations of (a) the cutting scheme used to obtain a $2(1-1/r)$-approximation (with $k = 4$), and (b) the cutting scheme used by our pseudo-approximation algorithm

$LP2$, note that $x^*(e) \leq 1$ for every edge e (even in the special case where e straddles the frontier at distance $k/2$ between two terminals), and also that $x^*(p) \geq k$ for any $p \in P$ connecting terminal t_i to t_j, since each of the $2k$ total sets $E_{i,\rho}$ and $E_{j,\rho}$ contributes $1/2$. By linearity of expectation, we have $\mathbf{E}[x^*(e)] \leq x(e)$ for each edge e, so letting C denote the cost of the solution x^*, we have $\mathbf{E}[C] = \mathbf{E}[\sum_e c(e)x^*(e)] \leq \sum_e c(e)x(e) = z_{LP2}$. However, since $C \geq z_{LP2}$ always holds, we conclude that $C = z_{LP2}$ irrespective of α.

We remark that if we replace edges with vertices, this same proof also establishes 1/2-integrality for the analogous LP relaxation of the *vertex* k-hurdle multiway cut problem. Garg et al. [13] have previously proved 1/2-integrality for the vertex version of the standard multiway cut problem, so our argument above gives not only gives a simpler alternative proof of this result, but it also provides a generalization to multiple layers of hurdles.

2.2 A $2(1-1/r)$-Approximation Algorithm Via Simplex Embeddings

The currently strongest approximation algorithms for the classical multiway cut problem are based on a stronger linear programming relaxation that serves to embed a graph G into the r-simplex, after which a randomized cutting scheme slices up the simplex using *side-parallel cuts* (hyperplane cuts parallel to the faces of the simplex), thereby inducing a multiway cut in G. In this section, we show how to generalize this approach to obtain a $2(1-1/r)$-approximation algorithm for the k-hurdle multiway cut problem. In addition, we show the (perhaps even more interesting) result that the integrality gap of the embedding increases to $2(1-1/r)$ in the k-hurdle case, thereby matching our approximation bound. One therefore cannot hope to achieve a stronger result by generalizing the stronger multiway cut approximation algorithms of [5,6,17] to the k-hurdle case, since all of these use the same linear program.

Let u_i denote the ith unit vector in $\mathbf{R}^r$ scaled up by k (with coordinate $x_i = k$, and all others zero), and let $\Delta_r = conv(u_1 \ldots u_r) = \{x \in \mathbf{R}^r : (x \geq 0) \wedge (\sum_{i=1}^r x_i = 1)\}$ denote the r-simplex. We wish to compute an embedding $y : V \to \Delta_r$ of minimum *volume* $Vol(y) = \frac{1}{2}\sum_{uv\in E} c(uv)||y(u) - y(v)||_1$ such that $y(t_i) = u_i$ for each terminal $i \in \{1, \ldots, r\}$, and such that $\frac{1}{2}||y(u) - y(v)||_1 \leq 1$ for all edges $uv \in E$. Note that the distance between two embedded vertices $y(u)$ and $y(v)$ is measured as $\frac{1}{2}||y(u) - y(v)||_1$, since the L_1 norm turns out to be the "natural" norm to use for this embedding, and scaling by 1/2 makes the corners of the simplex all lie at distance k from each-other. This geometric relaxation is stronger than $LP2$, since by setting $x(uv) = \frac{1}{2}||y(u) - y(v)||_1$, we can transform y into a feasible solution x for $LP2$ of cost $Vol(y)$. As shown in [5,6,17], an optimal embedding y can be computed by solving a simple linear program. Moreover, by appropriately subdividing edges in the embedding, one can assume each edge uv is *i-j aligned*, for some pair of terminals t_i and t_j; that is, $y(u)$ and $y(v)$ differ only in their ith and jth coordinates.

A valid *k-hurdle cut of the simplex* is a collection of surfaces in $\mathbf{R}^r$ such that (i) any continuous (possibly curved) path p through Δ_r from any u_i to any other u_j must cross at least k surfaces, and (ii) the crossing points must each be at least 1 unit apart, measuring the distance between any two points x and x' by $\frac{1}{2}||x - x'||_1$. Any such cut induces a feasible k-hurdle cut in G (by cutting every edge e crossing a surface), since any terminal-to-terminal path in G must include at least k edges in the cut. Condition (ii) above is particularly noteworthy, as it is not necessary for the standard multiway cut problem — without it, we could have a geometric cut comprised of k surfaces, but this might not cut as many as k edges along a path in G since a single embedded edge might cross two or more surfaces. It is this condition that will prevent us from obtaining an approximation bound stronger than $2(1-1/r)$. Let $C(i, \rho)$ denote the hyperplane $\{x \in \mathbf{R}^r : x_i = \rho\}$. We call $C(i, \rho)$ a side-parallel cut, since it runs parallel to the face of Δ_r opposite u_i. We henceforth focus only on side-parallel cuts.

A *cutting scheme*, is a probability distribution over k-hurdle cuts of the simplex. Consider the following such distribution D: pick $\alpha \in [0, 1]$ uniformly at random and select one terminal t_j uniformly at random to remove from consideration. For each of the $r - 1$ remaining terminals t_i, we include the family of k concentric side-parallel cuts $C(i, \alpha), C(i, 1 + \alpha), \ldots, C(i, k - 1 + \alpha)$. Figure 1(a) illustrates a the set of side-parallel cuts comprising one such cut from D. In the literature, a cutting scheme comprised of side-parallel cuts is sometimes known as a *SPARC*.

Theorem 3. *The cutting scheme D gives us a $2(1 - 1/r)$-approximation algorithm for the k-hurdle multiway cut problem.*

Proof. The expected cost of a k-hurdle multiway cut produced by D is

$$\sum_{uv\in E} c(uv)\mathbf{Pr}[uv \text{ cut}] = \left(1 - \frac{1}{r}\right)\sum_{uv\in E} c(uv)||y(u) - y(v)||_1 = 2\left(1 - \frac{1}{r}\right)Vol(y),$$

where the first equality uses Lemma 3. Since $Vol(y) \leq OPT$, the approximation bound follows.

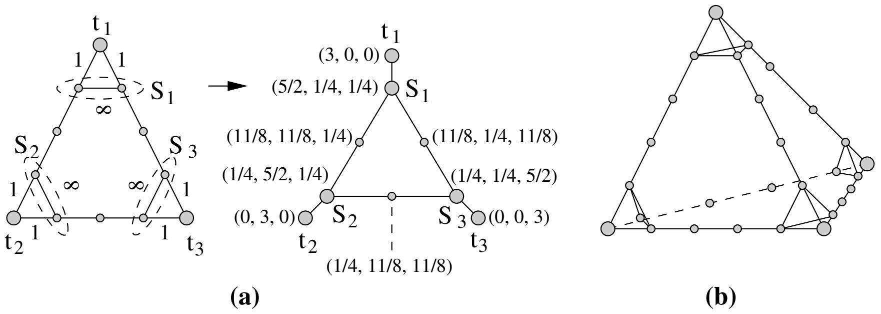


Fig. 2. Graphs illustrating the integrality gap of $2(1 - 1/r)$ in Theorem 4: (a) $G_{3,3}$, along with its simplex embedding, and (b) $G_{4,4}$

Lemma 3. *For any edge uv,* $\mathbf{Pr}[uv \text{ cut by } D] = (1 - 1/r)||y(u) - y(v)||_1$.

Proof. Let $x = y(u)$ and $x' = y(v)$. Assume without loss of generality that uv is i-j aligned. Only cuts $C(i, \cdot)$ or $C(j, \cdot)$ can potentially cut uv (the others are parallel to uv). The family of cuts $C(i, \cdot)$ will cut uv with total probability $(1 - 1/r)|x'_i - x_i|$ (this includes the probability that terminal i is used in D). Similarly, the probability uv is cut by the family $C(j, \cdot)$ is $(1 - 1/r)|x_j - x'_j|$. Applying a union bound, $\mathbf{Pr}[uv \text{ cut by } D] = (1 - 1/r)(|x'_i - x_i| + |x'_j - x_j|) = (1 - 1/r)||x - x'||_1$. Note that $x_k = x'_k$ for all $k \neq i, j$.

We note that our pseudo-approximation result from the preceding section is also obtainable via a simple randomized cutting scheme, illustrated in Figure 1(b): pick $\alpha \in [0, 1]$ uniformly at random, and include cuts $C(i, \alpha), C(i, 1 + \alpha), \ldots, C(i, \lfloor k/2 \rfloor - 1 + \alpha)$ for each terminal t_i. A similar argument to that of Lemma 3 shows that the expected cost of this cutting scheme is at most $Vol(y) \leq OPT$; however, it does not guarantee k hurdles along any t_i-t_j path. In the case that k is even, any such path p does indeed cross k cuts, but an edge in the middle of the path might cross two cuts, as shown in the figure. If k is odd, every path p only crosses $k - 1$ cuts; this is analogous to the proof of Lemma 2.

Theorem 4. *For $k \geq 2$ and $r \geq 3$, the integrality gap of our simplex embedding is precisely $2(1 - 1/r)$.*

Proof. Our approximation algorithm above provides an upper bound of $2(1 - 1/r)$, so we focus on the matching lower bound. For any $k \geq 2$ and any $r \geq 3$, we construct a graph G_{kr} having integrality gap $2(1-1/r)$ as follows. Start with an r-clique whose vertices are the r terminals $t_1 \ldots t_r$. Then subdivide each edge into $k + 1$ individual edges. Let S_i denote the set of vertices adjacent to t_i. For each $i = 1 \ldots r$, we form a clique of infinite-cost edges on the vertices in S_i. Set the cost of each remaining edge to zero, except the $r(r-1)$ edges incident to the terminals, which have unit cost. Figure 2 shows an example of $G_{3,3}$ and $G_{4,4}$.

We achieve an optimal integral k-hurdle multiway cut in G_{kr} by setting with $x_e = 1$ for all zero-cost and unit-cost edges e except the unit-cost edges adjacent to a single terminal, chosen arbitrarily. If there were two terminals t_i and t_j incident to edges not selected in our solution, then a path from t_i to t_j containing fewer than k selected edges would exist. Therefore, $OPT = (r-1)^2$. When we embed G_{kr} in a simplex, the cliques $S_1 \ldots S_r$ collapse into single points, the embedded length of each unit-cost edge becomes $1/2$ (again measured according to half the L_1 norm), and the zero-cost edges end up with unit length. Since the total volume of this embedding is $r(r-1)/2$, we obtain an integrality gap of $2(1-1/r)$.

3 *k*-Hurdle Multicut

As we have mentioned previously, the k-hurdle multicut problem seems somewhat more difficult to approximate than the classical multicut problem, particularly if we seek a solution that obtains all the required hurdles. In this section, we describe a pseudo-approximation algorithm based on the prominent "region-growing" algorithm of Garg et al. [11] that, for any constant $\varepsilon > 0$, outputs a solution providing a $1-\varepsilon$ fraction (rounded up) of the required hurdles for each commodity, whose cost is at most $O(\log n)$ times that of an optimal k-hurdle solution. After this, we describe a 2-approximation algorithm for the special case of trees. We begin with the integer programming formulation of the k-hurdle multicut problem,

$$\text{(IP3)} \qquad \begin{array}{rl} OPT = \text{Minimize} & \sum_{e \in E} x(e)c(e) \\ \text{Subject to} & x(p) \geq k_i \quad \forall i \in \{1,\ldots,r\}, p \in P_i \\ & x(e) \in \{0,1\} \; \forall e \in E, \end{array}$$

whose LP relaxation we denote by $LP3$. Our pseudo-approximation algorithm is as follows:

1. Solve $LP3$ to obtain an optimal fractional solution x (this can be done in polynomial time with the ellipsoid algorithm).
2. Set $\delta = \varepsilon^2$ and let $\overline{x} = x/\delta$.
3. Generate a new instance I' by adding a source-sink pair (u, v) for each $u, v \in V$ such that $|d_{\overline{x}}(s_i, u) - d_{\overline{x}}(s_i, v)| \geq 1$ for some existing terminal pair (s_i, t_i).
4. Use the region-growing algorithm of Garg, Vazirani, and Yannakakis [11] to round $\overline{x}$ to an integer solution x' for instance I'. Return x'.

Note that $\overline{x}$ would be an optimal fractional solution for $LP3$ for the instance I' if only $\overline{x}_e \leq 1$ for all $e \in E$. However, the GVY region-growing algorithm is not adversely affected by the fact that $\overline{x}_e > 1$ for some edges e — it still returns an integer solution of cost at most $O(\log R)$ times the objective value of our LP, where R is the number of commodities in the instance. In our case, since $R \leq n^2$ for the instance I', we obtain a solution of cose at most $O(\log n)$ times $z_{LP3}/\delta = z_{LP3}/\varepsilon^2$).

Theorem 5. *The algorithm above returns an integer solution x' for which $x'(p) \geq \lceil (1-\varepsilon)k_i \rceil$ for every $i \in \{1, \ldots, r\}$ and $p \in P_i$.*

Proof. We use the following fact about the GVY region-growing algorithm: for each commodity (s_i, t_i), it makes a cut at some radial distance $\rho_i \leq 1$ from either s_i or t_i. Consider now any commodity $i \in \{1, \ldots, r\}$ and any path $p \in P_i$. As we walk along p from s_i to t_i, we acquire at least one hurdle for every $1 + 1/\delta$ units of distance traveled. More precisely, we show how to obtain at least $\lceil (\overline{x}(p) - 1)/(1 + 1/\delta) \rceil$ hurdles, which is at least $(1-\varepsilon)k_i$ since $\overline{x}(p) \geq k_i/\delta$ and $\delta = \varepsilon^2$.

Define $q = \lceil \overline{x}(p)/(1 + 1/\delta) \rceil$. Let $v_0 = s_i$ and $v_q = t_i$, and define v_i for $i \in \{1, \ldots, q-1\}$ as the farthest vertex along p from s_i such s_i and v are no more than $q(1 + 1/\delta)$ units of distance apart on p. Let p_i, $i = 1 \ldots q$ denote the subpath of p from v_{i-1} up to v_i. We claim that $x'(p_i) \geq 1$ for each $i \in \{1 \ldots q-1\}$, and that $x'(p_q) \geq 1$ if $\overline{x}(p_q) \geq 1$. Consider any $i \in \{1, \ldots, q-1\}$. Since $\overline{x}(e) \leq 1/\delta$ for all $e \in E$, we must have $\overline{x}(p_i) \geq 1$; otherwise, the first edge in p_{i+1} would rightly belong to the end of p_i, since the length of p_i in this case would still be at most $1 + 1/\delta$. Finally, if $\overline{x}(p_i) \geq 1$, then p_i must be cut by the GVY algorithm, since a radial cut at distance ≤ 1 will be made from at least one endpoint of p_i.

By setting ε appropriately, we obtain a true $O(\log n)$-approximation algorithm for the special case where $k_i = O(1)$ for all commodities i. Since in practice we often wish to set up only two or three redundant layers of "checkpoints" for inspecting traffic through a network, we expect this special case to occur quite frequently.

3.1 A 2-Approximation for Trees

We now focus on the special case of a tree. We note with some interest that the well-known primal-dual algorithm of Garg et al. [12] does not seem to generalize in a straightforward fashion to the k-hurdle case, especially for the "non-uniform" case where k_i can vary by commodity. For the uniform case where all k_i are equal, the primal-dual approach does lead to a 3-approximation bound. However, by generalizing a more recent approach of Golovin et al. [15] and Levin and Segev [19], we show how to obtain 2-approximation algorithm.

Consider the integer programming formulation of the k-hurdle multicut problem in a tree,

$$\text{(IP4)} \qquad \begin{aligned} OPT = \text{Minimize} & \sum_{e \in E} x(e)c(e) \\ \text{Subject to } & x(P_i) \geq k_i \quad \forall i, 1 \leq i \leq r \\ & x(e) \in \{0,1\} \; \forall e \in E, \end{aligned}$$

whose LP relaxation we denote by $LP4$. Our approach is based on the following key property, mentioned in [15,19].

Lemma 4. *If each commodity (s_i, t_i) is* unidirectional *(s_i and t_i having an ancestor-descendant relationship), then $LP4$ is totally unimodular.*

We can therefore compute an integer optimal solution to $LP4$ in polynomial time if all commodities are unidirectional. In fact, we can compute such a solution in strongly polynomial time, since the dual of $LP4$ can be stated as a minimum-cost flow problem (we will include more detail on this issue in the full version of this paper). In the unit cost case, we can even use a simple combinatorial greedy algorithm in lieu of solving $LP4$ [2]: root the tree and perform a postorder scan over its edges, setting $x(e) = 1$ if $|P_i \cap P_e| - x(P_i \cap P_e) = k_i - x(P_i)$ for any commodity i (here, P_e denotes the path from e up to the root).

Let x be an optimal solution to $LP4$ of cost z_{LP4}. Although x may in general be non-integral, we can use x to construct a new instance of $LP4$, NLP, whose commodities are all unidirectional, and whose optimal (integral) solution gives us a 2-approximate solution to IP. For each commodity i that is not unidirectional, let u_i be the lowest common ancestor of s_i and t_i, and replace i with two commodities i' and i'', having source-sink pairs $(s_{i'}, t_{i'}) = (s_i, u_i)$ and $(s_{i''}, t_{i''}) = (u_i, t_i)$ and hurdle demands $k_{i'} = round(x(P_{i'}))$ and $k_{i''} = round(x(P_{i''}))$ The function $round(x)$ evaluates to $\lceil x \rceil$ if the fractional part of x is at least 0.5, and $\lfloor x \rfloor$ otherwise. This approach can be viewed as a strict generalization of [15,19] for the simpler unit hurdle case of $k_i = 1$, where i' and i'' are each included in NLP only if $x(P_{i'}) \geq 0.5$ or $x(P_{i''}) \geq 0.5$, respectively.

Consider now an optimal integer-valued solution x^* to NLP. We know that x^* is feasible for IP since $x^*(P_i) = x^*(P_{i'}) + x^*(P_{i''}) \geq k_{i'} + k_{i''} = round(x(P_{i'})) + round(x(P_{i''})) \geq \lfloor x(P_i) \rfloor \geq k_i$ for each original commodity i. We now only need to show that the cost of x^* is at most $2OPT$. In [15,19], this is easily achieved since $LP4$ in the unit hurdle case does require constraints of the form $x(e) \leq 1$, so $2x$ is a feasible solution for NLP, and therefore $z_{NLP} \leq 2z_{LP4} \leq 2OPT$. In our case, however, $2x$ may not be feasible for NLP, since doubling any $x(e) > 0.5$ will violate the constraint that $x(e) \leq 1$. However, by first truncating $2x$, we can show an analogous result.

Lemma 5. *The solution* $\overline{x} = \min(2x, \mathbf{1})$ *is feasible for* NLP.

Proof. Consider a particular unidirectional commodity j in NLP (so $j = i'$ or $j = i''$ for some commodity i in the original instance). We will show that $\overline{x}(P_j) \geq k_j$. Note that k_j was obtained by rounding $x(P_j)$ up or down. If k_j was obtained by rounding $x(P_j)$ down, then clearly $\overline{x}(P_j) \geq x(P_j) \geq k_j$. Henceforth, let us therefore assume k_j was obtained by rounding $x(P_j)$ up, which implies that $\lceil x(P_j) \rceil - x(P_j) \leq 0.5$. Let L_j denote the set of "large" edges $e \in P_j$ with $x(e) \geq 0.5$ and let S_j denote the "small" edges $e \in P_j$ with $x(e) < 0.5$. We can express $\overline{x}(P_j) = |L_j| + 2x(S_j)$. If $x(S_j) = 0$, then $\overline{x}(P_j) = |L_j| \geq \lceil x(L_j) \rceil = \lceil x(P_j) \rceil = k_j$. Therefore, we assume $x(S_j) > 0$ and consider two cases:

1. $\lceil x(S_j) \rceil - x(S_j) \leq 0.5$. Here, since $x(S_j) \geq \lceil x(S_j) \rceil - 0.5$ and $x(S_j) \geq 0.5$, we have $2x(S_j) \geq \lceil x(S_j) \rceil$. Therefore, $\overline{x}(P_j) = |L_j| + 2x(S_j) \geq |L_j| + \lceil x(S_j) \rceil = \lceil |L_j| + x(S_j) \rceil \geq \lceil x(L_j) + x(S_j) \rceil = \lceil x(P_j) \rceil = k_j$.
2. $\lceil x(S_j) \rceil - x(S_j) > 0.5$. By expanding out our assumption that $\lceil x(P_j) \rceil - x(P_j) \leq 0.5$, we obtain $0.5 \geq \lceil x(S_j) + x(L_j) \rceil - (x(S_j) + x(L_j))$, which implies that $x(L_j) > \lceil x(S_j) + x(L_j) \rceil - \lceil x(S_j) \rceil$. Since $|L_j| \geq x(L_j) >$

$\lceil x(S_j) + x(L_j) \rceil - \lceil x(S_j) \rceil$ and $|L_j|$ is an integer, we have $|L_j| \geq \lceil x(S_j) + x(L_j) \rceil - \lfloor x(S_j) \rfloor$. Therefore, $\overline{x}(P_j) = |L_j| + 2x(S_j) \geq |L_j| + \lfloor x(S_j) \rfloor \geq \lceil x(S_j) + x(L_j) \rceil = \lceil x(P_j) \rceil = k_j$.

Since $\overline{x}$ is feasible for NLP and its cost is at most $2z_{LP4}$, we have $z_{NLP} \leq 2z_{LP4} \leq 2OPT$, so our integer optimal solution x^* to NLP is a 2-approximation.

We close with one final note. 2-approximation algorithms for multicut on a tree are often phrased as 2-approximation algorithms for the set cover problem for the special case that we have a "tree representable" set system. Our result above admits a similar interpretation:

Theorem 6. *There exists a 2-approximation algorithm for the general covering integer program* $\min\{c^T x : Ax \geq b\}$ *in the special case where* A *encodes a "tree representable" set system.*

4 Concluding Remarks and Open Problems

The primary open question remaining is whether one can develop stronger approximation bounds for the k-hurdle multicut problem. Can one obtain a true approximation algorithm for non-constant k_i? Can one approximate an r-commodity instance and obtain an approximation bound in terms of r rather than n? Another interesting question one can address is whether a good approximation result can also be obtained for the k-hurdle version of the multi-multiway cut problem [1].

References

1. Avidor, A., Langberg, M.: The multi-multiway cut problem. In: Hagerup, T., Katajainen, J. (eds.) SWAT 2004. LNCS, vol. 3111, pp. 273–284. Springer, Heidelberg (2004)
2. Barany, I., Edmonds, J., Wolsey, L.A.: Packing and covering a tree by subtrees. Combinatorica 6(3), 221–233 (1986)
3. Burch, C., Carr, R., Krumke, S., Marathe, M., Phillips, C.: A decomposition-based pseudoapproximation algorithm for network flow inhibition. In: Woodruff, D.L. (ed.) Network Interdiction and Stochastic Integer Programming, pp. 51–68. Kluwer Academic Press, Dordrecht (2003)
4. Burch, C., Krumke, S., Marathe, M., Phillips, C., Sundberg, E.: Multicriteria approximation through decomposition, Technical Report (1997)
5. Călinescu, G., Karloff, H.J., Rabani, Y.: An improved approximation algorithm for MULTIWAY CUT. Journal of Computer and System Sciences 60(3), 564–574 (2000)
6. Cheung, K., Cunningham, W.H., Tang, L.: Optimal 3-terminal cuts and linear programming. Mathematical Programming 106(1) (2006)
7. Cunningham, W.H.: Optimal attack and reinforcement of a network. Journal of the ACM 32(3), 549–561 (1985)
8. Dahlhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., Yannakakis, M.: The complexity of multiterminal cuts. SIAM Journal on Computing 23, 864–894 (1994)

9. Ford, L.R., Fulkerson, D.R.: Flows in Networks. Princeton University Press, Princeton (1962)
10. Fulkerson, D.R., Harding, G.C.: Maximizing the minimum source-sink path subject to a budget constraint. Mathematical Programming 13, 116–118 (1977)
11. Garg, N., Vazirani, V.V., Yannakakis, M.: Approximate max-flow min-(multi)cut theorems and their applications. SIAM Journal on Computing 25(2), 235–251 (1996)
12. Garg, N., Vazirani, V.V., Yannakakis, M.: Primal-dual approximation algorithms for integral flow and multicut in trees. Algorithmica 18(1), 3–20 (1997)
13. Garg, N., Vazirani, V.V., Yannakakis, M.: Multiway cuts in node weighted graphs. Journal of Algorithms 50(1), 49–61 (2004)
14. Goldberg, A., Tarjan, R.: Finding minimum-cost circulations by successive approximation. Mathematics of Operations Research 15, 430–466 (1990)
15. Golovin, D., Nagarajan, V., Singh, M.: Approximation the k-multicut problem. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 621–630 (2006)
16. Israeli, E., Wood, R.K.: Shortest path network interdiction. Networks 40(2), 97–111 (2002)
17. Karger, D.R., Klein, P.N., Stein, C., Thorup, M., Young, N.E.: Rounding algorithms for a geometric embedding of minimum multiway cut. Mathematics of Operations Research 29(3), 436–461 (2004)
18. Krumke, S., Noltemeier, H., Ravi, R., Schwarz, S., Wirth, H.-C.: Flow improvement and flows with fixed costs. In: Proceedings of the International Conference on Operations Research (OR), pp. 158–167 (1998)
19. Levin, A., Segev, D.: Partial multicuts in trees. In: Erlebach, T., Persinao, G. (eds.) WAOA 2005. LNCS, vol. 3879, pp. 320–333. Springer, Heidelberg (2006)
20. Meignen, R., Berthoud, G., Schwarz, S., Krumke, S.: On budget-constrained flow improvement. Information Processing Letters 66(3), 291–297 (1998)
21. Nishihara, O., Inoue, K.: An algorithm for a multiple disconnecting set problem. Unpublished manuscript, Department of Aeronautical Engineering, Kyoto University (1988)
22. Nishihara, O., Kumamoto, H., Inoue, K.: An algorithm for a multiple cut problem and its application. In: 13th International Symposium on Mathematical Programming (1988)
23. Phillips, C., Swiler, L.P.: A graph-based system for network-vulnerability analysis. In: Proceedings of the DARPA Information Survivability Conference and Exposition, pp. 71–79 (2000)
24. Phillips, C.A.: The network inhibition problem. In: Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC), pp. 776–785 (1993)
25. Wagner, D.K.: Disjoint (s, t)-cuts in a network. Networks 20, 361–371 (1990)
26. Wagner, D.K., Wan, H.: A polynomial-time simplex method for the maximum k-flow problem. Mathematical Programming 30, 115–123 (1993)
27. Wood, R.K.: Deterministic network interdiction. Mathematical and Computer Modeling 17(2), 1–18 (1993)

Approximating Crossing Minimization in Radial Layouts*

Seok-Hee Hong[1] and Hiroshi Nagamochi[2]

[1] School of Information Technologies, University of Sydney
shhong@it.usyd.edu.au
[2] Department of Applied Mathematics and Physics,
Kyoto University
nag@amp.i.kyoto-u.ac.jp

Abstract. We study a crossing minimization problem of drawing a bipartite graph with a radial layout of two orbits. Radial layouts have strong application in social network visualization, displaying *centrality* of actors. The problem is called the one-sided crossing minimization if the positions of vertices in one of the two orbits are fixed, and is known to be NP-hard. We present the *first approximation* algorithm, proving that the one-sided crossing minimization in a radial layout is 15-approximable.

1 Introduction

Given a bipartite graph $G = (V, W, E)$ with two parallel straight lines L_1 and L_2, a 2-layered drawing consists of placing all vertices in the first vertex set V on L_1 and placing all vertices in the second vertex set W on L_2. In a 2-layered drawing, each edge is represented as a straight-line segment joining the end-vertices, as shown in Fig. 1(a). For a given ordering of vertices in W on L_2, the one-sided crossing minimization problem asks to find an ordering of vertices in V on L_1 so that the number of edge crossings between two layers is minimized, whereas the two-sided version asks to find orderings of W on L_2 and V on L_1 to minimize the number of edge crossings.

In a radial drawing of a bipartite graph, we consider orbits instead of horizontal lines. Given two orbits O_1 and O_2 with the common center in the plane, we draw a bipartite graph $G = (V, W, E)$ so that the vertices in V (resp., W) are placed on O_1 (resp., O_2) and each edge is drawn as a simple curve in the area between O_1 and O_2 (see Fig. 1(b)).

In horizontal 2-layered drawings, the embedding is fully determined by the vertex orderings of V and W. For radial drawings, however, it is also necessary to know where the orderings start (without loss of generality, counter clockwise) and end on each orbit. For this, we introduce a *ray* that indicates this borderline between the first and last vertices in the orderings. The ray is a straight-line segment that intersects each orbit exactly once, as shown in Fig. 1(b). An edge $e \in E$ is called a *cut edge* in a drawing if it intersects the ray of the drawing.

* This is an extended abstract. This research was partially supported by the Scientific Grant-in-Aid from Ministry of Education, Culture, Sports, Science and Technology of Japan.

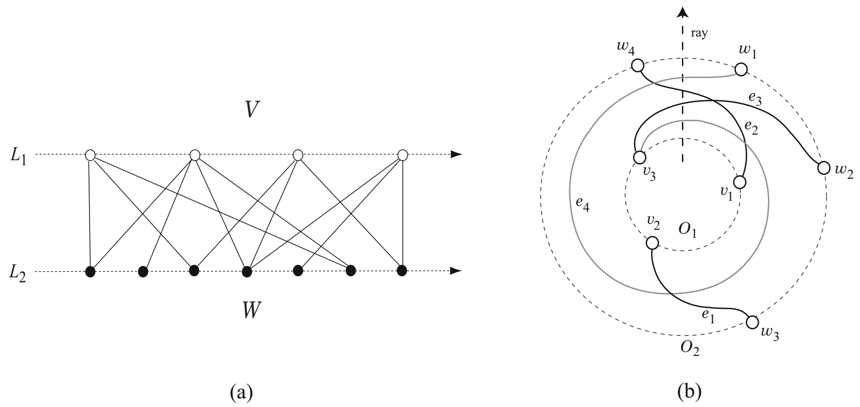


Fig. 1. (a) A horizontal drawing $D = (\pi, \sigma)$; (b) A radial drawing $D = (\pi, \sigma, \psi)$ with two orbits O_1 and O_2, where $\psi(e_1) = 0$, $\psi(e_2) = 1$, $\psi(e_3) = -1$, and $\psi(e_4) = -2$

Both 2-layered horizontal and radial drawings are used in VLSI layout design, and the fundamental building block for drawing graphs in a hierarchical fashion [1,4,8]. In particular, radial layout is one of popular drawing convention in social network visualization for displaying *centralities*. Centrality measures are one of the fundamental concepts in social network analysis to identify important actors in the social network [9]. In a radial drawing of a social network, the most prominent actor is placed in the center, and then other actors are embedded on the concentric circles, based on their centrality values.

Recently, human experiments suggested that edge crossing is one of the most important aesthetics criteria for understanding of a graph drawing. Further, in some applications such as VLSI layout design, layouts with less edge crossings can lead to a product with less cost and higher reliability. As a result, there is a rich literature on minimizing edge crossings of layouts in Graph Drawing.

Note that the crossing minimization problem in horizontal 2-layered drawings are well studied. Both two-sided and one-sided crossing minimization problems in horizontal layouts are NP-hard [2]. For the one-sided crossing minimization problem, a number of heuristics, approximation algorithms and exact algorithms have been proposed [2,3,5,8]. A well known lower bound $LB(G, \sigma)$ on the minimum number of crossings is obtained by summing up $\min\{\chi_{uv}, \chi_{vu}\}$ over all vertex pairs $u, v \in V$, where χ_{uv} denotes the number of crossings generated by edges incident to u and v when u precedes v in the ordering. A *median heuristic* produces a 3-approximate solution based on the lower bound $LB(G, \sigma)$. Currently, the best known algorithm for the problem delivers a drawing with at most $1.4664LB(G, \sigma)$ crossings [5]. For the two-sided crossing minimization, it is shown that the problem is $O(\log n)$-approximable, when the maximum degree over the minimum degree is bounded by a constant [7].

However, the crossing minimization problem in radial layouts has not been well studied. In this paper, we study the problem of finding a radial drawing D of a bipartite graph $G = (V, W, E)$ that minimizes the number of edge crossings in D, when a vertex ordering σ of W is fixed. We call this problem *the one-sided crossing minimization problem in a radial layout*. Recently, this problem was shown to be NP-hard, and a number of heuristics are presented [1]. However, it was left open to obtain a constant-factor approximation algorithm for radial layouts [1]. There is no approximation algorithm with a theoretical performance guarantee for the crossing minimization in radial layouts.

Note that the problem of crossing minimization in radial layout is more challenging, as it involves both vertex ordering and edge routing problems. That is, even if the orderings of vertices in both orbits are fixed, we still need to decide how to route (i.e. clockwise or counter clockwise) each edge around the inner orbit in order to minimize the number of edge crossings in a radial layout. It would be the fundamental observation to know whether the freedom in drawing edges around the inner orbit makes the one-sided crossing minimization hard to be approximated or not. The one-sided crossing minimization problem in horizontal layout admits the lower bound $LB(G, \sigma)$. However, no such effective lower bound is known to the one-sided crossing minimization problem in radial layouts.

The main contribution of this paper is to provide the first *constant-factor* approximation algorithm to the one-sided crossing minimization in a radial layout. More specifically, we prove the following.

Theorem 1. *The one-sided crossing minimization in a radial layout is 15-approximable.* □

Our main idea is to reduce a given instance of the one-sided crossing minimization in a radial layout to that of the one-sided crossing minimization in a horizontal layout. We present a new technique to convert a problem instance in a radial layout into that in a horizontal layout by introducing a bounded number of new edge crossings. This enables us to use the well-known lower bound on the one-sided crossing minimization in horizontal layouts and to derive a 15-approximation algorithm to the one-sided crossing minimization in a radial layout.

In Section 2, we formally define a horizontal layout and a radial layout of a bipartite graph, and review basic properties. Section 3 investigates a property of radial drawings in which at least one edge receives no edge crossings, and proves that among such drawings, there is a 3-approximate solution to the one-sided crossing minimization problem in a radial layout. Section 4 proves that the problem of finding an optimal radial drawing with a crossing-free edge is 5-approximable, by reducing the problem to the one-sided crossing minimization problem in a horizontal layout. The results in Sections 3 and 4 imply Theorem 1. Section 5 makes some concluding remarks.

2 Preliminaries

Let $G = (V, W, E)$ be a simple bipartite graph with vertex sets V and W and an edge set E. For a vertex $v \in V$ in G, let $E(v; G)$ denote the set of edges incident to v, $\Gamma(v;G)$ denote the set of *neighbors* of v (i.e., vertices adjacent to v) in G, and $d(v;G)$ denote the degree of a vertex v, i.e., $d(v;G) = |E(v;G)| = |\Gamma(v;G)|$. A subgraph $G' = (V', E')$ of $G = (V, E)$ is *induced* by V' if E' is given by $E' = \{e \in E \mid \text{the both end vertices of } e \text{ belong to } V'\}$, and G' may be denoted by $G[V']$. Throughout the paper, we assume that $d(u;G) \geq 1$ for all $u \in V \cup W$.

2.1 Horizontal Layouts

Let π and σ denote permutations of $\{1, 2, \ldots, |V|\}$ and $\{1, 2, \ldots, |W|\}$, respectively. A pair of π and σ defines a horizontal drawing of G in the plane in such a way that, for two parallel horizontal lines L_1 and L_2, the vertices in V (resp., in W) are arranged on L_1 (resp., L_2) according to π (resp., σ) and each edge is displayed by a straight line segment joining the end-vertices. For any choice of coordinates of points for vertices in $V \cup W$ in a horizontal drawing of G defined by (π, σ), two edges $(v, w), (v', w') \in E$ intersect properly (or create a *crossing*) if and only if $(\pi(v) - \pi(v'))(\sigma(w) - \sigma(w'))$ is negative.

Given a bipartite graph $G = (V, W, E)$ and a permutation σ on W, the *one-sided crossing minimization* problem asks to find a permutation π on V that minimizes the number of crossings in a horizontal drawing (π, σ) of G.

For an ordered pair of two vertices $u, v \in V$, let $\chi(u, v; G)$ denote the number of crossings generated by an edge incident to u and an edge incident to v when $\pi(u) < \pi(v)$ holds in a horizontal drawing $D = (\pi, \sigma)$. The total number $\chi(D;G)$ of edge crossings of $D = (\pi, \sigma)$ in G is given by

$$\chi(D;G) := \sum_{u,v \in V : \pi(u) < \pi(v)} \chi(u, v; G).$$

From this formula, we observe the next lower bound on the minimum $\chi(D;G)$.

Lemma 1. *Given a bipartite graph $G = (V, W, E)$ and a permutation σ on W, let $LB(G, \sigma) = \sum_{u,v \in V} \min\{\chi(u, v; G), \chi(v, u; G)\}$, and $\chi^*_h(G, \sigma) = \min\{\chi(D;G) \mid D = (\pi, \sigma)$, a permutation π on $V\}$. Then it holds $LB(G, \sigma) \leq \chi^*_h(G, \sigma)$* □

It is known that $\chi^*_h(G, \sigma) \leq 1.4664 LB(G, \sigma)$ always holds, and there is an instance (G, σ) with $\chi^*_h(G, \sigma) = 1.1818 LB(G, \sigma)$ [5].

Eades and Wormald [2] gave an algorithm, called the median heuristic, which produces a horizontal drawing $D = (\pi, \sigma)$ with $\chi(D;G) \leq 3LB(G, \sigma)$, i.e., a 3-approximate solution. Here we review the median heuristic with a slight modification. Let σ be a fixed permutation on W. We define the *median* $med(v)$ of a vertex in $v \in V$, where $d_v = d(v;G)$ and $\Gamma(v;G) = \{w_1, w_2, \ldots, w_{d_v}\}$ $(\sigma(w_1) < \sigma(w_2) < \cdots < \sigma(w_{d_v}))$, by

$$med(v) := \begin{cases} \sigma(w_{\frac{d_v+1}{2}}) & \text{if } d_v \text{ is odd,} \\ \frac{1}{2}(\sigma(w_{\frac{d_v}{2}}) + \sigma(w_{\frac{d_v}{2}+1})) & \text{if } d_v \text{ is positive and even,} \end{cases}$$

where $med(v)$ for a vertex v with $d_v = 0$ can take any value in $\{1, 2, \ldots, |V|\}$. Note that if $d_v \geq 2$ is even, then $med(v) \notin \{\sigma(w) \mid w \in \Gamma(v; G)\}$ ($med(v)$ was defined as $\sigma(w_{\frac{d_v}{2}})$ if d_v is even in [2]).

A permutation π has the *median property* with respect to σ if π is a total order obtained from med, i.e., $\pi(u) < \pi(v)$ implies $med(u) \leq med(v)$ for every two vertices $u, v \in V$. It is known that if π has the median property with respect to σ, then $\chi(D; G) \leq 3LB(G, \sigma)$ holds for $D = (\pi, \sigma)$ [2].

2.2 Radial Layouts

Let O_1 and O_2 be two orbits with the common center in the plane, where O_1 is the inner orbit and O_2 is the outer orbit. The positions of vertices in V (resp., W) is defined as a bijective function $\pi : V \to \{0, 1, \ldots, |V| - 1\}$, (resp., $\sigma : V \to \{0, 1, \ldots, |W| - 1\}$) where positions $0, 1, \ldots, |V| - 1$ (resp., $0, 1, \ldots, |W| - 1$) appear in this order when we traverse O_1 (resp., O_2). Each edge is drawn as a simple curve in the area between O_1 and O_2.

In horizontal drawings with two layers, a crossing between two edges only depends on the orderings of the end vertices. In radial drawings, however, it is also necessary to consider the direction in which the edges are wound around the inner orbit. Moreover, edges can also be wound around the inner orbit multiple times. We call this the *offset*, represented by a function $\psi_r : E \to \mathbb{Z}$, where $\mathbb{Z}$ is the set of integers. Thus, $|\psi(e)|$ counts the crossings of an edge $e \in E$ with the ray. An edge e is a cut edge if and only if $|\psi(e)| > 0$. If $\psi(e) < 0$ (resp., $\psi(e) > 0$), then e is a *clockwise* (resp., *counter-clockwise*) cut edge. Thus, the sign of $\psi(e)$ reflects the mathematical direction of rotation. Observe that in the above definition a cut edge cannot cross the ray clockwise and counter-clockwise simultaneously. See Fig 1(b). For simplicity, $\psi(e)$ for an edge $e = (u, v)$ may be denoted by $\psi(u, v)$.

We define a *radial drawing* D to consist of the vertex ordering of (π, σ) and the edge offset ψ, i.e., $D = (\pi, \sigma, \psi)$. Given a radial drawing and an edge e^*, let denote $\psi(e; e^*)$, $e \in E - \{e^*\}$ the offset of e when e^* is regarded as the ray. For example, $\psi(e_2; e_3) = 1$ and $\psi(e_1; e_3) = \psi(e_4; e_3) = 0$ hold in Fig 1(b).

We are now ready to describe crossings between edges in a radial drawing D. Note that there may be more than one crossing between two edges. Let $\chi(e_1, e_2; D)$ denote the number of crossings between two edges $e_1, e_2 \in E$. Let $sgn : \mathbb{R} \to \{-1, 0, 1\}$ denote the signum function.

Lemma 2. [1] *Let $D = (\pi, \sigma, \psi)$ be a radial drawing of a bipartite graph $G = (V, W, E)$. Then the number of crossings between two edges $e_1 = (u_1, v_1), e_2 = (u_2, v_2) \in E$ is*

$$\chi(e_1, e_2; D) = \max\left\{0, |\psi(e_2) - \psi(e_1) + \frac{b - a}{2}| + \frac{|a| + |b|}{2} - 1\right\},$$

where $a = sgn(\pi(u_2) - \pi(u_1))$ and $b = sgn(\sigma(v_2) - \sigma(v_1))$. □

We define

$$\chi(e; D) = \sum_{e' \in E - \{e\}} \chi(e, e'; D) \text{ for } e \in E,$$

$$\chi(D; G) = \sum_{e, e' \in E : e \neq e'} \chi(e, e'; D),$$

where $\chi(D; G) = \frac{1}{2} \sum_{e \in E} \chi(e; D)$ holds. We may write $\chi(D; G)$ as $\chi(D)$ if the underlying graph G is clear from the context.

In this paper, we consider a radial drawing with the minimum number of edge crossings, and hence we assume that a given radial drawing satisfies the following conditions:

(C1) Every two edges e and e' cross each other at most once.
(C2) No two edges e and e' incident to the same vertex cross each other.

This is because otherwise we can reduce $\chi(e, e'; D)$ in (C1) by two (resp., (C2) by one) without increasing the number of crossings between any other two edges.

In this paper, we consider the problem of finding a radial drawing $D = (\pi, \sigma, \psi)$ of a bipartite graph $G = (V, W, E)$ that minimizes $\chi(D)$ when a vertex ordering σ of W is fixed. We call this problem *the one-sided crossing minimization* in a radial layout. Let $\chi_r^*(G, \sigma)$ denote the optimal value, i.e., the minimum number of crossings over all radial drawings of G with a specified position σ of W.

3 Radial Drawings with Crossing-Free Edges

An edge e is called *crossing-free* in a radial drawing D if $\chi(e; D) = 0$.

Theorem 2. *For any radial drawing $D = (\pi, \sigma, \psi)$ of $G = (V, W, E)$, there is an offset ψ' of E such that the radial drawing $D' = (\pi, \sigma, \psi')$ of G has at least one crossing-free edge and satisfies*

$$\chi(D') \leq 3\chi(D).$$

Proof. Let $k = \min\{\chi(e; D) \mid e \in E\}$, and $\hat{e} \in \operatorname{argmin}\{\chi(e; D) \mid e \in E\}$. Hence

$$2\chi(D) = \sum_{e \in E} \chi(e; D) \geq k|E|.$$

Let $\widehat{E}$ be the set of edges that cross $\hat{e}$, where we can assume that each edge $e \in \widehat{E}$ crosses $\hat{e}$ exactly once by (C1). See Fig. 2(a). We now modify the offset $\psi(e)$ of each edge $e \in \widehat{E}$ so that the resulting radial drawing $D' = (\pi, \sigma, \psi')$ satisfy the lemma. We redraw each edge $e \in \widehat{E}$ by changing its offset $\psi(e)$ into a new offset $\psi'(e)$ so that it no longer crosses $\hat{e}$. See Fig. 2(b). For this, let $\psi'(e) = 0$ for $e \in \widehat{E}$ with $\psi(e) \in \{-1, 1\}$, and $\psi'(e) = -\psi(e; \hat{e})$ for $e \in \widehat{E}$ with $\psi(e) = 0$. Let $\psi'(e) = \psi(e)$ for all $e \in E - \widehat{E}$.

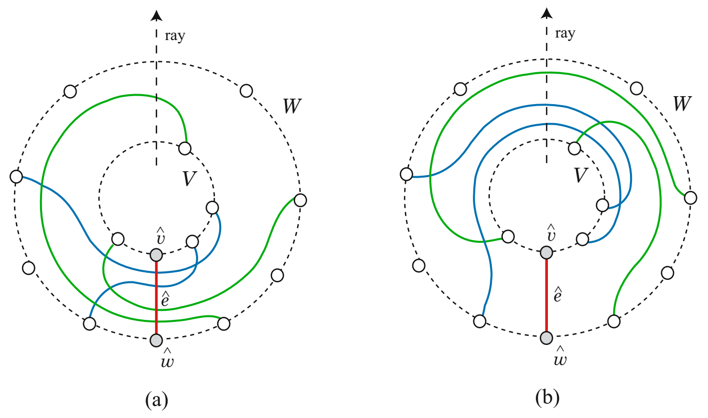


Fig. 2. (a) Edge $\hat{e}$ in a radial drawing D; (b) Edges in $\widehat{E}$ redrawn

Let ψ' be the resulting offsets of E, and let $D' = (\pi, \sigma, \psi')$. We easily see that $\chi(e, \hat{e}; D') = 0$ for all $e \in E - \{\hat{e}\}$ and that $\chi(e, e'; D) = \chi(e, e'; D)$ for all edges $e, e' \in \widehat{E}$. Since $|\psi'(e) - \psi(e)| = 1$ for all $e \in \widehat{E}$, Lemma 2 tells that redrawing edge $e \in \widehat{E}$ can increase the number of crossings between edges e and $e' \in E - \widehat{E}$ at most by $|E| - |\widehat{E}|$. Therefore, we have $\chi(D') \leq \chi(D) + k|E| \leq 3\chi(D)$. □

By Theorem 2, a given graph G contains an edge $\hat{e} = (\hat{v}, \hat{w}) \in E$ ($\hat{v} \in V$, $\hat{w} \in W$) such that $\chi(D) \leq 3\chi_r^*(G, \sigma)$ holds for some radial drawing $D = (\pi, \sigma, \psi)$ of G in which $\hat{e}$ is crossing-free. Our aim is now to consider the problem of finding a radial drawing D with crossing-free edge $\hat{e}$ that minimizes the number of edge crossings for a fixed position σ of W and a specified edge $\hat{e}$. Let $\mathcal{D}_{\hat{e}}$ be the set of all radial drawings with the position σ in which $\hat{e}$ is crossing-free. Then we prove the next result.

Theorem 3. *For a position σ of W in a bipartite graph $G = (V, W, E)$ and an edge $\hat{e} \in E$, a radial drawing $D_0 \in \mathcal{D}_{\hat{e}}$ such that*

$$\chi(D_0) \leq 5 \min\{\chi(D) \mid D \in \mathcal{D}_{\hat{e}}\}$$

can be obtained in polynomial time. □

By Theorems 2, if we apply Theorem 3 for all edges $\hat{e}$ in E, and choose the best drawing D_0 among the resulting drawings, then $\chi(D_0) \leq 5 \min\{\chi(D) \mid \hat{e} \in E,\ D \in \mathcal{D}_{\hat{e}}\} \leq 15\chi_r^*(G, \sigma)$ holds. Thus this implies Theorem 1.

In the next section, we discuss how to find a position π and an offset ψ such that $D_0 = (\pi, \sigma, \psi)$ satisfies Theorem 3.

In the rest of this section, we derive some conditions on offsets $\psi : E \to \{-1, 0, 1\}$ such that $D = (\pi, \sigma, \psi) \in \mathcal{D}_{\hat{e}}$ that minimizes $\chi(D)$ for given positions π and σ. Consider offset $\psi(v, w)$ of an edge $e = (v, w) \in E - \{\hat{e}\}$ with $v \in V$ and $w \in W$. If e is not adjacent to $\hat{e}$, then the offset of e is uniquely determined by

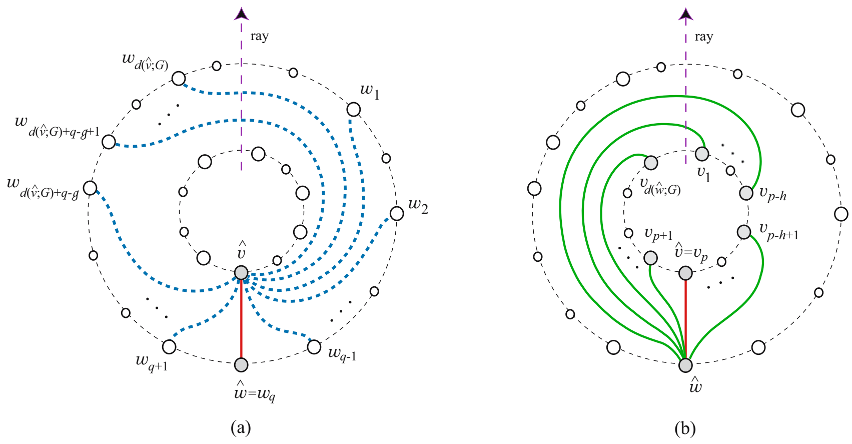


Fig. 3. (a) A partition of $E(\hat{v}; G)$, where $g > q$; (b) A partition of $E(\hat{w}; G)$, where $p > h$

the positions of v and w in π and σ, because it does not cross edge $\hat{e}$ in D. More precisely, it is given by

$$\psi(v, w) = \begin{cases} 0 & \text{if } (\pi(v) - \pi(\hat{v}))(\sigma(w) - \sigma(\hat{w})) > 0, \\ 1 & \text{if } \pi(v) < \pi(\hat{v}) \text{ and } \sigma(w) > \sigma(\hat{w}), \\ -1 & \text{if } \pi(v) > \pi(\hat{v}) \text{ and } \sigma(w) < \sigma(\hat{w}). \end{cases} \qquad (1)$$

However, if e is adjacent to $\hat{e}$, then it still has two possible ways of drawing without crossing $\hat{e}$, hence the offset of e cannot be determined only by (π, σ) (see Fig. 3(a) and (b)). We can assume that no two edges in $E(\hat{v}; G)$ (resp., $E(\hat{w}; G)$) cross each other by (C2). There are $d(\hat{v}; G)$ such drawings for $E(\hat{v}; G)$ (resp., $d(\hat{w}; G)$ such drawings for $E(\hat{w}; G)$). Thus, $E(\hat{v}; G) - \{\hat{e}\}$ is partitioned into two subsets $E_{\hat{v}}^{right}$ and $E_{\hat{v}}^{left}$, called the *right-set* and *left-set* (each of which may be empty), where edges in $E_{\hat{v}}^{right}$ (resp., $E_{\hat{v}}^{left}$) are drawn so that they enter $\hat{v}$ on the right of $\hat{e}$ (resp., the left of $\hat{e}$). We define the right-set $E_{\hat{w}}^{right}$ and the left-set $E_{\hat{w}}^{left}$ analogously. For $g \in \{1, 2, \ldots, d(\hat{v}; G)\}$ and $h \in \{1, 2, \ldots, d(\hat{v}; G)\}$, we denote by $\psi_{\pi,\sigma,\hat{e},g,h}$ the offset of E such that $\hat{e}$ is crossing-free, $|E_{\hat{v}}^{right}| = g - 1$, and $|E_{\hat{w}}^{right}| = h - 1$. More precisely, $\psi_{\pi,\sigma,\hat{e},g,h}$ is given as follows. For $\Gamma(\hat{v}; G) = \{w_1, w_2, \ldots, w_{d(\hat{v};G)}\}$ $(\sigma(w_1) < \sigma(w_2) < \cdots < \sigma(w_{d(\hat{v};G)})$ and $w_q = \hat{w})$, if $g \geq q$ then

$$\psi_{\pi,\sigma,\hat{e},g,h}(\hat{v}, w_i) = \begin{cases} 0 & \text{for } 1 \leq i \leq d(\hat{v}; G) + q - g, \\ 1 & \text{for } d(\hat{v}; G) + q - g < i \leq d(\hat{v}; G) \end{cases}$$

(see Fig. 3(a)), and if $g < q$, then

$$\psi_{\pi,\sigma,\hat{e},g,h}(\hat{v}, w_i) = \begin{cases} -1 & \text{for } 1 \leq i \leq q - g, \\ 0 & \text{for } q - g < i \leq d(\hat{v}; G). \end{cases}$$

The offset $\psi_{\pi,\sigma,\hat{e},g,h}(e)$ of edge e incident to $\hat{w}$ also can be given analogously (see Fig. 3(b)). For each edge $e \in E$ not adjacent to $\hat{e}$, its offset $\psi_{\pi,\sigma,\hat{e},g,h}(e)$ is defined by (1). From the above argument, to prove Theorem 3, it suffices to consider only radial drawings in the form of $D_{g,h} = (\pi, \sigma, \psi_{\pi,\sigma,\hat{e},g,h})$, $(g \in \{1, 2, \ldots, d(\hat{v}; G)\}$, $h \in \{1, 2, \ldots, d(\hat{w}; G)\})$.

Lemma 3. *For positions π of V and σ of W in a bipartite graph $G = (V, W, E)$ and an edge $\hat{e} = (\hat{v}, \hat{w}) \in E$ with $\hat{v} \in V$ and $\hat{w} \in W$, let $\psi_{\pi,\sigma,\hat{e},g,h}$ be the offset with respect to π, σ, $\hat{e}$, $g \in \{1, 2, \ldots, d(\hat{v}; G)\}$ and $h \in \{1, 2, \ldots, d(\hat{w}; G)\}$, and let $D_{g,h} = (\pi, \sigma, \psi_{\pi,\sigma,\hat{e},g,h})$. Then for any drawing $D^* = (\pi, \sigma, \psi^*) \in \mathcal{D}_{\hat{e}}$ of G, it holds*

$$\min\{\chi(D_{g,h}) \mid 1 \leq g \leq d(\hat{v}; G),\ 1 \leq h \leq d(\hat{w}; G)\} \leq \chi(D^*).$$

□

To prove Theorem 3, we also assume that $d(v; G) \geq 2$ for all $v \in \Gamma(\hat{w}; G) - \{\hat{v}\}$. We easily see that π attains the minimum $\chi(D)$ when all vertices $v \in \Gamma(\hat{w}; G)$ with $d(v; G) = 1$ appears consecutively including $\hat{v}$, in particular, they are contained in $E_{\hat{w}}^{right}$ or $E_{\hat{w}}^{left}$ completely. Thus, we only need to consider these two cases, and removing those vertices from such a drawing makes a fixed amount of change in the crossing number $\chi(D)$. Hence, in the next section, we assume that no vertex with degree 1 (except for $\hat{v}$) is adjacent to $\hat{w}$ for simplicity.

4 Reduction from Radial Drawings to Horizontal Drawings

Let $D = (\pi, \sigma, \psi)$ be a radial drawing of a bipartite graph $G = (V, W, E)$, and $\hat{e} = (\hat{v}, \hat{w}) \in E$ be an edge with $\hat{v} \in V$ and $\hat{w} \in W$. Consider a horizontal drawing $D_e = (\pi', \sigma')$ of the induced graph $G[V \cup W - \{\hat{v}, \hat{w}\}]$ as follows. Let π' (resp., σ') denote the permutation of $\{1, 2, \ldots, |V| - 1\}$ (resp., $\{1, 2, \ldots, |W| - 1\}$) induced from π by $V - \{\hat{v}\}$ (resp., σ by $W - \{\hat{w}\}$), i.e.,

$$\pi'(v) = \pi(v) - \pi(\hat{v}) \pmod{|V| - 1} \text{ for } v \in V - \{\hat{v}\},$$

$$\sigma'(w) = \sigma(w) - \sigma(\hat{w}) \pmod{|W| - 1} \text{ for } w \in W - \{\hat{w}\}.$$

Then we say that π has the *median property with respect to* $(\sigma, \hat{e})$ if the permutation π' has the median property with respect to σ' in $G[V \cup W - \{\hat{v}, \hat{w}\}]$. Note that such π can be computed from $(\sigma, \hat{e})$ in polynomial time *without* knowing the information on ψ.

As observed in the previous section, we assume that no vertex v with $d(v; G) = 1$ (except for $\hat{v}$) is adjacent to $\hat{w}$.

Lemma 4. *For a position σ of W in a bipartite graph $G = (V, W, E)$ and an edge $\hat{e} = (\hat{v}, \hat{w}) \in E$ with $\hat{v} \in V$ and $\hat{w} \in W$, let $D_{g,h} = (\pi, \sigma, \psi_{\pi,\sigma,\hat{e},g,h})$ and $D^*_{g,h} = (\pi^*, \sigma, \psi_{\pi^*,\sigma,\hat{e},g,h})$ be two radial drawings of G with crossing-free edge $\hat{e}$, where $1 \leq g \leq d(\hat{v}; G)$ and $1 \leq h \leq d(\hat{w}; G)$. If π has the median property with respect to $(\sigma, \hat{e})$, then, for each $g \in \{1, \ldots, d(\hat{v}; G)\}$, it holds*

$$\min\{\chi(D_{g,t}) \mid 1 \leq t \leq d(\hat{w}; G)\} \leq 5\chi(D^*_{g,h}).$$

□

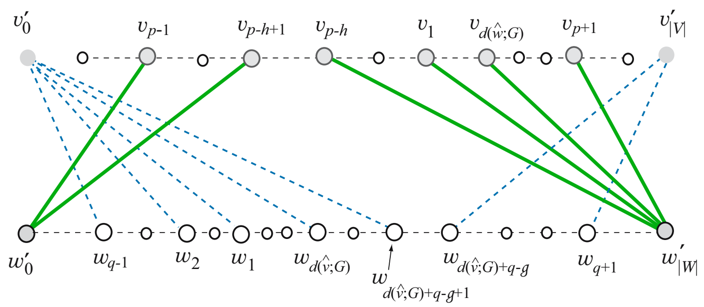


Fig. 4. Horizontal drawing $D_A = (\pi_A, \sigma_A)$ of G_A

Let $D^* \in \mathcal{D}_{\hat{e}}$ be a radial drawing with the minimum crossing number. By Lemma 3, we can assume that $D^* \in \mathcal{D}_{\hat{e}}$ is given as $D^*_{g,h} = (\pi^*, \sigma, \psi_{\pi^*,\sigma,\hat{e},g,h})$ for some $g \in \{1, \ldots, d(\hat{v}; G)\}$ and $h \in \{1, \ldots, d(\hat{w}; G)\}$. By applying Lemma 4 for all $g \in \{1, \ldots, d(\hat{v}; G)\}$, it holds $\min\{\chi(D_{g,t}) \mid 1 \le g \le d(\hat{v}; G),\ 1 \le t \le d(\hat{w}; G)\} \le 5\chi(D^*_{g,h}) = 5\chi(D^*)$, and the drawing $D_0 = D_{g',t'}$ attaining the minimum implies Theorem 3.

Now the remaining task is to prove Lemma 4. For this, we use the median property of π for a certain instance of the one-sided crossing minimization problem in a horizontal layout.

Based on G and $D^*_{g,h}$, we first define a graph G_A and a horizontal drawing D_A as follows. Remove edge $\hat{e} = (\hat{v}, \hat{w})$ from G. Then split vertex $\hat{v}$ into two new vertices v'_0 and $v'_{|V|}$ changing the end vertex $\hat{v}$ of each edge $(\hat{v}, w_i)$ in the right-set (resp., the left-set) of $E(\hat{v}; G) - \{\hat{e}\}$ to v'_0 (resp., $v'_{|V|}$). Analogously, split vertex $\hat{w}$ into w'_0 and $w'_{|W|}$ changing end vertex $\hat{w}$ of each edge $(v_i, \hat{w})$ in the right-set (resp., the left-set) of $E(\hat{w}; G) - \{\hat{e}\}$ to w'_0 (resp., $w'_{|W|}$). We denote the resulting bipartite graph by

$$G_A = ((V - \{\hat{v}\}) \cup \{v'_0, v'_{|V|}\}, (W - \{\hat{w}\}) \cup \{w'_0, w'_{|W|}\}, E - \{\hat{e}\}).$$

See Fig. 4. Let π_A and σ_A be the permutations of $\{0, 1, \ldots, |V|\}$ and $\{0, 1, \ldots, |W|\}$, respectively, such that $\pi_A(v'_0) = 0$, $\pi_A(v'_{|V|}) = |V|$, $\pi_A(v) = \pi^*(v) - \pi^*(\hat{v}) \pmod{|V|}$ for $v \in V - \{\hat{v}\}$, $\sigma_A(w'_0) = 0$, $\sigma_A(w'_{|V|}) = |V|$, and $\sigma_A(w) = \sigma(w) - \sigma(\hat{w}) \pmod{|W|}$ for $w \in W - \{\hat{w}\}$. Then the number of edge crossings in a horizontal drawing $D_A = (\pi_A, \sigma_A)$ of G_A is equal to that of $D^*_{g,h}$, i.e.,

$$\chi(D_A; G_A) = \chi(D^*_{g,h}; G).$$

We next consider the graph

$$G_B = (V_B = V - \{\hat{v}\}, W_B = (W - \{\hat{w}\}) \cup \{w'_0, w'_{|W|}\}, E_B = E - E(\hat{v}; G)),$$

i.e., G_B is obtained from G_A by removing vertices v'_0 and $v'_{|V|}$ together with the incident edges (see Fig. 5(a)). Let π_B be the permutation of $\{1, \ldots, |V| - 1\}$

such that $\pi_B(v) = \pi_A(v) = \pi^*(v)$ for $v \in V - \{\hat{v}\}$. Then a horizontal drawing $D_B = (\pi_B, \sigma_A)$ of G_B satisfies

$$\chi(D_B; G_B) + \chi_0 = \chi(D_A; G_A),$$

where $\chi_0 = \sum\{\chi(e, e'; D_B) \mid e \in E(v'_0; G_B) \cup E(v'_{|V|}; G_B), e' \in E - E(v'_0; G_B) \cup E(v'_{|V|}; G_B)\}$. Note that χ_0 with a fixed g is constant for any choice of π^* in $D^*_{g,h} = (\pi^*, \sigma, \psi_{\pi^*,\sigma,\hat{e},g,h})$.

Consider the induced graph

$$G[V \cup W - \{\hat{v}, \hat{w}\}] = (V - \{\hat{v}\}, W - \{\hat{w}\}, E - E(\hat{v}; G)).$$

Recall that σ' is a permutation of $\{1, 2, \ldots, |W| - 1\}$ such that $\sigma'(w) = \sigma(w)$ for $w \in W - \{\hat{w}\}$, and π' is a permutation of $\{1, \ldots, |V| - 1\}$, which has the median property with respect to σ' in graph $G[V \cup W - \{\hat{v}, \hat{w}\}]$ by the assumption on π in Lemma 4.

We eliminate all edge crossings between edges incident to w'_0 and $w'_{|W|}$ in a drawing D_C of G_B by modifying graph G_B as follows. Find two crossing edges (w'_0, v'_j) and $(w'_{|W|}, v'_i)$, remove such edges, and add two new edges (w'_0, v'_i) and $(w'_{|W|}, v'_j)$ (see Fig. 5(a) and (b)). Repeat this edge-exchange operation as long as it is applicable. Let $G_C = (V - \{\hat{v}\}, W - \{\hat{w}\}, E_C)$ denote the resulting graph (note that G_C may not be unique).

Consider horizontal drawing $D_C = (\pi', \sigma_A)$ of G_C. Observe that the drawing D_C in G_C corresponds to a radial drawing with crossing-free edge $\hat{e}$; i.e., for some $t' \in \{1, 2, \ldots, d(\hat{w}; G)\}$, $D_{g,t'} = (\pi, \sigma, \psi_{\pi,\sigma,\hat{e},g,t'})$ satisfies

$$\chi(D_{g,t'}) = \chi(D_C; G_C) + \chi_0.$$

To prove Lemma 4, it suffices to show that there is a way of constructing G_C from D_B such that

$$\chi(D_C; G_C) \le 5\chi(D_B; G_B), \tag{2}$$

from which we have

$$\begin{aligned} \min\{\chi(D_{g,t}) \mid 1 \le t \le d(\hat{w}; G)\} &\le \chi(D_{g,t'}) \le \chi(D_C; G_C) + \chi_0 \\ &\le 5\chi(D_B; G_B) + \chi_0 \\ &\le 5\chi(D_A; G_A) = 5\chi(D^*_{g,h}; G). \end{aligned}$$

Lemma 5. *A bipartite graph G_C which satisfies (2) can be constructed from a horizontal drawing $D_B = (\pi_B, \sigma_A)$ by applying edge-exchange operations to G_B.*

Proof. Omitted due to space limitation. □

Lemma 5 implies Lemma 4, from which Theorem 3 follows, as we already observed.

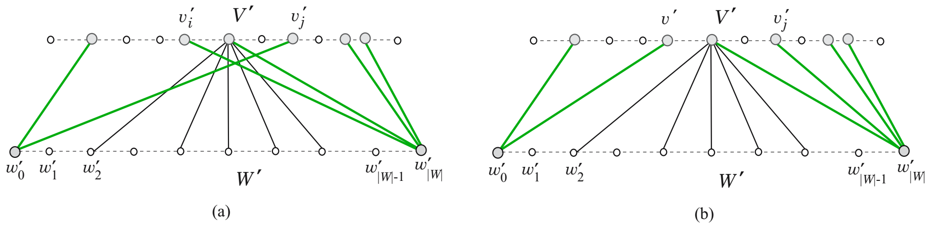


Fig. 5. (a) Horizontal drawing $D_C = (\pi', \sigma_A)$ in graph G_B; (b) Horizontal drawing $D_C = (\pi', \sigma_A)$ of graph G_C

5 Conclusion

In this paper, we have proved that the one-sided crossing minimization problem in radial 2-layered drawings is 15-approximable by converting an instance of the problem into an instance of the one-sided crossing minimization problem in horizontal 2-layered drawings where at least one edge is required to be crossing-free. This is the first constant-factor approximation algorithm.

To best of our knowledge, the problem of finding an optimal offset for given positions of vertices in the inner and outer orbits remains open. However, it is not difficult to see that a 3-approximate solution for the problem can be obtained by using Theorem 2 and Lemma 3.

References

1. Bachmaier, C.: A radial adaptation of the sugiyama framework for visualizing hierarchical information. IEEE Trans. Vis. Comput. Graph. 13, 583–594 (2007)
2. Eades, P., Wormald, N.C.: Edge crossing in drawing bipartite graphs. Algorithmica 11, 379–403 (1994)
3. Jünger, M., Mutzel, P.: 2-layer straight line crossing minimization: performance of exact and heuristic algorithms. J. Graph Algorithms and Applications 1, 1–25 (1997)
4. Leighton, F.T.: Complexity Issues in VLSI. MIT Press, Cambridge (1983)
5. Nagamochi, H.: An improved bound on the one-sided minimum crossing number in two-layered drawings. Discrete and Computational Geometry 33, 569–591 (2005)
6. Sarrafzadeh, M., Wong, C.K.: An Introduction to VLSI Physical Design. McGraw-Hill, New York (1996)
7. Shahrokhi, F., Sykora, O., Székly, L.A., Vrto, I.: On bipartite drawings and the linear arrangement problem. SIAM Journal on Computing 30, 1773–1789 (2001)
8. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Transactions on Systems, Man, and Cybernetics 11, 109–125 (1981)
9. Wasserman, S., Faust, K.: Social Network Analysis: Methods and Applications. Cambridge University Press, Cambridge (1994)

New Upper Bound on Vertex Folkman Numbers

Andrzej Dudek and Vojtěch Rödl

Department of Mathematics and Computer Science
Emory University
Atlanta, GA 30322, USA
{adudek,rodl}@mathcs.emory.edu

Abstract. In 1970, J. Folkman proved that for a given integer r and a graph G of order n there exists a graph H with the same clique number as G such that every r coloring of vertices of H yields at least one monochromatic copy of G. His proof gives no good bound on the order of graph H, i.e. the order of H is bounded by an iterated power function. A related problem was studied by Łuczak, Ruciński and Urbański, who gave some explicite bound on the order of H when G is a clique. In this note we give an alternative proof of Folkman's theorem with the relatively small order of H bounded from above by $O(n^3 \log^3 n)$. This improves Łuczak, Ruciński and Urbański's result.

Keywords: Ramsey theory, vertex Folkman numbers.

1 Introduction

For a given graph $H = (V, E)$, let $c : V(H) \to \{1, \ldots, r\}$ be an r-coloring of vertices of H. We write $H \to (G)^v_r$ (or $H \xrightarrow[ind]{} (G)^v_r$) if for every r-coloring c of vertices of H, there exists a copy of G (or an induced copy), say G', such that $V(G') \subseteq c^{-1}(i)$, for some color $i \in \{1, \ldots, k\}$. Moreover, let $cl(G)$ be the *clique number* of G, i.e. the order of a maximal clique in G. In [4], J. Folkman proved that for every graph G there exists a graph H such that $H \to (G)^v_r$ and $cl(H) = cl(G)$. Clearly $cl(H) \geq cl(G)$ for any graph with $H \to (G)^v_r$ and thus Folkman's theorem is in this sense the best possible. For $G = K_k$, i.e. for a clique of size k, a related question was studied e.g. in [6,8]. For positive integers r, k and l with $k < l$ the *vertex Folkman number* is

$$F(r, k, l) = \min\left\{|V(H)| \;\middle|\; H \to (K_k)^v_r \text{ and } cl(H) = l - 1\right\}.$$

Clearly Folkman's theorem yields that $F(r, k, l)$ is well-defined for any $k < l$. Determining the precise value of $F(r, k, l)$ is not an easy problem in general. Only few of these numbers are known and mostly they were found with the aid of computers (see e.g. [2]). Some special cases were considered in [6,8]. Obviously, the most restrictive and challenging case is to determine the exact value of $F(r, k, k+1)$ (or more realistic to estimate it). The upper bound on this number, based on Folkman's proof [4], is an iterated power function. Łuczak, Ruciński and Urbański [8] improved this bound and showed that for instance for 2 colors $F(2,k,k+1) = O(k!)$. In this note we will prove a relatively small upper bound on $F(r,k,l)$ (see comments after Theorem 1). In fact, we will prove a more general statement. In 1991, Brown and the second author [1] showed that for every natural number r there are constants c_1 and c_2 such that

$$c_1 n^2 \le \max\Big\{ \min\big\{|V(H)| \;\big|\; H \xrightarrow[ind]{} (G)^v_r\big\}\Big\} \le c_2 n^2 \log^2 n, \tag{1}$$

where the maximum is taken over all graphs G of order n. In this note we will not only enforce $H \to (G)^v_r$, but also require $cl(H) = cl(G)$. Following the idea from [1], we will show that adding this new constraint will increase the upper bound in (1) only by a factor of $n \log n$.

Theorem 1. *For a given natural number r there exists a constant $c = c(r)$ such that for every graph G of order n the following inequality holds*

$$\min\big\{|V(H)| \;\big|\; H \xrightarrow[ind]{} (G)^v_r \text{ and } cl(H) = cl(G)\big\} \le cn^3 \log^3 n.$$

For $G = K_k$ Theorem 1 immediately yields $F(r,k,l) \le ck^3 \log^3 k$. In fact, modifying the proof of Theorem 1, one can show a stronger result, which we state below without proof.

Theorem 2. *For a given natural number r and an arbitrarily small $\varepsilon > 0$ there exists a constant $c = c(r,\varepsilon)$ such that for any natural numbers k and l with $k < l$ the vertex Folkman number satisfies*

$$F(r,k,l) \le ck^{2+\varepsilon}.$$

We were not able to find any nontrivial lower bound on $F(r,k,l)$. It would be interesting to decide if for every r the ratio $\frac{F(r,k,k+1)}{k}$ tends to infinity as k tends to infinity. This work is currently in progress.

2 Generalized Quadrangles

In this section we describe some basic properties of generalized quadrangles, which we use to prove Theorem 1.

A *generalized quadrangle* (see e.g. [5]) is an incidence structure of a set $\mathcal{P}$ of points and a set $\mathcal{L}$ of lines such that:

(i) any two points are on at most one line,
(ii) if p is a point not on a line ℓ, then there is a unique point $\mathsf{p}' \in \ell$ collinear with p, and hence, no three lines form a triangle,
(iii) every line contains $q+1$ points, and every point lies on $q+1$ lines.

It is known that for every prime power q such incidence structure $\mathcal{Q}$ exists with $|\mathcal{P}| = |\mathcal{L}| = q^3 + q^2 + q + 1$ (see e.g. [7,9]). Let $\mathcal{Q}_I = (\mathcal{P}, \mathcal{L}, \mathcal{E})$ be a bipartite graph, which corresponds to the above incidence structure $\mathcal{Q}$, with the set of vertices $\mathcal{P} \cup \mathcal{L}$ and the set of edges $\mathcal{E} = \{(\mathsf{p}, \ell) \in \mathcal{P} \times \mathcal{L} \mid \mathsf{p} \text{ lies on line } \ell\}$. Note that $\mathcal{Q}_I$ is a $(q+1)$-regular graph. For a given graph G and disjoint subsets B and C of vertices of G we denote by $e(B,C)$ the number of edges that connect a vertex of B with a vertex of C. The following statement is an easy consequence of the fact that the graph $\mathcal{Q}_I$ is $(q+1)$-regular.

Fact 3. *Let $\mathcal{Q}_I = (\mathcal{P}, \mathcal{L}, \mathcal{E})$ be the bipartite graph defined above with $|\mathcal{P}| = |\mathcal{L}| = N$. Suppose that $Y \subseteq \mathcal{P}$ with $|Y| = \alpha N$, for some $0 < \alpha < 1$. Then,*

$$e(Y, \mathcal{L}) = \alpha N^{\frac{4}{3}}(1 + o(1)).$$

3 Proof of Theorem 1

As we mentioned in the introduction, the proof of Theorem 1 goes along the lines of the proof of Theorem 2.2 from [1].

Fix a natural number r, $r \geq 2$ (for $r = 1$ Theorem 1 holds trivially). Let α be a real number satisfying $0 < \alpha \leq \frac{1}{2}$. Let $G = (V, E)$ be a graph of order n with $V = \{v_1, \ldots, v_n\}$. We always assume that n is sufficiently large. We will show that there exists a graph H of order $cn^3 \log^3 n$, $c = c(r)$, such that $cl(H) = cl(G)$ and any subgraph of H induced by a set of cardinality $\lfloor \alpha |V(H)| \rfloor$ contains an induced copy of G. For $\alpha = \frac{1}{r}$ this will obviously imply the statement of Theorem 1.

By Bertrand's postulate we know that for any number $z \geq 1$ there exists a prime number between z and $2z$. In particular for a given n there is a prime q such that

$$\frac{4}{\alpha} n \log n \leq q + 1 \leq \frac{8}{\alpha} n \log n.$$

For such q let $t = q + 1$ and let x be such that

$$t = xn + m, \tag{2}$$

where $0 \leq m < n$. Consequently,

$$\frac{3}{\alpha} \log n \leq x \leq \frac{8}{\alpha} \log n. \tag{3}$$

Let $\mathcal{Q}$ be a generalized quadrangle from the previous section with $|\mathcal{P}| = |\mathcal{L}| = N$, where $N = q^3 + q^2 + q + 1$. We construct a "random graph" H with the vertex set $\mathcal{P}$ as follows. In view of (2) one can partition each line into sets of size x and $x + 1$, respectively. For each line ℓ we choose one such ordered partition $\ell_1, \ldots, \ell_n$ randomly and uniformly from the sets of all

$$\gamma = \frac{t!}{(x!)^{n-m}((x+1)!)^m} \tag{4}$$

partitions. For each $\mathsf{u} \in \ell_i$ and $\mathsf{w} \in \ell_j$ we join $\{\mathsf{u}, \mathsf{w}\}$ by an edge if and only if $\{v_i, v_j\} \in E(G)$. Note that H is well-defined because of condition (i). Moreover, condition (ii) yields that $cl(H) = cl(G)$. In fact, more is true: every triangle of H is contained entirely within some ℓ. Observe that there are γ^N graphs H constructed this way. We will show that the graph randomly chosen from the space of all such graphs has the following property. Every set $Y \subseteq V(H)$, $|Y| = \lfloor \frac{1}{r} N \rfloor$ induces a subgraph $H[Y]$, which contains G as an induced subgraph.

For $Y \subseteq V(H)$ with cardinality $|Y| = \lfloor \frac{1}{r} N \rfloor = \lfloor \alpha N \rfloor$ let A_Y be the event that G is an induced subgraph of $H[Y]$. For each $\ell \in \mathcal{L}$ let A_ℓ be the event that some ℓ_i in the partition of ℓ is disjoint from Y. Note that if A_Y fails, then all events A_ℓ, $\ell \in \mathcal{L}$, must occur. Consequently,

$$\bar{A}_Y \subseteq \bigcap_{\ell \in \mathcal{L}} A_\ell.$$

Furthermore, since all events A_ℓ are independent we obtain

$$\Pr(\bar{A}_Y) \le \prod_{\ell \in \mathcal{L}} \Pr(A_\ell). \tag{5}$$

For each $\ell \in \mathcal{L}$ we will bound from above the probability $\Pr(A_\ell)$. Recall that $|\ell| = t$ and γ is the number of all ordered partitions of ℓ into n sets ($n - m$ of size x and m of size $x + 1$). Let $|Y \cap \ell| = y_\ell$. For a fixed i, the number of partitions of ℓ for which ℓ_i, $|\ell_i| = x$, is disjoint from Y is at most

$$\binom{t - y_\ell}{x} \frac{(t-x)!}{(x!)^{n-m-1}\big((x+1)!\big)^m} = \frac{\binom{t-y_\ell}{x}}{\binom{t}{x}} \gamma. \tag{6}$$

Similarly, the number of partitions of ℓ for which ℓ_i, $|\ell_i| = x + 1$, is disjoint from Y is at most

$$\binom{t - y_\ell}{x+1} \frac{(t-x-1)!}{(x!)^{n-m}\big((x+1)!\big)^{m-1}} = \frac{\binom{t-y_\ell}{x+1}}{\binom{t}{x+1}} \gamma. \tag{7}$$

Hence, (4), (6) and (7) yield

$$\Pr(A_\ell) \le (n-m)\frac{\binom{t-y_\ell}{x}}{\binom{t}{x}} + m\frac{\binom{t-y_\ell}{x+1}}{\binom{t}{x+1}} \le (n-m)\mathrm{e}^{\frac{-xy_\ell}{t}} + m\mathrm{e}^{\frac{-(x+1)y_\ell}{t}} \le n\mathrm{e}^{\frac{-xy_\ell}{t}}.$$

The last inequality holds, since for for any natural numbers a, b, c with $a - b \ge c$, the following is true

$$\frac{\binom{a-b}{c}}{\binom{a}{c}} = \frac{(a-b-c+1)\cdots(a-b)}{(a-c+1)\cdots a} \le \Big(\frac{a-b}{a}\Big)^c \le \mathrm{e}^{-\frac{bc}{a}}.$$

Consequently, by (5) we get

$$\Pr(\bar{A}_Y) \le n^N \exp\Big(-\frac{x}{t}\sum_{\ell \in \mathcal{L}} y_\ell\Big).$$

Moreover, Fact 3 infers that $\sum_{\ell\in\mathcal{L}} y_\ell = e(Y,\mathcal{L}) \geq \frac{\alpha}{2}N^{\frac{4}{3}}$, and hence

$$\Pr(\bar{A}_Y) \leq n^N \exp\Big(-\frac{\alpha}{2}\frac{x}{t}N^{\frac{4}{3}}\Big).$$

Thus,

$$\begin{aligned}\Pr\big(\bigcup_Y \bar{A}_Y\big) &\leq \binom{N}{\lfloor\alpha N\rfloor} n^N \exp\Big(-\frac{\alpha}{2}\frac{x}{t}N^{\frac{4}{3}}\Big)\\ &\leq \Big(\frac{\mathrm{e}}{\alpha}\Big)^{\alpha N} n^N \exp\Big(-\frac{\alpha}{2}\frac{x}{t}N^{\frac{4}{3}}\Big)\\ &= \exp\bigg(N\Big(\alpha-\alpha\log\alpha+\log n-\frac{\alpha}{2}\frac{x}{t}\sqrt[3]{N}\Big)\bigg)\\ &\leq \exp\bigg(N\Big(\alpha-\alpha\log\alpha+\log n-\frac{\alpha}{2}\frac{3}{\alpha}\log n\frac{\sqrt[3]{N}}{t}\Big)\bigg), \qquad (8)\end{aligned}$$

where the last inequality follows from (3).

Since $\frac{\sqrt[3]{N}}{t} \sim 1$ we obtain that (8) tends to 0 as n goes to infinity. This yields that

$$\Pr\big(\bigcap_Y A_Y\big) > 0,$$

i.e. there is a graph H of order $N = q^3 + q^2 + q + 1 = O(n^3 \log^3 n)$ for which every subgraph of order $\lfloor\alpha N\rfloor$ contains G as an induced subgraph. In particular, for $\alpha = \frac{1}{r}$ we have $H \xrightarrow[ind]{} (G)^v_r$ and $cl(H) = cl(G)$. This completes the proof of Theorem 1.

4 Concluding Remarks

With some additional work (see e.g. [3]) one can reduce the factor $n^3 \log^3 n$ from Theorem 1 to $n^3 \log n$.

References

1. Brown, J.I., Rödl, V.: A Ramsey type problem concerning vertex colourings. J. Comb. Theory, Ser. B 52(1), 45–52 (1991)
2. Coles, J., Radziszowski, S.: Computing the Folkman number $F_v(2,2,3;4)$. J. Combinatorial Mathematics and Combinatorial Computing 58, 13–22 (2006)
3. Eaton, N., Rödl, V.: A canonical Ramsey theorem. Random Structures and Algorithms 3(4), 427–444 (1992)
4. Folkman, J.: Graphs with monochromatic complete subgraphs in every edge coloring. SIAM J. Appl. Math. 18, 19–24 (1970)
5. Royle, G., Godsil, C.: Algebraic graph theory. Springer, New York (2001)

6. Kolev, N., Nenov, N.: New upper bound for a class of vertex Folkman numbers. Electronic J. Comb. 13, #R14 (2006)
7. Lazebnik, F., Ustimenko, V.A., Woldar, A.J.: Polarities and $2k$-cycle-free graphs. Disc. Math. 197/198, 503–513 (1999)
8. Łuczak, T., Ruciński, A., Urbański, S.: On minimal vertex Folkman graphs. Disc. Math. 236, 245–262 (2001)
9. Thas, J.A.: Generalized polygons. In: Buekenhout, F. (ed.) Handbook on incidence geometry, ch. 9, North Holland, Amsterdam (1995)

Ptolemaic Graphs and Interval Graphs Are Leaf Powers

Andreas Brandstädt and Christian Hundt

Institut für Informatik, Universität Rostock, D-18051, Germany
{Andreas.Brandstaedt,Christian.Hundt}@uni-rostock.de

Abstract. Motivated by the problem of reconstructing evolutionary history, Nishimura, Radge and Thilikos introduced the notion of k-leaf powers as the class of graphs $G = (V, E)$ which have a k-leaf root, i.e., a tree T with leaf set V where $xy \in E$ if and only if the T-distance between x and y is at most k. It is known that leaf powers are strongly chordal (i.e., sun-free chordal) graphs. Despite extensive research, the problem of recognizing leaf powers, i.e., to decide for a given graph G whether it is a k-leaf power for some k, remains open. Much less is known on the complexity of finding the leaf rank of G, i.e., to determine the minimum number k such that G is a k-leaf power. A result by Bibelnieks and Dearing implies that not every strongly chordal graph is a leaf power. Recently, Kennedy, Lin and Yan have shown that dart- and gem-free chordal graphs are 4-leaf powers. We generalize their result and show that ptolemaic (i.e., gem-free chordal) graphs are leaf powers. Moreover, ptolemaic graphs have unbounded leaf rank. Furthermore, we show that interval graphs are leaf powers which implies that leaf powers have unbounded clique-width. Finally, we characterize unit interval graphs as those leaf powers having a caterpillar leaf root.

Keywords and Classification: Leaf powers; leaf roots; strongly chordal graphs; ptolemaic graphs; graph powers; graph class inclusions; (unit) interval graphs; clique-width.

1 Introduction

Motivated by the problem of reconstructing evolutionary history, and by the notion of *k-th phylogenetic power* and *k-root phylogeny* of a graph G introduced by Lin, Kearney and Jiang [23] (see also [10]), Nishimura, Ragde and Thilikos [26] introduced the following notion:

A tree $T = (V_T, E_T)$ is a *k-leaf root* of a finite undirected graph $G = (V, E)$ if the set of leaves of T is V and for any two vertices $x, y \in V$, $xy \in E$ if and only if the distance of x and y in T is at most k. Graph G is a *k-leaf power* if it has a k-leaf root. In general, G is a *leaf power* if it is a k-leaf power for some k, and a k-leaf root T of G is also called *leaf root*.

The vertex set V_T of T consists of *leaves* and *internal nodes*. Note that internal nodes may have degree two; in phylogenetic roots, internal nodes have degree larger than two but edges are weighted. The two models are equivalent because a path of degree two nodes can be contracted to one edge with the length of the corresponding path as its weight. For a leaf power G, its *leaf rank* $lr(G)$ is the smallest k such that G has a k-leaf root. For a class $\mathcal{G}$ of leaf powers, let $lr(\mathcal{G})$ be the maximum $lr(G)$ over all $G \in \mathcal{G}$, and infinite if it is unbounded.

Obviously, a graph G is a 2-leaf power if and only if it is the disjoint union of cliques, i.e., G is P_3-free (where P_3 denotes the path with three vertices and two edges). The 3-leaf powers are exactly the bull-, dart-, and gem-free chordal graphs [14] (see Figure 1 for bull, dart and gem); equivalently, 3-leaf powers are exactly the result of substituting cliques into the nodes of a tree. For this and other characterizations of 3-leaf powers see [3,27]. A characterization of 4-leaf powers in terms of forbidden subgraphs is much more complicated [27,6]. For 5-leaf powers, a polynomial time recognition was given in [9] but no structural characterization of 5-leaf powers is known. Recently, [4] characterized distance-hereditary 5-leaf powers (without true twins) in terms of forbidden induced subgraphs. The complexity of characterizing and recognizing leaf powers in general is a major open problem.

However, leaf powers are strongly chordal (i.e., sun-free chordal) but not vice versa. Moreover, in [21], Kennedy, Lin and Yan showed that strictly chordal (i.e., dart- and gem-free chordal) graphs are 4-leaf powers (but not vice versa). This leaves a huge gap for a precise localization and characterization of the class of leaf powers. The aim of this paper is to narrow this gap by showing the following results (which also improve and extend various results in [18,21]):

(i) Every ptolemaic (i.e., gem-free chordal) graph is a leaf power (but not vice versa as the gem shows).
(ii) In contrast to strictly chordal graphs (which are 4-leaf powers), the leaf rank of ptolemaic graphs is unbounded.
(iii) Every interval graph is a leaf power; thus, as for strongly chordal graphs (but unlike ptolemaic graphs), the clique-width of leaf powers is unbounded.
(iv) Unit interval graphs are exactly the leaf powers which have a caterpillar as leaf root.

Due to space limitations, most of the proofs are omitted.

2 Notations and Basic Facts

Throughout this paper, let $G = (V, E)$ be a finite undirected graph without self-loops and multiple edges with vertex set V and edge set E, and let $|V| = n$, $|E| = m$. For a vertex $v \in V$, let $N(v) = \{u \mid uv \in E\}$ denote the (*open*) *neighborhood* of v in G, and let $N[v] = \{v\} \cup N(v)$ denote the *closed neighborhood* of v in G. A *clique* is a vertex set of mutually adjacent vertices. An *independent vertex set* is a set of mutually nonadjacent vertices. A vertex is *simplicial* in G if its neighborhood $N(v)$ is a clique.

For a subset $U \subseteq V$, let $G[U] = (U, E_U)$ denote the *induced subgraph* of G where E_U consists of all edges in E with both end vertices in U.

If $xy \in E$ then we also say that x *sees* y and vice versa. Two vertices $x, y \in V$ are *true twins* (*false twins*, respectively) if $N(x) = N(y)$ and $xy \in E$ ($xy \notin E$, respectively).

Let $\mathcal{F}$ denote a set of graphs. A graph G is $\mathcal{F}$*-free* if none of its induced subgraphs is in $\mathcal{F}$. A sequence $P = (v_1, \ldots, v_k)$ of pairwise distinct vertices is an *induced path* if its edge set is $\{v_1v_2, \ldots, v_{k-1}v_k\}$; such paths with k vertices will be denoted by P_k. If additionally, v_kv_1 is an edge then P is an *induced cycle*; such cycles will be denoted by C_k. The *length* $|P_k|$ of P_k ($|C_k|$ of C_k, respectively) is $k - 1$ (k, respectively). Subsequently, we only consider induced paths. Let the *distance* $d_G(x, y)$ (or $d(x, y)$ for short if G is understood) be the length of a shortest path in G between x and y.

For $k \geq 1$, let $G^k = (V, E^k)$ with $xy \in E^k$ if and only if $d_G(x, y) \leq k$ denote the *k-th power of* G. If the edges get weights then $d_G(x, y)$ is the minimum weight sum on any path between x and y.

Figure 1 contains some graphs which are important for this paper. For $k \geq 3$, a *k-sun* is a graph of $2k$ vertices $u_1, \ldots, u_k$ and $w_1, \ldots, w_k$ such that $u_1, \ldots, u_k$ is a clique, $w_1, \ldots, w_k$ is an independent stable set and for all $i \in \{1, \ldots, k\}$, $N(w_i) = \{u_i, u_{i+1}\}$ (index arithmetic modulo k). A *sun* is a k-sun for some $k \geq 3$.

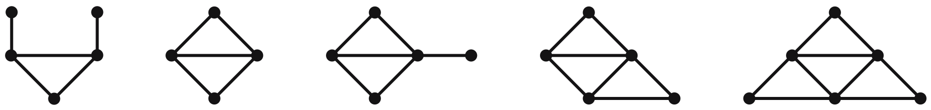


Fig. 1. Bull, diamond, dart, gem and 3-sun

Figure 1 shows the 3-sun. A graph is

- *chordal* if it contains no induced cycle of length more than three,
- *strongly chordal* if it is chordal and sun-free [16] (see also [5] for various characterizations of (strongly) chordal graphs),
- *ptolemaic* if it is chordal and gem-free [19,20],
- *strictly chordal* if it is chordal, dart- and gem-free [21],
- a *block graph* if it is chordal and diamond-free (equivalently, a graph is a block graph if and only if its blocks are cliques),
- a tree if it is cycle-free and connected.

It is easy to see that each of the above graph classes is properly contained in the preceding one. Note that there is a close relationship between these graph classes and certain acyclicity conditions of hypergraphs (which were called Berge-, γ-, β- and α-acyclicity) motivated by desirable properties of relational database schemes which are described by Fagin [15].

A graph G is *distance hereditary* if for every induced connected subgraph H of G, the distance function in H is the same as in G. For various characterizations of distance-hereditary graphs see [19,1]. It is well known that a chordal graph is distance hereditary if and only if it is gem-free chordal (i.e., ptolemaic) ([19,20,1,5]). Ptolemaic graphs G were characterized by Bandelt and Mulder [1] in terms of three operations adding a new vertex y to G with respect to an existing vertex x:

- *pendant vertex* (*pv*): add y adjacent only to x.
- *true twin* (*tt*): add y as a true twin to x.
- *restricted false twin* (*rft*): add y as a false twin to x if x is simplicial.

Theorem 1 ([1]). *A graph is ptolemaic if and only if it can be obtained from a single vertex by recursively applying the operations pv, tt, and rft.*

In this paper we will also use the following new characterization of ptolemaic graphs found by Kloks [22]:

Lemma 1 ([22]). *A graph is ptolemaic if and only if all its connected induced subgraphs are a clique or contain a pair of true twins or contain a cut vertex.*

Interval graphs, i.e., the intersection graphs of intervals on the real line, are another important subclass of strongly chordal graphs (see e.g. [5]). If the intervals have unit length then their intersection graphs are called *unit interval graphs.*

Subsequently, we also need the following notions: A graph is a *split graph* if its vertex set has a partition into a clique and an independent set. A *caterpillar* T is a tree consisting of a path (the *backbone* of T) and some leafs attached to the backbone.

We frequently use the weighted version of a k-leaf root which is defined as follows. A *weighted k-leaf root* of a finite undirected graph $G = (V, E)$ is a tree $T = (V_T, E_T)$ where V is the set of leaves and edges have weights $\omega(e)$, $e \in E_T$ such that for any two vertices $x, y \in V$, $xy \in E$ if and only if the sum of the edge weights on the path between x and y is at most k.

We say that a leaf root T is *basic* if at most one leaf is attached to each node of T, and leaf power G is *basic* if it has a basic leaf root. Obviously, any set of leaves with the same parent is a clique module whenever $k \geq 2$, and thus, every k-leaf power G results from a basic k-leaf power G' by substituting cliques into the vertices of G'.

Let $G = (V, E)$ be a leaf power and $T = (V_T, E_T)$ a leaf root of G. For all distinct vertices $u, v \in V_T$ we denote by $T[u, v]$ the path between u and v in T. For any $U \subseteq V$ we also denote by $T[U]$ the subtree of T *spanned by* U, i.e., the union of the paths $T[u, v]$ for all pairs $u, v \in U$. Proposition 1 collects some useful facts on leaf powers (see e.g. [3,6]):

Proposition 1

(i) *Every induced subgraph of a k-leaf power, $k \geq 2$, is a k-leaf power.*
(ii) *A graph is a k-leaf power if and only if each of its connected components is a k-leaf power.*
(iii) *Graph G is a basic $(k+2)$-leaf power if and only if G is an induced subgraph of the k-th power T^k of a tree T.*

(iv) *For every $k \geq 1$, graph G is a k-leaf power if and only if G results from a basic k-leaf power G' by substituting cliques into the vertices of G'.*

Subsequently, we need the following notion from [7]: A graph $G = (V, E)$ is a (k, l)*-leaf power* if there is a tree T with leaf set V such that for all $xy \in E$, $d_T(x, y) \leq k$, and for all $xy \notin E$, $d_T(x, y) \geq l$.

Theorem 2 ([7]). *For graph G, the following conditions are equivalent:*

(i) *G is strictly chordal;*
(ii) *G is a $(4, 6)$-leaf power;*
(iii) *G results from a block graph by substituting cliques into its vertices.*

It is known that the class of strongly chordal graphs is closed under powers:

Theorem 3 ([12,24,28]). *If G is strongly chordal then for every $k \geq 1$, G^k is strongly chordal.* □

Moreover, strongly chordal graphs are closed under substitution of cliques. Since trees are strongly chordal, by Proposition 1 (iii), basic leaf powers are induced subgraphs of powers of trees and induced subgraphs of strongly chordal graphs are strongly chordal, Theorem 3 and Proposition 1 imply:

Proposition 2. *Leaf powers are strongly chordal.*

Thus, it is natural to ask whether this is a strict inclusion. The following theorem is based on results in [8] and [2].

Theorem 4. *There are strongly chordal graphs which are no k-leaf power for any $k \geq 2$.*

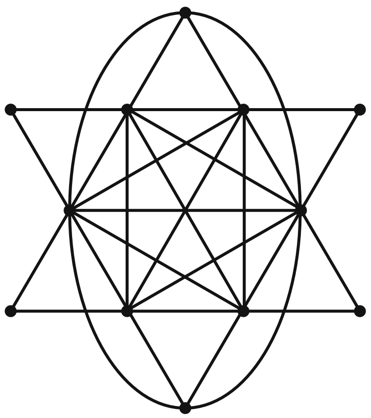


Fig. 2. A strongly chordal graph which is no k-leaf power for any k

3 Ptolemaic Graphs Are Leaf Powers

In this section we use Lemma 1 to introduce the following simple recursive construction of a basic $(2k, 2k+2)$-leaf root $T = (V_T, E_T)$ for a given ptolemaic graph $G = (V, E)$ and a number $k \geq |V|$:

Construction 1. *For given $G = (V, E)$ and $k \geq |V|$, this construction builds as an intermediate step a weighted $(2k, 2k+2)$-leaf root T for G. If G contains only a single vertex v, then G is itself the weighted $(2k, 2k+2)$-leaf root T. If G is a clique we construct a weighted $(2k, 2k+2)$-leaf root T which contains one internal node r parent to all leaves V of G and for all $v \in V$ the edge rv has the weight $\omega(rv) = k$. If G is not a clique but contains a*

pair x, y of true twins, *we recursively apply Construction 1 on the graph $G' = G[V \setminus \{y\}]$ and the same number k to build a weighted $(2k, 2k+2)$-leaf root T' for G'. Let p_x be the parent node of x in T'. We obtain a weighted $(2k, 2k+2)$-leaf root T for G by attaching y as a new leaf to p_x and setting $\omega(p_x y) = \omega(p_x x)$.*

cut vertex x, *we recursively apply Construction 1 on each of the components $G'_1 = (V'_1, E'_1), \ldots, G'_\ell = (V'_\ell, E'_\ell)$ of G incident to x and the same number k to build weighted $(2k, 2k+2)$-leaf roots $T'_1, \ldots, T'_\ell$ for $G'_1, \ldots, G'_\ell$. Define c_1 and c_2 to be the two largest cardinalities of the sets V'_1 to V'_ℓ. For all $i \in \{1, \ldots, \ell\}$ we identify the parent node p_i of x in the tree T'_i, replace the node x by a new internal node p_x and set $\omega(p_i p_x) = \omega(p_i x) - (k - (c_1 + c_2) + 3)$. Then we merge the trees $T'_1, \ldots, T'_\ell$ to a single tree T' by identifying their p_x nodes. We obtain from T' a weighted $(2k, 2k+2)$-leaf root T for G by appending the leaf x to its parent node p_x and setting $\omega(p_x x) = k - (c_1 + c_2) + 3$.*

Finally, to construct a basic $(2k, 2k+2)$-leaf root we replace in T all edges e by a path of length $\omega(e)$.

Theorem 5. *Every ptolemaic graph $G = (V, E)$ is a basic $(2|V|, 2|V| + 2)$-leaf power, and a corresponding basic $(2|V|, 2|V| + 2)$-leaf root $T = (V_T, E_T)$ can be obtained by Construction 1 with given $k = |V|$.*

Proof. For Construction 1 with given ptolemaic graph $G = (V, E)$ and $k \geq |V|$ we show by induction on n that (a) T is a weighted $(2k, 2k+2)$-leaf root for G and (b) that for any leaf $v \in V$ with parent node p_v in T it is true $k - |V| + 2 \leq \omega(p_v v) \leq k$. This is trivially true for the base cases of G being a single vertex or a clique. Otherwise, if G contains a

pair x, y of true twins, then by induction hypothesis T' is a weighted $(2k, 2k+2)$-leaf root for $G' = G[V \setminus \{y\}]$ which fulfills the above weight criterion. Now consider the tree T. Since $\omega(p_x x) = \omega(p_x y) \leq k$, it follows that x and y see each other and have equal T-distances to all other leaves. Therefore, T is a weighted $(2k, 2k+2)$-leaf root for G which fulfills the weight criterion (even for y).

cut vertex x**,** then by induction hypothesis $T'_1, \dots, T'_\ell$ are weighted $(2k, 2k+2)$-leaf roots for the components $G'_1 = (V'_1, E'_1), \dots, G'_\ell = (V'_\ell, E'_\ell)$ of G incident to x such that for all $i \in \{1, \dots, \ell\}$ it is true $k - |V'_i| + 2 \leq \omega(p_v v) \leq k$ for all leaves v in T'_i and p_v the parent node of v. Now consider the tree T. Trivially, for all $i \in \{1, \dots, \ell\}$ the T'_i-distance between nodes $u, v \in V'_i \setminus \{x\}$ equals the T-distance between u and v. Moreover the T-distance between x and $v \in V'_i \setminus \{x\}$ is

$$\begin{aligned} d_T(v, x) &= d_T(v, p_i) + \omega(p_i p_x) + \omega(p_x x) \\ &= d_T(v, p_i) + \omega(p_i x) - (k - (c_1 + c_2) + 3) + (k - (c_1 + c_2) + 3) \\ &= d_T(v, p_i) + \omega(p_i x) = d_{T'_i}(v, p_i) + \omega(p_i x) = d_{T'_i}(v, x) \end{aligned}$$

and hence, equals the T'_i-distance between x and v.
For T being a weighted $(2k, 2k+2)$-leaf root for G it remains to show that for all $1 \leq i < j \leq \ell$, all $u \in V'_i \setminus \{x\}$ and all $v \in V'_j \setminus \{x\}$ the T-distance between u and v is at least $2k+2$. Since $(c_1 + c_2) \geq (V'_i + V'_j)$ it follows:

$$\begin{aligned} d_T(u, v) &= d_T(u, p_i) + \omega(p_i p_x) + \omega(p_x p_j) + d_T(p_j, v) \\ &\geq \omega(p_u u) + \omega(p_i x) + \omega(p_j x) + \omega(p_v v) \quad - \quad 2(k - (c_1 + c_2) + 3) \\ &\geq 2(k - |V'_i| + 2) \quad + \quad 2(k - |V'_j| + 2) \quad - \quad 2(k - (c_1 + c_2) + 3) \\ &= 2k - 2(|V'_i| + |V'_j|) + 2(c_1 + c_2) + 2 \geq 2k + 2 \end{aligned}$$

Moreover, T fulfills the weight criterion for pendant edges since $\omega(q_x x) = k - (c_1 + c_2) + 3 \geq k - |V| + 2$ which follows from $(c_1 + c_2 - 1) \leq |V|$.

Finally, T is basic after weighted edges have been replaced by paths. Since $k \geq |V|$ it follows for all leaves v with parent node p_v that $\omega(p_v v) \geq k - |V| + 2 \geq 2$. Hence, each leaf has a unique parent node. □

Theorem 5 has the following two implications:

Corollary 1. *Let $G = (V, E)$ be a ptolemaic graph.*

(i) *The leaf rank of G is at most $2|V|$.*
(ii) *G is a k-leaf power for all $k \geq 2|V|$.*

4 Ptolemaic Graphs Have Unbounded Leaf Rank

In this section we show that the class of ptolemaic graphs has unbounded leaf rank. Hence, in contrast to strictly chordal graphs there is no natural number k such that for all ptolemaic graphs G, $lr(G) \leq k$ holds. In particular, we will define a sequence G_i, $i = 1, 2, \dots$ of ptolemaic graphs with linear lower bound. We consider the following graphs:

Definition 1. *Let $G_0 = (\{x_0\}, \emptyset)$ be the graph with only one vertex x_0. For all $k \in \mathbb{N}$, G_k results from G_{k-1} by*

(i) *adding a true twin* y_k *of* x_{k-1} *and*
(ii) *adding a false twin* x_k *of* y_k.

Let $X_k = \{x_0, \dots, x_k\}$ *and* $Y_k = \{y_1, \dots, y_k\}$. *Obviously, for all* k, X_k *and* Y_k *give a partition of the vertex set of* G_k.

Lemma 2. *Every graph* $G_k, k \in \mathbb{N}$, *is a ptolemaic split graph with partition into clique* X_k *and independent set* Y_k.

Since all $G_k, k \in \mathbb{N}$, are ptolemaic, they are also leaf powers. In the following we provide a recursive construction giving a $(k+2)$-leaf root for each graph G_k:

Construction 2. *For* G_1 *to* G_4 *we have depicted the* 3- *to* 6-*leaf roots in Figure* 3. *Now for* $k > 4$ *we assume that* T' *is a* $(k+1)$-*leaf root for* G_{k-1} *with the following properties. Let* $T'[X_{k-1}]$ *denote the subtree of* T' *spanned by* X_{k-1}.

($*$) *If* $k = 2i+1$ *then the diameter of* $T'[X_{k-1}]$ *is* $2i+2$ *and* $T[x_{2i}, x_{2i-1}]$ *is a diametral path; let* z *denote a central node of* $T'[X_{k-1}]$. *Then* $d_{T'}(z, x_j) \leq i+1$ *for all* $j \in \{1, \dots, 2i\}$, *and* $d_{T'}(z, x_j) = i+1$ *for* $j = 2i$ *and* $j = 2i-1$.
($**$) *If* $k = 2i$ *then the diameter of* $T'[X_{k-1}]$ *is* $2i+1$ *and* $T[x_{2i-2}, x_{2i-1}]$ *is a diametral path; let* $z_1 z_2$ *denote a central edge of* $T'[X_{k-1}]$ *such that* z_1 *is closer to* x_{2i-2} *and* z_2 *is closer to* x_{2i-1}. *Then* $d_{T'}(z_1, x_{2i-2}) = d_{T'}(z_2, x_{2i-1}) = i$.

Then a $(k+2)$-*leaf root* T *for* G_k *is constructed from* T' *as follows:*
First we subdivide all pendant edges in T' *with leaves* $y_j \in Y_{k-1}$, $j \in \{1, \dots, k-1\}$ *by a new internal node. Then we add the nodes* x_k *and* y_k *as leaves of new paths in the following way: If* k *is*

odd, *i.e.,* $k = 2i+1$, *then* $T'[X_{k-1}]$ *has a central node* z. *We attach new paths* P_{x_k} (P_{y_k}, *respectively*) *of length* $i+2$ *with new internal nodes and* x_k (y_k, *respectively*) *as leaf to* z.
even, *i.e.,* $k = 2i$, *then* $T'[X_{k-1}]$ *has a central edge* $z_1 z_2$. *We attach a new path* P_{x_k} (P_{y_k}, *respectively*) *of length* $i+1$ *with new internal nodes and* x_k (y_k, *respectively*) *as leaf to* z_1 (z_2, *respectively*).

Lemma 3. *For all* $k \in \mathbb{N}$, *Construction* 2 *gives a* $(k+2)$-*leaf root* T *of* G_k.

Lemma 4 says that the upper bound in Lemma 3 cannot be improved.

Lemma 4. *Let* $k \in \mathbb{N}$. *If* T *is a leaf root of* G_k, *then there exist two adjacent vertices* $x_s, x_t \in X_k$ *such that* $d_T(x_s, x_t) \geq k+2$.

The following is a straightforward implication of Lemma 4:

Corollary 2. *The leaf rank of ptolemaic graphs is unbounded.*

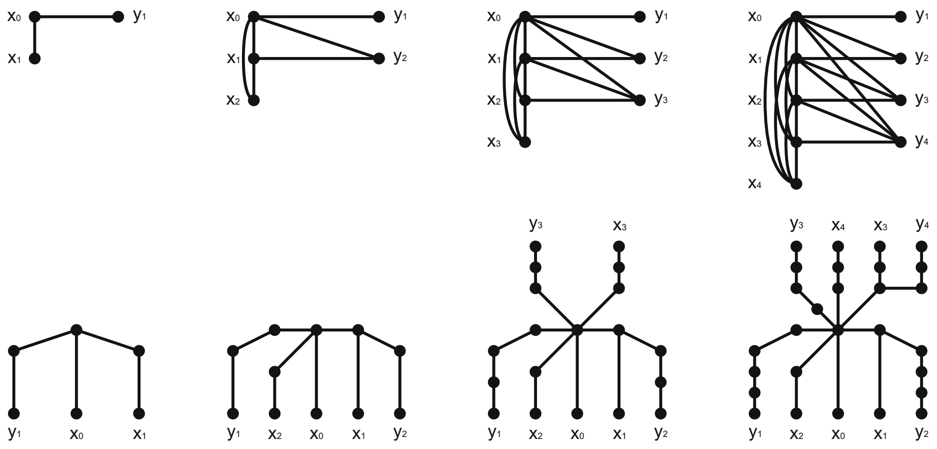


Fig. 3. The first row shows the graphs G_1, G_2, G_3 and G_4. The second row depicts the 3-, 4-, 5- and 6-leaf roots obtained by Construction 2.

5 Interval Graphs Are Leaf Powers

Interval graphs are an important subclass of strongly chordal graphs and interestingly they are leaf powers.

Theorem 6. *Interval graphs are leaf powers (but not vice versa).*

Proof. Let $G = (V, E)$ be an interval graph with interval model $(I_v)_{v \in V}$, and let m_v denote the midpoint of interval I_v. Without loss of generality, we can assume that

(i) for all $1 \le i < j \le n$ it is true $m_{v_i} < m_{v_j}$,
(ii) all midpoints have rational values and
(iii) the lengths of all intervals are at most one and have rational values.

Obviously, $uv \in E$, if and only if the two points m_u and m_v on the real line have distance at most $\frac{|I_u|+|I_v|}{2}$. From the interval model we construct a weighted caterpillar $T' = (V_{T'}, E_{T'})$ with leaf set V such that $uv \in E$ if and only if the distance of the two leaves u and v is at most one. The backbone of T' is the path $(p_1, p_2, \ldots, p_n)$ and we attach to each node $p_i, i \in \{1, \ldots, n\}$ the leaf v_i. Now we define the edge weights $\omega : E_{T'} \to \mathbb{R}^+$: For all $i \in \{1, \ldots, n-1\}$ let $\omega(p_i p_{i+1}) = m_{v_{i+1}} - m_{v_i}$ and for all $i \in \{1, \ldots, n\}$ let $\omega(p_i v_i) = \frac{1-|I_{v_i}|}{2}$.

Let $v_i, v_j \in V$ with $i < j$ and assume that $v_i v_j \in E$, i.e., $m_{v_j} - m_{v_i} \le \frac{|I_{v_i}|+|I_{v_j}|}{2}$. The distance between the leaves v_i and v_j in T' is

$$\omega(p_i v_i) + m_{v_j} - m_{v_i} + \omega(p_j v_j) = 1 + m_{v_j} - m_{v_i} - \frac{|I_{v_i}| + |I_{v_j}|}{2} \le 1.$$

Conversely, if the T'-distance between v_i and v_j is at most one then

$$\frac{1-|I_{v_i}|}{2} + m_{v_j} - m_{v_i} + \frac{1-|I_{v_j}|}{2} - 1 = m_{v_j} - m_{v_i} - \frac{|I_{v_i}|+|I_{v_j}|}{2} \leq 0$$

and hence, $m_{v_j} - m_{v_i} \leq \frac{|I_{v_i}|+|I_{v_j}|}{2}$. Thus, for all $u, v \in V$ the two intervals I_u and I_v intersect if and only if the T'-distance between leaves u and v is at most one.

Let N be the least common multiple of the values given by the denominators of the edge weights $\omega(e), e \in E_{T'}$ and let $k = N + 2$. We obtain a k-leaf root $T = (V_T, E_T)$ of G from T' if we replace for all $i \in \{1, \ldots, n-1\}$ the edge $p_i p_{i+1}$ by a path of length $N \cdot \omega(e)$ and replace for all $i \in \{1, \ldots, n\}$ the edge $p_i v_i$ by a path of length $N \cdot \omega(e) + 1$ with leaf v_i. It is easy to see that $uv \in E$ if and only if $d_T(u, v) \leq k$. Trees are leaf powers but in general no interval graphs. □

Recently, the clique-width of graphs, as an important width measure on graphs, has attracted considerable attention. It generalizes treewidth and leads to efficient algorithms [11]. In [17], it was shown that the clique-width of ptolemaic graphs is at most three and hence, it is natural question to ask whether leaf powers have bounded clique-width. Golumbic and Rotics [17] showed that unit interval graphs have unbounded clique-width and thus, as a byproduct, Theorem 6 also shows the following:

Corollary 3. *Leaf powers have unbounded clique-width.*

From a result of Todinca [30], which implies that k-leaf powers have bounded clique-width for every fixed k, it follows that unit interval graphs have unbounded leaf rank. See also [18] for upper bounds on clique-width of k-leaf powers.

An interesting subclass of leaf powers are the graphs which have caterpillar leaf roots. It turns out that this is exactly the class of unit interval graphs.

Theorem 7. *For a connected graph G, the following conditions are equivalent:*

(i) *G has a leaf root which is a caterpillar.*
(ii) *G results from an induced subgraph of the power of some path by substituting cliques.*
(iii) *G is a unit interval graph.*

Proof. Let $G = (V, E)$ be a connected graph.

(i) $\Longrightarrow$ (ii): Let caterpillar T be a k-leaf root of G for some $k \geq 3$ (without loss of generality, we can assume that T is a basic leaf root) and let B be the backbone path of T. For every $v \in V$ let b_v denote the backbone parent of v. Now, $uv \in E$ if and only if $d_T(u, v) \leq k$ if and only if $d_B(b_u, b_v) \leq k - 2$. This shows that G is an induced subgraph of B^{k-2}. If more than one vertex of G is attached to a backbone node of T then corresponding clique substitutions give the desired graph.

(ii) $\Longrightarrow$ (iii): Again, without loss of generality, assume that G itself is an induced subgraph of the k-th power of some path P_l. In [25], Theorem 3.8 says that a graph is a proper interval graph if and only if it is a unit interval graph, and Theorem 3.10 says that G is a proper interval graph if and only if the vertex-maxclique incidence matrix $M(G)$ has the consecutive ones property for both rows and columns (which is mentioned in [13] and goes back to [29]). Obviously, powers of paths have the last property, and the property is hereditary for induced subgraphs.

(iii) $\Longrightarrow$ (i): Let $G = (V, E)$ be a unit interval graph with interval model $(I_v)_{v \in V}$, and let m_v denote the midpoint of interval I_v. Without loss of generality, we can assume that the midpoints have rational values. Let N be the least common multiple of the denominators of $m_v, v \in V$. By definition, $uv \in E$ if and only if $d(m_u, m_v) \leq 1$. Thus, by multiplying the midpoints by N, we obtain a path containing n midpoints m'_v such that $uv \in E$ if and only if $d(m'_u, m'_v) \leq N$. This is the backbone B of a caterpillar T where we attach a leaf v to midpoint m'_v such that $uv \in E$ if and only if $d_T(u, v) \leq N + 2$. $\square$

6 Discussion and Outlook

In this paper, we investigated the relationship of the class of leaf powers to ptolemaic, (unit) interval and strongly chordal graphs. Meanwhile, Peter Wagner has shown that rooted directed path graphs, which contain ptolemaic graphs as well as interval graphs, are a proper subclass of leaf powers, too. It remains for future work to pinpoint leaf powers between these classes. In particular we hope that our results will lead to characterizations and efficient recognition algorithms for leaf powers.

Far less is known on how to find the leaf rank of a graph. In this paper we gave narrow upper and lower bounds on how the leaf rank of ptolemaic graphs grows linearly with the number of vertices. Moreover we showed that ptolemaic and interval graphs have unbounded leaf rank.

Acknowledgement

The first author thanks Chính T. Hoàng for discussions on the relationship between leaf powers and strongly chordal graphs and gratefully acknowledges partial support by Wilfrid Laurier University. Moreover, thanks go to Peter Wagner for a discussion on powers of paths and clique-width and to Dieter Rautenbach for a remark and the reference [2] on constant neighborhood subtree tolerance graphs.

References

1. Bandelt, H.J., Mulder, H.M.: Distance hereditary graphs. J. Combinatorial Theory (B) 41, 182–208 (1986)
2. Bibelnieks, E., Dearing, P.M.: Neighborhood subtree tolerance graphs. Discrete Applied Math. 43, 13–26 (1993)

3. Brandstädt, A., Le, V.B.: Structure and linear time recognition of 3-leaf powers. Information Processing Letters 98, 133–138 (2006)
4. Brandstädt, A., Le, V.B., Rautenbach, D.: Distance-hereditary 5-leaf powers (manuscript, 2006)
5. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. In: SIAM Monographs on Discrete Math. Appl., vol. 3, SIAM, Philadelphia (1999)
6. Brandstädt, A., Le, V.B., Sritharan, R.: Structure and linear time recognition of 4-leaf powers (manuscript, 2006)
7. Brandstädt, A., Wagner, P.: On (k, ℓ)-Leaf Powers. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 525–535. Springer, Heidelberg (2007)
8. Broin, M.W., Lowe, T.J.: A dynamic programming algorithm for covering problems with (greedy) totally balanced constraint matrices. SIAM J. Alg. Disc. Meth. 7, 348–357 (1986)
9. Chang, M.-S., Ko, T.: The 3-Steiner Root Problem. In: Extended abstract in: Proceedings 33rd International Workshop on Graph-Theoretic Concepts in Computer Science WG 2007. LNCS, vol. 4769, pp. 109–120. Springer, Heidelberg (2007)
10. Chen, Z.-Z., Jiang, T., Lin, G.: Computing phylogenetic roots with bounded degrees and errors. SIAM J. Computing 32, 864–879 (2003)
11. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique width. Theory of Computing Systems 33, 125–150 (2000)
12. Dahlhaus, E., Duchet, P.: On strongly chordal graphs. Ars Combinatoria 24 B, 23–30 (1987)
13. Deogun, J.S., Gopalakrishnan, K.: Consecutive Retrieval Property - Revisited. Information Processing Letters 69, 15–20 (1999)
14. Niedermeier, R., Guo, J., Hüffner, F., Dom, M.: Error Compensation in Leaf Root Problems. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 389–401. Springer, Heidelberg (2004); Algorithmica 44(4), 363-381 (2006)
15. Fagin, R.: A painless introduction. In: Protasi, M., Ausiello, G. (eds.) CAAP 1983. LNCS, vol. 159, pp. 65–89. Springer, Heidelberg (1983)
16. Farber, M.: Characterizations of strongly chordal graphs. Discrete Math. 43, 173–189 (1983)
17. Golumbic, M., Rotics, U.: On the clique-width of some perfect graph classes. International J. Foundat. Computer Science 11(3), 423–443 (2000)
18. Gurski, F., Wanke, E.: The clique-width of tree powers and leaf-power graphs. Extended abstract In: Proceedings 33rd International Workshop on Graph-Theoretic Concepts in Computer Science WG 2007. LNCS, vol. 4769, pp. 76–85. Springer, Heidelberg (2007)
19. Howorka, E.: A characterization of distance-hereditary graphs. Quart. J. Math. Oxford 2(28), 417–420 (1977)
20. Howorka, E.: A characterization of ptolemaic graphs. J. Graph Theory 5, 323–331 (1981)
21. Kennedy, W., Lin, G., Yan, G.: Strictly chordal graphs are leaf powers. J. Discrete Algorithms 4, 511–525 (2006)
22. Kloks, T.: Private communication (2007)
23. Lin, G.-H., Kearney, P.E., Jiang, T.: Phylogenetic k-root and Steiner k-root. In: Lee, D.T., Teng, S.-H. (eds.) ISAAC 2000. LNCS, vol. 1969, pp. 539–551. Springer, Heidelberg (2000)
24. Lubiw, A.: Γ-free matrices, Master Thesis, Dept. of Combinatorics and Optimization, University of Waterloo, Canada (1982)

25. McKee, T.A., McMorris, F.R.: Topics in Intersection Graph Theory. In: SIAM Monographs on Discrete Math. Appl., vol. 2, SIAM, Philadelphia (1999)
26. Nishimura, N., Ragde, P., Thilikos, D.M.: On graph powers for leaf labeled trees. J. Algorithms 42, 69–108 (2002)
27. Rautenbach, D.: Some remarks about leaf roots. Discrete Math. 306(13), 1456–1461 (2006)
28. Raychaudhuri, A.: On powers of strongly chordal and circular arc graphs. Ars Combinatoria 34, 147–160 (1992)
29. Roberts, F.S.: Representations of Indifference Relations, Ph.D. thesis, Standford University, Standford, CA (1968)
30. Todinca, I.: Coloring powers of graphs of bounded clique-width. In: Bodlaender, H.L. (ed.) WG 2003. LNCS, vol. 2880, pp. 370–382. Springer, Heidelberg (2003)

A Representation Theorem for Union-Difference Families and Application[⋆]
(Extended Abstract)[⋆⋆]

B.-M. Bui-Xuan[1] and M. Habib[2]

[1] CNRS - LIRMM - Université Montpellier II
161 rue Ada, 34392 Montpellier Cedex 05, France
buixuan@lirmm.fr
[2] CNRS - LIAFA - Université Paris Diderot
Paris 7 Case 7014, 75205 Paris Cedex 13, France
habib@liafa.jussieu.fr

Abstract. We give a quadratic $O(|X|^2)$ space representation based on a canonical tree for any subset family $\mathcal{F} \subseteq 2^X$ closed under the union and the difference of its overlapping members. The cardinality of $\mathcal{F}$ is potentially in $O(2^{|X|})$, and the total cardinality of its members even higher. As far as we know this is the first representation result for such families. As an application of this framework we obtain a unique digraph decomposition that not only captures, but also is strictly more powerful than the well-studied modular decomposition. A polynomial time decomposition algorithm for this case is described.

1 Introduction

Many combinatorial decompositions lead to interesting subset families, such as crossing families for minimum cuts in network flows theory (see e.g. [21]), and partitive families for modular decomposition in graph theory [4,10,19]. Cross-free families as defined in [21] using the famous Edmonds-Giles's theorem [9] admit a tree structure and arise in many combinatorial decompositions such as the split decomposition [6,7,8,18] and also in phylogeny [22].

For a given set family $\mathcal{F} \subseteq 2^X$, it is worth studying its distance from a tree structure, namely to examine if it can be represented via a tree. Such a representation must allow the enumeration of all members of the family in $O(|\mathcal{F}|)$ time. Let us define the complexity of a family as the size of its minimal tree. At first level one can find simple hierarchies (c.f. laminar in [21]) and cross-free families. Then, (weakly) partitive families which admit a unique tree decomposition with 3 types of nodes (prime, complete and linear) also have complexity $O(|X|)$. For crossing families only a representation tree in $O(|X|^2)$ space is known [13].

This paper deals with union-difference families – families closed under the union and the difference of its overlapping elements – which is a natural generalisation

⋆ Research supported by the ANR project *Graph Decompositions and Algorithms*.
⋆⋆ Full version available at http://hal-lirmm.ccsd.cnrs.fr/lirmm-00175766.

of partitive families. We show the existence of a canonical tree representation in $O(|X|^2)$ space. Furthermore, from this we obtain a new polynomial and unique decomposition of directed graphs, generalising modular decomposition. A polynomial time decomposition algorithm for this case is then depicted in the last section.

2 Representation Theorem

Let X be a finite set. Two sets A and B *overlap*, denoted by $A \between B$, if none among $A \cap B$, $A \setminus B$, and $B \setminus A$ is empty. They *cross*, if we have both $A \between B$ and $\overline{A} \between \overline{B}$, where $\overline{A} = X \setminus A$. A family $\mathcal{F} \subseteq 2^X$ is a *union-difference* family if: $\mathcal{F}$ contains the *trivial members* X and $\{x\}$ (for all $x \in X$), and $\mathcal{F}$ is closed under the union and the difference of its overlapping members. If a union-difference family is also closed under the symmetric difference of its overlapping members, then it is closed under the intersection of its overlapping members too. Union-difference-intersection families are well-studied under the name of *partitive families* [4], and are fundamental for modular graph decomposition [10,19].

Henceforth $\mathcal{F}$ is a union-difference family. Notice that if $|X| \leq 2$ then $\mathcal{F} = 2^X$ and representing $\mathcal{F}$ is trivial. We assume throughout the paper that $|X| \geq 3$. $A \in \mathcal{F}$ is a *strong member of* $\mathcal{F}$ if A does not overlap any $B \in \mathcal{F}$. Likewise, $A \in \mathcal{F}$ is a *semi-strong member of* $\mathcal{F}$ if it does not cross any $B \in \mathcal{F}$. Let $\mathcal{S} \subseteq \mathcal{F}$ be the family of semi-strong members of $\mathcal{F}$. For sake of simplicity, X is *excluded* from $\mathcal{S}$ although it is clearly semi-strong. By definition, no two members of $\mathcal{S}$ cross, and $\mathcal{S}$ is called *cross-free.*

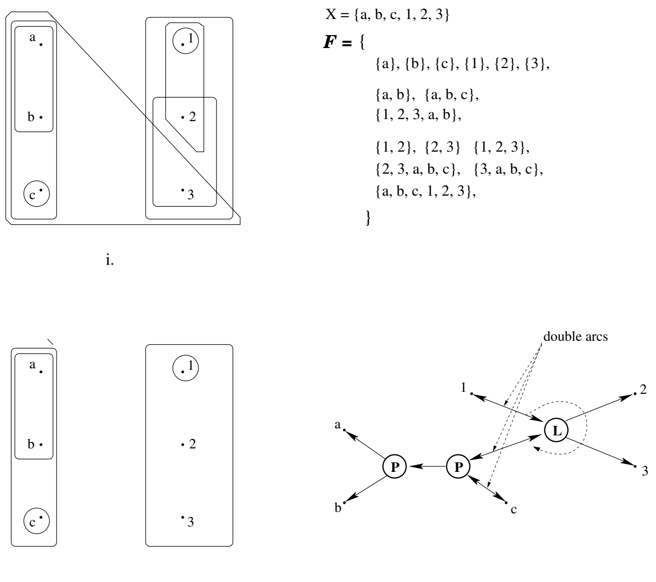


Fig. 1. i. A union-difference family, circles represent their complement. ii. The semi-strong subfamily excluding X. iii. Decomposition tree.

Let us now recall a cross-free family representation [9] which is widely used in combinatorial optimisation research areas (refer to e.g. [21]). Let $x \in X$, we consider $\mathcal{S}' = \{A \mid A \in \mathcal{S} \wedge x \notin A\} \cup \{\overline{A} \mid A \in \mathcal{S} \wedge x \in A\}$. No two members of $\mathcal{S}'$ overlap, and their inclusionwise ordering results in a tree rooted at $X \setminus \{x\}$. We then add x to the children of the root and unroot the tree. The set of leaves is now in bijection with X: by abusiveness we confound the two sets. In this tree, deleting any edge gives rise to two connected components. If each component is regarded as the set of its leaves, then at least one of them is a member of $\mathcal{S}$. Then, edge orientation can denote which ones belong to $\mathcal{S}$ (see Fig. 1). On the other hand, each member of $\mathcal{S}$ corresponds to one edge of the tree.

Definition 1 (Decomposition tree). *We define the* decomposition tree *of a union-difference family $\mathcal{F} \subseteq 2^X$ as the Edmonds-Giles's tree representation [9] of its semi-strong members, X is excluded. Such a tree has no degree 2 node.*

We shall label this tree to obtain an enumerating object of all members of $\mathcal{F}$. In the tree, the deletion of an internal node n gives rise to $k = d(n)$ connected components, which can also be seen as a $k-$partition of X. Let $\{X_1, \ldots, X_k\}$ denote this partition. For instance, the node labelled "L" in Fig. 1 yields $\{\{1\}, \{2\}, \{3\}, \{a, b, c\}\}$. For later use, notice that we always have $k \geq 3$. (This can also be seen as a quick proof that the unlabelled decomposition tree is of linear $O(|X|)$ size.) Let us consider $Y = \{X_1, \ldots, X_k\}$ as a set, and define the *quotient of $\mathcal{F}$ with respect to node n* as the family $\mathcal{Q}(n) \subseteq 2^Y$ such that

$$\begin{cases} \{X_i\} \text{ belongs to } \mathcal{Q}(n) \text{ for all } 1 \leq i \leq k, \\ Q = \{X_i \mid i \in I\} \text{ with } |Q| \neq 1 \text{ belongs to } \mathcal{Q}(n) \;\Leftrightarrow\; \cup_{i \in I} X_i \text{ belongs to } \mathcal{F}. \end{cases}$$

The membership of X_i in $\mathcal{F}$ (resp. exclusion of X_i from $\mathcal{F}$) is *already* stored by the edge orientation of the decomposition tree. Roughly, each member Q of the quotient $\mathcal{Q}(n)$ corresponds to one and only one member of $\mathcal{F}$, except for the singletons $\{X_i\}$. Moreover, it is not obvious but folklore that the converse holds:

Proposition 1. *For all member $A \in \mathcal{F}$ of a subset family $\mathcal{F} \subseteq 2^X$, there exists a node n in the semi-strong tree of $\mathcal{F}$ such that A corresponds to a member of the quotient $\mathcal{Q}(n)$ of $\mathcal{F}$ with respect to n. This node is unique.*

Consequently, if there is a way to describe $\mathcal{Q}(n)$ for every node n of the decomposition tree, then one can rebuild the initial family $\mathcal{F}$ in an exact manner. As a step towards this aim, we say that a member A of $\mathcal{F}$ is *quasi-trivial* if $|A| = |X| - 1$, and notice a second non obvious but folklore fact:

Proposition 2. *Trivial and quasi-trivial members are semi-strong by vacuity. On the other hand, any semi-strong member of a quotient $\mathcal{Q}(n) \subseteq 2^Y$ is either trivial or quasi-trivial.*

Remark 1. Both Propositions 1 and 2 hold for arbitrary subset families.

Definition 2 (Quotient property). *We say that a subset family satisfies the* quotient property *if all its semi-strong members are either trivial or quasi-trivial.*

Obviously the quotient of a union-difference family is also a union-difference family. We thus focus on families satisfying both union-difference and quotient properties, which form a super-class of the quotient nodes of a union-difference decomposition tree. We shall prove that there are at most 5 types of them. Moreover, each type will be proved to be of "small enough" size, that is

Main Representation Theorem. *There is a node-labelling of the decomposition tree of a union-difference family $\mathcal{F} \subseteq 2^X$ such that every member of $\mathcal{F}$ can be retrieved from the tree and its labels. Moreover, the size of the tree and its labels is in $O(|X|^2)$ space.*

Proof. We shall consider two main categories. *Simply-linked* quotients (see further in Definition 3) will be characterised by Theorem 1 into 4 types. Section 2.2 addresses the remaining ones. Theorem 2 proves the quadratic global size. $\square$

Before continuing, let us highlight a useful tool from previous works on partitive families. A subset family $\mathcal{F} \subseteq 2^X$ can also be seen as an undirected hypergraph with vertex set X. Let us define the $2-$*graph of* $\mathcal{F}$ as its restriction to size 2 hyperedges: $G_{\mathcal{F}} = (X, E)$ with $E = \{A \in \mathcal{F}$ and $|A| = 2\}$. Though the following property was discovered for partitive families, its proof only requires the union and difference closures. (The proof given in [10] is recalled in the full version [1].)

Lemma 1. *(c.f. [4,10] with partitive families) Let $\mathcal{F}$ be a union-difference family. If its $2-$graph $G_{\mathcal{F}}$ is connected then $G_{\mathcal{F}}$ is either a clique, a path, or a cycle.*

2.1 Simply-Linked Quotients

We first focus on a case of "easy" decomposition, fully exploiting Lemma 1. While a quasi-trivial member is clearly semi-strong, it is not necessarily strong. Moreover, we say that

Definition 3 (Simply-linked Property). *A subset family is* simply-linked *if none of its quasi-trivial members is strong.*

For simply-linked quotients, the following nice theorem holds. A family is *prime* if it has only trivial and quasi-trivial members.

Theorem 1. *If a union-difference family $\mathcal{F}$ satisfies both quotient and simply-linked properties, then one and only one of the following holds:*

- *$G_{\mathcal{F}}$ is a clique (we say that $\mathcal{F}$ is complete),*
- *$G_{\mathcal{F}}$ is a path (we say that $\mathcal{F}$ is linear),*
- *$G_{\mathcal{F}}$ is a cycle of length at least 4 (we say that $\mathcal{F}$ is circular),*
- *$\mathcal{F}$ is prime.*

Proof. First we have to prove the two lemmas 2 and 3 (below). Then, notice by Lemma 1 that if $G_{\mathcal{F}}$ is connected, it is either a clique, a path, or a cycle. $\square$

By union closure, $G_{\mathcal{F}}$ is a clique if and only if $\mathcal{F} = 2^X$, and we say that $\mathcal{F}$ is *complete*. Likewise, $G_{\mathcal{F}}$ is a path (resp. cycle) if and only if there is a linear (resp. circular) ordering of X such that $\mathcal{F}$ is exactly the family of all intervals (resp. circular intervals) of this ordering. $\mathcal{F}$ is then *linear* (resp. *circular*).

Corollary (Representing simply-linked quotients). Let $X_1, \ldots, X_k$ denote the resulting connected components of a decomposition tree when deleting a quotient node. Representing a *complete* quotient node is easily done with $O(1)$ label, stating the quotient is the family of every union of some X_i's. For a *linear* or *circular* node, we also need to code an ordering on the incident edges. Then, an $O(1)$ label can state the quotient is the family of every union of some consecutive X_i's (Fig. 1 gives an illustration on the node labelled "L"). Except for the special case of X, members of a *prime* quotient node are *already* stored in the edge orientation of the decomposition tree (they are semi-strong, and belong to $\mathcal{S}$). Accordingly, we only need an $O(1)$ label for all prime nodes, stating there are no members bound to the node other than those given by the edge orientation.

Let us head back to the proof of Theorem 1. A *chain* of length k of $\mathcal{F}$ is a sequence $(A_1, \ldots, A_k)$ of members of $\mathcal{F}$ such that $A_i \oplus A_{i+1}$ for all i, and $A_i \cap A_j = \emptyset$ for all $|i-j| > 1$. The chain is *covering* if $A_1 \cup \cdots \cup A_k = X$, and *irreducible* if $|A_i| = 2$ for all $1 \leq i \leq k$. An irreducible and covering chain of $\mathcal{F}$ can also be seen as a Hamiltonian path in the $2-$graph $G_{\mathcal{F}}$, which would imply its connectivity, and enable the use of Lemma 1.

Lemma 2. *If a union-difference family $\mathcal{F}$ satisfies both quotient and simply-linked properties, then either $\mathcal{F}$ is prime, or $\mathcal{F}$ has a length 3 covering chain.*

Proof. Suppose that $\mathcal{F}$ is not prime, and let $A \in \mathcal{F}$ be neither trivial nor quasi-trivial. We take A maximal by inclusion. The quotient property provides us with $B \in \mathcal{F}$ such that A and B cross. The closure under union implies $A \cup B \in \mathcal{F}$. Moreover, A is maximal. Hence $A \cup B$ is either trivial or quasi-trivial. However, $A \cup B$ cannot be trivial since A and B cross. Then, the simply-linked property implies that $A \cup B$ is not strong. Hence it overlaps some member $C \in \mathcal{F}$. Here, all cases lead to either $D = C \cup B \setminus A$ or $E = C \cup A \setminus B$ is a member of $\mathcal{F}$. Then, either (A, B, D) or (B, A, E) is a covering chain of length 3. □

Lemma 3. *If a union-difference family $\mathcal{F}$ satisfies both quotient and simply-linked properties, and has a covering chain of length at least 3, then $\mathcal{F}$ has an irreducible covering chain (then $G_{\mathcal{F}}$ is connected).*

By lack of space, the proof of Lemma 3 is omitted. Please refer to the full version [1] for any detail.

2.2 Other Quotients

We now address a family $\mathcal{F} \subseteq 2^X$ that is not simply-linked. By definition it has a quasi-trivial member that is strong. We note $Y = X \setminus \{x\}$ that member. Since Y is strong, except for X and $\{x\}$, $\mathcal{F}$ has no other member containing x. Let us consider the sub-family $\mathcal{G} = \mathcal{F} \setminus \{X, \{x\}\}$, which holds $\mathcal{G} \subseteq 2^Y$. Obviously, if $\mathcal{F}$ is a union-difference family, so is $\mathcal{G}$. Fainthearted, we represent $\mathcal{F}$ with the member

$\{x\}$ and the union-difference decomposition tree of $\mathcal{G}$. We process the same way with all quotient nodes that are not simply-linked. Therefore, such a tree may have recursive levels. Fortunately enough, its total size still is polynomial:

Theorem 2. *The global size of the labelled decomposition tree of a given union-difference family $\mathcal{F} \subseteq 2^X$ is in $O(|X|^2)$.*

Proof. By induction on $n = |X|$. Let $f(n)$ be the maximum size of all decomposition trees of n leaves. Obviously, $f(1)$ and $f(2)$ are non null constants. Let $f(k) \leq \alpha \times k^2$ hold for all $k < n$. We suppose without loss of generality that α is greater than any other constant in this proof. Let us consider a decomposition tree of n leaves and let N be the set of its internal nodes. For each $i \in N$, let n_i be its degree. The label of i is either of constant size (c.f. prime and complete nodes), of linear size on n_i (c.f. linear and circular nodes), or of size bounded by $f(n_i - 1) + \beta$ (c.f. nodes that are not simply-linked). In all cases, it is bounded by $\alpha \times (n_i - 1)^2 + \alpha$ since $n_i \geq 3$ and $\alpha \geq \beta$. The total size of leaves, edges, and orientations is linear on n, hence bounded by $\alpha \times n$. We deduce that

$$f(n) \leq \alpha \times \left(\sum_{i \in N} ((n_i - 1)^2 + 1) + n \right) \leq \alpha \times \left(\sum_{i \in N} (n_i - 1)^2 + n' + n \right),$$

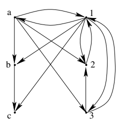
where $n' = |N|$. Notice that $\sum_{i \in N} n_i = n + 2 \times (n' - 1)$ (the n pendant edges are counted once while other edges are counted twice). In other words, $S = \sum_{i \in N} (n_i - 1) = n + n' - 2$. Then, the greatest value that $\sum_{i \in N} (n_i - 1)^2$ can reach happens when one among the n_i gets the greatest value possible. Since $n_i - 1 \geq 2$, we have $\sum_{i \in N} (n_i - 1)^2 \leq (n' - 1) \times 2^2 + (S - (n' - 1) \times 2)^2$. Then, $f(n) \leq \alpha \times (n^2 + n'^2 + 5n' + n(1 - 2n') - 4)$. Besides, that there are no degree 2 nodes in the tree provides us with $n \geq n' + 2$. Finally, combining the previous facts and $1 - 2n' \leq 0$ allows to conclude. □

Conjecture: *There is for every union-difference family a representation of smaller space than quadratic on the size of the ground set. The reason for the conjecture comes from the brute-force aspects of this section.*

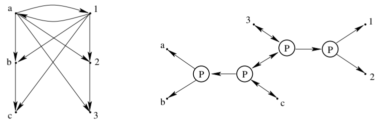
3 Application to Graphs: Sesquimodular Decomposition

In graph theory modular decomposition is now a well-studied notion [4,10,15,19], as well as some of its generalisations [6,7,8,17,18,20]. As having been rediscovered in other fields, the notion also appears under various names, including intervals, externally related sets, autonomous sets, partitive sets, and clans. Direct applications of modular decomposition include tractable constraint satisfaction problems [5], computational biology [14], graph clustering for network analysis, and graph drawing. This rich research field lays heavily on the nice combinatorial properties of modules. Among most important ones, that modules form a partitive family allows representing them compactly with a tree [4,10,19].

Besides, in the area of social networks, several vertex partitioning have been introduced in order to catch the idea of putting in the same part all vertices

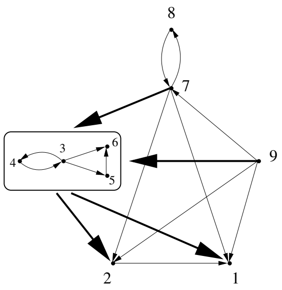


(a) The family of sesquimodules of this digraph is the one of Fig. 1 (refer therein for the decomposition tree). Notice that $\{2, 3\}$ is also a module.

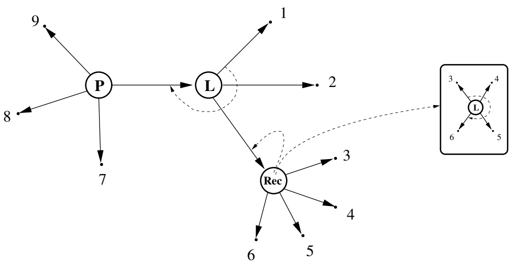


(b) A modular prime digraph with its sesquimodular decomposition tree.

Fig. 2. Modules v.s. sesquimodules



(a) A directed graph, bold arcs denote all-to-all arcs.



(b) Its sesquimodular decomposition tree.

Fig. 3. Sesquimodular decomposition. The family of non-trivial sesquimodules of the digraph is $\{\{1, 2\}, \{1, 2, 3, 4, 5, 6\}, \{2, 3, 4, 5, 6\}, \{2, 3, 4, 5, 6, 7, 8, 9\}, \{3, 4, 5, 6, 7, 8, 9\}, \{3, 4\}, \{3, 4, 5\}, \{3, 4, 5, 6\}, \{4, 5\}, \{4, 5, 6\}, \{5, 6\}\}$.

acknowledging similar behaviour, in other words finding regularities [23]. Modular decomposition provides such a partitioning, yet seemingly too restrictive for real life applications. The concept of a role [11] on the other hand seems promising, however its computation is unfortunately $NP-$hard [12]. As a natural consequence, there is need for the search of *relaxed*, but *tractable*, variations of the modular decomposition scheme. We here investigate the case of directed graphs, and propose a weakened definition of module in order to further decompose. Fortunately enough, we still obtain a well-structured variation, thanks to union-difference families.

Digraphs here refer to loopless simple directed graphs where $2-$cycles are allowed. Let $G = (V, A)$ be a digraph, $M \subseteq V$ is a *sesquimodule* if:

- $\forall x, y \in M$, $N^-(x) \setminus M = N^-(y) \setminus M$, and
- $\forall x, y \in M$, either $N^+(x) \setminus M = N^+(y) \setminus M$ or $N^+(x) \setminus M = \overline{N^+(y)} \setminus M$.

In an undirected graph, there is only one requirement to be a module, which is $\forall x, y \in M$, $N(x)\setminus M = N(y)\setminus M$. The classical generalisation to directed graphs requires *two full conditions*, one on in-neighbours and one on out-neighbours: $\forall x, y \in M$, $N^-(x) \setminus M = N^-(y) \setminus M$ and $N^+(x) \setminus M = N^+(y) \setminus M$. In the new definition, there is a full condition on in-neighbours, and a relaxed one on out-neighbours: the exterior still has to be partitioned into out-/non-out-neighbour vertices, however, their order is irrelevant. This is the reason for the terminology. Fig. 2(a) exemplifies an instance where the sesquimodules form the family given in Fig. 1, while Fig. 2(b) shows that the generalisation of modules to sesquimodules is proper. A more complex example of sesquimodular decomposition tree is given in Fig. 3. We have the following theorem.

Theorem 3 (Uniqueness Decomposition Theorem). *There is a unique unrooted tree associated to a digraph $G = (V, A)$ such that: the leaves of the tree are in one-to-one correspondence with the vertices of G; the edges of the tree are oriented; the internal nodes of the tree are marked with at most 4 types of labels; and all sesquimodules of G can be generated from this tree without the knowledge of the graph. The size of this tree and its labels is in $O(|V|^2)$.*

This theorem lays on the simple fact that

Proposition 3. *The sesquimodules of a digraph form a union-difference family. Furthermore there are no circular nodes in its decomposition tree.*

Proof. Let $G = (V, A)$ be a digraph. Clearly, the trivial vertex subsets are sesquimodules of G. Let X and Y be two overlapping sesquimodules of G. It follows straight from definition that $X \cup Y$ is a sesquimodule. We only need to prove that $Z = X \setminus Y$ is also a sesquimodule.

First suppose that there exist an exterior vertex $s \notin Z$ and two vertices $x, y \in Z$ s.t. $(s, x) \in A$ and $(s, y) \notin A$. We shall denote arc $(x, y) \in A$ by xy, and non-arc $(x, y) \notin A$ by $\overline{xy}$. Since X is a sesquimodule s belongs to $X \cap Y$. Moreover, that X and Y overlap implies there is a vertex t belonging to $Y \setminus X$. Notice that $s, t \in Y$ and $x, y \notin Y$. Additionally, we have sx and $\overline{sy}$. Since Y is a sesquimodule, we have either $tx \wedge \overline{ty}$ or $\overline{tx} \wedge ty$. But then X no more is a sesquimodule as $t \notin X$ and $x, y \in X$. Hence, for all $x, y \in Z$, $N^-(x) \setminus Z = N^-(y) \setminus Z$.

Now let $x, y \in Z$ and $s, t \notin Z$. For convenience, we refer to the fact that either $xs \wedge xt$ or $\overline{xs} \wedge \overline{xt}$ by "*x is not a splitter of* $\{s, t\}$", denoted by $x|st$. We need to prove that $x|st \Leftrightarrow y|st$. If none of s and t belong to X, that X is a sesquimodule allows to conclude. If both s and t belong to Y, that Y is a sesquimodule allows to conclude. By symmetry, the only remaining case is when $s \in X \cap Y$ and $t \notin X \cup Y$. In this case, let $u \in Y \setminus X$. Since X is a sesquimodule, we already have $x|tu \Leftrightarrow y|tu$, but we would like the same property with vertex u replaced by vertex s. For this, notice that $x \notin Y$, but $s, u \in Y$, and Y is a sesquimodule. Therefore, $x|su$. Likewise, $y|su$. Then, combining the two latter facts and $x|tu \Leftrightarrow y|tu$ leads to the desired property.

Finally, a circular sesquimodule quotient node would be a complete one. □

Remark 2. A 2−structure is roughly an edge-coloured complete digraph (see e.g. [10]). Graph modules can be generalised to *clans* of a 2−structure: a vertex subset M of a 2−structure is a clan if for all $x, y \in M$, for all $s \notin M$, the arcs (s, x) and (s, y) are of same colour, and the arcs (x, s) and (y, s) are also of same colour (though the two colours may differ). Likewise, graph sesquimodules can also be generalised to sesquimodules of a 2−structure as follows. M is a sesquimodule of a 2−structure if it holds two following conditions. For all

$x, y \in M$, for all $s \notin M$, the arcs (s, x) and (s, y) are of the same colour. For all $x, y \in M$, for all $s, t \notin M$, (x, s) and (x, t) are of the same colour if and only if (y, s) and (y, t) are of the same colour. Then, one can check that the family of sesquimodules of any $2-$structure is a union-difference family. Hence, for $2-$structures we have a similar decomposition theorem as what has been said for digraphs. However, the algorithm described in the next section wont apply.

Remark 3. We newly pointed out that the family of sesquimodules of a digraph is also closed under the intersection of its *crossing* members. Based on this, we improved the representation of the sesquimodules of a digraph $G = (V, A)$ from $O(|V|^2)$ (as in this paper) down to an $O(|V|)$ space decomposition tree. We also showed that the latter tree can be computed in polynomial time. All these new results can be found in [2]. This does not apply to the family of sesquimodules of a $2-$structure since such a family could fail the closure under intersection of its crossing members.

4 Polynomial Time Algorithm for Sesquimodular Decomposition

This section describes a brute-force algorithm to compute in polynomial time the sesquimodular decomposition tree of a given digraph $G = (V, A)$. We divide the computation into two main steps, generalising the two-step scheme introduced by [3] for modular graph decomposition.

Definition 4 (Factoring Permutation). *[3] A* factoring permutation *of a decomposition tree is the visit order of the leaves of the underlying decomposition tree by some depth-first graph search.*

For sesquimodular decomposition tree, which is unrooted, we define the factoring permutation as a circular permutation. This notion dues its name to the fact that every node of the tree is a (circular) interval of the (circular) permutation. In the following, results of Section 4.1 can not be used unless we meet a certain notion of *splitter*. However, all the remaining is a scheme that can be used to compute the semi-strong tree of any arbitrary subset family. Indeed, the union-difference property will only be used for eventually typing the nodes.

4.1 Computing a Factoring Permutation

Like modular decomposition, we use a partition refinement technique (c.f. [16]) based on the notion of a *splitter*. There are two kinds of sesquimodule splitters:

– If there are $s \notin M$ and $x, y \in M$ with $(s, x) \in A$ (denoted by sx) and $(s, y) \notin A$ (denoted by $\overline{sy}$) then M is not a sesquimodule. We say that vertex s *splits* x and y.
– If there are $x, y \in M$ such that there are $s, t \notin M$ with $x|st$ and $\overline{y|st}$ then M is not a sesquimodule, where $x|st$ denotes $(xs \wedge xt) \vee (\overline{xs} \wedge \overline{xt})$ and $\overline{x|st}$ denotes its negation. We say that vertex pair (x, y) is a *self-splitter*.

Let us consider first category splitters. We begin with picking a vertex $x \in V$ and considering the *ordered* partition $\mathcal{P} = \{\overline{N^+(x)}, \{x\}, N^+(x)\}$, which will be seen as an ordered circular partition. We then perform a *round*, which consists of performing the refinement operation (see right below) for all vertex $y \neq x$ until this modifies the partition $\mathcal{P}$. The round ends and we restart a new one whenever $\mathcal{P}$ is modified. If, through some whole round, the partition $\mathcal{P}$ remains unchanged for all $y \neq x$, then the process stops.

The refinement operation w.r.t. a vertex $y \neq x$ consists of splitting each part P of the partition $\mathcal{P}$ into two parts: $P^+ = P \cap N^+(y)$ and $\overline{P^+} = P \setminus N^+(y)$. The reorganisation of the split parts is as follows. Though obviously the part $Q = \{x\}$ of the partition $\mathcal{P}$ is not split, we have to consider whether $x \in N^+(y)$. Let P and R be the neighbour parts of Q in $\mathcal{P}$: $\mathcal{P} = (\dots, P, Q, R, \dots)$. If $x \in N^+(y)$, replace P with $(\overline{P^+}, P^+)$ and replace R with $(R^+, \overline{R^+})$, else, replace P with $(P^+, \overline{P^+})$ and R with $(\overline{R^+}, R^+)$. If there are some empty set, we act as if they were present, but skip storing them to the partition list. Thus, the elements that y "sees" the same way as how y "sees" x are locally stick together. We do the same processing for the parts before P and those after R in $\mathcal{P}$ (elements of same vision by y are locally stick together). That there is in the initial partition an odd number of parts – actually 3 – guarantees no conflict when closing the circle.

At this point, we obtain a partition $\mathcal{P}$ such that for all vertex v, and for all part P of the partition $\mathcal{P}$, v is not a first category splitter of P. Since each refinement can be done in $O(|N^+(y)|)$ time (see [16]), a round takes $O(n^2)$ time, where $n = |V|$. Since each round decreases the partition $\mathcal{P}$ to a thinner partition, there are at most n rounds. The total time is in $O(n^3)$.

We now consider second category splitters, with the computed partition $\mathcal{P}$. While there is in $\mathcal{P}$ some part P containing a *self-splitter* $(x, y) \subseteq P$, replace P with (P_x, P_y), which is defined as follows. First, push x in P_x and y in P_y. Let $s, t \notin P$ such that $x|st$ and $\overline{y|st}$. Then, for every other vertex z of P, either $z|st$ or $\overline{z|st}$, and we push z in P_x or P_y accordingly. Testing for *self-splitters* can be done by just testing all vertex quadruplets. This would globally cost $O(n^5)$ time.

At the end, we obtain a circularly ordered partition $\mathcal{P} = (P_1, \dots, P_k)$ of unordered parts P_i's. Then, ordering arbitrary the P_i's results in a circular permutation of V, which is a factoring permutation.

4.2 Computing the Decomposition Tree

We here constantly need to test if a subset is member of the initial family. Let τ denotes the time for such a test. For digraphs, given a vertex subset, we can test in $\tau = O(n^4)$ time if the subset is member of the sesquimodule family by checking every vertex quadruplets. Then, the shape of the decomposition tree can easily be constructed in a brute-force manner as follows. Compute a factoring permutation. For each interval of the factoring permutation, test if it is a member of the initial family. For each pair of the latter members, if they cross, then remove both. Represent the remaining members in a cross-free tree-representation as explained at the beginning of Section 2. Since there are at most n^2 intervals in a circular permutation, these operations take $O(n^5 + n^2\tau)$ time.

The only remaining thing is to type the nodes. The main difficulty is how to test for nodes that are not simply-linked. Actually, we avoid this test by elimination of cases. For each internal node i of the decomposition tree:

Compute the $2-$graph of the quotient w.r.t. node i (quadratic number of tests for membership). If this is a clique, a path, or a cycle, conclude accordingly, and stop. Compute all quasi-trivial members of the quotient. If there are more than one or none of such, report a prime node, and stop. Else either the node is prime or it is not simply-linked with that unique quasi-trivial member which is strong. Let $\{c\}$ be the complement of the unique quasi-trivial member. Assume node i is not simply-linked and recursively compute the decomposition tree of the quotient excluding $\{c\}$ (refer to Section 2.2 for details). If the latter tree is anything except a single prime node then node i effectively was not simply-linked, we conclude and stop. The latter tree is a single prime node. If there is some quasi-trivial member therein then node i effectively was not simply-linked, we conclude and stop. Otherwise node i was simply-linked. We report a prime node.

Without recursive calls the process is in $O(n^3\tau) = O(n^7)$ time. Then, an inductive argument similar to the proof of Theorem 2 gives an $O(n^8)$ time bound.

Theorem 4. *The sesquimodular decomposition tree of a given digraph $G = (V, A)$ can be computed in $O(|V|^8)$ time.*

5 Conclusion and Perspectives

We have shown that union-difference families can be represented via a unique tree, and applied this result to a new directed graph decomposition. Of course the polynomial decomposition algorithm proposed here for this variation of modular decomposition has to be improved for a practical use. Another interesting investigation could be on the properties of the family of complements of members of a union-difference family. Such a family owns a quadratic representation straight from the result of union-difference families. However, their intrinsic properties are unclear, as the closure under difference does not behave symmetrically via complementary. Besides, representing families satisfying a number of closure operations remains an interesting question, and we are convinced that some other combinatorial decompositions can be expressed in this framework, as in [2].

Acknowledgements. The first author is grateful to S. Thomassé for helpful discussions and pointers.

References

1. http://hal-lirmm.ccsd.cnrs.fr/lirmm-00175766
2. Bui-Xuan, B.-M., Habib, M., Rao, M.: Representing partitive crossing Families and union-difference Families, with Application to Sesquimodular Decomposition. Available at: http://hal-lirmm.ccsd.cnrs.fr/lirmm-00199916

3. Capelle, C., Habib, M.: Graph decompositions and factorizing permutations. In: 5th Israel Symposium on Theory of Computing and Systems (ISTCS 1997), pp. 132–143. IEEE Computer Society, Los Alamitos (1997)
4. Chein, M., Habib, M., Maurer, M.C.: Partitive hypergraphs. Discrete Mathematics 37(1), 35–50 (1981)
5. Cohen, D.A., Cooper, M.C., Jeavons, P.G.: Generalising submodularity and Horn clauses: Tractable optimization problems defined by tournament pair multimorphisms. Technical Report CS-RR-06-06, Oxford University (2006)
6. Crespelle, C.: Représentations dynamiques de graphes. PhD thesis, Université Montpellier II (2007)
7. Cunningham, W.: Decomposition of directed graphs. SIAM Journal on Algebraic and Discrete Methods 3, 214–228 (1982)
8. Cunningham, W., Edmonds, J.: A combinatorial decomposition theory. Canadian Journal of Mathematics 32, 734–765 (1980)
9. Edmonds, J., Giles, R.: A min-max relation for submodular functions on graphs. Annals of Discrete Mathematics 1, 185–204 (1977)
10. Ehrenfeucht, A., Harju, T., Rozenberg, G.: The Theory of 2-Structures- A Framework for Decomposition and Transformation of Graphs. World Scientific, Singapore (1999)
11. Everett, M.G., Borgatti, S.P.: Regular Equivalence: General Theory. Journal of Mathematical Sociology 18, 29–52 (1994)
12. Fiala, J., Paulusma, D.: A complete complexity classification of the role assignment problem. Theoretical Computer Science 349(1), 67–81 (2005)
13. Gabow, H.: Centroids, Representations, and Submodular Flows. Journal of Algorithms 18(3), 586–628 (1995)
14. Gagneur, J., Krause, R., Bouwmeester, T., Casari, G.: Modular decomposition of protein-protein interaction networks. Genome Biology 5(8) (2004)
15. Gallai, T.: Transitiv orientierbare Graphen. Acta Mathematica Academiae Scientiarum Hungaricae 18, 25–66 (1967)
16. Habib, M., Paul, C., Viennot, L.: Partition refinement techniques: An interesting algorithmic tool kit. International Journal of Foundations of Computer Science 10(2), 147–170 (1999)
17. Hsu, W.-L., McConnell, R.M.: PC-trees and circular-ones arrangements. Theoretical Computer Science 296, 99–116 (2003)
18. Lanlignel, J.-M.: Autour de la décomposition en coupes. PhD thesis, Université Montpellier II (2001)
19. Möhring, R.H., Radermacher, F.J.: Substitution decomposition for discrete structures and connections with combinatorial optimization. Annals of Discrete Mathematics 19, 257–356 (1984)
20. de Mongolfier, F., Rao, M.: The bi-join decomposition. In: 7th International Colloquium on Graph Theory (ICGT 2005) (2005)
21. Schrijver, A.: Combinatorial Optimization - Polyhedra and Efficiency. Springer, Heidelberg (2003)
22. Semple, C., Steel, M.: Phylogenetics. Oxford Lecture Series in Mathematics and Its Applications, vol. 24 (2003)
23. White, D.R., Reitz, K.P.: Graph and Semigroup Homomorphisms on Networks of Relations. Social Networks 5, 193–234 (1983)

Algorithms to Locate Errors Using Covering Arrays[⋆]

Conrado Martínez[1], Lucia Moura[2], Daniel Panario[3], and Brett Stevens[3]

[1] Universitat Politècnica de Catalunya
[2] University of Ottawa
[3] Carleton University

Abstract. In this paper, we define error locating arrays (ELAs), which can be used to locate faulty interactions between parameters or components in a software system. We give constructions of ELAs based on covering arrays. Under certain assumptions on the structure of the faulty interactions, we design and analyse efficient algorithms that locate errors. Under the assumption of known "safe values", our algorithm performs a number of tests that is polynomial in $\log k$ and d, where k is the number of parameters in the system and d is an upper bound on the number of faulty pairwise interactions. For the binary alphabet case, we provide an algorithm that does not require safe values and runs in expected polynomial time in $\log k$ whenever $d \in O(\log \log k)$.

1 Introduction

Consider a complex system whose behavior depends on the values of k parameters or *factors*. To temporarily simplify matters, suppose each of the k factors may take any of two values. In order to exhaustively test the system, 2^k tests are required, rendering it infeasible in a practical setting, even when k is only moderately large.

An alternative to exhaustive testing is provided by covering arrays. A *binary covering array* (CA) is a 0-1 matrix with n rows and k columns. Each of its columns represents a parameter and each of its rows gives a test to be performed. The number of rows, n, is called the *size* of the array. The array is said to be of *strength* t if for any t-subset of the k factors, the corresponding columns exhaustively cover all possible 2^t combinations. In other words, if we define a t-way interaction to be the assignment of specific values to each factor from set of t factors, a covering array tests each t-way interaction in some of its rows.

[⋆] Part of this research was done while the first author was on sabbatical leave at Carleton Univ. and later while the second and third authors were on sabbatical leave at Univ. Politècnica de Catalunya. C. Martínez was supported by projects ALINEX (TIN2005-05446 and TIN2006-11345) of the Spanish Min. of Education and Science. L. Moura was supported by NSERC. D. Panario was supported by NSERC and the Spanish Min. of Education and Science. B. Stevens was supported by NSERC, CFI, OIT, and Ontario MRI.

System:

factors:	printer	file format	colours	file size (MBs)
values:	0 =P1	0 = JPEG	0 =black & white	0 = (≤ 50)
	1 =P2	1 = PDF	1 = colour	1 = (> 50 and < 500)
		2 = PS		2 = (≥ 500)

factors:	(1) printer	(2) file format	(3) colours	(4) file size	outcome
test 1	0	0	0	0	pass
test 2	0	0	1	1	fail
test 3	1	0	1	2	fail
test 4	1	1	1	0	pass
test 5	1	1	0	1	pass
test 6	0	1	0	2	fail
test 7	0	2	0	0	fail
test 8	0	2	1	1	pass
test 9	1	2	1	2	fail

Fig. 1. An MCA covering all pairwise interactions in a printing system

Since in many practical settings it is enough to test 2-, 3- or 4-way interactions, we can tackle these problems with a moderately sized CA. The minimal size of a binary covering array of strength t for k factors is denoted by $CAN(t, k, 2)$. An important result with practical implications establishes that this number grows logarithmically in k (see [2] for binary arrays).

There is a vast literature on covering arrays, their properties and efficient constructions (see Colbourn's survey [7]). They have a wide range of applications, in particular software and hardware testing [9,12,13,15,19], genomics [18] and material sciences [6]. Besides the simplified version of CAs presented above, there are immediate generalizations when each factor can take any of g different values, or when each factor i has g_i possible values, the so-called *mixed covering arrays* (MCAs) [17]. Another generalization of CAs and MCAs targets situations where some combinations need not [16] or must not be tested [10].

CAs are useful in software interaction testing, where individual software components have been thoroughly tested but interaction among the components can cause faults [22]. Empirical results show that testing all pairwise interactions in a software system finds most of its faults [9,12,13]; some authors also link pairwise coverage to good "code coverage" [5,11]. While there are situations in which considering t-way interactions for $t > 2$ give additional benefits [12], in this extended abstract we concentrate on pairwise interactions, i.e., $t = 2$.

Figure 1 shows a toy example of a system with $k = 4$ factors. Two of the factors can have two possible values, while the other two can take any of three different values. Testing exhaustively such a system would require $36 = 2 \times 2 \times 3 \times 3$ tests, but an MCA of size 9 allows us to test all pairwise interactions. We also record in the rightmost column the (alleged) outcome of each test.

In this study, our goal is to define new combinatorial designs that help us not only to test the system (i.e., to determine whether there are interactions that cause failures) but also to identify and to determine which are the faulty interactions causing the failing tests. For instance, in the test suite given in Figure 1, does test 2 fail because we printed a JPEG file (0) in colour (1), or because we printed a JPEG file (0) of medium size (1)? Or is this failure caused by a faulty 3-way interaction?

In a recent paper, Colbourn and McClary introduce $(\overline{d}, t)$-locating arrays and $(\overline{d}, t)$-detecting arrays. They are special types of CAs that allow for the location of faulty t-way interactions, provided that a bound d on the number of faulty interactions is given. These arrays are built to locate any set of up to d errors, which unfortunately makes their existence very limited to small d with respect to the number of values g_i for each factor [8]. For instance, if all factors are binary, there exist no detecting arrays if there is more than one pairwise faulty interaction.

In this paper, we introduce *error locating arrays* (ELAs), a generalization of $(\overline{d}, t)$-detecting arrays in which we guarantee the location and detection of errors whenever the structure, rather than the number, of the faulty interactions satisfies some assumptions. In particular, since in this extended abstract we only consider pairwise interactions, the faulty interactions can be conveniently modeled as the edges of a graph. We study the existence and efficient construction of ELAs when the structure of the errors belongs to particular families of graphs. Furthermore, we consider here two types of testing: *non-adaptive testing*, when we have to construct the full test suite in advance, without prior knowledge of the outcome of any test; and *adaptive testing*, when the choice of each new row/test can be based on the outcome of all previous tests. While in [8] the parameter d is required to be constant in order to achieve a number of tests that is polynomial in $\log k$, our results allow d to grow as $O(\log \log k)$ and, in the case of one algorithm, as $O(\log k)$ to achieve polynomial growth in $\log k$.

We summarize our contributions and the organization of the paper in what follows. In Section 2, we give some definitions and background. In Section 3, we study conditions for the existence of ELAs and for their logarithmic growth in k. We introduce the notion of *locatable graphs*, the most general family of graphs for which ELAs can exist. We also consider a special family of graphs, which we call *graphs with safe values*, and which turns out to be always locatable. Briefly, these graphs are such that every factor has a value for which the corresponding vertex is not part of any edge/error. In the testing application, this models a situation in which each parameter has a value not present in any faulty interaction. We observe that for a random error graph with $d \in o(k)$ the probability that the graph has safe values goes to 1 as $k \to \infty$ [14]. Therefore, the assumption that a graph has safe values is not too restrictive in an asymptotic sense. In Section 4, we give an adaptive algorithm that locates faults when safe values are known; for this particular algorithm the graph may have loops, which corresponds to errors given by single factor values. The algorithm perform tests adaptively, which corresponds to building an ELA for the error graph, and performs $O(d(\log k)^2 +$

$d^2 \log k$) tests. While this algorithm is very efficient when we know safe values, this may be too strong an assumption in many testing situations. Ultimately, we would like to eliminate such a strong assumption, which in this paper is accomplished for the binary case in Section 5. We characterize the family of locatable graphs when $g_i = g = 2$ for all i. Using this characterization, we give an algorithm that adaptively builds tests that locate all errors, for any locatable "binary" graph. This algorithm performs an expected number of tests in $O(d \log k + (\log k)^{2c'})$, when $d \leq c' \log\log k$, for some constant c'. In Section 6, we discuss some open problems.

2 Definitions and Preliminaries

Let us define the testing problem more formally. Consider a system with k *factors* (parameters or components) $1, \ldots, k$. Each factor i can take one of g_i possible *values*, which we consider w.l.o.g. to be in the set $\{0, \ldots, g_i - 1\}$, denoted by $[0, g_i - 1]$. A *test* is an assignment of values to factors, i.e., a k-tuple in $[0, g_1 - 1] \times \cdots \times [0, g_k - 1]$. The execution of a test can have two outcomes: *pass* or *fail*; we call it a *passing* or a *failing* test, respectively. An *interaction* is a set of values assigned to distinct factors: $I = \{(f_1, a_1), \ldots, (f_t, a_t)\}$, $f_i \neq f_j$ for $i \neq j$, and $a_i \in [0, g_{f_i} - 1]$, $1 \leq i \leq t$. An interaction I is a *t-way interaction* if $|I| = t$. We say that a test (or a k-tuple) $T = (T_1, \ldots, T_k)$ *covers* interaction $I = \{(f_1, a_1), \ldots, (f_t, a_t)\}$, if $T_{f_i} = a_i$ for $1 \leq i \leq t$. Thus, a test covers exactly $\binom{k}{t}$ t-way interactions, $1 \leq t \leq k$. We assume that failures are caused by faulty interactions, that is, the execution of a test fails if and only if it covers one or more faulty interactions. Covering arrays are combinatorial designs that correspond to test suits that cover all t-way interactions of factor values, and consequently all s-way interactions with $1 \leq s \leq t$.

Definition 1. *A* mixed covering array*, A, denoted by $MCA(n; t, (g_1, \ldots, g_k))$, is an $n \times k$ array, such that each column i (corresponding to a factor) has values from the alphabet $[0, g_i - 1]$, and every possible t-way interaction is covered by some row, or in other words, for every t-set of factors $\{f_1, \ldots, f_t\}$ and every t-tuple of values $(a_1, \ldots, a_t) \in [0, g_{f_1} - 1] \times \cdots \times [0, g_{f_t} - 1]$, there exists at least one row r (corresponding to a test) such that $A[r, f_j] = a_j$ for all $j \in [1, t]$. Given t and $g_1, \ldots, g_k$, the* mixed covering array number*, denoted by $MCAN(t, (g_1, \ldots, g_k))$, is the smallest n for which an $MCA(n; t, (g_1, \ldots, g_k))$ exists. When $g_i = g$ for all $1 \leq i \leq k$, we call the objects simply* covering arrays *and simplify the notation to $CA(n; t, k, g)$, and $CAN(t, k, g)$.*

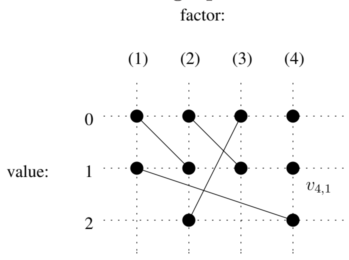
The test suits in Figures 1 and 2 give examples of $MCA(9; 2, (2, 3, 2, 3))$. Since $g_2 g_4 = 9$ is a lower bound for n, this gives $MCAN(2, (2, 3, 2, 3)) = 9$.

Consider a graph whose edges represent the faulty pairwise interactions in a system. Let $G = G_{(g_1, \ldots, g_k)}$ denote a k-partite simple graph with k parts of sizes $g_1, \ldots, g_k$, respectively. The vertices of G are indexed by i, a_i where $i \in [1, k]$ and $a_i \in [0, g_i - 1]$. If $g_1 = \cdots = g_k = g$, then we simplify the notation to $G = G_{k,g}$. Denote by $G \setminus (e = \{v, w\})$ the graph with vertex set $V(G)$ and edge

Faulty interactions:

printer P1 with file format PDF: $\{(1,0),(2,1)\}$,
printer P2 with file size ≥ 500: $\{(1,1),(4,2)\}$,
file format JPEG with colour: $\{(2,2),(3,0)\}$,
file format PS with black&white: $\{(2,0),(3,1)\}$.

Error graph G:



factors:	(1) printer	(2) file format	(3) colours	(4) file size	outcome
test 1	**0**	**1**	1	0	fail
test 2	1	**0**	**1**	0	fail
test 3	1	**2**	**0**	0	fail
test 4	**1**	1	0	**2**	fail
test 5	0	0	0	2	pass
test 6	0	2	1	2	pass
test 7	1	1	1	1	pass
test 8	0	0	0	1	pass
test 9	1	2	1	1	pass

Fig. 2. Error graph G and a test suit corresponding to an $ELA(9;G)$

set $E(G) \setminus \{e\}$. A k-tuple $T = (T_1,\ldots,T_k) \in [0,g_1-1] \times \cdots \times [0,g_k-1]$ is said to *avoid* $G = G_{(g_1,\ldots,g_k)}$ if for all $i,j \in [1,k]$, we have $\{v_{i,T_i}, v_{j,T_j}\} \notin E(G)$. We say that an interaction $\{(i,a),(j,b)\}$ is *locatable* if there exists a k-tuple T with $T_i = a$ and $T_j = b$ that avoids $G \setminus (\{v_{i,a}, v_{j,b}\})$, if $\{v_{i,a}, v_{j,b}\} \in E(G)$, or avoids G, otherwise. We say that T *locates* interaction $\{(i,a),(j,b)\}$ with respect to G, and that $\{(i,a),(j,b)\}$ is *located* by T. A graph is *locatable* if all pairwise interactions are locatable.

Definition 2. *An* error-locating array *for $G = G_{(g_1,\ldots,g_k)}$ is an $n \times k$ array, A, with each column i having symbols from the alphabet $[0,g_i-1]$, and denoted by $ELA(n;G)$, such that each pairwise interaction $\{(i,a),(j,b)\}$, $1 \leq i < j \leq k$, $0 \leq a < g_i$, $0 \leq b < g_j$, is located by some row of A. If $\mathcal{G}$ is a family of graphs with each $G \in \mathcal{G}$ of the form $G = G_{(g_1,\ldots,g_k)}$, an* error-locating array *for $\mathcal{G}$, denoted by $ELA(n;\mathcal{G})$, is simply an $ELA(n;G)$ for all $G \in \mathcal{G}$.*

It is easy to see that there exists an error-locating array for G if and only if G is locatable. Also, every $ELA(n;G)$ is a covering array with $t = 2$. In Figure 2, we show a graph G and an $ELA(9;G)$; note that each faulty interaction (marked in bold in the array) appears in a distinct test. The covering array shown in Figure 1 is not an $ELA(9,G)$, since interaction $\{(3,1),(4,2)\}$ is not located by any of its rows. The definition of ELAs can be easily adapted to locate t-way interactions or even s-way interactions for all $s \leq t$, given t, by using hypergraphs in place of graphs [14].

3 Existence of ELAs and Logarithmic Growth

Given G, the existence of an $ELA(n; G)$ for some n is equivalent to G being locatable. We can show that the decision problem of whether G is locatable for general g is NP-complete, while for $g = 2$ it is in P [14]. A necessary condition for G to be locatable is that G does not have two vertices in different factors such that the union of their neighbourhoods contains all the vertices of another factor. We look at a sufficient condition.

Definition 3. *(Graphs with safe values) A k-partitite graph $G = G_{g_1,\ldots,g_k}$ with parts $V_1, \ldots V_k$ of cardinalities $g_1, \ldots, g_k$, respectively, is said to have* safe values *if for every $i \in [1, k]$ there exists a vertex $v_{i,s_i} \in V_i$ such that v_{i,s_i} is not incident to any edge. The values $s_1, s_2, \ldots, s_k$ are said to be* safe values *of G.*

Proposition 4. *If G is a k-partite graph with safe values, then G is locatable.*

Proof. Take any interaction $I = \{(i, a_i), (j, a_j)\}$ and a k-tuple $T = (T_1, \ldots, T_k)$ with $T_i = a_i$ and $T_j = a_j$, and $T_\ell = s_\ell$, for $\ell \in [1, k] \setminus \{i, j\}$. We note that T locates I. □

Consider a family of k-partite graphs with g vertices per part and d edges. Assume that we choose the d edges at random among the $g^2\binom{k}{2}$ possible edges. It is possible to show that for constant g and $d \in o(k)$, the probability that a graph has safe values goes to 1 as $k, d \to \infty$ [14].

Next, we give constructions of ELAs based on covering arrays of strength higher than 2. Theorem 5 generalizes a result by Colbourn and McCleary (Theorem 4.5 in [8]) about $(\overline{d}, 2)$-detecting arrays, by removing the requirement that $d < g_i$, while adding the weaker requirement that the error graph has safe values. Theorem 6 removes the hypothesis that the graphs have safe values. The proofs are omitted due to lack of space [14].

Theorem 5. *Fix d and k with $d + 2 \leq k$ and $g_1, \ldots, g_k$. Let $\mathcal{SG} = \mathcal{SG}^d_{(g_1,\ldots,g_k)}$ be the class of graphs of the form $G_{(g_1,\ldots,g_k)}$ with at most d edges and that have safe values. Then, every $MCA(n, d + 2, (g_1, \ldots, g_k))$ is an $ELA(n; \mathcal{SG})$.*

Theorem 6. *Fix k and d, with $2(d+1) \leq k$ and $g_1, \ldots, g_k$. Let $\mathcal{LG} = \mathcal{LG}^d_{(g_1,\ldots,g_k)}$ be the class of* **locatable** *graphs of the form $G_{(g_1,\ldots,g_k)}$ with at most d edges. Then, any $MCA(n; 2(d+1), (g_1, \ldots, g_k))$, A, is an $ELA(n; \mathcal{LG})$. Moreover, if G is not locatable then the outcomes of the tests in A are enough to detect this.*

The next theorem implies that the size n of an ELA grows logarithmically in k for fixed d and g. Moreover, even if d grows as $O(\log \log k)$, n remains polynomial in $\log k$.

Theorem 7. *(Size of ELAs for bounded g_i) Fix g. Let $\mathcal{G}(k, d) = \mathcal{G}^d_{k,g}$ be a class of graphs of the form $G_{(g_1,\ldots,g_k)}$, with at most d edges, with $g_i \leq g$, for all $i \in [1, k]$, and satisfying extra conditions specified below. Then,*

1. *if all graphs in $\mathcal{G}(k,d)$ are locatable and $2(d+1) \leq k$, then there exists an $ELA(n;\mathcal{G}(k,d))$ for $n \in O(dg^{2d}\log k)$; and*
2. *if all graphs in $\mathcal{G}(k,d)$ have safe values and $d+2 \leq k$, then there exists an $ELA(n;\mathcal{G}(k,d))$ for $n \in O(dg^{d}\log k)$.*

Proof. It follows from the application of Theorems 5 and 6, using the construction in [4], which gives an $MCA(n;s,k,g)$ with $n \in O(sg^{s}\log k)$, for fixed g. □

Proposition 4 and Theorems 5-7 can be generalized for $t \geq 2$, which will be included in the complete article version of this extended abstract [14].

4 Algorithm for Locating Graphs with Safe Values

In this section, we give an efficient algorithm for locating errors for graphs that have safe values: Algorithm ErrorLocateWithSafeValues. Unlike previous definitions in this paper, the graphs are allowed to have loops, i.e., faulty 1-way interactions, as long as no faulty 1-way interaction is contained in a faulty 2-way interaction. In this case, Theorem 7 would provide a solution for non-adaptive testing (which can be shown to work for graphs with loops) that uses $O(dg^{d}\log k)$ tests and requires the knowledge of d. In contrast, Algorithm ErrorLocateWithSafeValues uses an adaptive testing approach that does not require the knowledge of d, does require the knowledge of the safe values s_i, $1 \leq i \leq k$, and uses a polynomial number of tests in d and $\log k$, more specifically $O(d(\log k)^2 + d^2 \log k)$ tests.

Let $G = G_{(g_1,\ldots,g_k)}$ be a graph with edge set that corresponds to faulty 1-way and 2-way interactions, such that no faulty 1-way interaction is contained in a 2-way interaction. Let $s = (s_1,\ldots,s_k)$ be a k-tuple of safe values for G, and $T = (T_1,\ldots,T_k)$ be a k-tuple corresponding to a failing test for G. Let $A \subseteq [1,k]$ be the *inspection set of* T, that is, a set of factors f such that v_{f,T_f} may be an endpoint of an edge. In general, $|A| \leq k$. The algorithms presented here detect whether T is a passing or failing test by calling procedure Test(T).

First, we give the main steps of Algorithm ErrorLocateWithSafeValues (k,$(s_1,\ldots,s_k)$, $(g_1,\ldots,g_k)$), which locates failing interactions given safe values. It starts by building $\mathcal{A}$, an $MCA(n;2,(g_1,\ldots,g_k))$, using the greedy density method by Bryce and Colbourn [3], which guarantees $n \in O(\log k)$ when the g_i's are bounded by a constant. Since $\mathcal{A}$ is a mixed covering array of strength 2, it is guaranteed that each 1-way and 2-way interaction is covered by some of its rows. For each test T given by a row of $\mathcal{A}$, run procedure Test(T). Finally, for each failing test T found in the previous step, run procedure LocateErrorsInTest ($(s_1,\ldots,s_k), T, A = [1,k]$) to identify the failing interactions covered by T. Now, what is left to describe is procedure LocateErrorsInTest as well as its auxiliary procedures given in Algorithm 1 (page 512).

In Algorithm 1, procedure LocateErrorsInTest(s, T, A) is recursive and works as follows. At each call, we assume T is a failing test with A its inspection set. If $|A| = 1$, then the unique factor of A corresponds to an error of order 1, which is a loop in the error graph. If $|A| \geq 2$, we solve the problem recursively by partitioning A into approximately equal sized sets A' and A''. Edges can involve factors within A', within A'' or with one end in A' and the other one in A''. The first type of edge we say to be *within* partition (A', A'') and the second type we say to be *across* partition (A', A''). We use recursion to locate the edges within the partition. Now, with knowledge of the edge sets, E' and E'', within the partition, we use procedure AcrossLocate(s, T, A', A'', E', E'') to determine the edges across the partition. Procedure AcrossLocate(s, T, A', A'', E', E'') partitions A' and A'' into parts that do not include within edges. To do this, we first remove the loops from A' and A'' creating B' and B'', respectively. Then, we partition B' into $(A'_1, \ldots, A''_{c'})$ and B'' into $(A''_1, \ldots, A''_{c''})$, such that there are no edges within A'_i for $1 \leq i \leq c'$ or within A''_j for $1 \leq j \leq c''$; in other words, we c'-colour (c''-colour) the graph corresponding to A'_i (A''_j, respectively). This can be done using a greedy colouring algorithm. Now, the only unknown edges have one endpoint from each of A'_i and A''_j, for some i, j, with $1 \leq i \leq c'$, $1 \leq j \leq c''$. For each such candidate pair of sets, we apply procedure AcrossLocateAux to discover the edges between them. The objective of AcrossLocateAux(s, T, A, B) is to find edges that have exactly one end in inspection set A and one end in inspection set B. It is assumed that T is a failing test and (A, B) does not contain within edges nor loops. If both A and B consist of single factors a and b, respectively, we conclude that the only edge across (A, B) is $\{v_{a,T_a}, v_{b,T_b}\}$. Otherwise, at least one of A and B has more than one factor. Assume w.l.o.g. that A does. So, we partition A into parts C' and C'', and we reduce the problem to recursively finding edges across (B, C') and across (B, C''); in the next recursive call we partition B, if it has more than one factor. In this way, A and B are alternatingly partitioned from one iteration to the next, as long as their size is non-trivial. This alternation is controlled by the selection of C and D in AcrossLocateAux.

The cost of Algorithm 1 is measured by the number of times a test is performed via a call to Test(). This analysis follows from the next two lemmas.

Lemma 8. *Let (A, B) be a partition without edges within it, and let d_a be the total number of edges (with respect to values in test T) across partition (A, B). Then, the number of times* Test() *is called by procedure* AcrossLocateAux(s, T, A, B) *is at most $4 d_a \log_2 k$.*

Proof. Let $\mathcal{C}(e)$ be the number of times AcrossLocateAux calls Test() with a test that covers edge e. Consider two consecutive levels of recursion in a call AcrossLocateAux(A, B), which involves six recursive calls and produces four possible recursion tree nodes at the second level. Any of these four nodes have an input size essentially half of $|A \cup B|$; so the recursion tree height is at most $2(\lceil \log_2 k \rceil - 1)$. Additionally, of the six recursion calls mentioned above, only two can have input sets such that e has an endpoint in each set. So, $\mathcal{C}(e) \leq 2(\lceil \log_2 k \rceil - 1)$. Let $\mathcal{D}$ be the total number of calls to Test(). Now, each Test() either covers some edge (fails), or it passes and the other call to Test() in the same recursion-tree node covers some edge (fails). Therefore, $\mathcal{D} \leq 2 \sum_{i=1}^{d_a} \mathcal{C}(e_i) \leq 2 \sum_{i=1}^{d_a} 2(\lceil \log_2 k \rceil - 1) \leq 4 d_a \log_2 k$. □

Algorithm 1. Returns the errors "exposed" by test T, given safe tuple s and inspection set A

```
procedure LocateErrorsInTest($s$, $T$, $A$)    $\triangleright$ Main Procedure of Algorithm 1
    if $|A| = 1$, say $A = \{a\}$, then return $\{\{(a, T_a), (a, T_a)\}\}$
    else
        partition $A$ into $(A', A'')$ of approximately equal sizes
        $\triangleright$ find edges within $A'$:
        Define: $T'$ by: $T'_f = T_f$, if $f \in A'$, and $T'_f = s_f$, otherwise.
        if Test($T'$) = fail then
            $E' \leftarrow$ LocateErrorsInTest($s, T', A'$)
        $\triangleright$ find edges within $A''$:
        Define: $T''$ by: $T''_f = T_f$, if $f \in A''$, and $T''_f = s_f$, otherwise.
        if Test($T''$) = fail then
            $E'' \leftarrow$ LocateErrorsInTest($s, T'', A''$)
        $\triangleright$ find edges across $(A', A'')$:
        $E''' \leftarrow$ AcrossLocate($s, T, A', A'', E', E''$)
        return $E' \cup E'' \cup E'''$

procedure AcrossLocate($s$, $T$, $A'$, $A''$, $E'$, $E''$)
    $B' = A' \setminus \{x \in A' : \{x, x\} \in E'\}$
    partition $B'$ into $c'$ colour classes $(A'_1, \ldots, A'_{c'})$
    $B'' = A'' \setminus \{x \in A'' : \{x, x\} \in E''\}$
    partition $B''$ into $c''$ colour classes $(A''_1, \ldots, A''_{c''})$
    $E''' \leftarrow \emptyset$
    for $i \leftarrow 1$ to $c'$ do
        for $j \leftarrow 1$ to $c''$ do
            Define $T'$ by: $T'_f = T_f$, if $f \in A'_i \cup A''_j$, and $T'_f = s_f$, otherwise.
            if Test($T'$) = fail then
                $E''' \leftarrow E''' \cup$ AcrossLocateAux($s, T', A'_i, A''_j$)
    return $E'''$

procedure AcrossLocateAux($s$, $T$, $A$, $B$)
    if $|A| = |B| = 1$, say $A = \{a\}$ and $B = \{b\}$, then
        return $\{\{(a, T_a), (b, T_b)\}\}$
    else
        if $|A| > 1$ then $C \leftarrow A$; $D \leftarrow B$ $\triangleright$ always partition first set, if possible
        else $C \leftarrow B$; $D \leftarrow A$
        Partition $C$ into $(C', C'')$ of approximately equal sizes
        $\triangleright$ Note that the recursive calls below invert the order of the sets.
        $E' \leftarrow E'' \leftarrow \emptyset$
        Define $T'$ by: $T'_f = s_f$, if $f \in C''$, and $T'_f = T_f$, otherwise.
        if Test($T'$) = fail then $E' \leftarrow$ AcrossLocateAux($s, T', D, C'$)
        Define $T''$ by: $T''_f = s_f$, if $f \in C'$, and $T''_f T_f$, otherwise.
        if Test($T''$) = fail then $E'' \leftarrow$ AcrossLocateAux($s, T'', D, C''$)
        return $E' \cup E''$
```

Lemma 9. *Let (A', A'') be a partition with d' edges within part A', d'' edges within part A'', and d_a edges across partition (A', A''). Then, the number of times* Test() *is called by procedure* AcrossLocate(s, T, A', A'', E', E'') *is* $O(d_a \log k + d'd'')$.

Proof. Procedure AcrossLocate performs $c'c''$ calls to Test() in its main loop. Since the number of colours needed to greedy colour a graph with e edges is at most $e+1$, we have $c'c'' \leq (d'+1)(d''+1)$. Let $d_a^{i,j}$ be the number of edges across partition (A'_i, A''_j). Since $\sum_{i=1}^{c'} \sum_{j=1}^{c''} d_a^{i,j} = d_a$ and, by Lemma 8, each call to AcrossLocateAux costs at most $4d_a^{i,j} \log k$ calls to Test(), we have in total at most $4d_a \log k$ such calls. □

Theorem 10. *Let $\hat{d}$ be the total number of failing 1-way or 2-way interactions of values covered by test T. The number of times* Test() *is called by Algortihm 1 is* $O(\hat{d} \log k + (\hat{d})^2)$.

Proof. Associate with each possible subtree τ of the recursion tree of LocateErrorInTest, the total number of failing interactions that are detected on this subtree, and denote it by $d(\tau)$. Given a subtree τ, denote by τ_1 and τ_2, its left and right subtrees, respectively. We consider a full binary tree of the maximum height $\lceil \log k \rceil - 1$, instead of a possibly smaller tree, by assigning $d(\tau) = 0$ for a subtree τ that is not generated by the recursion. Note that for the recursion tree root r, $d(r) = \hat{d}$, and that, considering the recursive calls at subtree τ, we have $d(\tau_1) > 0$ if and only if (Test(T')=fail), and $d(\tau_2) > 0$ if and only if (Test(T'')=fail). Moreover, letting $d_a(\tau)$ denote the number of edges across the partition found by the call to AcrossLocate performed at the root of subtree τ, we get $d(\tau) = d(\tau_1) + d(\tau_2) + d_a(\tau)$.

So the number of tests $\mathcal{C}(\ell, \tau)$ performed by Algorithm 1 at subtree τ with $|A| = \ell$ is given by: $\mathcal{C}(1, \tau) = 0$; $\mathcal{C}(\ell, \tau) = 0$, if $d(\tau) = 0$; and

$$\mathcal{C}(\ell,\tau) \leq \mathcal{C}(\lceil \ell/2 \rceil, \tau_1) + \mathcal{C}(\lfloor \ell/2 \rfloor, \tau_2) + C(d_a(\tau)\log \ell + d(\tau_1)d(\tau_2)) + 2, \text{if } d(\tau) > 0 \text{ and } \ell > 1;$$

where C is a constant given by Lemma 9. Let Δ_j be the set of subtrees rooted at level j of the recursion tree, and denote $d_a^j = \sum_{\tau \in \Delta_j} d_a(\tau)$. Let N denote the number of internal nodes in the recursion tree. Iterating the equations that define $\mathcal{C}$, and defining ϕ such that $\phi(\tau) = 0$, if $d(\tau) = 0, 1$, and $\phi(\tau) = \phi(\tau_1) + \phi(\tau_2) + d(\tau_1)d(\tau_2)$, if $d(\tau) \geq 2$, we get for the root node r:

$$\mathcal{C}(k, r) \leq C \left(\sum_{j=0}^{\lceil \log k \rceil - 1} d_a^j \log \frac{k}{2^j} \right) + C\phi(r) + 2N. \quad (1)$$

We observe that $\sum_{j=0}^{\lceil \log k \rceil - 1} d_a^j = \hat{d}$, since each edge is identified as an across edge in a distinct recursion tree node. This gives that the expression between parenthesis in (1) is at most $\hat{d} \log k$. Now, using $d(\tau_1) + d(\tau_2) \leq d(\tau)$, it is easy to show by induction that $\phi(\tau) \leq d(\tau)^2/2$, which gives $\phi(r) \leq (\hat{d})^2/2$. In addition,

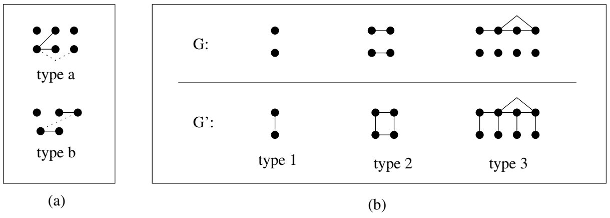


Fig. 3. Forbidden structures and connected components in locatable graphs with $g = 2$

at each level of the tree there cannot be more than $\hat{d}$ nodes, and the tree has at most $\lceil \log k \rceil + 1$ levels, so we get $N \leq \hat{d}\lceil \log k \rceil$. Substituting these bounds in (1), gives:

$$\mathcal{C}(k, r) \leq C\hat{d}\log k + C(\hat{d})^2/2 + 2\hat{d}\lceil \log k \rceil \in O(\hat{d}\log k + (\hat{d})^2). \qquad \square$$

Finally, we analyse Algorithm ErrorLocateWithSafeValues(k, $(s_1, \ldots, s_k)$, $(g_1, \ldots, g_k)$), our main algorithm described at the begining of this section.

Theorem 11. *Let d be the total number of failing 1-way and 2-way interactions, k be the number of factors, and g be a constant such that $g \geq \max_{i=1}^{k}\{g_i\}$. Then, the number of calls to* Test() *by Algorithm* ErrorLocateWithSafeValues *is in $O(d(\log k)^2 + d^2 \log k)$.*

Proof. Initially, we do n calls to Test() where $n \in O(\log k)$ is the number of rows in the mixed covering array $\mathcal{A}$. Then, for each row of $\mathcal{A}$ that gives a failing test (at most $O(\log k)$ of them), we call procedure LocateErrorsInTest. Since, by Theorem 10, each such call performs at most $O(\hat{d}\log k + (\hat{d})^2)$ calls to Test(), and $\hat{d} \leq d$, the result follows. $\square$

5 Algorithm for Locating Graphs with g=2

The following theorem characterizes locatable graphs for the binary case (see proof in [14]).

Theorem 12. *Let $G = G_{k,2}$. Then, G is not locatable if and only if it contains as a subgraph one of the two forbidden structures given by the solid lines in Figure 3a. The dashed lines in Figure 3a indicate interactions that cannot be located.*

From the previous theorem we can deduce the structure of the connected components of a locatable graph G. Define an auxiliary graph G' such that $V(G') = V(G)$ and $E(G') = E(G) \cup \{\{v_{i,0}, v_{i,1}\} : 1 \leq i \leq k\}$. Since G does not contain any of the forbidden graphs with 2 edges, then G' is formed by

```
procedure SearchEndPoint($T$, $D$)
    if $|D| = 1$, say $D = \{f\}$, then return $\{f\}$
    else
        $V' \leftarrow V'' \leftarrow \emptyset$
        Partition $D$ into $(D', D'')$ of approximately equal sizes
        ▷ Flip bit in $D''$ to relative safe value:
        Define $T'$ by: $T'_f = \neg(T_f)$, if $f \in D''$, and $T'_f = T_f$, otherwise
        if Test($T'$) = fail then $V' \leftarrow$ SearchEndPoint($T', D'$)
        ▷ Flip bit in $D'$ to relative safe value:
        Define $T''$ by: $T''_f = \neg(T_f)$, if $f \in D'$, and $T'_f = T_f$, otherwise
        if Test($T''$) = fail then $V'' \leftarrow$ SearchEndPoint($T'', D''$)
        return $V' \cup V''$
```

Fig. 4. Auxiliary procedure used in Algorithm DiscoverEdges

connected components that can be of one of the following types, as illustrated in Figure 3b: 1) a single edge $\{\{v_{i,a}, v_{i,\overline{a}}\}\}$ (type 1); 2) a cycle of the form $(v_{i,a}, v_{j,b}, v_{j,\overline{b}}, v_{i,\overline{a}}, v_{i,a})$, with $i \neq j$ (type 2); 3) a connected component with each vertex of degree ≥ 1 belonging to a different factor of G' with the other vertex for each factor being a degree-one vertex hanging out of it (type 3). Using the knowledge of this structure, we now give an efficient algorithm for locatable graphs for $g = 2$. Algorithm DiscoverEdges is based on partitioning the set of factors into subsets according to properties of their edges in relation to the vertices associated with a passing test. Then, an investigation of which pair of factor subsets can have edges between them guides our discovery of each type of edge in the various steps of the algorithm. We assume w.l.o.g. that the test with all factors set to value 0 is a passing test, and then define the following sets that partition the set of factors: the set A of factors where 1 is an endpoint of a 1–0 edge, $A = \{f \in [1,k] :$ there exists $j \in [1,k]$ such that $\{v_{f,1}, v_{j,0}\} \in E(G)\}$; the set B of factors where 1 that is an endpoint of a 1–1 edge and are not in A, $B = \{f \in [1,k] \setminus A :$ there exists $f' \in [1,k] \setminus A$ such that $\{v_{f,1}, v_{f',1}\} \in E(G)\}$; and all the remaining factors $C = [1,k] \setminus (A \cup B)$. The set A is further partitioned into the sets $A^P = \{f \in A :$ there exists $f' \in A$ such that $\{v_{f,1}, v_{f',0}\} \in E(G)\}$ and $A^S = A \setminus A^P$. The set A^P is called the set of endpoints of "parallel edges", whereas A^S is called the set of "single factors" factors in A. Finally, the set C is partitioned into three sets $C^0 = \{f \in C :$ there exists $f' \in A^S$ such that $\{v_{f,0}, v_{f',1}\} \in E(G)\}$, $C^1 = \{f \in C :$ there exists $f' \in A^S$ such that $\{v_{f,1}, v_{f',1}\} \in E(G)\}$ and $C^I = C \setminus (C^0 \cup C^1)$.

Note that C^1 and C^0 must be disjoint, for otherwise, a forbidden configuration of type a or b (Figure 3a) would be present. By the above definition, the only types of possible edges are: $1-0$ inside A^P, $1-1$ inside A^S, $1-1$ inside B, $1-1$ between A^S and B, $1-0$ between A^S and C^0, and $1-1$ between A^S and C^1.

Our main algorithm uses the auxiliary procedure SearchEndPoint(T, D), which is detailed in Figure 4. Here $\neg a$ denotes the complement of the binary value a. This method starts with a failing test T, and values in $\hat{D} = [1,k] \setminus D$ are fixed. We also know that $\neg T_f$ has no edges to $\hat{D}$, for all $f \in D$. This procedure finds each $f \in D$ such that v_{f,T_f} is the endpoint of an edge to $v_{\hat{f},T_{\hat{f}}}$ for some $\hat{f} \in \hat{D}$. The main algorithm is given next.

Algorithm DiscoverEdges:

Step 0. Discover a passing test:
Build $\mathcal{A}$, a $CA(n; 2, k, 2)$ with $n \in O(\log k)$ rows, and run $\text{Test}(T)$ on each row T of A. If $\mathcal{A}$ gives no failing test, return $E(G) = \emptyset$. Otherwise, if $\mathcal{A}$ contains a passing test, call it P. Otherwise, run $\text{Test}(P)$ on random tests P until a passing test P is found. Relabel the values of $\mathcal{A}$ in such a way that the passing test P becomes $[0, 0, \ldots, 0]$.
Step 1. Discover the set A:
This is done via binary searching for the 1-ends of $1-0$ edges, using as a starting point each failing test T given by a row of the covering array $\mathcal{A}$, and then calling $\text{SearchEndPoint}(T, F_1(T))$, where $F_1(T)$ is the set of factors with value 1 in T.
Step 2. Discover E^P (parallel edges), and so A^P and A^S:
We build E^P by testing possible end points in A, two by two. For each $i, j \in A$, consider the test T with values set to 0 except $T_i = T_j = 1$. It is easy to see that $\text{Test}(T) = \text{pass}$ if and only if $\{v_{i,1}, v_{j,0}\}, \{v_{i,0}, v_{j,1}\} \in E^P$, so if $\text{Test}(T) = \text{pass}$, then add i and j to A^P.
Step 3. Discover B:
Let T be a test that fixes factors in A to zero, and initially have factors in $[1, k] \setminus A$ set to 1. This reduces the problem to edges internal to levels in B for which 0s are safe values. Thus, we discover the edges internal to B (and thus B itself) by running $\text{LocateErrorsInTest}(s = (0, 0, \ldots, 0), T, [1 : k] \setminus A)$.
Step 4. Determine the set E of edges with exactly one endpoint in A^S (the other endpoint corresponds to a factor that is in B or either in C^1 or C^0):

```
$E \leftarrow \emptyset$; $C \leftarrow [1, k] \setminus (A \cup B)$
for all $f \in A^S$ do
    Fix $T_f = 1$ and $T_i = 0$ for all $i \in (A \setminus \{f\}) \cup B$
    repeat
        Randomly pick the values of $T_j$ for $j \in C$
    until $\text{Test}(T) = \text{pass}$
    for all $b \in B$ do
        $T' \leftarrow T; T'_b \leftarrow 1$
        if $\text{Test}(T') = \text{fail}$ then $E \leftarrow E \cup \{\{v_{f,1}, v_{b,1}\}\}$
    ▷ Binary search for mates of $f$ in $C$ using values in $T$ as safe values for $C$
    Set $T''$ with $T''_i = T_i$ for $i \in A \cup B$, and $T''_i = \neg T_i$ for $i \in C$
    $L \leftarrow \text{SearchEndPoint}(T'', C)$
    $E \leftarrow E \cup \{\{v_{f,1}, v_{c,T''_c}\} : c \in L\}$
```

Step 5. Find the set E' of all edges with both ends in A^S:
For each $i, j \in A^S$, define test T with all factors set to 0, except $T_i = T_j = 1$ and $T_c = 1$ for all $c \in C^0$. If $\text{Test}(T) = \text{fail}$, then add $\{v_{i,1}, v_{j,1}\}$ to E'.

Theorem 13. *If $G = G_{(2^k)}$ is locatable then algorithm* DiscoverEdges *is correct. In addition, letting $\mathcal{C}(d, k)$ be the expected number of tests performed by algorithm* DiscoverEdges *and δ_{max} denote the maximum degree of a vertex in G, we have $\mathcal{C}(d, k) \in O(d^2 + d \log k + d2^{\delta_{max}} + 2^{2d})$. Moreover, we get that $\mathcal{C}(d, k)$ is polynomial on both d and $\log k$, under extra assumptions:*

1. *If a passing test is found in the covering array $\mathcal{A}$ in Step 0, and $\delta_{max} \leq c \log\log k$, for some c, then $\mathcal{C}(d, k) \in O(d^2 + d \log k + d(\log k)^c)$.*
2. *If $d \leq c' \log\log k$, for some c', then $\mathcal{C}(d, k) \in O(d \log k + (\log k)^{2c'})$.*

Proof. The correctness of the algorithm is based on the definitions of sets A^P, A^S, B, C^0, C^1, and C^I and the possible edges that can be found between factors in these sets (see page 515); the proof of correctness is omitted here (see [14]). Now we justify the running time of the algorithm. The running time of SearchEndPoint is in $O(d \log |D|)$, since at each level of its recursion tree the sum of the tests performed does not exceed d, and that the number of tree levels does not exceed $\lceil \log |D| \rceil$. Now, we analyse steps 0-5.

Step 0: First, we call Test() once per row of covering array $\mathcal{A}$, i.e., $O(\log k)$ times. Let x be the number of factors involved in some edge, that is, the number of factors that index vertices that are endpoints of some edge. Then, since the graph is locatable, at least one of the $2^x \leq 2^{2d}$ combinations of possible values for these factors determines a passing test, yielding the probability that a passing test is selected to be at least $1/2^{2d}$; therefore the expected number of random trials until a passing test is found is in $O(2^{2d})$. Thus, the expected number of calls to Test() for this step is $O(\log k + 2^{2d})$.
Step 1: There are $O(\log k)$ tests given by the covering array $\mathcal{A}$ which might fail, and for each of them, SearchEndPoint is run with $|D| \leq k$, each contributing to $O(d \log k)$ tests. Thus, the number of calls to Test() in this step is $O(d(\log k)^2)$.
Step 2: The number of tests for this step is $|A|^2 \leq d^2$.
Step 3: This step is an application of LocateErrorsInTest while restricting the problem to levels in set B, for which 0's are safe values. By Theorem 10, the number of tests performed is $O(d_B^2 + d_B \log |B|)$, where d_B is the number of edges with both ends in B. Since $d_B \leq d$ and $|B| \leq k$, we get $O(d^2 + d \log k)$.
Step 4: Let $f \in A^S$ as selected in the main loop. Once we fix the test to value 0 for all factors in $A \cup B$, except for f that has value 1, by definition of C, the only edges that can occur in any completion of this test involves factor f set to 1 and at most one of the values (either 0 or 1) for each factor c in C. This means that the probability of not hitting an edge for each attempt at completing C is $1/2^{\delta(v_{f,1})}$, which gives an expected number of $2^{\delta(v_{f,1})}$ tests until a passing test is found. Moreover, the number of edges that can be found within SearchEndPoint is $\delta(v_{f,1})$, and so yields $O(\delta(v_{f,1}) \log k)$. As we sum over all iterations, we get a total of $O((\sum_{f \in A^S} 2^{\delta(v_{f,1})}) + \log k \sum_{f \in A^S} \delta(v_{f,1})) \subseteq O(d2^{\delta_{max}} + d \log k)$.
Step 5: The number of tests for this step is $|A^S|^2 \leq d^2$.

Therefore, if the covering array $\mathcal{A}$ in step 0 yields a passing test, we get $\mathcal{C}(k, d) \in O(d^2 + d \log k + d2^{\delta_{max}})$, while if we need to find a passing test, then $\mathcal{C}(k, d) \in O(d^2 + d \log k + d2^{\delta_{max}} + 2^{2d})$. The theorem follows easily from this. □

6 Conclusion and Further Work

In this paper, we generalize recent work on locating faulty interactions in system testing. We provide new results for *non-adaptive* testing (Section 3) and the first algorithms for *adaptive testing* (Sections 4 and 5). The definitions and results in Sections 2 and 3 can be generalized for locating t-way interactions with $t \geq 2$, as well as for locating s-way interactions with $1 \leq s \leq t$, for some $t \geq 2$ (see full version of this paper [14]). Important open problems include generalizing the algorithm in Section 4 for $t > 2$, and generalizing the algorithm in Section 5 for $g > 2$.

References

1. Aigner, M., Triesch, E.: Searching for an edge in a graph. J. Graph Theory 12, 45–57 (1988)
2. Azar, Y., Motwani, R., Naor, J.: Approximating probability distributions using small sample spaces. Combinatorica 18, 151–171 (1998)
3. Bryce, R.C., Colbourn, C.J.: A density algorithm for pairwise interaction testing. Softw. Test. Verif. Reliab. 17, 159–182 (2007)
4. R.C. Bryce and C.J. Colbourn, A density-based greedy algorithm for higher strength covering arrays, 17 pages, (preprint, July 2007)
5. Burr, K., Young, W.: Combinatorial test techniques: Table-based automation, test generation, and code coverage. In: Proc. Intl. Conf. on Soft. Test. Anal. and Rev., October 1998, pp. 503–513. ACM, New York (1998)
6. J.N. Cawse, Experimental design for combinatorial and high throughput materials development, GE Global Research Technical Report 29, 769–781 (2002)
7. Colbourn, C.J.: Combinatorial aspects of covering arrays. Le Matematiche (Catania) 58, 121–167 (2004)
8. Colbourn, C.J., McClary, D.W.: Locating and detecting arrays for interaction faults. Journal of Combinatorial Optimization (accepted, April 2007) (to appear)
9. Dalal, S.R., Karunanithi, A.J.N., Leaton, J.M.L., Patton, G.C.P., Horowitz, B.M.: In: Model-based testing in practice, In Proc. Intl. Conf. on Software Engineering (ICSE 1999), pp. 285–294 (1999)
10. Danziger, P., Mendelsohn, E., Moura, L., Stevens, B.: Covering arrays without forbidden pairs, p. 10 (preparation, 2007)
11. Dunietz, S., Ehrlich, W.K., Szablak, B.D., Mallows, C.L., Iannino, A.: Applying design of experiments to software testing. In: Proc. Intl. Conf. on Software Engineering (ICSE 1997), October 1997, pp. 205–215. IEEE, Los Alamitos (1997)
12. Kuhn, D.R., Reilly, M.: An investigation of the applicability of design of experiments to software testing. In: Proc. 27th Annual NASA Goddard/IEEE Software Engineering Workshop, October 2002, pp. 91–95. IEEE, Los Alamitos (2002)
13. Kuhn, D.R., Wallace, D.R., Gallo, A.M.: Software fault interactions and implications for software testing. IEEE Trans. Soft. Eng. 30, 418–421 (2004)
14. Martínez, C., Moura, L., Panario, D., Stevens, B.: Locating errors using ELAs, covering arrays and adaptive testing algorithms, p. 35 (in preparation, 2007) (full article version of this extended abstract)
15. Mandl, R.: Orthogonal latin squares: An application of experiment design to compiler testing. Communic. of the ACM 28, 1054–1058 (1985)

16. Meagher, K., Stevens, B.: Covering arrays on graphs. J. Combin. Theory. Ser. B 95, 134–151 (2005)
17. Moura, L., Stardom, J., Stevens, B., Williams, A.W.: Covering arrays with mixed alphabet sizes. J. Combin. Des. 11, 413–432 (2003)
18. Shasha, D.E., Kouranov, A.Y., Lejay, L.V., Chou, M.F., Coruzzi, G.M.: Using combinatorial design to study regulation by multiple input signals: A tool for parsimony in the pos-genomics era. Plant Physiology 127, 1590–2594 (2001)
19. Seroussi, G., Bshouty, N.H.: Vector sets for exhaustive testing of logic circuits. IEEE Transactions on Information Theory 34, 513–522 (1988)
20. Tang, D.T., Chen, C.L.: Iterative exhaustive pattern generation for logic testing. IBM Journal Research and Development 28, 212–219 (1984)
21. Torney, D.C.: Sets pooling designs. Ann. Comb. 3, 95–101 (1999)
22. Williams, A.W., Probert, R.L.: A measure for component interaction test coverage. In: Proc. ACS/IEEE Intl. Conf. Comput. Syst. & Applic., pp. 301–311 (2001)

On Injective Colourings of Chordal Graphs

Pavol Hell[1,⋆], André Raspaud[2], and Juraj Stacho[1]

[1] School of Computing Science, Simon Fraser University
8888 University Drive, Burnaby, B.C., Canada V5A 1S6
{pavol,jstacho}@cs.sfu.ca
[2] LaBRI, Université Bordeaux I
351 Cours de la Libération, F33405 Talence Cedex, France
raspaud@labri.fr

Abstract. We show that one can compute the injective chromatic number of a chordal graph G at least as efficiently as one can compute the chromatic number of $(G-B)^2$, where B are the bridges of G. In particular, it follows that for strongly chordal graphs and so-called power chordal graphs the injective chromatic number can be determined in polynomial time. Moreover, for chordal graphs in general, we show that the decision problem with a fixed number of colours is solvable in polynomial time. On the other hand, we show that computing the injective chromatic number of a chordal graph is NP-hard; and unless $NP = ZPP$, it is hard to approximate within a factor of $n^{1/3-\epsilon}$, for any $\epsilon > 0$. For split graphs, this is best possible, since we show that the injective chromatic number of a split graph is $\sqrt[3]{n}$-approximable. (In the process, we correct a result of Agnarsson et al. on inapproximability of the chromatic number of the square of a split graph.)

1 Introduction

In this paper, a graph is always assumed to be undirected, loopless and simple. An *injective colouring* of a graph G is a colouring c of the vertices of G that assigns different colours to any pair of vertices that have a common neighbour. (That is, for any vertex v, if we restrict c to the (open) neighbourhood of v, this mapping will be injective; whence the name.) Note that injective colouring is not necessarily a proper colouring, i.e., it is possible for two adjacent vertices to receive the same colour. The injective chromatic number of G, denoted $\chi_i(G)$, is the smallest integer k such that G can be injectively coloured with k colours.

Injective colourings are closely related to (but not identical with) the notions of locally injective colourings [9] and $L(h,k)$-labellings [2,3,11]. In particular, $L(0,1)$-labellings unlike injective colourings assign distinct colours only to non-adjacent vertices with a common neighbour.

Injective colourings were introduced by Hahn, Kratochvíl, Širáň and Sotteau in [12]. They attribute the origin of the concept to complexity theory on Random Access Machines. They prove several interesting bounds on $\chi_i(G)$, and also

⋆ Supported by a Discovery Grant from NSERC.

show that, for $k \geq 3$, it is NP-complete to decide whether the injective chromatic number of a graph is at most k. Here we look at the complexity of this problem when the input graphs G are restricted to be chordal. A graph is *chordal* if it does not contain any induced cycle of length four or more [10]. Several difficult combinatorial problems that are NP-complete in general (including graph colouring [10], and many variants [5,7,13]) admit a polynomial time solution in chordal graphs.

In Section 4, we show that determining $\chi_i(G)$ is still difficult when restricted to chordal graphs. In fact, it is not only NP-hard, but unless $NP = ZPP$, the injective chromatic number of a chordal graph cannot be efficiently approximated within a factor of $n^{1/3-\epsilon}$, for any $\epsilon > 0$. (Here ZPP is the class of languages decidable by a randomized algorithm that makes no errors and whose expected running time is polynomial.) For split graphs, this is best possible since we show an $\sqrt[3]{n}$-approximation algorithm for the injective chromatic number of a split graph.

On the positive side, we show in Section 5 that for any fixed number k, one can in linear time determine whether a chordal graph can be injectively coloured using no more than k colours. Moreover, we describe large subclasses of chordal graphs that allow computing the injective chromatic number efficiently. We show that for a chordal graph G, one can efficiently compute the injective chromatic number of G from the chromatic number of the square of $G - \mathcal{B}(G)$, that is, the graph G with its bridges $\mathcal{B}(G)$ removed. It follows that for strongly chordal graphs and power chordal graphs (the graphs whose powers are all chordal) the problem is polynomial time solvable.

2 Preliminaries

We follow the terminology of [4,20]. For a subset S of the vertices (edges) of G, we denote $G[S]$ the subgraph of G induced on the vertices (edges) of S, and $G-S$ the subgraph of G that is obtained by removing from G the vertices (edges) of S. In the case that S consists only of a single element x, we write $G - x$ instead of $G - \{x\}$.

For a connected graph G, a vertex u is a *cutpoint* of G if the graph $G - u$ is disconnected. An edge $e = uv$ is a *bridge* of G if the graph $G - e$ is disconnected. A subset S of vertices of G is a *separator* of G if $G - S$ is disconnected. As usual, a *clique* of G is a complete subgraph of G, and an *independent set* of G is a subgraph of G having no edges. For any graph G, we denote by $\chi(G)$ and $\alpha(G)$, the chromatic number of G, and the size of a maximum independent set in G, respectively. We denote by G^k the k-th power of G, i.e., the graph obtained from G by making adjacent any two vertices in distance at most k in G. We denote by n, respectively m, the number of vertices, respectively edges of G. For a vertex u in G, we denote by $N(u)$ the set of vertices of G adjacent to u (the *neighbourhood* of u); and for a subset S of vertices of G, we denote by $N(S)$ the set of vertices of $G - S$ adjacent to at least one vertex of S. We let $\deg(u) = |N(u)|$ be the degree of u, and let $\Delta(G)$ be the maximum degree among the vertices of G.

A *split graph* is a graph which can be partitioned into a clique and an independent set with no other restriction on the edges between the two. Any split graph is also chordal. A *tree-decomposition* (T, X) of a connected graph G is a pair (T, X) where T is a tree and X is a mapping from $V(T)$ to the subsets of $V(G)$, such that *(i)* for any edge $ab \in E(G)$, there exists $u \in V(T)$ with $a, b \in X(u)$, and *(ii)* for any vertex $a \in V(G)$, the vertices $u \in V(T)$ with $a \in X(u)$ induce a connected subgraph in T. A *clique-tree* of a chordal graph G is a tree-decomposition (T, X) of G where $\{X(u) \mid u \in V(T)\}$ is precisely the set of all maximal cliques of G.

3 Basic Properties

We have the following simple observation.

Observation 1. *For any graph G, $\chi_i(G) \geq \Delta(G)$ and $\chi(G^2) \geq \Delta(G) + 1$.*

For trees this is also an upper bound.

Proposition 2. *For any tree T, $\chi_i(T) = \Delta(T)$ and $\chi(T^2) = \Delta(T) + 1$.*

Proof. Let u be a leaf in T and v the parent of u. Then clearly, $\chi(T^2) = \max\big\{\deg(v)+1, \chi\big((T-u)^2\big)\big\}$ and $\chi_i(T) = \max\{\deg(v), \chi_i(T-u))\}$. The claim follows by induction on $|V(T)|$. $\square$

Now we look at the general case. Let $G^{(2)}$ be the *common neighbour graph* of a graph G, that is, the graph on the vertices of G in which two vertices are adjacent if they have a common neighbour in G. It is easy to see that the injective chromatic number of G is exactly the chromatic number of $G^{(2)}$. In general, as we shall see later, properties of the graph $G^{(2)}$ can be very different from those of G. For instance, even if G is efficiently colourable, e.g. if G is perfect, it may be difficult to colour $G^{(2)}$. Note that any edge of $G^{(2)}$ must be also an edge of G^2 (but not conversely). This yields the following inequality.

Proposition 3. *For any graph G, we have $\chi_i(G) \leq \chi(G^2)$.*

In fact, this inequality can be strengthened. Let $\mathcal{F}(G)$ be the set of edges of G that do not lie in any triangle. Note that an edge of G is also an edge of $G^{(2)}$ if and only if it belongs to a triangle of G. This proves the following proposition.

Proposition 4. *For any graph G, we have $\chi_i(G) = \chi(G^2 - \mathcal{F}(G))$.*

Now we turn to chordal graphs. The following is easy to check.

Observation 5. *Any edge in a bridgeless chordal graph lies in a triangle.*

Let $\mathcal{B}(G)$ be the set of bridges of G. Since a bridge of a graph can never be in a triangle, we have the following fact.

Proposition 6. *For any chordal G, we have $\chi_i(G) = \chi(G^2 - \mathcal{B}(G))$.*

Now since $\mathcal{B}(G - \mathcal{B}(G)) = \emptyset$, we have the following corollary.

Corollary 7. *For any chordal G, $\chi_i(G - \mathcal{B}(G)) = \chi\big((G - \mathcal{B}(G))^2\big)$.*

It turns out that there is a close connection between $\chi_i(G - \mathcal{B}(G))$ and both $\chi_i(G)$ and $\chi(G^2)$.

Proposition 8. *For any G, $\chi(G^2) = \max\big\{\Delta(G) + 1, \chi\big((G - \mathcal{B}(G))^2\big)\big\}$*

Proof. Let $k = \max\big\{\Delta(G)+1, \chi\big((G-\mathcal{B}(G))^2\big)\big\}$. It follows from Observation 1 and Corollary 7 that $\chi(G^2) \geq k$. Now fix a set of k colours ($k \geq \chi\big((G-\mathcal{B}(G))^2\big)$), and consider a colouring of $(G - \mathcal{B}(G))^2$ using these k colours. We now add the bridges of G one by one, modifying the colouring accordingly. Let uv be a bridge of G and let X and Y be the connected components which become connected by the addition of uv. Suppose that $u \in X$ and $v \in Y$. We can permute the colours of X and Y independently so that u and v obtain the same colour i. Since we have $k \geq \Delta(G) + 1$ colours, there must be a colour $j \neq i$ not used in the neighbourhood of v in Y. By the same argument for u, we may assume that j is not used in the neighbourhood of u in X. Finally, we exchange in X the colours i and j. It is easy to see that after adding all bridges of G one by one, we obtain a proper colouring of G^2. □

A similar argument proves the next proposition.

Proposition 9. *For any split graph G, $\chi_i(G) = \max\{\Delta(G), \chi_i(G - \mathcal{B}(G))\}$*

Finally, combining Corollary 7, and Propositions 8 and 3, we obtain the following tight lower bound on the injective chromatic number of a chordal graph.

Proposition 10. *For any chordal graph G, we have*

$$\chi(G^2) - 1 \leq \max\{\Delta(G), \chi_i(G - \mathcal{B}(G))\} \leq \chi_i(G) \leq \chi(G^2)$$

4 Hardness and Approximation Results

In this section, we focus on hardness results for the injective chromatic number problem. We begin by observing that it is NP-hard to compute the injective chromatic number of a split graph. This also follows from a similar proof in [12]; we include our construction here, since we shall extend it to prove an accompanying inapproximability result in Theorem 13.

Theorem 11. *It is NP-complete for a given split (and hence chordal) graph G and an integer k, to decide whether the injective chromatic number of G is at most k.*

Proof. First, we observe that the problem is clearly in NP. We show it is also NP-hard. Consider an instance of the graph colouring problem, namely a graph G and an integer l. We may assume that G is connected and contains no bridges. Let H_G be the graph constructed from G by first subdividing each edge of G and then connecting all the new vertices. That is, $V(H_G) = V(G) \cup \{x_{uv} \mid uv \in E(G)\}$ and $E(H_G) = \{ux_{uv}, vx_{uv} \mid uv \in E(G)\} \cup \{x_{st}x_{uv} \mid uv, st \in E(G)\}$. The graph H_G can clearly be constructed in polynomial time. It is not difficult to see that H_G is a split graph, hence it is also chordal. Moreover, one can check that the subgraph of H_G^2 induced on the vertices of G is precisely the graph G. Since G is bridgeless, H_G is also bridgeless, hence using Proposition 6 we have the following.

$$\chi_i(H_G) = \chi(H_G^2) = \chi(G) + m$$

Therefore $\chi_i(H_G)$ is at most $k = l + m$ if and only if $\chi(G)$ is at most l. That concludes the proof. □

By Proposition 10, for any chordal graph G, the injective chromatic number of G is either $\chi(G^2)$ or $\chi(G^2) - 1$. Interestingly, merely distinguishing between these two cases is already NP-complete.

Theorem 12. *It is NP-complete to decide, for a given split (and hence chordal) graph G, whether $\chi_i(G) = \chi(G^2) - 1$.* □

Now we extend the proof of Theorem 11 to show that under a certain complexity assumption, it is not tractable to approximate the injective chromatic number of a split (chordal) graph within a factor of $n^{1/3-\epsilon}$ for all $\epsilon > 0$.

Theorem 13. *Unless $NP = ZPP$, for any $\epsilon > 0$, it is not possible to efficiently approximate $\chi(G^2)$ and $\chi_i(G)$ within a factor of $n^{1/3-\epsilon}$, for any split (and hence chordal) graph G.*

Proof. In [8], it was shown that for any fixed $\epsilon > 0$, unless $NP = ZPP$, the problem of deciding whether $\chi(G) \leq n^\epsilon$ or $\alpha(G) < n^\epsilon$ for a given graph G is not solvable in polynomial time. Consider an instance of this problem, namely a graph G. Again, as in the proof of Theorem 11, we may assume that G is connected and bridgeless. Let $H_{k,G}$ be the split graph constructed from k copies of H_G (the graph used in the proof Theorem 11) by identifying, for each $uv \in E(G)$, all copies of x_{uv}. That is, if $v_1, v_2, \ldots, v_n$ are the vertices of G, we have in $H_{k,G}$ vertices $V(H_{k,G}) = \bigcup_{i=1}^k \{v_1^i, v_2^i, \ldots, v_n^i\} \cup \{x_{uv} \mid uv \in E(G)\}$, and edges $E(H_{k,G}) = \bigcup_{i=1}^k \{u^i x_{uv},\ v^i x_{uv} \mid uv \in E(G)\} \cup \{x_{uv}x_{st} \mid uv, st \in E(G)\}$.

Now since G is bridgeless, $H_{k,G}$ is also bridgeless. Consider an independent set I of $H_{k,G}^2$. It is not difficult to check that either I trivially contains only a single vertex x_{uv}, or for each pair of vertices $u^i, v^j \in I$, the vertices u and v are not adjacent in G. Hence it follows that from any colouring of $H_{k,G}^2$ one can construct a fractional k-fold colouring of G (i.e., a collection of independent sets covering each vertex of G at least k times) by projecting each non-trivial colour class of $H_{k,G}^2$ to G, i.e., mapping each u^i to u. Using this observation we obtain the following inequalities.

$$\frac{k \cdot n}{\alpha(G)} + m \leq k \cdot \chi_f(G) + m \leq \chi(H_{k,G}^2) = \chi_i(H_{k,G}) \leq k \cdot \chi(G) + m$$

Therefore if $\chi(G) \leq n^\epsilon$ then $\chi(H^2_{k,G}) \leq k \cdot n^\epsilon + m$, and if $\alpha(G) < n^\epsilon$ then $\chi(H^2_{k,G}) > k \cdot n^{1-\epsilon} + m$. Now we fix $k = m$, and denote by N the number of vertices in $H_{m,G}$. For $n \geq 2^{1/\epsilon}$ we obtain the following.

$$\frac{m \cdot n^{1-\epsilon} + m}{m \cdot n^\epsilon + m} \geq \frac{1}{2} n^{1-2\epsilon} \geq n^{1-3\epsilon} \geq (m \cdot n + m)^{\frac{1}{3}(1-3\epsilon)} = N^{\frac{1}{3}-\epsilon}$$

Hence if we can efficiently $(N^{\frac{1}{3}-\epsilon})$-approximate the colouring of $H^2_{m,G}$ then we can decide whether $\chi(G) \leq n^\epsilon$ or $\alpha(G) < n^\epsilon$. That concludes the proof. □

Note that a seemingly stronger result appeared in [1]. Namely, the authors claim that the chromatic number of the square of a split graph is not $(n^{1/2-\epsilon})$-approximable for all $\epsilon > 0$. However this result is not correct. In fact, we show below that there exists a polynomial time algorithm $\sqrt[3]{n}$-approximating the chromatic number of the square of a split graph G, and also $\sqrt[3]{n}$-approximating the injective chromatic number of G. Note that this is also a strengthening of best known $\sqrt{n}$-approximation algorithm for the chromatic number of the square in general graphs (cf. [1]). We need the following lemma.

Lemma 14. *For chordal graphs, the injective chromatic number is α-approximable if and only if the chromatic number of the square is α-approximable.* □

Theorem 15. *There exists a polynomial time algorithm that given a split graph G approximates $\chi(G^2)$ and $\chi_i(G)$ within a factor of $\sqrt[3]{n}$.*

Proof. Let G be a connected split graph with a clique X and an independent set Y. Denote by H the subgraph of G^2 induced on Y. Let $p = |X|$, $N = |V(H)|$, and $M = |E(H)|$. Clearly, $\chi(G^2) = p + \chi(H)$. Consider an optimal colouring of H with colour classes $V_1, V_2, \ldots, V_{\chi(H)}$. Let E_{ij} be the edges of H between V_i and V_j. Clearly, for each edge $uv \in E_{ij}$ there must exist a vertex x_{uv} in X adjacent to both u and v. Moreover, for any two edges $uv, st \in E_{ij}$ we have $x_{uv} \neq x_{st}$, since otherwise we obtain a triangle in $H[V_i \cup V_j]$ which is bipartite. Hence $p \geq |E_{ij}|$ and considering all pairs of colours in H we conclude that $p \geq M/\binom{\chi(H)}{2} \geq 2M/\chi^2(H)$.

A simple edge count shows that any graph with t edges can be coloured with no more than $1/2 + \sqrt{2t + 1/4}$ colours. Such a colouring can be found by a simple greedy algorithm [4]. We can apply this algorithm to H, and use additional p colours to colour the vertices of X. This way we obtain a colouring c of G^2 using at most $p + 1 + \sqrt{2M}$ colours. Using the lower bound from the previous paragraph one can prove the following inequalities (assuming $M \geq 6$ or $p \geq 17$).

$$\frac{p + 1 + \sqrt{2M}}{\chi(G^2)} \leq \frac{p + 1 + \sqrt{2M}}{p + \sqrt{\frac{2M}{p}}} \leq (2M)^{1/6} \leq N^{1/3} \leq n^{1/3}$$

Hence, the colouring c is an $\sqrt[3]{n}$-approximation of $\chi(G^2)$, and by Lemma 14 we can obtain a $\sqrt[3]{n}$-approximation of $\chi_i(G)$. □

5 Exact Algorithmic Results

Now we focus on algorithms for injective colouring of chordal graphs. Although, computing the injective chromatic number of a chordal graph is hard, the associated decision problem with a fixed number of colours has a polynomial time solution, i.e., the problem is fixed parameter tractable. We need the following lemma.

Lemma 16. *For any chordal G, the treewidth of G^2 is at most $\frac{1}{4}\Delta(G)^2 + \Delta(G)$.* □

Theorem 17. *Given a chordal graph G and a fixed integer k, one can decide in time $O\big(n \cdot k \cdot k^{(k/2+1)^2}\big)$ whether $\chi_i(G) \leq k$ and also whether $\chi(G^2) \leq k$.*

Proof. It is easy to see that if $\chi_i(G) \leq k$ or if $\chi(G^2) \leq k$, then $\Delta(G)$ must be at most k. Thus if $\Delta(G) > k$, we can reject G immediately. Using Lemma 16, we can construct in time $O(nk^2)$ a tree decomposition (T, X) of G^2 whose width is at most $k^2/4 + k$. Now, using standard dynamic programming techniques on the tree T (cf. [4,7]), we can decide in time $O(n \cdot k \cdot k^{(k/2+1)^2})$ whether $\chi(G^2) \leq k$ and whether $\chi_i(G) \leq k$. □

Now we show that for certain subclasses of chordal graphs, the injective chromatic number can be computed in polynomial time (in contrast to Theorem 11). First, we summarise the results; the details are presented in subsequent sections.

We call a graph G a *power chordal* graph if all powers of G are chordal. Recall that in Propositions 8 and 9, we showed how, from the chromatic number of the square of the graph $G - \mathcal{B}(G)$, one can compute $\chi(G^2)$ for any graph G, respectively $\chi_i(G)$ for a split graph G. The following theorem describes a similar property for the injective chromatic number in chordal graphs. The proof will follow from Corollary 25 and Theorem 28 which we prove in sections 5.2 and 5.4 respectively.

Theorem 18. *There exists an $O(n + m)$ time algorithm that computes $\chi_i(G)$ given a chordal graph G and $\chi_i(G - \mathcal{B}(G))$. Using this algorithm one can also construct an optimal injective colouring of G from an optimal injective colouring of $G - \mathcal{B}(G)$ in time $O(n + m)$.*

A class $\mathcal{C}$ of graphs is called *induced-hereditary*, if $\mathcal{C}$ is closed under taking induced subgraphs. For an induced-hereditary subclasses of chordal graphs we have the following property.

Proposition 19. *Let $\mathcal{C}$ be an induced-hereditary subclass of chordal graphs. Then the following statements are equivalent.*

(i) One can efficiently compute $\chi(G^2)$ for any $G \in \mathcal{C}$.
(ii) One can efficiently compute $\chi_i(G - \mathcal{B}(G))$ for any $G \in \mathcal{C}$.
(iii) One can efficiently compute $\chi_i(G)$ for any $G \in \mathcal{C}$.

This follows from Theorem 18, Proposition 8, and the fact that each connected component of $G - \mathcal{B}(G)$ must be in $\mathcal{C}$. In some cases, e.g., in the class of power chordal graph, this is true even if $\mathcal{C}$ is not induced-hereditary. The following corollary will follow from Theorem 18 and Corollary 27 which we prove in section 5.3.

Corollary 20. *The injective chromatic number of a power chordal graph can be computed in polynomial time.*

Thus the injective chromatic number of a strongly chordal graph can also be computed in polynomial time.

Finally, observe that due to Theorem 12 one cannot expect the property from Proposition 19 to hold for any subclass of chordal graphs.

5.1 Injective Structure

In order to prove Theorem 18, we investigate the structural properties of graphs G that allow efficient computation of $\chi_i(G)$. In this section, G refers to an arbitrary connected graph (not necessarily chordal).

A *clique separator* of G is a separator of G which induces a clique in G. A tree decomposition (T, X) of G is a *decomposition by clique separators*, if for any $uv \in E(T)$, the set $X(u) \cap X(v)$ induces a clique in G. This type of decomposition of graphs was introduced and studied by Tarjan [19]. The decomposition turns out to be particularly useful for the graph colouring problem; namely, one can efficiently construct an optimal colouring of G from optimal colourings of $G[X(u)]$ for all $u \in V(T)$. We define and study a similar concept for the injective colouring problem. Recall that $G^{(2)}$ denotes the common neighbour graph of G defined in section 3.

We say that a subset S of vertices of G is *injectively closed*, if for any two vertices $x, y \in S$ having a common neighbour in G, there exists a common neighbour of x and y that belongs to S. A subset S of vertices of G is called an *injective clique*, if S induces a clique in $G^{(2)}$. Note that an injective clique is not necessarily injectively closed in G. A subset of vertices S of G is called an *injective separator* of G, if S is injectively closed in G, S is a separator of $G^{(2)}$, and $G^{(2)}$ is connected. Note that $G^{(2)}$ can be disconnected even if G is connected, e.g., if G is bipartite. An *injective decomposition* of G is a tree decomposition (T, X) of G such that for any $uv \in E(T)$, the set $X(u) \cap X(v)$ is an injective separator of G. An injective separator S is an *injective clique separator*, if S is also an injective clique. An *injective clique decomposition* is an injective decomposition (T, X) such that for any $uv \in E(T)$, the set $X(u) \cap X(v)$ is an injective clique. Note that any injective clique decomposition of G is a decomposition of $G^{(2)}$ and G^2 by clique separators.

We have the following properties.

Lemma 21. *Let (T, X) be an injective decomposition of a graph G. Then for each $u \in V(T)$, the set $X(u)$ is injectively closed.* □

Theorem 22. *Let (T, X) be an injective clique decomposition of a graph G. Then*

$$\chi_i(G) = \chi(G^{(2)}) = \max_{u\in V(T)} \chi\Big(G^{(2)}\big[X(u)\big]\Big) = \max_{u\in V(T)} \chi_i\Big(G\big[X(u)\big]\Big).$$

Proof. The first equality is by definition. We obtain the second equality from the fact that (T, X) is a decomposition of $G^{(2)}$ by clique separators. The last equality follows easily, since by Lemma 21, we have $G^{(2)}[X(u)] = G[X(u)]^{(2)}$, and by definition $\chi\big(G[X(u)]^{(2)}\big) = \chi_i\big(G[X(u)]\big)$. □

5.2 Computing $\chi_i(G)$ in Chordal Graphs

In this section, we focus on injective clique decompositions of chordal graphs. The following is easy to check.

Observation 23. *Let H be a bridgeless graph having a dominating vertex. Then H is an injective clique.* □

We say that a graph G is *decomposable*, if G contains an an injective clique separator S; we say that S *decomposes* G. A graph G is *indecomposable*, if it is not *decomposable*. A graph G is called *perfectly tree-dominated*, if G contains an induced tree T, such that any vertex and any connected component of $G - V(T)$ has exactly one neighbour in T. For such T, we say that T *perfectly dominates* G, or that G is *perfectly dominated* by T.

The following statement relates indecomposable chordal graphs and perfectly tree-dominated graphs.

Proposition 24. *Any perfectly tree-dominated graph is indecomposable. Any indecomposable chordal graph is either perfectly tree-dominated or bridgeless.* □

The property above has an important corollary.

Corollary 25. *For any chordal graph G, there exists an injective clique decomposition (T, X) of G, such that for any $u \in V(T)$, the set $X(u)$ induces either a bridgeless graph or a perfectly tree-dominated graph. This decomposition can be constructed in time $O(n + m)$.*

Proof. First, we find the bridges $\mathcal{B}(G)$ of G. Then, we construct a tree decomposition (T, X) of G such that for $u \in V(T)$, the set $X(u)$ is either a connected component of $G - \mathcal{B}(G)$, or a connected component T of $G[\mathcal{B}(G)]$ augmented with the neighbours of T in G. It follows from the proof of Proposition 24 that (T, X) is a injective clique decomposition. □

5.3 Bridgeless Chordal Graphs

In this section, we describe some classes of chordal graphs G that allow efficiently computing $\chi(G^2)$.

We focus on chordal graphs whose square is also a chordal graph. Clearly, for any such graph G, one can efficiently colour the square of G. Chordal graphs whose powers are also chordal were already studied in the past. In particular, it was shown by Duchet [16] that for any k, if G^k is chordal, then also G^{k+2} is chordal. Therefore, if a chordal graph G has a chordal square, then any power of G must be chordal, that is, G is power chordal. Interestingly, many known subclasses of chordal graphs, e.g. trees, interval graphs, and strongly chordal graphs, were shown to be power chordal [1]. Moreover, Laskar and Shier [16] found the following subgraph characterisation of power chordal graphs. A k-sun is a graph formed by a cycle $v_0, v_1, \ldots, v_{k-1}$ with edges $v_i v_{i+1}$ (and possibly other edges), and an independent set $w_0, w_1, \ldots w_{k-1}$, where w_i is adjacent only to v_i and v_{i+1} (all indices are taken modulo k). A k-sun of a graph G is *suspended* in G, if there exists a vertex z in G adjacent to w_i and w_j where $j \neq i$ and $j \neq i \pm 1$ modulo k.

Theorem 26. [16] *A graph G is power chordal if and only if any k-sun of G, $k \geq 4$, is suspended.*

Based on this characterisation, it is easy to check the following.

Corollary 27. *If G is power chordal, the graph $G - \mathcal{B}(G)$ is also power chordal.* □

Note that by Theorem 26, strongly chordal graphs are trivially power chordal, since no strongly chordal graph can contain an induced k-sun, $k \geq 3$ [6]. Also notice, that the class of power chordal graphs is not induced-hereditary (closed under taking induced subgraphs), since a graph that contains a k-sun can be power chordal, but the k-sun itself (taken as an induced subgraph) is not.

5.4 Perfectly Tree-Dominated Graphs

In this section, we show how to efficiently compute the injective chromatic number of a perfectly tree-dominated graph.

Let G be a perfectly tree-dominated graph. If G is a tree, then by Proposition 2, we have $\chi_i(G) = \Delta(G)$, and a greedy injective colouring of G will be optimal. Otherwise, let T be a minimal tree perfectly dominating G. We define a tree decomposition (T_G, X) of G as follows. We set $T_G = T$, and for $u \in V(T)$, we set $X(u) = N(u) \cup \{u\}$. Clearly, $X(u) \cap X(v) = \{u, v\}$ is injectively closed, and the set $X(u) \cap X(v)$ is a separator of $G^{(2)}$. Hence (T, X) is an injective decomposition of G. Note that for any $u \in V(T)$, the graph $G[X(u)]$ admits only $\deg(u)$ different injective colourings, up to renaming colours. It follows, that using dynamic programming on the rooted tree T, one can determine $\chi_i(G)$ and an optimal injective colouring of G, by computing, for each $u \in V(T)$ and each colouring of $G[X(u)]$, the minimum number of colours needed to injectively colour the subgraph of G induced on the union of $X(u)$ and the sets $X(v)$ for all descendants v of u. Using an additional simple argument it can be shown that the algorithm we just described can be performed in time $O(n + m)$. Hence we have the following theorem.

Theorem 28. *The injective chromatic number $\chi_i(G)$ and an optimal injective colouring of a perfectly tree-dominated graph G can be computed in time $O(n + m)$.*

The above algorithm turns out to be an instance of a more general approach to graph colouring problems [18].

Note added in proof

We have just learned of a related result of Král'[15] showing that $\chi(G^2) = O(\Delta(G)^{3/2})$ for any chordal G. This is easily seen to allow strengthening Theorem 15 from split to chordal graphs.

References

1. Agnarsson, G., Greenlaw, R., Halldorsson, M.: On Powers of Chordal Graphs and Their Colorings. Congress Numerantium 100, 41–65 (2000)
2. Bodlaender, H.L., Kloks, T., Tan, R.B., van Leeuwen, J.: Approximations for λ-Coloring of Graphs. The Computer Journal 47, 193–204 (2004)
3. Calamoneri, T.: The $L(h, k)$-Labelling Problem: A Survey and Annotated Bibliography. The Computer Journal 49, 585–608 (2006)
4. Diestel, R.: Graph Theory, 3rd edn. Graduate Texts in Mathematics, vol. 173. Springer, Heidelberg (2005)
5. Ekim, T., Hell, P., Stacho, J., de Werra, D.: Polarity of Chordal Graphs. Discrete Applied Mathematics (to appear)
6. Farber, M.: Characterizations of strongly chordal graphs. Discrete Mathematics 43, 173–189 (1983)
7. Feder, T., Hell, P., Klein, S., Nogueira, L.T., Protti, F.: List matrix partitions of chordal graphs. Theoretical Computer Science 349, 52–66 (2005)
8. Feige, U., Killian, J.: Zero Knowledge and the Chromatic Number. Journal of Computer and System Sciences 57, 187–199 (1998)
9. Fiala, J., Kratochvíl, J.: Partial covers of graphs. Discussiones Mathematicae Graph Theory 22, 89–99 (2002)
10. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, New York (1980)
11. Griggs, J.R., Yeh, R.K.: Labelling graphs with a condition at distance 2. SIAM Journal on Discrete Mathematics 5, 586–595 (1992)
12. Hahn, G., Kratochvíl, J., Širáň, J., Sotteau, D.: On the injective chromatic number of graphs. Discrete Mathematics 256, 179–192 (2002)
13. Hell, P., Klein, S., Nogueira, L.T., Protti, F.: Partitioning chordal graphs into independent sets and cliques. Discrete Applied Mathematics 141, 185–194 (2004)
14. Jensen, T.R., Toft, B.: Graph coloring problems. Wiley Interscience Series in Discrete Mathematics. Wiley, New York (1995)
15. Král̆, D.: Coloring Powers of Chordal Graphs. SIAM Journal on Discrete Mathematics 18, 451–461 (2005)
16. Laskar, R., Shier, D.: On powers and centers of chordal graphs. Discrete Applied Mathematics 6, 139–147 (1983)
17. Robertson, N., Sanders, D.P., Seymour, P., Thomas, R.: Efficiently four-coloring planar graphs. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (STOC), pp. 571–575.
18. J. Stacho: Ph.D. Thesis, Simon Fraser University (2008)
19. Tarjan, R.: Decomposition by clique separators. Discrete Mathematics 55, 221–232 (1985)
20. West, D.: Introduction to Graph Theory. Prentice Hall, Englewood Cliffs (1996)

Spanning Trees with Many Leaves in Graphs without Diamonds and Blossoms

Paul Bonsma* and Florian Zickfeld**

Technische Universität Berlin, Fak. II, Str. des 17. Juni 136, 10623 Berlin, Germany
bonsma,zickfeld@math.tu-berlin.de

Abstract. It is known that graphs on n vertices with minimum degree at least 3 have spanning trees with at least $n/4+2$ leaves and that this can be improved to $(n+4)/3$ for cubic graphs without the diamond $K_4 - e$ as a subgraph. We generalize the second result by proving that every graph with minimum degree at least 3, without diamonds and certain subgraphs called blossoms, has a spanning tree with at least $(n+4)/3$ leaves. We show that it is necessary to exclude blossoms in order to obtain a bound of the form $n/3+c$.

We use the new bound to obtain a simple FPT algorithm, which decides in $O(m)+O^*(6.75^k)$ time whether a graph of size m has a spanning tree with at least k leaves. This improves the best known time complexity for Max-Leaves Spanning Tree.

1 Introduction

This paper is concerned with finding spanning trees with maximum number of leaves. This is a well-studied problem with many applications. Different types of algorithms have been proposed for this $\mathcal{NP}$-hard problem. Constant factor approximation algorithms appear e.g. in [4,9]. In addition constructive methods exist that guarantee a certain fraction of the vertices to become leaves, when the graph satisfies certain properties [2,7,8]. Hence these results give lower bounds, which are *extremal* in the sense that examples are given which show that they are tight for their graph classes. Thirdly, FPT algorithms for the related decision problem have also received much attention, see [1,2,3,5]. These results are closely related; lower bounds have been used to find good approximations [4] and fast FPT algorithms [2,3]. Our contribution is a new extremal lower bound that generalizes and strengthens previous results, and a fast and simple FPT algorithm based on this bound.

We first introduce the extremal setting and explain our result. We then explain how this helps to obtain an improved FPT algorithm. Throughout this paper G is assumed to be a simple and connected graph on $n \geq 2$ vertices. Other graphs may be multi-graphs, disconnected, or a K_1. The minimum vertex degree of G is denoted by $\delta(G)$.

* Supported by the Graduate School "Methods for Discrete Structures" in Berlin, DFG grant GRK 1408.

** Supported by the Studienstiftung des deutschen Volkes.

Linial and Sturtevant first observed that every graph G with $\delta(G) \geq 3$ has a spanning tree with at least $n/4 + 2$ leaves and that this bound is best possible (unpublished). Kleitman and West give a proof in [8] and also give stronger bounds for graphs of higher minimum degree.

The examples showing that the bound $n/4 + 2$ is best possible for graphs of minimum degree 3 are all obtained from a cycle by replacing every vertex by a cubic diamond. A *diamond* is the graph K_4 minus one edge, and a diamond subgraph of a graph G is a *cubic diamond* if its four vertices all have degree 3 in G, see Figure 1 (a).

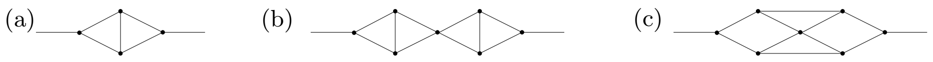


Fig. 1. A cubic diamond (a), a 2-necklace (b), and a 2-blossom (c)

Since these examples are very restricted it is natural to ask if better bounds can be obtained for graphs without cubic diamonds. This question was answered by Griggs, Kleitman and Shastri [7] for cubic graphs. They show that a cubic graph G without diamonds always admits a spanning tree with at least $n/3 + 4/3$ leaves and that this is best possible. For minimum degree 3 the following bound is proved in [2]. A graph G with $\delta(G) \geq 3$, without cubic diamonds, contains a spanning tree with at least $2n/7 + 12/7$ leaves. Replacing every vertex of a cycle by the graph from Figure 1 (b) shows that the factor $2/7$ cannot be improved. The following conjecture from [2] therefore seems natural: it was conjectured that every graph G with $\delta(G) \geq 3$ and without *2-necklaces* contains a spanning tree with at least $n/3+4/3$ leaves. Informally speaking, a 2-necklace is a concatenation of $k \geq 1$ diamonds with only two outgoing edges, see Figure 1 (b). In Definition 1 in the next section a precise definition is given. This statement, if true, would improve the bound $2n/7 + 12/7$ with only a minor extra restriction, and generalize the $n/3 + 4/3$ bound for cubic graphs from [7].

In Section 2 we disprove this conjecture by constructing graphs with $\delta(G) = 3$ without 2-necklaces, which do not admit spanning trees with more than $4n/13 + 24/13$ leaves. On the positive side, we prove that the statement is true after only excluding one more very specific structure, called a *2-blossom*, see Figure 1 (c). A precise definition is given in Definition 2. We generalize the statement further by removing any restriction on the minimum degree. This yields Theorem 1, our main theorem, which is proved in Section 3. Let $V_{\geq 3}(G)$ denote the set of vertices in G with degree at least 3 and $n_{\geq 3}(G)$ its cardinality. Let $\ell(T)$ be the number of leaves of a graph T.

Theorem 1. *Let G be a simple, connected, non-trivial graph which contains neither 2-necklaces nor 2-blossoms. Then, G has a spanning tree T with $\ell(T) \geq n_{\geq 3}(G)/3 + \alpha$, where $\alpha = 2$ if $\delta(G) \leq 2$, and $\alpha = 4/3$ otherwise.*

The proof is constructive and can be turned into a polynomial time algorithm for the construction of a spanning tree. The main technical contribution of this paper is that we prove this generalization of the statement in [7], and improvement of the statement in [2], without a proof as lengthy as the proofs in these two papers. This is made possible by extending the techniques and proofs from [7]. In Section 3 we argue that the long case study in [7] actually proves a strong new lemma, which we use as an important step in the proof of Theorem 1. We share the opinion expressed in [7] that a significantly shorter proof of the bound for cubic graphs might not exist. Therefore using that result in order to prove the more general statement seems appropriate.

In Section 4 we explain the consequences of Theorem 1 for FPT algorithms (short for *fixed parameter tractable*) for the following decision problem.

Max-Leaves Spanning Tree (MaxLeaf):
INSTANCE: A graph G and integer k.
QUESTION: Does G have a spanning tree T with $\ell(T) \geq k$?

When choosing k as a parameter, an algorithm for MaxLeaf is called an *FPT algorithm* if its complexity is bounded by $f(k)g(n)$, where $g(n)$ is a polynomial. See [6] for an introduction to FPT algorithms. $f(k)$ is called the *parameter function* of the algorithm. Usually, $g(n)$ will turn out to be a low degree polynomial, thus to assess the speed of the algorithm it is mainly important to consider the growth rate of $f(k)$. Bodlaender [1] constructed the first FPT algorithm for MaxLeaf with a parameter function of roughly $(17k^4)!$. Since then, considerable effort has been put in finding faster FPT algorithms for this problem. The fastest algorithms can be found in [3,5,2], which also give an overview of older results. These papers also establish a strong connection between extremal graph-theoretic results and fast FPT algorithms. The $n/4 + 2$ bound mentioned above is an essential ingredient in [3]. With the same techniques, the $2n/7 + 12/7$ bound is used in [2] to obtain the so far fastest algorithm with a parameter function in $O^*(\binom{3.5k}{k}) \subset O^*(8.12^k)$. Here the O^* notation ignores polynomial factors. Similarly Theorem 1 yields a new FPT algorithm for MaxLeaf.

Theorem 2. *There exists an FPT algorithm for* MaxLeaf *with time complexity $O(m) + O^*(6.75^k)$, where m denotes the size of the input graph and k the desired number of leaves.*

This algorithm is the new fastest FPT algorithm for MaxLeaf, both optimizing the dependency on the input size and the parameter function. It simplifies the ideas introduced by Bonsma, Brueggemann and Woeginger [3] and therefore is also significantly simpler than the other recent fast FPT algorithms. Hardly any preprocessing of the input graph is needed, since Theorem 1 is already formulated for a very broad graph class.

We end in Section 5 with a discussion of further improvements and applications of our results. Due to space constraints we cannot include complete proofs of Theorems 1 and 2 in this extended abstract. Instead we refer the reader to the full version of this paper for more details.

2 Obstructions for Spanning Trees with Many Leaves

As mentioned in the introduction, 2-necklaces have been identified as an obstruction for the existence of spanning trees with $n/3 + c$ leaves in graphs with minimum degree 3, see [7,8,2]. In this section we show that they are not the only such obstruction. We start by precisely defining 2-necklaces and 2-blossoms.

The degree of a vertex v in a graph G is denoted by $d_G(v)$ and by $d(v)$ if ambiguities can be excluded. A vertex v of a subgraph H of G with $d_H(v) < d_G(v)$ is called a *terminal* of H.

Definition 1 (2-Necklace). *The graph K_4 minus one edge is called a* diamond *and denoted by N_1. The degree 3 vertices are the* inner vertices *of the diamond.*

For $k \geq 2$ the diamond necklace *N_k is obtained from the graph N_{k-1} and a vertex disjoint N_1 by identifying a degree 2 vertex of N_1 with a degree 2 vertex of N_{k-1}. The two unique degree 2 vertices of N_k are denoted by c_1 and c_2.*

An N_k subgraph of G is a 2-necklace *if it only has c_1 and c_2 as terminals, which both have degree 3 in G. See Figures 1 (a) and (b). If G contains an N_1 this way, this N_1 subgraph is also called a* cubic diamond *of G.*

Definition 2 (2-Blossom). *The graph B on seven vertices shown in Figure 2 (a) is the blossom graph. A blossom subgraph B of G is a* 2-blossom *if c_1 and c_2 are its only terminals, and they both have degree 3 in G, see Figure 1 (c).*

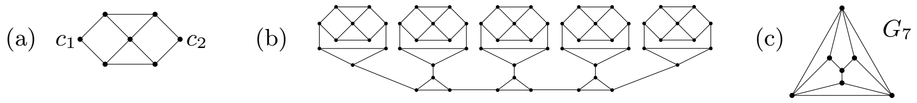


Fig. 2. A blossom, five flowers in a tree, and G_7

The two outgoing edges of 2-necklaces and 2-blossoms may in fact be the same edge, in that case G is just a 2-necklace or 2-blossom plus an edge.

The building block for the graph family that shows that the bound $n/3 + c$ cannot be achieved when only necklaces are excluded, is a graph on ten vertices called a *flower*. The graphs obtained from ternary trees by replacing inner vertices with triangles, and leaves with flowers, have no spanning tree with more than $4n/13 + 24/13$ leaves. Figure 2 (b) shows such a graph and it can be checked that these graphs have no more than four leaves per flower.

An important graph for our proofs, called G_7, is shown in Figure 2 (c). This is a non-cubic graph without 2-necklaces or 2-blossoms that does not admit a spanning tree with at least $n_{\geq 3}(G)/3+2$ leaves; only four leaves can be obtained. In fact, a more detailed proof of Theorem 1 shows that G_7 is the only such graph. The cubic examples from [7] show that the bound in Theorem 1 is best possible (ignoring small differences in the constant, see Section 5). These examples can also be modified to obtain extremal examples that have vertices of degree 1, 2 and 4, thus Theorem 1 is best possible in a strong sense.

3 Proof of the Main Theorem

This section is devoted to the proof of Theorem 1. The proof approach is based on [7]. We first sketch an overview of the proof of [7], then mention how we extend this method, and later give precise definitions. The two main techniques of the proof are *tree extensions* and *reduction rules*. For certain graphs G, the following approach can be used to find a tree with the desired number of leaves: start with a small tree subgraph T of G, which has the proper ratio between the number of leaves, and the number of vertices of $V_{\geq 3}(G)$ that it contains. Loosely speaking, it needs at least one leaf for every three vertices from $V_{\geq 3}(G)$. This is expressed more precisely by the value $\mathcal{P}_G(T)$ defined below, which should remain above a certain value (usually 0). Then the initial tree T is *extended* iteratively, without decreasing $\mathcal{P}_G(T)$. Proving that this is always possible is done with a case study, which considers the 'outside' of T, i.e. the subgraph of G that is not yet covered by T. When T becomes spanning, the bound follows easily from the initial lower bound on $\mathcal{P}_G$.

However there are certain substructures that are problematic for this approach. These structures are handled instead by *reduction rules*, before building the spanning tree. Every reduction rule works on a certain graph structure, and removes it. These reduction rules are applied to the graph G as long as possible, in arbitrary order. The resulting graph G' is called *irreducible*, and may be disconnected. These rules are chosen such that if a tree with the desired number of leaves is given for every component of G', it can be used to construct a tree with the right number of leaves for G. It follows that the extension procedure only has to be applied for irreducible graphs.

For our proof, it does not suffice to only consider tree extensions: it may be necessary to start new tree components, so we will actually extend a *forest* F. This is no problem when we demand an extra number of leaves for every new tree component in F. We continue with these extensions until F is a spanning forest. We add edges to obtain a spanning tree T, for which the bound will follow from $\mathcal{P}_G(F) \geq 0$.

We now state the necessary notions and lemmas more precisely, and use these to prove Theorem 1 formally. The proofs of the lemmas and details of the reduction rules are postponed to later sections, or omitted. A vertex with degree at most 2 will be called a *goober*. One important convention is that when we consider a subgraph H of G, goobers in H are always defined with respect to G, not with respect to H. In our figures, e.g. Figure 3 white vertices indicate goobers. A *high-degree vertex* is a vertex of degree at least 4.

In Section 3 the reduction rules are introduced, and it will be shown that all of them maintain the proper leaf ratio of a spanning tree, when reversed. This is expressed by the following lemma. Note that F is a maximal forest for G' if and only if it consists of a spanning tree for every component of G. A *trivial component* is an isolated vertex.

Lemma 1 (Reconstruction Lemma). *Let G' be the result of applying a reduction rule to a connected graph G, and let k be the number of non-trivial com-*

ponents of G', and $\beta \geq 0$. If G' has a maximal forest with at least $n_{\geq 3}(G')/3 + 2k - \beta$ leaves, then G has a spanning tree with at least $n_{\geq 3}(G)/3 + 2 - \beta$ leaves.

This lemma will be used with either $\beta = 0$ or $\beta = 2/3$. For showing that the graph G' in this lemma indeed has a spanning tree with at least $n_{\geq 3}(G')/3 + 2k - \beta$ leaves, it is important that the reduction rules maintain the conditions stated in Theorem 1. This is shown by the next lemma, more details about the lemma are given in Section 3. The multi-graph with two vertices and two parallel edges is denoted by $K_2 + e$.

Lemma 2. *Let G' be obtained from a simple, connected graph G without 2-necklaces or 2-blossoms by the application of a reduction rule. Then (i) G' is connected, or every component of G' contains a goober, and (ii) every component of G' is either simple or it is a $K_2 + e$, and (iii) G' contains neither 2-necklaces nor 2-blossoms, and (iv) if G contains a goober, then G' contains a goober.*

We now state the definitions and lemmas used in proving extendibility of a forest subgraph. When F and G are graphs, $F \subseteq G$ and $F \subset G$ denote the subgraph resp. proper subgraph relation. Let $F \subseteq G$. By $n_G(F)$ we denote the number of non-goober vertices of G that are in $V(F)$. By $\ell_d(F)$ we denote the number of *dead leaves* of F, that is leaves of F, which have no neighbor in $V(G) \setminus V(F)$. Let $cc(F)$ denote the number of connected components of F.

Definition 3 (Leaf-Potential). *The* leaf-potential *of a subgraph $F \subseteq G$ is $\mathcal{P}_G(F) = 2.5\ell(F) + 0.5\ell_d(F) - n_G(F) - 6cc(F)$.*

We now show that if G has a spanning subgraph F with $\mathcal{P}_G(F) \geq 0$, then a tree satisfying the desired bound exists. We may assume that F is a forest. Since all leaves of F are dead, we have $0 \leq \mathcal{P}_G(F) = 3\ell(F) - n_{\geq 3}(G) - 6cc(F)$, and thus $\ell(F) \geq n_{\geq 3}(G)/3 + 2cc(F)$. We can now add $cc(F) - 1$ edges to F to obtain a spanning tree T, losing at most $2(cc(F) - 1)$ leaves, so $\ell(T) \geq n_{\geq 3}(G)/3 + 2$.

Definition 4 (Extendible). *Let F be a subgraph of a graph G. Then F is called* extendible *if there exists an F' with $F \subset F' \subseteq G$ and $\mathcal{P}_G(F') \geq \mathcal{P}_G(F)$, and F' is called an* extension*.*

Above we already informally mentioned the subgraph of G 'outside' a subgraph $F \subset G$. This graph may formally be defined as an edge induced graph as follows.

Definition 5 (Graph Outside F). *Let F be a non-spanning subgraph of G. The* subgraph of G outside of F *is $F^C = G[\{uv \in E(G) : u \notin V(F)\}]$.*

Note that no edges between two vertices that are both in $V(F)$ are included in F^C. We can now formulate the two lemmas that together show that every forest subgraph in an irreducible graph without 2-necklaces and 2-blossoms is extendible. More details are given in Section 3.

Lemma 3 (Start Lemma). *Let $G \neq G_7$ be an irreducible graph, and let F be a (possibly empty) subgraph of G such that F^C contains at least one high-degree vertex and contains neither 2-necklaces nor 2-blossoms. Then F is extendible.*

Lemma 4 (Extension Lemma). *Let G be an irreducible graph, and let F be a non-empty subgraph of G such that F^C has maximum degree 3 and contains no 2-necklaces. Then F is extendible.*

The following theorem appears in [7] as Theorem 3 (reformulated slightly for our purposes).

Theorem 3. *Every simple, connected, irreducible graph G of maximum degree exactly 3 has a spanning tree with at least $n_{\geq 3}(G)/3 + \alpha$ leaves, where $\alpha = 4/3$ if G is cubic and $\alpha = 2$ otherwise.*

We now have collected all the necessary tools to prove Theorem 1.

Proof of Theorem 1. We prove, by induction over the number of edges, that every simple, connected, non-trivial graph G without 2-necklaces or 2-blossoms, has a spanning tree T with $\ell(T) \geq n_{\geq 3}(G)/3 + \alpha$, where $\alpha = 4/3$ if $\delta(G) \geq 3$, and $\alpha = 2$ if $\delta(G) \leq 2$.

First suppose G is irreducible. If G has maximum degree exactly 3, Theorem 1 follows immediately from Theorem 3. If G has maximum degree at most 2, G has a spanning tree with at least two leaves (since we assumed that G is not a K_1), which suffices.

If $G = G_7$, then a spanning tree with $4 = n_{\geq 3}(G)/3 + 5/3$ leaves can be obtained. So we may now assume that G contains at least one high degree vertex, and is not equal to G_7.

We start with an empty subgraph F of G which has $\mathcal{P}_G(F) = 0$. The Start Lemma (Lemma 3) shows that, as long as there is at least one high degree vertex not in F, we can extend F while maintaining $\mathcal{P}_G(F) \geq 0$. When all high degree vertices are included in F, the Extension Lemma (Lemma 4) can be applied iteratively, until a spanning subgraph F' is obtained with $\mathcal{P}_G(F') \geq 0$. By our observation following Definition 3, it then follows that G has a spanning tree with at least $n_{\geq 3}(G)/3 + 2$ leaves.

It remains to consider the case that G is reducible (the induction step). We can apply a reduction rule, such that the resulting graph G' again contains no 2-necklaces or 2-blossoms, and such that every component is either simple or a $K_2 + e$ (Lemma 2). First suppose G' is connected. By Lemma 2, if $\delta(G) \leq 2$, then $\delta(G') \leq 2$, and by induction G' has a spanning tree with at least $n_{\geq 3}(G')/3 + 2$ leaves. Lemma 1 then shows that G admits a spanning tree with at least $n_{\geq 3}(G)/3 + 2$ leaves. Similarly, if $\delta(G) \geq 3$ then it follows that G has a spanning tree with at least $n_{\geq 3}(G)/3 + 4/3$ leaves. Now suppose the reduction rule yields a disconnected graph G'. Then every resulting component has a goober (Lemma 2). So by induction, every non-trivial component C of G' has a spanning tree with at least $n_{\geq 3}(C)/3 + 2$ leaves. Thus Lemma 1 implies that G has a spanning tree with at least $n_{\geq 3}(G)/3 + 2$ leaves. □

Reducible Structures. We now present the reduction rules. We introduce five reduction rules that will be called the *high-degree reduction rules.* Each consists of an operation on a certain subgraph, and conditions for when it may be applied.

Figure 3 shows the graph operations for the five rules. For each rule, on the left the subgraph is shown that is reduced by the rule, and on the right the resulting subgraph, which has the same terminal set. The encircled vertices are the terminals, which may have further incidences, unlike the other vertices. In some cases outgoing half edges are added to indicate conditions on minimum vertex degrees. None of the vertices in the figures may coincide, but there are no restrictions on outgoing edges sharing end vertices. Goobers are shown as white vertices, but in the case of (R3), vertex v or w may also become a goober, depending on the original degree. The numbers above the arrows indicate the decrease in $n_{\geq 3}$.

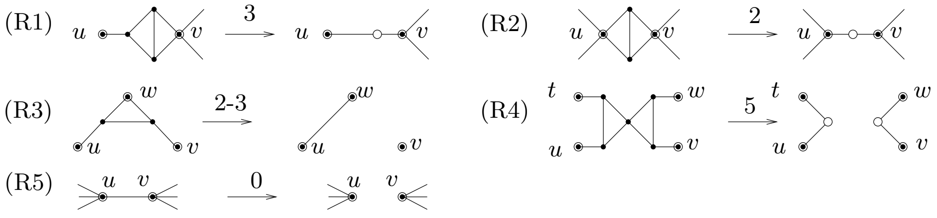


Fig. 3. The high-degree reduction rules

The following restrictions are imposed on applying these operations to a graph G. First, no reduction rule may be applied if (i) it introduces multi-edges that are incident with a non-goober, or if (ii) it introduces a 2-necklace or 2-blossom. In addition, the following rule-specific restrictions are imposed: for (R1), $d_G(v) \geq 4$ is required. For (R2), $d_G(u) \geq 4$ and $d_G(v) \geq 4$ are required. Rule (R3) may not disconnect a component, and may introduce at most one new goober. Rule (R4) may only be applied if it *does* disconnect a component. For (R5), $d_G(u) \geq 4$ and $d_G(v) \geq 4$ are required, and uv may not be a bridge. A *bridge* is an edge which upon deletion increases the number of components.

Considering these conditions on the applicability of the rules, it is obvious that Lemma 2 holds for the high-degree reduction rules. For the proof of Lemma 1, consider Figure 4. This figure shows the *tree reconstructions* for the high-degree reduction rules. If the application of a reduction rule on G gives a graph G', then without loss of generality, a spanning tree T' of G' has one of the forms shown on the left. On the right it is shown how to adapt T' to obtain a spanning tree of G. Dashed edges are present in the resulting tree if and only if they are part of T'. For the rules (R1), (R2) and (R3), one more leaf is gained, which is enough since $n_{\geq 3}(G) - n_{\geq 3}(G') \leq 3$ for these rules and G' is connected, i.e. $k = 1$. For rule (R4), no leaves are gained, but (R4) disconnects the graph and $k = 2$. Thus the increase of $n_{\geq 3}$ by 5 is compensated since each of the two components brings with it an additive term of 2. For (R5), nothing has to be proved, so Lemma 1 holds for the high-degree reduction rules.

Besides the rules (R1)-(R5) we use the seven reduction rules that are defined in [7], and we call them the *low-degree reduction rules.* These rules are shown in

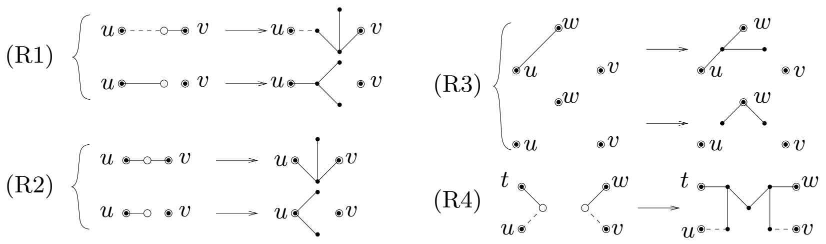


Fig. 4. Spanning tree constructions when reversing the new reduction rules

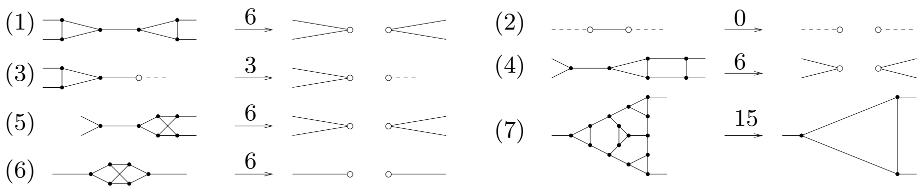


Fig. 5. The seven low-degree reduction rules

Figure 5. Conditions on their applicability are given in [7] which ensure that both Lemmas 1 and 2 hold for them. A graph to which none of the twelve reductions can be applied is called *irreducible.*

Extension Lemmas. We now sketch the proofs of Lemmas 3 and 4, which handle the construction of an extension F' of a subgraph $F \subseteq G$. Lemma 3 is proved by considering all possible neighborhoods of a high-degree vertex v of G that is not yet included in F. For every possibility, we can either give an extension of the forest that does not decrease $\mathcal{P}_G$, or we can identify a reducible structure around v, which is a contradiction with the irreducibility of G. For details of this case study we refer to the full version of this paper. We first consider the cases where v is at distance at most two of a vertex that is already in F. These cases are easily handled by extending existing trees. However when v is at a larger distance from F, we instead build a new tree around v, and add it to the forest F. Note that for instance adding a star consisting of non-goobers around a vertex of degree 5 increases $\mathcal{P}_G$ by $2.5 \cdot 5 - 6 - 6 = 0.5$, which thus gives a valid extension. Higher degrees and goobers increase $\mathcal{P}_G$ even further. For degree 4 vertices v, more cases need to be considered, but a valid extension can always be found unless v is part of two edge-disjoint triangles and all its neighbors have degree 3. However, most of these cases are reducible, provided v is not the center of a 2-blossom. In the remaining cases an extension can be found.

We now argue that the long case study in [7] in fact proves the Extension Lemma (Lemma 4). First of all we remark that in [7], goobers are defined differently, namely as vertices of degree at most two *resulting from reduction rules.* Considering the reduction rules, it can be seen that this extra condition adds no information (for instance about the possible neighborhoods of goobers). Indeed, no such information is used in the proofs in [7], and thus goobers may simply be defined as we do. The case study in Section 4 of [7] proves the following statement, expressed using our notations.

Lemma 5. *Let G be a graph with maximum degree exactly 3, without diamonds, that is irreducible with respect to the low-degree reduction rules. Let F be a non-empty tree subgraph of G. Then there exists a tree F' with $F \subset F' \subseteq G$ and*

$$2.5(\ell(F') - \ell(F)) + 0.5(\ell_d(F') - \ell_d(F)) - (n_G(F') - n_G(F)) \geq 0.$$

The most important observation is that nowhere in the case study that proves Lemma 5, any information about the current tree F is used; only information about what we defined as F^C is used. In particular, the fact that F is connected is never used in the proof, and neither are upper bounds on degrees of vertices already included in F. So it suffices to state the maximum degree 3 condition for F^C, and the condition that F is a tree may be removed. Furthermore, an irreducible graph with maximum degree 3 that contains no 2-necklaces does not contain any diamonds as subgraphs. So we may replace the 'no diamond' condition by the 'no 2-necklace' condition. Our definition of irreducible implies irreducibility with respect to the low-degree reduction rules, so this change is also not a problem. Finally, the graph F' that is constructed by Lemma 5 has the same number of components as F, so the expression in Lemma 5 simply means that $\mathcal{P}_G(F') \geq \mathcal{P}_G(F)$. Altogether this yields Lemma 4.

4 A Fast FPT Algorithm for MaxLeaf

In this section we present a fast and relatively simple FPT algorithm for MaxLeaf, which uses Theorem 1 as an essential ingredient. The other two ingredients are a short preprocessing step, consisting of two reduction rules, and an enumerative procedure, which is similar to the one introduced in [3], and also applied in [2].

We start by presenting the two reduction rules that constitute the preprocessing phase. Recall that in 2-necklaces and 2-blossoms both terminals have degree 3 in G. The rules we introduce now also reduce diamonds and blossoms whose two terminals have arbitrary degree. However the two terminals of the subgraph must still be the two vertices that have degree 2 in the diamond necklace or blossom itself. Such a subgraph of G will be called a *2-terminal diamond* respectively a *2-terminal blossom*. Rule (F1) in Figure 6, which resembles rule (R2), reduces 2-terminal diamonds. Since a 2-necklace N_k consists of k 2-terminal diamonds, those are reduced as well by rule (F1). Rule (F2) in Figure 6 reduces 2-terminal blossoms.

Lemma 6. *Let G' be the result of applying reduction (F1) or (F2) to G. Then $(G', k-1)$ is a YES-instance for* MaxLeaf *if and only if (G, k) is a YES-instance for* MaxLeaf.

Throughout this section we will denote the set of leaves of a graph G by $L(G)$. We now explain how to obtain a graph $S(G)$ from a graph G by suppressing

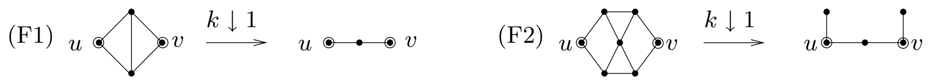


Fig. 6. Two reduction rules for an instance (G, k) of MaxLeaf

vertices. *Suppressing* a vertex u of degree 2 means deleting u and adding an edge between the two end vertices of the incident edges. We allow this operation to introduce parallel edges and loops, so the degrees of non-suppressed vertices are maintained. If $n_{\geq 3}(G) = 0$, that is G is a path or cycle, then $S(G)$ is the empty graph. If $n_{\geq 3}(G) > 0$ then $S(G)$ is obtained from G by suppressing all degree 2 vertices. So $V(S) = L(G) \cup V_{\geq 3}(G)$, and G is a subdivision of $S(G)$. Hence loops and non-loop edges of $S(G)$ correspond to cycles and paths of G respectively. Let uv be a non-loop edge of $S(G)$ where the corresponding path P_{uv} in G has i internal vertices. We define a cost function c on the non-loop edges of $S(G)$ which assigns cost $c(uv) = \min\{i, 2\}$ to uv. Thus $c(uv)$ is the maximum possible number of leaves that a spanning tree of G can have among the internal vertices of P_{uv}. Now we are ready to present the FPT algorithm in Algorithm 1.

Algorithm 1. An FPT algorithm for MaxLeaf

```
INPUT: a MaxLeaf instance $(G, k)$.
1) while $G$ has a 2-terminal diamond or 2-terminal blossom subgraph do
       $G$ :=the result of applying (F1) or (F2) to $G$
       $k := k - 1$
2) if $n_{\geq 3}(G) \geq 3k$ or $|L(G)| \geq k$ or $k \leq 2$ then return(YES)
3) construct $S(G)$ and $c$
4) for all $L \subseteq V_{\geq 3}(G)$ with $|L| \leq k$ do
       if $G$ has a spanning tree $T$ with $L \subseteq L(T)$ and $|L| + |L(T) \backslash V_{\geq 3}(G)| \geq k$ then
           return(YES)
5) return(NO)
```

The decision in Step 4 can be made in polynomial time in the size of $S(G)$. The essential step is to solve a minimum weight spanning tree problem on $S(G) - L$, using edge costs c. We omit the proof of the following lemma's, noting that the algorithm is similar to the ones in [3] and [2].

Lemma 7. *Let (G, k) be a MaxLeaf instance for which $S(G)$ and c are non-empty and known. For any $L \subseteq V_{\geq 3}(G)$, deciding whether G has a spanning tree T with $L \subseteq L(T)$ and $|L| + |L(T) \backslash V_{\geq 3}(G)| \geq k$ can be done in time polynomial in the size of $S(G)$.*

Lemma 8. *Algorithm 1 returns YES if and only if its input (G, k) is a YES-instance.*

Proof sketch for Theorem 2. It only remains to prove the complexity bound. The first three steps can be done in linear time by building the proper data structures. For this it is essential that the degree of non-terminal vertices of 2-terminal diamonds and blossoms is bounded by a constant. We omit the details. Since the reductions in Step 1 do not increase the number of vertices or the value of k, we may assume that n and k are the number of vertices and parameter of the reduced instance, as it is after Step 1.

Step 4 of the algorithm is only executed when $n_{\geq 3}(G) < 3k$ and $|L(G)| < k$. Furthermore $V(S(G)) = L(G) \cup V_{\geq 3}(G)$, so every iteration of the for-loop of Step 4 takes time polynomial in k (Lemma 7). This for-loop is executed once for every subset $L \subseteq V_{\geq 3}(G)$ with $|L| \leq k$. Using $|V_{\geq 3}(G)| \leq 3k$, the number of such sets can be verified to be $O(k\binom{3k}{k})$. Using Stirling's approximation $x! \approx x^x e^{-x}\sqrt{2\pi x}$, we obtain that the loop is executed $O^*(6.75^k)$ times. $\square$

5 Conclusions

We conclude with some remarks about possible improvements and further applications. Theorem 1 can be strengthened at the cost of lengthier proofs. A full version of this paper shows that $n_{\geq 3}(G)/3 + c$ leaves can be obtained where $c = 4/3$ when $G = Q_3$, the 3-dimensional cube, $c = 5/3$ when $G = G_7$ or $G \neq Q_3$ is cubic, and $c = 2$ otherwise. In addition it can be shown that any graph G has a spanning tree with at least $(n_{\geq 3}(G) - x - y)/3 + c$ leaves, where x is the number of 2-necklaces in G and y is the number of 2-blossoms in G. This is a strong statement since firstly it holds for all graphs (barring the trivial conditions that G should be simple, connected and non-trivial), and secondly it not only generalizes the bound from [7], but also the $n/4 + 2$ bound for graphs with minimum degree three, and the $2n/7 + 12/7$ bound from [2] (see Section 1), when substituting the appropriate upper bounds for x and y. The usefulness of bounds of this form was recently demonstrated in [4], where a similar bound was used to obtain an improved approximation algorithm.

Theorem 1 can be used to show that the 'flower tree' example from Section 2 is extremal: when only 2-necklaces are forbidden, it is always possible to obtain at least $4n_{\geq 3}(G)/13 + 24/13$ leaves in non-cubic graphs.

References

1. Bodlaender, H.L.: On linear time minor tests with depth-first search. J. Algorithms 14(1), 1–23 (1993)
2. Bonsma, P.S.: Sparse cuts, matching-cuts and leafy trees in graphs. PhD thesis, University of Twente, Enschede, the Netherlands (2006), `http://purl.org/utwente/57117`
3. Bonsma, P.S., Brueggemann, T., Woeginger, G.J.: A faster FPT algorithm for finding spanning trees with many leaves. In: Rovan, B., Vojtáš, P. (eds.) MFCS 2003. LNCS, vol. 2747, pp. 259–268. Springer, Heidelberg (2003)

4. Correa, J.R., Fernandes, C.G., Matamala, M., Wakabayashi, Y.: A 5/3-approximation for finding spanning trees with many leaves in cubic graphs. In: WAOA 2007 (to appear, 2007)
5. Estivill-Castro, V., Fellows, M.R., Langston, M.A., Rosamond, F.A.: FPT is P-time extremal structure I. In: ACiD 2005. Texts in algorithmics, vol. 4, pp. 1–41. King's College Publications
6. Flum, J., Grohe, M.: Parameterized complexity theory. Springer, Berlin (2006)
7. Griggs, J.R., Kleitman, D.J., Shastri, A.: Spanning trees with many leaves in cubic graphs. J. Graph Theory 13(6), 669–695 (1989)
8. Kleitman, D.J., West, D.B.: Spanning trees with many leaves. SIAM J. Discrete Math. 4(1), 99–106 (1991)
9. Solis-Oba, R.: 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In: Bilardi, G., Pietracaprina, A., Italiano, G.F., Pucci, G. (eds.) ESA 1998. LNCS, vol. 1461, pp. 441–452. Springer, Heidelberg (1998)

On 2-Subcolourings of Chordal Graphs

Juraj Stacho

School of Computing Science, Simon Fraser University
8888 University Drive, Burnaby, B.C., Canada V5A 1S6
jstacho@cs.sfu.ca

Abstract. A 2-subcolouring of a graph is a partition of the vertices into two subsets, each inducing a P_3-free graph, i.e., a disjoint union of cliques. We give the first polynomial time algorithm to test whether a chordal graph has a 2-subcolouring. This solves (for two colours) an open problem of Broersma, Fomin, Nešetřil, and Woeginger, who gave an $O(n^5)$ time algorithm for interval graphs. Our algorithm for the larger class of chordal graphs has complexity only $O(n^3)$.

1 Introduction

A *k-subcolouring* of a graph G is a partition of the vertices of G into k subsets $V(G) = V_1 \cup V_2 \cup \ldots \cup V_k$, such that each V_i induces a disjoint union of *cliques* (complete graphs) in G, i.e., each V_i induces a P_3-free graph. A graph G is called *k-subcolourable* if there exists a k-subcolouring of G. The smallest integer k such that G is k-subcolourable is called the *subchromatic number* of G, and is denoted by $\chi_s(G)$.

The k-subcolourings and the subchromatic number were first introduced by Albertson, Jamison, Hedetniemi and Locke in [1]. Initially, the main focus was on bounds for $\chi_s(G)$. More recently, the complexity of recognizing k-subcolourable graphs has become a focus of attention. It follows from the result in [2] that for $k \geq 2$ this problem is NP-complete for general graphs. In [3] (and also in [4]) the authors show that it remains NP-complete for $k \geq 2$ even if the graph is triangle-free and of maximum degree four. On the other hand, there are several natural classes of graphs for which the problem has a polynomial time solution for any fixed k, e.g., graphs of bounded treewidth [3]. In another paper [5], the authors show that the problem is NP-complete for $k \geq 2$ when restricted to the class of comparability graphs, whereas for interval graphs there is an $O(n^{2k+1})$ time algorithm. In fact, it is easy to check that their algorithm also works for the case of *list k-subcolouring*, where each vertex of the input graph G has a list of admissible colours and the task is to determine whether or not there exists a k-subcolouring of G that obeys these lists. In this paper, we also deal with list k-subcolourings.

In [5], the authors formulated the following open problem. Determine the complexity of the k-subcolouring problem for the class of chordal graphs. This seems interesting, since the class of chordal graphs is strictly between the class of perfect graphs (for which the problem is NP-complete) and the class of interval graphs (for which the problem is polynomial time solvable), and colouring problems for chordal graphs often lead to interesting insights [6,7,8]. In this paper, we develop a novel technique that allows us to extract the essential properties of a 2-subcolouring, to solve this problem for $k = 2$.

In the following, we give a polynomial time algorithm for testing list 2-subcolourability of chordal graphs. In fact, our algorithm is $O(n^3)$; that also improves the complexity of the algorithm from [5] for the smaller class of interval graphs.

Instead of considering a k-subcolouring of G as a partition $V(G) = V_1 \cup V_2 \cup \ldots \cup V_k$, one can view it as a mapping $c : V(G) \rightarrow \{1, 2, \ldots, k\}$, where for every $i \in \{1, 2, \ldots, k\}$, the vertices of V_i are mapped to i. Therefore we shall employ the terminology of colourings and refer to c as a colouring of G, and refer to the elements of $\{1, 2, \ldots, k\}$ as colours. (Note that c is not necessarily a proper colouring.) For a 2-subcolouring $V(G) = V_\mathsf{r} \cup V_\mathsf{b}$, the associated colouring c is a mapping $c : V(G) \rightarrow \{\mathsf{r}, \mathsf{b}\}$, and we refer to the elements of V_r and V_b as red and blue vertices respectively.

A graph is *chordal* if it does not contain an induced cycle of length more than three [8,9]. A *clique-tree* T of a chordal graph G is a tree with the following properties [8,9].

(i) Each vertex u in T corresponds to a maximal clique C_u of G
(ii) For every edge $ab \in E(G)$, there exists a vertex $u \in V(T)$ such that $a, b \in C_u$
(iii) For every vertex $a \in V(G)$, the set of vertices u of $V(T)$ such that $a \in C_u$ induces a connected subgraph of T.

As usual, we denote by n and m the number of vertices and edges of an input graph G respectively. It is known [9] that recognizing a chordal graph, and constructing a clique-tree of a (connected) chordal graph, can both be performed in time $O(n + m)$.

The paper is structured as follows. Before describing our algorithm we investigate, in sections 2 and 3, the general properties of subcolourings of chordal graphs. In particular, in section 2, we introduce the key structure, called the *subcolouring digraph*, that encodes important properties of a given subcolouring of a chordal graph G (based on a fixed clique-tree of G). In section 3, we describe the necessary conditions for a 2-subcolouring c implied by the structure of its subcolouring digraph. Finally, in section 4, we describe our algorithm, which uses dynamic programming on the subcolouring digraph, and we discuss the complexity and efficient implementation of our algorithm.

2 The Subcolouring Digraph

Observe first that G is k-subcolourable if and only if each component of G is k-subcolourable. Throughout the paper, unless otherwise indicated, we shall always deal with a connected chordal graph G, a fixed clique-tree T of G, and with c, a colouring of the vertices of G (not necessarily a subcolouring or a proper colouring).

Therefore, let G be a connected chordal graph, and let T be a fixed clique-tree of G. Let C_u for $u \in V(T)$ denote the maximal clique of G associated with u, and let $\mathcal{C}(X)$ denote the union of cliques associated with the vertices of a set $X \subseteq V(T)$, i.e., $\mathcal{C}(X) = \bigcup_{u \in X} C_u$. In this section, we shall not consider T to be rooted. We shall use parentheses $(,)$ to denote the edges of T, to distinguish them from the edges of G. The removal of an edge (u, v) splits T into two subtrees; we shall denote by $T_{u,v}$ the subtree containing the vertex v, and by $T_{v,u}$ the subtree containing the vertex u. We denote $G_{u,v} = \mathcal{C}(T_{u,v})$ and $G_{v,u} = \mathcal{C}(T_{v,u})$.

Observe that in any k-subcolouring c of G, a vertex a in a clique C of G can have neighbours of the same colour as a in at most one connected component of $G \setminus C$. Based on this, we construct a multidigraph $\mathcal{D}_c(G)$ with coloured edges to capture the properties of the colouring c. We shall refer to $\mathcal{D}_c(G)$ as a *subcolouring digraph* for c. To avoid ambiguity, the edges of $\mathcal{D}_c(G)$ shall be referred to as *arcs* and denoted using angle brackets $\langle , \rangle$ to distinguish them from the edges of T and the edges of G. In particular, $\langle u, v \rangle_i$ shall denote an arc from u to v coloured i, and we shall write $\langle u, v \rangle$ for an arc from u to v (of some colour). The digraph $\mathcal{D}_c(G)$ is constructed as follows (cf. Figure 1). The vertices of $\mathcal{D}_c(G)$ are the vertices of T, and for vertices u, v that are adjacent in T, there is an arc $\langle u, v \rangle_i$ in $\mathcal{D}_c(G)$, if there exist vertices $a \in C_u$ and $b \in G_{u,v} \setminus C_u$ such that $ab \in E(G)$ and both a and b have the same colour i in c. Note that we have arcs in $\mathcal{D}_c(G)$ only between vertices that are adjacent in T.

Formally, we define $\mathcal{D}_c(G)$ as follows.

(i) $V(\mathcal{D}_c(G)) = V(T)$

(ii) $E(\mathcal{D}_c(G)) = \left\{ \langle u, v \rangle_i \;\middle|\; \begin{array}{ll} \exists\, a \in C_u & (u, v) \in E(T) \\ \exists\, b \in G_{u,v} \setminus C_u & ab \in E(G) \\ & c(a) = c(b) = i \end{array} \right\}$

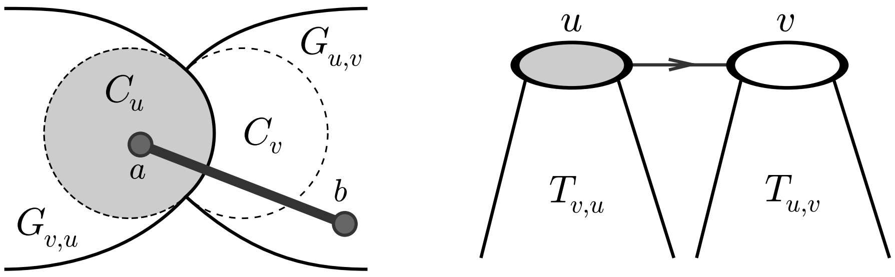


Fig. 1. Illustrating the case when there is an arc $\langle u, v \rangle$ in $\mathcal{D}_c(G)$

We have the following observations about $\mathcal{D}_c(G)$.

Proposition 1. *Let u, v, w be vertices of $\mathcal{D}_c(G)$ and let i be a colour from $\{1, 2, \ldots, k\}$. If c is a k-subcolouring then $\mathcal{D}_c(G)$*

(i) cannot contain both the arc $\langle u,v\rangle_i$ and the arc $\langle v,u\rangle_i$,
(ii) cannot contain both the arc $\langle u,v\rangle_i$ and the arc $\langle u,w\rangle_i$,
(iii) cannot contain all of the arcs $\langle u,v\rangle_1, \langle u,v\rangle_2, \ldots, \langle u,v\rangle_k$.

Proof. Suppose that (i) is false. Let a, b and a', b' be the vertices of G that caused the arcs $\langle u,v\rangle_i$ and $\langle v,u\rangle_i$ respectively to appear in $\mathcal{D}_c(G)$. It is not difficult to see that $a, a' \in C_u \cap C_v$ and $bb' \notin E(G)$. Hence the graph induced on a, b, a', b' must contain an induced P_3 coloured i, a contradiction. One can easily repeat this same argument for pairs a, b and a', b' that falsify (ii).

Finally, let $a_1, b_1,\ a_2, b_2,\ \ldots,\ a_k, b_k$ be the pairs of vertices that falsify (iii). Since C_u and C_v are two different maximal cliques of G, there exists a vertex $d \in C_u \setminus C_v$. Again, it is not difficult to see that $a_i \in C_u \cap C_v$ for all i, and d is not a neighbour of any b_i. Now clearly, any colour j assigned to d creates an induced P_3 coloured j on vertices d, a_j, b_j, yielding a contradiction. □

3 2-Subcolourings

From now on we focus on the case $k = 2$, i.e., 2-subcolourings. In what follows, we shall assume that G is a connected 2-subcolourable chordal graph, T a fixed clique-tree of G, and c is a 2-subcolouring of G. As remarked earlier, for a 2-subcolouring c of G, we shall refer to the vertices of G as red and blue vertices and use letters r and b respectively to denote the two colours.

We shall call an edge (u, v) in T a *strong* edge of T if $\mathcal{D}_c(G)$ contains both $\langle u,v\rangle_{\mathsf{r}}$ and $\langle v,u\rangle_{\mathsf{b}}$, or both $\langle u,v\rangle_{\mathsf{b}}$ and $\langle v,u\rangle_{\mathsf{r}}$. We shall call an edge (u, v) in T a *weak* edge of T if there is at most one arc between u and v in $\mathcal{D}_c(G)$. It follows from Proposition 1 that every edge in T must be either strong or weak.

Let u, v be adjacent vertices in T. Let $I_{u,v} = C_u \cap C_v$, and let $N_{u,v}$ be the set of all vertices of $G_{u,v} \setminus C_u$ which are neighbours of $C_u \cap C_v$. Furthermore, let $L_{u,v} = G_{v,u} \setminus C_v$ and $R_{u,v} = G_{u,v} \setminus (I_{u,v} \cup N_{u,v})$. (Note that $C_v \subseteq I_{u,v} \cup N_{u,v}$.) We have the following observation.

Proposition 2. *Let u, v be adjacent vertices in T.*

(i) If $\langle u,v\rangle_{\mathsf{r}} \in E(\mathcal{D}_c(G))$, or $\langle u,v\rangle_{\mathsf{b}} \in E(\mathcal{D}_c(G))$, then the vertices of $C_u \setminus C_v$ are all blue, or all red, respectively.
(ii) If $\langle u,v\rangle \notin E(\mathcal{D}_c(G))$ then the vertices of $I_{u,v}$ and $N_{u,v}$ are all red and all blue respectively, or all blue and all red respectively.
(iii) G has no induced P_3 having both a vertex of $L_{u,v}$ and a vertex of $R_{u,v}$.

Proof. For (i) suppose that $\langle u,v\rangle_{\mathsf{r}} \in E(\mathcal{D}_c(G))$ (the other case is clearly symmetric), and let a, b be the red vertices that caused this arc. It is easy to see, that $a \in C_u \cap C_v$ and b is not adjacent to any vertex in $C_u \setminus C_v$. Hence, no vertex d of $C_u \setminus C_v$ can be red, since otherwise d, a, b is an induced red P_3.

For (ii) let $a \in I_{u,v}$ and $b \in N_{u,v}$ be adjacent. Then a and b must have different colours, since otherwise we would have an arc $\langle u,v\rangle \in E(\mathcal{D}_c(G))$. Since C_u and C_v are different maximal cliques, there exists $d \in C_v \setminus C_u$. Now the claim follows, because d is adjacent to all vertices of $I_{u,v}$, and hence any $a' \in I_{u,v}$ must have different colour from d.

Finally, for (iii) let b, a, d be an induced P_3 in G with edges ba and ad, that contains both a vertex of $L_{u,v}$ and a vertex of $R_{u,v}$. Now since $L_{u,v}$ and $R_{u,v}$ are completely non-adjacent, it follows that $a \notin L_{u,v} \cup R_{u,v}$. Hence we can assume that $b \in L_{u,v}$ and $d \in R_{u,v}$. Now if $a \in I_{u,v}$ then we must have $d \in N_{u,v}$ but $N_{u,v} \cap R_{u,v} = \emptyset$. Hence $a \in N_{u,v}$ and $b \in I_{u,v}$ but $L_{u,v} \cap I_{u,v} = \emptyset$. Therefore no such vertices b, a, d exist in G. □

Note that it follows from the above observation that for an edge (u, v) in T, the 2-subcolourings induced by the fixed c on $L_{u,v}$ and $R_{u,v}$, are independent of each other in the sense that they only depend on the colours assigned to $I_{u,v} \cup N_{u,v}$. Furthermore, if (u, v) is a weak edge such that $\langle u, v\rangle \notin E(\mathcal{D}_c(G))$, then the 2-subcolouring of $I_{u,v} \cup N_{u,v}$ is unique (up to exchanging the colours red and blue). That allows one to consider independently the subgraphs of T that no longer contain any weak edges.

For strong edges in T we have the following observations.

Observation 3. *Every vertex of T has at most two incident strong edges, i.e., the connected components formed by the strong edges of T are paths.*

Proof. It is not difficult to see that if a vertex u in T has three adjacent strong edges, then for at least two of them, say (u, v) and (u, w), we have arcs $\langle u, v\rangle$ and $\langle u, w\rangle$ of the same colour in $\mathcal{D}_c(G)$. By Proposition 1(ii) this is not possible. □

Proposition 4. *Let u be a vertex in T with distinct neighbours v, w, z.*

(i) If (v, u) is a strong edge and $\langle w, u\rangle \notin E(\mathcal{D}_c(G))$, then (z, u) is not a strong edge.
(ii) If (v, u) is a strong edge, $\langle w, u\rangle \notin E(\mathcal{D}_c(G))$, and $\langle z, u\rangle \notin E(\mathcal{D}_c(G))$, then $I_{u,w} = I_{u,z}$.
(iii) If $\langle v, u\rangle \notin E(\mathcal{D}_c(G))$, $\langle w, u\rangle \notin E(\mathcal{D}_c(G))$, and $\langle z, u\rangle \notin E(\mathcal{D}_c(G))$, then $I_{u,v} = I_{u,w}$ or $I_{u,w} = I_{u,z}$ or $I_{u,z} = I_{u,v}$.

Proof. For (i) suppose that the edges (v, u) and (z, u) are strong and that $\langle w, u\rangle \notin E(\mathcal{D}_c(G))$. Since G is connected, there exists $a \in I_{u,w}$. Without loss of generality we may assume that $\langle u, z\rangle_{\mathsf{r}} \in E(\mathcal{D}_c(G))$. Hence by Proposition 2$(i)$ we obtain that $C_z \setminus C_v$ is all red, $C_v \setminus C_z$ is all blue and $C_u \subseteq C_v \cup C_z$. Also since C_u, C_z and C_v are different maximal cliques we have $d \in C_z \setminus C_u$ and $b \in C_v \setminus C_u$. Now clearly, $a \in C_v \cup C_z$ hence if a is red then $a \in C_z$ and hence $\langle w, u\rangle_{\mathsf{r}} \in E(\mathcal{D}_c(G))$, and if a is blue then $a \in C_v$ and hence $\langle w, u\rangle_{\mathsf{b}} \in E(\mathcal{D}_c(G))$, a contradiction.

For (ii) suppose that (v, u) is strong, $\langle w, u\rangle \notin E(\mathcal{D}_c(G))$, and $\langle z, u\rangle \notin E(\mathcal{D}_c(G))$ but $I_{u,w} \neq I_{u,z}$. Without loss of generality we may assume that $I_{u,w} \not\subseteq I_{u,z}$ and that $\langle u, v\rangle_{\mathsf{r}}, \langle v, u\rangle_{\mathsf{b}} \in E(\mathcal{D}_c(G))$. Hence there must exist a vertex $a \in I_{u,w} \setminus I_{u,z}$, a vertex $b \in I_{u,z}$ (since G is connected), and a vertex $d \in C_v \setminus C_u$ (since the cliques are maximal). Clearly, $c(a) \neq c(b)$ otherwise we have $\langle z, u\rangle \in E(\mathcal{D}_c(G))$.

Now since $\langle v, u\rangle_{\mathsf{b}} \in E(\mathcal{D}_c(G))$ it follows that the vertex d is red. Similarly, since $\langle u, v\rangle_{\mathsf{r}} \in E(\mathcal{D}_c(G))$ we have that $C_u \setminus C_v$ is all blue, hence if a is red, then $a \in I_{u,v}$ and hence $\langle w, u\rangle_{\mathsf{r}} \in E(\mathcal{D}_c(G))$, and if b is red then $b \in I_{u,v}$ and hence $\langle z, u\rangle_{\mathsf{r}} \in E(\mathcal{D}_c(G))$, a contradiction.

Finally, for (iii) suppose that none of $\langle v, u\rangle$,$\langle w, u\rangle$, and $\langle z, u\rangle$ is in $E(\mathcal{D}_c(G))$, but the three sets $I_{u,v}$, $I_{u,w}$ and $I_{u,z}$ are pairwise different. Without loss of generality we may assume that $I_{u,v} \not\subseteq I_{u,w} \not\subseteq I_{u,z}$ and either $I_{u,v} \not\subseteq I_{u,z}$ or $I_{u,v} \not\supseteq I_{u,z}$. If $I_{u,v} \not\subseteq I_{u,z}$, suppose first that $J \neq \emptyset$ where $J = I_{u,v} \setminus (I_{u,w} \cup I_{u,z})$. It follows that we must have a vertex $a \in J$, a vertex $b \in I_{u,w} \setminus I_{u,z}$ and a vertex $c \in I_{u,z}$. Now at least two of the vertices a, b, c must have the same colour and that gives us one of the edges $\langle v, u\rangle$, $\langle w, u\rangle$, $\langle z, u\rangle$ in $E(\mathcal{D}_c(G))$, a contradiction. If $J = \emptyset$, we similarly obtain a contradiction for vertices $a \in (I_{u,w} \cap I_{u,v}) \setminus I_{u,z}$, $b \in (I_{u,z} \cap I_{u,v}) \setminus I_{u,w}$ and $c \in C_u \setminus C_v$. Now if $I_{u,v} \not\supseteq I_{u,z}$, it follows that we have a vertex $a \in I_{u,v} \setminus I_{u,w}$, a vertex $b \in I_{u,w} \setminus I_{u,z}$ and $c \in I_{u,z} \setminus I_{u,v}$, and again a contradiction follows. □

Let $P_{u,v}$ denote the (unique) path from u to v in T. We shall call the path $P_{u,v}$ *strong* if it is formed only by strong edges of T. Note that we also allow paths of zero length (i.e., paths $P_{u,v}$ with $u = v$); all such paths are trivially strong. A strong path is *maximal* if it is not properly contained in another strong path. A vertex z in T adjacent to a vertex u is a *special neighbour* of u if $\langle z, u\rangle \notin E(\mathcal{D}_c(G))$. The following claim is a direct consequence of Proposition 4.

Lemma 5. *For any strong path $P_{u,v}$ in T (possibly with $u = v$) there exist sets $A_{u,v}$ and $A'_{u,v}$ (both possibly empty) such that for any special neighbour s of some $t \in P_{u,v}$ we have $I_{s,t} = A_{u,v}$ or $I_{s,t} = A'_{u,v}$.*

Proof. If $u \neq v$ then by Proposition 4(i) only u and v can have special neighbours. Hence, if u has a special neighbour z, we let $A_{u,v} = I_{z,u}$ and $A_{u,v} = \emptyset$ otherwise. If v has a special neighbour w, we let $A'_{u,v} = I_{w,v}$ and $A'_{u,v} = \emptyset$ otherwise. Now the claim follows from Proposition 4(ii).

If $u = v$ and u has two special neighbours z and w with $I_{z,u} \neq I_{w,u}$, we define $A_{u,v} = I_{z,u}$, $A'_{u,v} = I_{w,u}$. Otherwise, we let $A_{u,v} = I_{z,u}$ and $A'_{u,v} = \emptyset$ if u has a special neighbour z but does not satisfy the previous condition. Finally, we let $A_{u,v} = A'_{u,v} = \emptyset$ if u has no special neighbours. Now the claim follows from Proposition 4(iii). □

Let $B_{u,v}$ and $B'_{u,v}$ denote the sets of neighbours of $A_{u,v}$ and $A'_{u,v}$ in $\mathcal{C}(P_{u,v})$ respectively. We now give a complete characterisation of the structure of the colouring c on the vertices of $\mathcal{C}(P_{u,v})$.

Theorem 6. *For any strong path $P_{u,v}$ in T (possibly with $u = v$) we have*

(i) $\mathcal{C}(P_{u,v}) = C_u \cup C_v$,

(ii) the vertices of $C_u \setminus C_v$ and $C_v \setminus C_u$ are all red and all blue respectively, or all blue and all red respectively,

(iii) for every weak edge (s, t) incident to $P_{u,v}$ the vertices of $I_{s,t}$ are all red or all blue,

(iv) the vertices of $A_{u,v} \cup B'_{u,v}$ and $A'_{u,v} \cup B_{u,v}$ are all red and all blue respectively, or all blue and all red respectively, and

(v) if in addition $P_{u,v}$ is maximal, then any colouring c' of $\mathcal{C}(P_{u,v})$ satisfying $(ii)-(iv)$ is a 2-subcolouring of $\mathcal{C}(P_{u,v})$ and can be extended to a 2-subcolouring of G.

Proof. We prove (i) and (ii) by induction on the length of the path $P_{u,v}$. If $u=v$ then there is nothing to prove. Hence, let w be the neighbour of v on $P_{u,v}$ and assume that $\mathcal{C}(P_{u,w}) = C_u \cup C_w$ and that the vertices of $C_u \setminus C_w$ and $C_w \setminus C_u$ are all red and all blue respectively. Since (w,v) is a strong edge, by Proposition 2(i) we have that $C_v \setminus C_w$ is all blue and $C_w \setminus C_v$ is all red. From this we deduce $C_w \setminus (C_u \cup C_v) = \emptyset$ which implies $C_w \subseteq C_u \cup C_v$ and the claim follows.

Now claims (iii) and (iv) follow directly from Proposition 2(ii) and Lemma 5 since for any special neighbour s of $t \in P_{u,v}$ we have $I_{s,t} = A_{u,v}$ and $N_{s,t} \cap \mathcal{C}(P_{u,v}) = B_{u,v}$ or $I_{s,t} = A'_{u,v}$ and $N_{s,t} \cap \mathcal{C}(P_{u,v}) = B'_{u,v}$.

Finally, let c' be any colouring of $\mathcal{C}(P_{u,v})$ satisfying $(ii)-(iv)$. Let c'' be a colouring of G constructed from the colouring c as follows. First we exchange the colours red and blue on the vertices of $G_{t,s}$ for each neighbour $s \notin P_{u,v}$ of $t \in P_{u,v}$ so that the colours of $I_{s,t}$ match the colouring c'. (Note that since $P_{u,v}$ is maximal, the edge (s,t) is weak.) Then we replace the colours of $\mathcal{C}(P_{u,v})$ by c'. Clearly, c'' extends c'. We show that c'' is a 2-subcolouring of G. Suppose otherwise and let b,a,d be an induced P_3 in G with edges ba and ad such that $c''(b) = c''(a) = c''(d)$. If $b,a,d \in \mathcal{C}(P_{u,v})$ then by (i) and (ii) it follows that the vertices b,a,d are all in C_u or all in C_v, but that is not possible since $bd \notin E(G)$. On the other hand, if $a \notin \mathcal{C}(P_{u,v})$ then it follows from the construction of c'' that $c(b) = c(a) = c(d)$ which is not possible since c is a 2-subcolouring. Hence for some neighbour $s \notin P_{u,v}$ of $t \in P_{u,v}$ we have that $a \in I_{t,s}$ and $b \in N_{t,s}$ and $d \in N_{s,t} \cap \mathcal{C}(P_{u,v})$. Now if $\langle t,s \rangle \notin E(\mathcal{D}_c(G))$ then by Proposition 2(ii) we have that $c(a) \neq c(b)$ hence $c''(a) \neq c''(b)$ because $a,b \in G_{t,s}$. On the other hand, if $\langle t,s \rangle \in E(\mathcal{D}_c(G))$, then s must be a special neighbour of t but then by (iv) we have that $a \in A_{u,v}$ and $d \in B_{u,v}$ or $a \in A'_{u,v}$ and $d \in B'_{u,v}$, and hence $c''(a) \neq c''(d)$, a contradiction. □

4 The Algorithm

Now we are ready to describe the algorithm for deciding (list) 2-subcolourability for chordal graphs. We assume that we are given a chordal graph G and a fixed clique-tree T of G, and we want to decide whether or not G is 2-subcolourable. Later, we describe how to obtain a list version of the algorithm.

This time, we consider T rooted at an arbitrary fixed vertex r. Therefore, we write $p[v]$ to denote the parent of a vertex v in T. For a vertex v in T, we denote by T_v the subtree of T rooted at v.

We shall say that T_v is $(-)$ colourable if there exists a 2-subcolouring c_v of $\mathcal{C}(T_v)$ such that the vertices of $I_{p[v],v}$ are all red or all blue. Similarly, T_v is $(+)$ colourable if there exists a 2-subcolouring c_v of $\mathcal{C}(T_v)$ such that the vertices of $I_{p[v],v}$ and $N_{p[v],v}$ are all red and all blue respectively, or all blue and all red respectively. In the special case of the root r, when $p[v]$ does not exist, we shall say that T_r is $(-)$ colourable if there exists a 2-subcolouring c_r of $\mathcal{C}(T_r) = G$.

Note that by Lemma 5, for every strong path, we only need to consider up to two special neighbours z and w. If the path has only one such neighbour (or none), we use nil as the value of z or w. Therefore, we always view a path $P_{u,v}$ having two special neighbours z and w of u and v respectively, but allow one (or both) of z and w to be nil. We shall say that the path $P_{u,v}$ is (z, w)-colourable if there exists a 2-subcolouring $c_{u,v}$ of $\mathcal{C}(P_{u,v})$ such that for every incident edge (s, t) of $P_{u,v}$ in T, the vertices of $I_{s,t}$ are all red or all blue, and such that if z is not nil (w is not nil) then the vertices of $N_{z,u}$ ($N_{w,v}$ respectively) in $\mathcal{C}(P_{u,v})$ are all red or all blue.

The algorithm works as follows. It processes the vertices of T in a bottom-up order and identifies which edges of T could be weak, for some 2-subcolouring of G. This is done by testing and recording for every vertex v in T, whether or not the subtree T_v is (+) colourable, and whether or not the subtree T_v is (−) colourable. (Note that T_v must be either (+) colourable or (−) colourable if $(v, p[v])$ is a weak edge in T for some 2-subcolouring of G.)

For a vertex x of T, the test for colourability of T_x is done as follows. First, if we are testing (−) colourability, we precolour the vertices of $I_{p[x],x}$ red or blue, otherwise we precolour the vertices of $I_{p[x],x}$ and $N_{p[x],x}$ red and blue respectively, or blue and red respectively. Then we choose a strong path $P_{u,v}$ in T_x that passes through x and we choose special neighbours z and w of u and v respectively. (See the above remark about special neighbours.) Then we test for colourability of $P_{u,v}$ with respect to the chosen special neighbours by applying Theorem 6. Finally, we recursively test, for every special neighbour y of $P_{u,v}$, whether or not the corresponding tree T_y is (−) colourable or (+) colourable, and for all other neighbours of $P_{u,v}$, whether or not their corresponding trees are (+) colourable. We declare T_x (−) colourable (or (+) colourable, depending on the particular case) if and only if the above tests succeed for some choice of u, v and some choice of special neighbours of u and v. Note that since we process the vertices in a bottom-up order, each recursive call amounts to a constant time table look-up.

If the algorithm succeeds to declare T_r (−) colourable then the graph G is 2-subcolourable, otherwise G is not 2-subcolourable. The correctness can be shown to follow from Proposition 2 and Theorem 6. A more precise description of an efficient implementation of the algorithm can be found on pages 552-553. Below, we discuss some details of this implementation.

Note that in the procedure for testing colourability of a strong path $P_{u,v}$ (see Algorithm 2 on page 553), instead of independently precolouring the sets $I_{s,t}$ either red or blue, for each incident edge of $P_{u,v}$ (as follows from Theorem 6), we construct a collection of sets $\mathcal{L}$ containing the unions of the sets $I_{s,t}$ that intersect. Note that if sets $I_{s,t}$ and $I_{s',t'}$ intersect, their union must also be all red or all blue. After constructing $\mathcal{L}$ we can independently decide the colours of the sets in $\mathcal{L}$, since they no longer intersect. To find $\mathcal{L}$ we use an efficient variant of the Union-Find algorithm which has time and space complexity $O(n)$.

It is not difficult to see that this algorithm can be easily extended to solve the list 2-subcolouring problem. Recall that this is the problem where each vertex has a list of admissible colours and the task is to decide whether G has a

2-subcolouring that obeys these lists. Whenever in the algorithm a vertex is being precoloured by some colour, this colour is checked against the list of that vertex and if it is not in the list, we exit the current procedure with a negative answer. (Note that this only happens in the procedure for testing strong paths, see Algorithm 2 on page 553.)

The following theorem summarizes the complexity of the above algorithm and is followed by its formal description.

Theorem 7. *There exists an $O(n^3)$ time algorithm deciding, for a given chordal graph G, whether G admits a (list) 2-subcolouring; the algorithm also constructs a 2-subcolouring of G if one exists.*

Proof. As noted before, one can determine the maximal cliques of G and construct a clique-tree T of G in time $O(n + m)$. It follows from the remark above that for any pair of vertices u, v and choice of special neighbours z, w of u, v respectively, one can determine (z, w)-colourability of the path $P_{u,v}$ in time $O(n^2)$. In the first part of the algorithm, this test is performed for every pair of vertices (including the choice of their special neighbours). This step can be implemented more efficiently by reusing the results for the subpaths, i.e., starting from some vertex v and computing all paths from v, altogether in time $O(n^2)$. Therefore the total running time for the first part of the algorithm is $O(n^3)$. In the second part, note that during the course of the algorithm (in the procedures for testing the colourability of a subtree T_x, see Algorithms 3,4 on page 553), we consider every path (including the choice of special neighbours) in T only once. Each path is processed in time $O(n)$, so in total we have $O(n^3)$ time. Finally, a 2-subcolouring can be easily found by keeping track of which paths were used to colour the subtrees of T and backtracking from the root r. □

```
     Input: A chordal graph $G$ and a clique tree $T$ of $G$ rooted at $r$
     Output: Decide whether $G$ is 2-subcolourable

 1   for every two vertices $u, v$ in $T$ do
 2       for every neighbour $z$ and $w$ (including nil) of $u$ and $v$ respectively do
 3           test and record whether $P_{u,v}$ is $(z, w)$-colourable

 4   initialize $S \leftarrow \emptyset$    ($S$ is the set of processed vertices)
 5   while $S \neq V(T)$ do
 6       pick a vertex $v \notin S$ whose all children are in $S$
 7       test and record whether $T_v$ is $(-)$ colourable
 8       test and record whether $T_v$ is $(+)$ colourable (if $v \neq r$)
 9       $S \leftarrow S \cup \{v\}$

10   if $T_r$ is $(-)$ colourable then
         return "G is 2-subcolourable"
11   else return "G is not 2-subcolourable"
```

Algorithm 1. The test for 2-subcolourability of G

```
    **Input**: Vertices $u, v$ of $T$, vertices $z, w$ neighbours of $u, v$ respectively or nil
    **Output**: Decide whether $P_{u,v}$ is $(z, w)$-colourable

 1  compute $\mathcal{C}(P_{u,v})$
 2  if $\mathcal{C}(P_{u,v}) \neq C_u \cup C_v$ then return "$P_{u,v}$ is not $(z, w)$-colourable"
 3  precolour the vertices of $C_u \setminus C_v$ and $C_v \setminus C_u$ by red and blue respectively
                                                   (or blue and red respectively)
 4  if $z \neq$ nil then precolour the vertices of $N_{z,u}$ red (or blue)
 5  if $w \neq$ nil then precolour the vertices of $N_{w,v}$ blue (or red)
 6  initialize the set $\mathcal{L} \leftarrow \emptyset$
 7  for each edge $(s, t)$ incident to $P_{u,v}$ do
 8      compute the set $\mathcal{L}_s$ consisting of those sets from $\mathcal{L}$ which intersect $I_{s,t}$
 9      $\mathcal{L} \leftarrow \mathcal{L} \setminus \mathcal{L}_s \cup \left\{ I_{s,t} \cup \left( \bigcup_{L \in \mathcal{L}_s} L \right) \right\}$
10  if some $L \in \mathcal{L}$ contains both a precoloured red and a precoloured blue vertex
        then return "$P_{u,v}$ is not $(z, w)$-colourable"
11  else return "$P_{u,v}$ is $(z, w)$-colourable"
```

Algorithm 2. The test whether $P_{u,v}$ is (z, w)-colourable

```
    **Input**: A vertex $x$ in $T$
    **Output**: Decide whether $T_x$ is $(-)$ colourable

 1  for each $u, v \in T_x$ such that $x \in P_{u,v}$ do
 2      for every child $z$ and $w$ (including nil) of $u$ and $v$ respectively do
 3          if $P_{u,v}$ is $(z, w)$-colourable
              and for each child $s \neq w, z$ of $P_{u,v}$ the tree $T_s$ is $(+)$ colourable
                and either $z =$ nil (resp. $w =$ nil) or $T_z$ (resp. $T_w$) is $(+)$ or $(-)$
                  colourable then return "$T_x$ is $(-)$ colourable"
 4  return "$T_x$ is not $(-)$ colourable"
```

Algorithm 3. The test whether T_x is $(-)$ colourable

```
    **Input**: A vertex $x$ in $T$
    **Output**: Decide whether $T_x$ is $(+)$ colourable

 1  for each $u \in T_x$ do
 2      for every child $z$ (including nil) of $u$ do
 3          if $P_{u,x}$ is $(z, p[x])$-colourable
              and for each child $s \neq z$ of $P_{u,x}$ the tree $T_s$ is $(+)$ colourable
                and either $z =$ nil or $T_z$ is $(+)$ or $(-)$ colourable
                  then return "$T_x$ is $(+)$ colourable"
 4  return "$T_x$ is not $(+)$ colourable"
```

Algorithm 4. The test whether T_x is $(+)$ colourable

Acknowledgements

The author would like to thank his advisor Pavol Hell for directing this research and for his help with the preparation of this paper.

Note added in proof

The algorithm presented in this paper answers an open question from [5] for the case $k = 2$, while it also extends a result from [5] and improves the complexity for the larger class of chordal graphs. Recently, we were able to show that for all other values of $k \geq 3$, the problem of k-subcolouring of chordal graphs is NP-complete, thus completely answering the open question of Broersma et al. [5].

References

1. Albertson, M.O., Jamison, R.E., Hedetniemi, S.T., Locke, S.C.: The subchromatic number of a graph. Discrete Mathematics 74, 33–49 (1989)
2. Achlioptas, D.: The complexity of G-free colorability. Discrete Mathematics 165/166, 21–30 (1997)
3. Fiala, J., Jansen, K., Le, V.B., Seidel, E.: Graph subcoloring: Complexity and algorithms. In: Brandstädt, A., Le, V.B. (eds.) WG 2001. LNCS, vol. 2204, pp. 154–165. Springer, Heidelberg (2001)
4. Gimbel, J., Hartman, C.: Subcolorings and the subchromatic number of a graph. Discrete Mathematics 272, 139–154 (2003)
5. Broersma, H., Fomin, F.V., Nešetřil, J., Woeginger, G.J.: More about subcolorings. Computing 69, 187–203 (2002)
6. Hell, P., Klein, S., Nogueira, L.T., Protti, F.: Partitioning chordal graphs into independent sets and cliques. Discrete Applied Mathematics 141, 185–194 (2004)
7. Feder, T., Hell, P., Klein, S., Nogueira, L.T., Protti, F.: List matrix partitions of chordal graphs. Theoretical Computer Science 349, 52–66 (2005)
8. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, New York (1980)
9. Spinrad, J.P.: Efficient Graph Representations. American Mathematical Society (2003)

Collective Additive Tree Spanners of Homogeneously Orderable Graphs
[Extended Abstract]

Feodor F. Dragan, Chenyu Yan, and Yang Xiang

Algorithmic Research Laboratory, Department of Computer Science
Kent State University, Kent, OH 44242, USA
dragan@cs.kent.edu, cyan1@kent.edu, yxiang@cs.kent.edu

Abstract. In this paper we investigate the (*collective*) *tree spanners problem* in homogeneously orderable graphs. This class of graphs was introduced by A. Brandstädt et al. to generalize the dually chordal graphs and the distance-hereditary graphs and to show that the Steiner tree problem can still be solved in polynomial time on this more general class of graphs. In this paper, we demonstrate that every n-vertex homogeneously orderable graph G admits

– a spanning tree T such that, for any two vertices x, y of G, $d_T(x, y) \leq d_G(x, y) + 3$ (i.e., an *additive tree 3-spanner*) and
– a system $\mathcal{T}(G)$ of at most $O(\log n)$ spanning trees such that, for any two vertices x, y of G, a spanning tree $T \in \mathcal{T}(G)$ exists with $d_T(x, y) \leq d_G(x, y) + 2$ (i.e, a *system of at most $O(\log n)$ collective additive tree 2-spanners*).

These results generalize known results on tree spanners of dually chordal graphs and of distance-hereditary graphs. The results above are also complemented with some lower bounds which say that on some n-vertex homogeneously orderable graphs any system of collective additive tree 1-spanners must have at least $\Omega(n)$ spanning trees and there is no system of collective additive tree 2-spanners with constant number of trees.

1 Introduction

A spanning tree T of a graph G is called a *tree spanner* of G if T provides a "good" approximation of the distances in G. More formally, for $r \geq 0$, T is called an *additive tree r-spanner* of G if for any pair of vertices u and v their distance in T is at most r plus their distance in G (see [17,19]). A similar definition can be given for multiplicative tree t-spanners (see [6]); however in this paper we are only concerned with additive spanners. Tree spanners have many applications in various areas. They occur in biology and they can be used in message routing and as models for broadcast operations in communication networks. Tree spanners are useful also from the algorithmic point of view - many algorithmic problems are easily solvable on trees. If one needs to solve an NP-hard optimization problem concerning distances in a graph G and G admits a good tree spanner

T, then an efficient solution to that problem on T would provide an approximate solution to the original problem on G.

The problem to decide for a graph G whether G has a multiplicative tree t–spanner (the *multiplicative tree t–spanner problem*) is NP–complete for any fixed $t \geq 4$ [6], and it remains NP-complete even on some rather restricted graph families. Fortunately, there is also a number of special classes of graphs where additive or multiplicative variants of the tree spanner problem are polynomial time solvable. Here we will mention only the results on two families of graphs which are relevant to our paper. Every dually chordal graph admits an additive tree 3-spanner [2] and every distance-hereditary graph admits an additive tree 2-spanner [19], and such tree spanners can be constructed in linear time.

In [12] we generalized the notion of tree spanners by defining a new notion of *collective tree spanners*. We say that a graph G *admits a system of μ collective additive tree r-spanners* if there is a system $\mathcal{T}(G)$ of at most μ spanning trees of G such that for any two vertices u, v of G a spanning tree $T \in \mathcal{T}(G)$ exists such that the distance in T between u and v is at most r plus their distance in G. We say that system $\mathcal{T}(G)$ *collectively r-spans* the graph G and r is the *(collective) additive stretch factor*. Clearly, if G admits a system of μ collective additive tree r-spanners, then G admits an additive r-spanner with at most $\mu \times (n-1)$ edges, and if $\mu = 1$ then G admits an additive tree r-spanner. Note that, an induced cycle of length k provides an example of a graph which does not have any additive tree $(k-3)$-spanner, but admits a system of two collective additive tree 0-spanners. Furthermore, for any $r \geq 1$ there is a chordal graph which does not have any additive tree r-spanner [19]; on the other hand, any n-vertex chordal graph admits a system of $O(\log n)$ collective additive tree 2-spanners [12]. These two examples demonstrate the power of this new concept of collective tree spanners. One of the motivations to introduce this new concept stems from the problem of designing compact and efficient routing schemes in graphs. In [14,20], a shortest path routing labeling scheme for trees is described that assigns each vertex of an n-vertex tree a $O(\log^2 n/\log\log n)$-bit label. Given the label of a source vertex and the label of a destination, it is possible to compute in constant time, based solely on these two labels, the neighbor of the source that heads in the direction of the destination. Clearly, if an n-vertex graph G admits a system of μ collective additive tree r-spanners, then G admits a routing labeling scheme of deviation (i.e., additive stretch) r with addresses and routing tables of size $O(\mu \log^2 n/\log\log n)$ bits per vertex. Once computed by the sender in μ time, headers of messages never change, and the routing decision is made in constant time per vertex (for details see [11,12]). Other motivations stem from the generic problems of efficient representation of the distances in "complicated" graphs by the tree distances and of algorithmic use of these representations [1,7,13]. Approximating graph distance d_G by a simpler distance (in particular, by tree–distance d_T) is useful in many areas such as communication networks, data analysis, motion planning, image processing, network design, and phylogenetic analysis.

Previously, collective tree spanners of particular classes of graphs were considered in [8,10,11,12,16]. Paper [12] showed that any chordal graph or chordal bipartite graph admits a system of at most $\log_2 n$ collective additive tree 2–spanners. These results were complemented by lower bounds, which say that any system of collective additive tree 1–spanners must have $\Omega(\sqrt{n})$ spanning trees for some chordal graphs and $\Omega(n)$ spanning trees for some chordal bipartite graphs. Furthermore, it was shown that any c-chordal graph admits a system of at most $\log_2 n$ collective additive tree $(2\lfloor c/2 \rfloor)$–spanners and any circular-arc graph admits a system of two collective additive tree 2–spanners. Paper [11] showed that any AT-free graph (graph without asteroidal triples) admits a system of two collective additive tree 2-spanners and any graph having a dominating shortest path admits a system of two collective additive tree 3-spanners and a system of five collective additive tree 2-spanners. In paper [8], it was shown that no system of constant number of collective additive tree 1-spanners can exist for unit interval graphs, no system of constant number of collective additive tree r-spanners can exist for chordal graphs and $r \leq 3$, and no system of constant number of collective additive tree r-spanners can exist for weakly chordal graphs and any constant r. On the other hand, [8] proved that any interval graph of diameter D admits an easily constructible system of $2\log(D-1)+4$ collective additive tree 1-spanners.

Only papers [10,16] have investigated (so far) collective tree spanners in the *weighted graphs*. Paper [10] demonstrated that any weighted graph with tree-width at most $k-1$ admits a system of $k\log_2 n$ collective additive tree 0–spanners, any weighted graph with clique-width at most k admits a system of $k\log_{3/2} n$ collective additive tree $(2\mathsf{w})$–spanners, and any weighted graph with size of largest induced cycle at most c (i.e., a c-chordal graph) admits a system of $4\log_2 n$ collective additive tree $(2(\lfloor c/3 \rfloor + 1)\mathsf{w})$–spanners (here, w is the maximum edge weight in G). The latter result was refined for weighted weakly chordal graphs: any such graph admits a system of $4\log_2 n$ collective additive tree $(2\mathsf{w})$-spanners. In [16], it was shown that any n–vertex planar graph admits a system of $O(\sqrt{n})$ collective multiplicative tree 1-spanners (equivalently, additive tree 0-spanners) and a system of at most $2\log_{3/2} n$ collective multiplicative tree 3–spanners.

In this paper we study collective additive tree spanners in homogeneously orderable graphs. The class of homogeneously orderable graphs was introduced in [4] to generalize the dually chordal graphs and the distance-hereditary graphs and to show that the Steiner tree problem can still be solved in polynomial time on this more general class of graphs. The follow up to [4] paper [9] showed also that both the connected r-domination problem and the r-dominating clique problem are polynomial time solvable on homogeneously orderable graphs. Homogeneously orderable graphs (HOGs) is a large family of graphs which comprises a number of well-known graph classes including Dually Chordal graphs, House-Hole-Domino-Sun-free graphs (HHDS-free graphs), Distance-Hereditary graphs, Strongly Chordal graphs, Interval graphs and others (see Figure 1). In Section 3, we show that every homogeneously orderable graph admits an additive tree 3-spanner constructible in linear time. In Section 4, we demonstrate

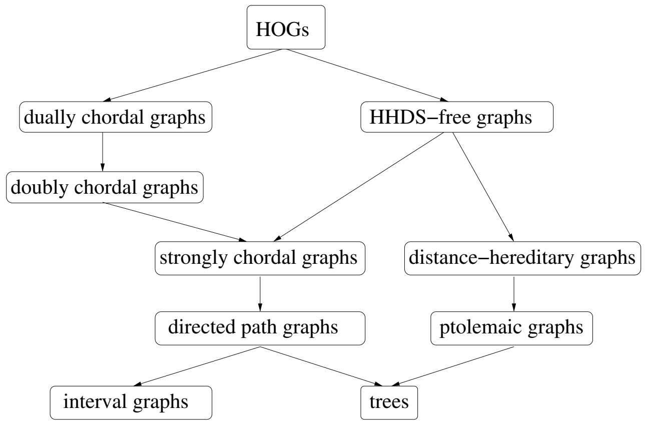


Fig. 1. Hierarchy of Homogeneously Orderable Graphs. For definitions of graph classes included see [5].

that every homogeneously orderable graph admits a system of $O(\log n)$ collective additive tree 2-spanners constructible in polynomial time. These results generalize known results on tree spanners of dually chordal graphs and of distance-hereditary graphs (see [2] and [19], respectively). Table 1 summarizes the results of this paper.

Table 1. Collective additive tree spanners of n-vertex homogeneously orderable graphs

additive stretch factor	upper bound on number of trees	lower bound on number of trees
3	1	1
2	$\log_2 n$	$c < \mu \leq \log_2 n$
1	$n - 1$	$\Omega(n)$
0	$n - 1$	$\Omega(n)$

2 Preliminaries

All graphs occurring in this paper are connected, finite, undirected, unweighted, loopless and without multiple edges. In a graph $G = (V, E)$ $(n = |V|, m = |E|)$ the *length* of a path from a vertex v to a vertex u is the number of edges in the path. The *distance* $d_G(u, v)$ between the vertices u and v is the length of a shortest path connecting u and v. The i-th *neighborhood* of a vertex v of G is the set $N_i(v) = \{u \in V : d_G(v, u) = i\}$. For a vertex v of G, the sets $N(v) = N_1(v)$ and $N[v] = N(v) \cup \{v\}$ are called the *open neighborhood* and the *closed neighborhood* of v, respectively. For a set $S \subseteq V$, by $N[S] = \bigcup_{v \in S} N[v]$ we denote the *closed neighborhood* of S and by $N(S) = N[S] \setminus S$ the *open neighborhood* of S. The *disk* of radius k centered at v is the set of all vertices of distance at most k to v, i.e., $D_k(v) = \{u \in V : d_G(u,v) \leq k\} = \bigcup\{N_i(v) : i = 0, \dots, k\}$.

Denote by $\mathcal{D}(G) = \{D_r(v) : v \in V,\ r$ a non-negative integer$\}$ the *family of all possible disks* of G and by $L(\mathcal{D}(G))$ *the intersection graph of those disks*, i.e., the vertices of $L(\mathcal{D}(G))$ are disks from $\mathcal{D}(G)$ and two vertices are adjacent if and only if the corresponding disks share a common vertex. Note that two disks $D_p(x)$ and $D_q(y)$ intersect if and only if $d_G(x,y) \leq p+q$. The *k–th power* G^k of a graph $G = (V,E)$ is the graph with vertex set V and edges between vertices u,v with distance $d_G(u,v) \leq k$. In what follows, a subset U of V is a *k–set* if U induces a clique in the power G^k, i.e., for any pair x,y of vertices of U we have $d_G(x,y) \leq k$. A graph G is called *chordal* if it does not have any induced cycle of length greater than 3.

We say that a graph $G = (V,E)$ *admits a system of* μ *collective additive tree r-spanners* if there is a system $\mathcal{T}(G)$ of at most μ spanning trees of G such that for any two vertices x,y of G a spanning tree $T \in \mathcal{T}(G)$ exists such that $d_T(x,y) \leq d_G(x,y) + r$. If $\mu = 1$, then the tree T such that $\mathcal{T}(G) = \{T\}$ is called a *tree r-spanner* of G.

A nonempty set $H \subseteq V$ is *homogeneous* in $G = (V,E)$ if all vertices of H have the same neighborhood in $V \setminus H$, i.e., $N(u) \cap (V \setminus H) = N(v) \cap (V \setminus H)$ for all $u,v \in H$, (any vertex $w \in V \setminus H$ is adjacent to either all or none of the vertices from H). A homogeneous set H is *proper* if $|H| < |V|$. Trivially for each $v \in V$ the singleton $\{v\}$ is a proper homogeneous set. Let U_1, U_2 be disjoint subsets of V. If every vertex of U_1 is adjacent to every vertex of U_2 then U_1 and U_2 form a *join*, denoted by $U_1 \bowtie U_2$. A set $U \subseteq V$ is *join–split* if U can be partitioned into two nonempty sets U_1, U_2 such that $U = U_1 \bowtie U_2$.

Next we recall the definition of homogeneously orderable graphs as given in [4]. A vertex v of $G = (V,E)$ with $|V| > 1$ is *h–extremal* if there is a proper subset $H \subset D_2(v)$ which is homogeneous in G and for which $D_2(v) \subseteq N[H]$ holds, i.e., H dominates $D_2(v)$. Thus, the sets H and $D_2(v) \setminus H$ form a join. A sequence $\sigma = (v_1, \dots, v_n)$ is an *h–extremal ordering* of a graph G if for any $i = 1, \dots, n-1$ the vertex v_i is h–extremal in $G_i := G(\{v_i, \dots, v_n\})$ (induced subgraph of G formed by vertices of $\{v_i, \dots, v_n\}$). A graph G is *homogeneously orderable* if G has an h–extremal ordering.

In [4] it is proved that a graph G is homogeneously orderable if and only if the square G^2 of G is chordal and each maximal two–set of G is join–split. This local structure of homogeneously orderable graphs implies a simple $O(n^3)$ recognition algorithm of this class, which uses the chordality of the squares (see [4]). Since the square G^2 of a graph G is chordal if and only if the graph $L(\mathcal{D}(G))$ is chordal [3], we can reformulate that characterization in the following way.

Theorem 1. [4] *A graph G is a homogeneously orderable graph if and only if the graph $L(\mathcal{D}(G))$ of G is chordal and each maximal two-set of G is join-split.*

This characterization will be very useful in Section 3 and Section 4. In Section 4, the following theorem from [9] will also be of use.

Theorem 2. [9] *For any homogeneously orderable graph $G = (V, E)$ with vertex function $r : V \to N$ and for any subset S of V, we have that S is r-dominated by some clique C of G (i.e, $d_G(v, C) \leq r(v)$ for every $v \in S$) if and only if $d_G(x, y) \leq r(x) + r(y) + 1$ for all $x, y \in S$.*

Finally, it is well known that the following lemma for chordal graphs holds.

Lemma 1 (Cycle Lemma for Chordal Graphs). *Let $C = (v_0, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_{k-1})$ be a cycle of a chordal graph G with $k \geq 4$. Then, for any vertex v_i of C, $v_i v_j \in E(G)$, for $j \neq i-1, i, i+1$ (mod k), or $v_{i-1}v_{i+1} \in E(G)$.*

3 Additive Tree 3-Spanners

In this section, we show that every homogeneously orderable graph G admits an additive tree 3-spanner. We prove that an algorithm similar to one presented in [2] for constructing an additive tree 3-spanner of a dually chordal graph constructs an additive tree 3-spanner of a homogeneously orderable graph. However, its correctness proof (Lemma 3 and Lemma 4) for homogeneously orderable graphs is more involved.

Assume u is an arbitrary vertex of a homogeneously orderable graph G and $k \in \{1, 2, \ldots, ecc(u)\}$, where $ecc(u) = \max\{d_G(u, v) : v \in V(G)\}$. Let $F_1^k, \ldots, F_{p_k}^k$ be the connected components of the graph obtained from G by removing vertices of $D_{k-1}(u)$. Denote $S_i^k = F_i^k \cap N_k(u)$ and $M_i^{k-1} = N(S_i^k) \cap N_{k-1}(u)$. M_i^{k-1} is called the *projection* of S_i^k to layer $N_{k-1}(u)$. Clearly, any two vertices $x, y \in S_i^k$ are connected by a path outside the disk $D_{k-1}(u)$. Denote by H_{k-1} the graph with $\bigcup_{i=1}^{p_k} M_i^{k-1}$ as the vertex set and two vertices x, y are adjacent in H_{k-1} if and only if they belong to a common set M_i^{k-1}.

Lemma 2. *Every connected component A of the graph H_k is a two-set in G.*

Proof. Let x, y be two vertices of a connected component A of the graph H_k. Then, we can find a collection of projections $M_{i_1}^k, M_{i_2}^k, \ldots, M_{i_h}^k$ such that $x \in M_{i_1}^k$, $y \in M_{i_h}^k$ and $M_{i_j}^k \cap M_{i_{j+1}}^k \neq \emptyset$ for all $j = 1, \ldots, h-1$. Pick $z_j \in M_{i_j}^k \cap M_{i_{j+1}}^k$, $j = 1, \ldots, h-1$ and let $z_0 := x$ and $z_h := y$. Since $z_{j-1}, z_j \in M_{i_j}^k$, $j = 1, \ldots, h$, we can find two vertices $v_j', v_j'' \in S_{i_j}^{k+1}$ adjacent to z_{j-1} and z_j, respectively. Let P_j be a path of $F_{i_j}^{k+1}$ connecting the vertices v_j' and v_j''. The disk $D_{k-1}(u)$ together with $D_1(x), D_1(y)$ and the disks of the family $\{D_1(z) : z \in \cup_{j=1}^h P_j\}$ forms a cycle in the intersection graph $L(\mathcal{D}(G))$. From chordality of this graph (see Theorem 1) and since $D_{k-1}(u) \cap D_1(z) = \emptyset$ holds for all $z \in \cup_{j=1}^h P_j$, we deduce (see the Cycle Lemma for Chordal Graphs) that $D_1(x) \cap D_1(y) \neq \emptyset$, i.e. $d_G(x, y) \leq 2$. □

Next, we are going to show that for every connected component A of the graph H_k there is a vertex $z \in N_{k-1}(u)$ such that $A \subseteq N(z)$. To show that, the following lemma is needed, proof of which is omitted in this version.

Lemma 3. *If $k \geq 2$ and A is a connected component of the graph H_k, then there is a vertex $t \in N_{k-2}(u)$ such that $A \subseteq N_2(t)$.*

Now, we are ready to prove the following lemma.

Lemma 4. *For every connected component A of the graph H_k there is a vertex $z \in N_{k-1}(u)$ such that $A \subseteq N(z)$.*

Proof. If $k = 1$, then the lemma clearly holds. Hence, assume $k \geq 2$. According to Lemma 3, there is a vertex $t \in N_{k-2}(u)$ such that $A \subseteq N_2(t)$. Hence, $A \cup \{t\}$ is a two-set of G. Let U be a maximal two-set of G such that $A \subseteq U$. By Theorem 1, U consists of two subsets U_1 and U_2 and $U_1 \bowtie U_2$. Clearly, $A \cup \{t\}$ must be in either U_1 or U_2, since $d_G(x, t) = 2$ for each $x \in A$. Without loss of generality, assume $A \cup \{t\}$ is in U_1. Then, U_2 must contain a vertex z which is adjacent to all the vertices in $A \cup \{t\}$. This vertex z can only be in $N_{k-1}(u)$. This concludes our proof. □

From the above lemmata, one can give the following linear time algorithm to construct an additive tree 3-spanner for a homogeneously orderable graph G.

PROCEDURE 1. Construct an additive tree 3-spanner for a HOG G

Input: A homogeneously orderable graph $G = (V, E)$.
Output: An additive tree 3-spanner T of G.
Method:

```
set $E' = \emptyset$;
pick an arbitrary vertex $u$ in $G$;
set $i = ecc(u)$;
for each vertex $x \in N_i(u)$ do
  arbitrarily choose a vertex $y \in N_{i-1}(u)$ such that $xy \in E(G)$;
  add $xy$ into $E'$;
set $i$ to $i - 1$;
while $i \geq 1$ do
  construct the graph $H_i$ and find its connected components;
  for each connected component $A$ of $H_i$ do
    find a vertex $z \in N_{i-1}(u)$ such that $A \subseteq N(z)$;
    for each vertex $x \in A$ add $xz$ into $E'$;
  for other vertices $x$ which are in $N_i(u)$ but not in $H_i$;
    arbitrarily choose a vertex $y \in N_{i-1}(u)$ such that $xy \in E(G)$;
    add $xy$ into $E'$;
output $T = (V, E')$.
```

It is not hard to show that Procedure 1 can be implemented to run in linear time. The most complex step is the construction of the connected components of the graphs H_i ($i = ecc(u) - 1, \ldots, 2, 1$). We start from the layer $N_k(u)$, where $k = ecc(u)$, find its connected components $F_1^k, \ldots, F_{p_k}^k$ and contract each of them into a vertex. Then find the connected components $F_1^{k-1}, \ldots, F_{p_{k-1}}^{k-1}$ in the graph induced by $N_{k-1}(u)$ and the set of contracted vertices. To find the connected components of the graph H_{k-1}, we construct a special bipartite graph $B_{k-1} = (W, K; U)$. In this graph $W = \{f_1, \ldots, f_{p_k}\}$ (a vertex f_j represents component F_j^k), and K is the vertex set of H_{k-1} (which is $\bigcup_{i=1}^{p_k} M_i^{k-1} = \bigcup_{i=1}^{p_k}(N(F_i^k) \cap N_{k-1}(u))$. A vertex $f_j \in W$ and a vertex $v \in K$ are adjacent in B_{k-1} if and only if $v \in N(F_j^k)$. The graph B_{k-1} can be constructed in $O(\sum_{v \in N_k(u) \cup N_{k-1}(u)} deg(v))$ time. Note that a vertex $v \in N_{k-1}(u)$ belongs to H_{k-1} if and only if it has a neighbor in $N_k(u)$. The connected components of H_{k-1} are exactly the intersections of the connected components of B_{k-1} with the set K. After performing for H_{k-1} all operations prescribed in the other lines of Procedure 1, we contract each of $F_1^{k-1}, \ldots, F_{p_{k-1}}^{k-1}$ into a vertex and descend to the lower level. We repeat the above until we come to the vertex u.

Theorem 3. *The spanning tree T constructed by Procedure 1 is an additive tree 3-spanner of G.*

Proof. Let x, y be two arbitrary vertices of G. Assume $x \in N_{l_x}(u)$ and $y \in N_{l_y}(u)$. Without loss of generality, assume $l_x \leq l_y$. Let $P = (x_0 = x, x_1, \cdots, x_p = y)$ be a shortest path between x and y in G. Let k be the smallest integer such that $P \cap N_k(u) \neq \emptyset$. There are two cases to consider.

Case 1: *There is exactly one vertex x_i in $P \cap N_k(u)$.*
First, consider the subcase when $i = 0$, i.e., $x_i = x_0 = x$. Let P_T be the path between y and u in T. Let y' be the vertex of P_T from $N_k(u)$. Clearly, x and y' belong to the projection M_j^k of that connected component F_j^{k+1} (of the induced subgraph of G formed by vertices $V \setminus D_k(u)$) which contains vertex y. By the way T was constructed, $d_T(x, y') \leq 2$ and $d_T(y, y') = l_y - l_x$ must hold. This implies $d_T(x, y) \leq d_T(x, y') + d_T(y', y) \leq 2 + l_y - l_x \leq 2 + d_G(x, y)$ since $d_G(x, y) \geq l_y - l_x$.

Let now $i \neq 0$. Since x_i is the only vertex in $P \cap N_k(u)$, x_{i-1} and x_{i+1} must be in $N_{k+1}(u)$. Let P'_T be the path between x and u and P_T be the path between y and u in T. Let x' and y' be the vertices of P'_T and P_T taken from $N_k(u)$. Clearly, x' and x_i belong to the projection $M_{j_x}^k$ of that connected component $F_{j_x}^{k+1}$ (of the induced subgraph of G formed by vertices $V \setminus D_k(u)$) which contains vertex x, and y' and x_i belong to the projection $M_{j_y}^k$ of that connected component $F_{j_y}^{k+1}$ which contains vertex y (it is possible that $F_{j_y}^{k+1} = F_{j_x}^{k+1}$). Since these projections $M_{j_y}^k$ and $M_{j_x}^k$ share a common vertex x_i, they must belong to the same connected component of the graph H_k. By the way T was constructed, $d_T(x', y') \leq 2$ and $d_T(y, y') = l_y - k$, $d_T(x, x') = l_x - k$ must hold. This implies $d_T(x, y) \leq d_T(x, x') + d_T(x', y') + d_T(y', y) \leq l_x - k + 2 + l_y - k \leq d_G(x, y) + 2$ since $d_G(x, y) = d_G(x, x_i) + d_G(x_i, y) \geq l_x - k + l_y - k$.

Case 2: *There are at least two vertices x_i, x_j in $P \cap N_k(u)$.*
In this case, $d_G(x, y) > l_x - k + l_y - k$ (i.e., $d_G(x, y) \geq l_x + l_y - 2k + 1$) must hold. Again, let P'_T be the path between x and u and P_T be the path between y and u in T. Let x' and y' be the vertices of P'_T and P_T, respectively, taken from $N_k(u)$. Clearly, x', y', x_i and x_j are in one connected component F_t^k of the induced subgraph of G formed by vertices $V \setminus D_{k-1}(u)$. By the way T was constructed, this implies $d_T(x', y') \leq 4$. Hence, $d_T(x, y) \leq d_T(x, x') + d_T(y, y') + d_T(x', y') \leq$

$l_y + l_x - 2k + 4 \leq d_G(x, y) + 3$. This proves the second case and concludes the proof of the theorem. □

Note that in [2] an example of a dually chordal graph (and hence of a homogeneously orderable graph) is presented which does not have any additive tree 2-spanner. Thus, the additive stretch factor 3 for the tree spanners presented in Theorem 3 is best possible if one wants to achieve it only with one tree.

4 Collective Additive Tree 2-Spanners

In this section, we show that every homogeneously orderable graph H admits a system of $O(\log n)$ collective additive tree 2-spanners. According to [4], if a graph H is homogeneously orderable, then $G = H^2$ is a chordal graph. Unfortunately, the method developed in [12] for constructing collective tree spanners in some hereditary classes of graphs (so-called (α, r)–*decomposable graphs*) cannot be directly applied to the class of homogeneously orderable graphs as this class is not hereditary (see [4]). We will work first on the square $G = H^2$ of a homogeneously orderable graph H and then will move down to the original graph H. To the best of our knowledge this is the first non-trivial result on collective tree spanners of a non-hereditary family of graphs.

The following theorem is known for chordal graphs.

Theorem 4. [15] *Every n-vertex chordal graph G contains a maximal clique S such that if the vertices in S are deleted from G, every connected component in the graph induced by any remaining vertices is of size at most $n/2$.*

A linear time algorithm for finding for a chordal graph G a separating clique S satisfying the condition of the theorem is also given in [15]. We will call S a *balanced separator* of G.

Using the above theorem, one can construct for any chordal graph G a *(rooted) balanced decomposition tree* $\mathcal{BT}(G)$ as follows. If G is a complete graph, then $\mathcal{BT}(G)$ is a one-node tree. Otherwise, find a balanced separator S in G, which exists according to Theorem 4. Let $G_1, G_2, \ldots, G_p$ be the connected components of the graph $G \setminus S$ obtained from G by removing vertices of S. For each graph G_i $(i = 1, \ldots, p)$, which is also a chordal graph, construct a balanced decomposition tree $\mathcal{BT}(G_i)$ recursively, and build $\mathcal{BT}(G)$ by taking S to be the root and connecting the root of each tree $\mathcal{BT}(G_i)$ as a child of S. Clearly, the nodes of $\mathcal{BT}(G)$ represent a partition of the vertex set V of G into *clusters* $S_1, S_2, \ldots, S_q$ which are cliques. For a node X of $\mathcal{BT}(G)$, denote by $G(\downarrow X)$ the (connected) subgraph of G induced by vertices $\bigcup\{Y : Y$ is a descendent of X in $\mathcal{BT}(G)\}$ (here we assume that X is a descendent of itself).

Consider two arbitrary vertices x and y of a chordal graph G and let $S(x)$ and $S(y)$ be the nodes of $\mathcal{BT}(G)$ containing x and y, respectively. Let also $NCA_{\mathcal{BT}(G)}(S(x), S(y)) = X_t$ be the nearest common ancestor and $X_0, X_1, \ldots,$ X_t be the sequence of common ancestors of $S(x)$ and $S(y)$ in $\mathcal{BT}(G)$ starting from the root X_0 of $\mathcal{BT}(G)$. We will need the following lemma from [12].

Lemma 5. [12] *Any path $P^G_{x,y}$, connecting vertices x and y in G, contains a vertex from $X_0 \cup \cdots \cup X_t$.*

Let now $H = (V, E)$ be a homogeneously orderable graph. By Theorem 1, $G = H^2$ is a chordal graph. We construct a *rooted balanced decomposition tree* $\mathcal{BT}(G)$ for G as described above. It is also known that any maximal clique of G is a join-split two–set of H. For any maximal clique C of G, let $C = U_1 \cup U_2$ such that, in H, $U_1 \bowtie U_2$. Note that X_i $(i = 0, 1, \ldots, t)$ may not be a maximal clique in G. Let C'_i be a maximal clique of G such that $X_i \subseteq C'_i$. Assume $C'_i = U^i_1 \cup U^i_2$ with $U^i_1 \bowtie U^i_2$ in H.

There are two cases to consider:

(a) $U^i_1 \cap X_i \neq \emptyset$ and $U^i_2 \cap X_i \neq \emptyset$;
(b) either $U^i_1 \cap X_i = \emptyset$ or $U^i_2 \cap X_i = \emptyset$ (without loss of generality, we assume $U^i_1 \cap X_i = \emptyset$ in this case).

For each connected graph $G(\downarrow X_i)$, let $H(\downarrow X_i)$ be the induced subgraph of H which has the same vertex set as $G(\downarrow X_i)$. Note that $H(\downarrow X_i)$ may not be connected as not all edges of $G(\downarrow X_i)$ are present in $H(\downarrow X_i)$. We construct a tree T_i for $H(\downarrow X_i)$ in the following way.

If $U^i_1 \cap X_i \neq \emptyset$ and $U^i_2 \cap X_i \neq \emptyset$ holds, then we choose two vertices $r_1 \in X_i \cap U^i_1, r_2 \in X_i \cap U^i_2$ and add r_1r_2 into $E(T_i)$. For each $x \in X_i \cap U^i_1$, we add xr_2 into $E(T_i)$. For each $y \in X_i \cap U^i_2$, we add yr_1 into $E(T_i)$. Then we extend the tree constructed so far by building a breadth-first-search-tree in $H(\downarrow X_i)$ starting at set X_i and spanning that connected component of $H(\downarrow X_i)$ which contains r_1 and r_2.

If $U^i_1 \cap X_i = \emptyset$ holds, then we choose a vertex r from U^i_1 (which may not be even in $H(\downarrow X_i)$) and construct a tree T_i as follows. For every vertex $x \in X_i$, we put xr into $E(T_i)$. Then, we extend the tree constructed so far by building a breadth-first-search-tree in $H(\downarrow X_i)$ starting at set X_i and spanning the vertices that can reach r via vertices in $H(\downarrow X_i)$.

Below we will prove that for some vertices $x, y \in H(\downarrow X_i)$ the tree T_i will guarantee $d_{T_i}(x, y) \leq d_H(x, y) + 2$. Let $x \in V(H(\downarrow X_i))$ and $x' \in X_i$ be vertices such that $d_{T_i}(x, x') = d_{H(\downarrow X_i)}(x, X_i)$. Vertex x' is called the *projection* of x in T_i. We use $P_{T_i}(x)$ to denote the shortest path between x' and x in T_i.

Consider any two vertices $x, y \in V$. Let as before $X_0, X_1, \ldots, X_t$ be the common ancestors of $S(x)$ and $S(y)$ in $\mathcal{BT}(G)$, and let X_i be the common ancestor with minimum i such that there is a shortest path $P^H_{x,y}$ between x and y in H with $P^H_{x,y} \cap X_i \neq \emptyset$. It is easy to see that such X_i must exist. Indeed, according to Lemma 5, any path $P^G_{x,y}$ between x and y in G must intersect $X_0 \cup \cdots \cup X_t$ and any path $P^H_{x,y}$ is also a path between x and y in G (as $G = H^2$ and therefore the edge set of H is a subset of the edge set of G). By the choice of X_i, any shortest path in H connecting x and y must be in $H(\downarrow X_i)$. Let T_i be the tree constructed as above for $H(\downarrow X_i)$. The following lemma can be proved to be true.

Lemma 6. *$d_{T_i}(x, y) \leq d_H(x, y) + 2$ must hold.*

Proof. Let $P^H_{x,y}$ be a shortest path of H connecting x and y. By the choice of X_i, this path is entirely in $H(\downarrow X_i)$ and intersects X_i. Let z_x, z_y be vertices of this path and X_i closest to x and y, respectively.

Let x', y' be the projections of x and y in T_i. According to the way T_i was constructed, $d_{T_i}(x', y') \leq 3$, $d_{T_i}(x, x') = d_{H(\downarrow X_i)}(x, x') \leq d_{H(\downarrow X_i)}(x, z_x) = d_H(x, z_x)$ and $d_{T_i}(y, y') = d_{H(\downarrow X_i)}(y, y') \leq d_{H(\downarrow X_i)}(y, z_y) = d_H(y, z_y)$. If $d_{T_i}(x', y') \leq 2$ or $z_x \neq z_y$, then we are done since $d_{T_i}(x, y) \leq d_{T_i}(x, x') + d_{T_i}(x', y') + d_{T_i}(y, y') \leq d_H(x, z_x) + d_{T_i}(x', y') + d_H(y, z_y) = d_H(x, y) - d_H(z_x, z_y) + d_{T_i}(x', y') \leq d_H(x, y) + 2$. Hence, assume that $d_{T_i}(x', y') = 3$ and $d_H(x, y) = d_H(x, z_x) + d_H(y, z_y)$, i.e., $z_x = z_y$. Denote $s := z_x = z_y$, $d_H(x, s) := l_x$ and $d_H(y, s) := l_y$. Since $d_{T_i}(x', y') = 3$, by the way T_i was constructed, we have $U^i_1 \cap X_i \neq \emptyset$ and $U^i_2 \cap X_i \neq \emptyset$. Furthermore, vertices x' and y' can not be both in $U^i_1 \cap X_i$ or both in $U^i_2 \cap X_i$ (otherwise, $d_{T_i}(x', y') \leq 2$). Without loss of generality, assume $x' \in U^i_1 \cap X_i$, $y' \in U^i_2 \cap X_i$ and $s \in U^i_1 \cap X_i$. Consider the following radius function for vertices s, y, y' in H: $r(s) = 0, r(y') = 0$ and $r(y) = l_y - 1$. According to Theorem 2, there is a clique C in H such that C r-dominates the set $\{s, y', y\}$. Hence, there must exist a vertex $y'' \in (C \setminus X_i)$ such that $sy'', y'y'' \in E(H)$ and $d_H(y'', y) = l_y - 2$. This implies that y'' is on a shortest path between x and y in H and, by the choice of X_i, y'' is in $H(\downarrow X_i)$. Since $sy'', y'y'' \in E(H)$, we have $d_H(y'', z) \leq 2$ for any vertex $z \in U^i_1 \cup U^i_2$. Therefore, $\{y''\} \cup X_i$ is a clique in $G(\downarrow X_i)$. The latter contradicts with the fact that X_i was a maximal clique in $G(\downarrow X_i)$ (see Theorem 4 and paragraph on the construction of $\mathcal{BT}(G)$ after that theorem). Thus, the case that $d_{T_i}(x', y') = 3$ and $z_x = z_y$ is impossible. This concludes our proof. □

Let now $B^i_1, \ldots, B^i_{p_i}$ be the nodes on depth i of the tree $\mathcal{BT}(G)$. For each subgraph $H^i_j = H(\downarrow B^i_j)$ of H $(i = 0, 1, \ldots, depth(\mathcal{BT}(G)), j = 1, 2, \ldots, p_i)$, denote by T^i_j the tree constructed for H^i_j as above. Clearly, T^i_j can be constructed in linear time. We call T^i_j a *local tree of* H. Since any two local trees T^i_j and $T^i_{j'}$ share at most one vertex, $T^i = \bigcup\{T^i_j, j = 0, \ldots, p_i\}$ is a forest. T^i can be extended to T'^i to span all the vertices in H. Tree T'^i is called a *global spanning tree of* H. By Lemma 6, we immediately get the following result.

Lemma 7. *For any two vertices $x, y \in V$, there exists a global spanning tree T'^i such that $d_{T'^i}(x, y) \leq d_H(x, y) + 2$.*

By the way $\mathcal{BT}(G)$ was constructed, the depth of $\mathcal{BT}(G)$ is at most $\log_2 n$. Hence, there are at most $\log_2 n$ global spanning trees of H. Obviously, $G = H^2$ can be obtained in $O(nm)$ time. According to [12], $\mathcal{BT}(G)$ can be constructed in $O(n^2 \log_2 n)$ time (note that G may have $O(n^2)$ edges). Each T'^i is constructible in linear time. Therefore, the following theorem is true.

Theorem 5. *Any n-vertex homogeneously orderable graph admits a system of $\log_2 n$ collective additive tree 2-spanners which can be constructed in $O(nm + n^2 \log_2 n)$ time.*

5 Conclusion

In this paper we studied collective additive tree spanners in homogeneously orderable graphs. We showed that every n-vertex homogeneously orderable graph admits an additive tree 3-spanner constructible in linear time and a system of $O(\log n)$ collective additive tree 2-spanners constructible in polynomial time. These results generalize known results on tree spanners of dually chordal graphs and of distance-hereditary graphs. We mentioned also that there are homogeneously orderable graphs which do not admit any additive tree 2-spanners. Hence, it is natural to ask how many spanning trees are necessary to achieve collective additive stretch factor of 2 in homogeneously orderable graphs. We know that this number is not a constant (a proof of this is presented in the full version of the paper) and is not larger than $\log_2 n$. One may ask also how many spanning trees are necessary and how many are sufficient to achieve collective additive stretch factor of 1 or 0 in homogeneously orderable graphs. The answer to this question is simple. Any graph G on n vertices admits a system of at most $n-1$ collective additive tree 0-spanners. On the other hand, it is easy to see that any system of collective additive tree 1–spanners of homogeneously orderable graphs will need to have at least $\Omega(n)$ spanning trees (a proof of this is presented in the full version).

References

1. Bartal, Y.: On approximating arbitrary metrices by tree metrics. In: STOC 1998, pp. 161–8 (1998)
2. Brandstädt, A., Chepoi, V., Dragan, F.F.: Distance Approximating Trees for Chordal and Dually Chordal Graphs. J. Algorithms 30, 166–184 (1999)
3. Brandstädt, A., Dragan, F.F., Chepoi, V.D., Voloshin, V.I.: Dually chordal graphs. SIAM J. Discrete Math. 11, 437–455 (1998)
4. Brandstädt, A., Dragan, F.F., Nicolai, F.: Homogeneously orderable graphs. Theoretical Computer Science 172, 209–232 (1997)
5. Brandstädt, A., Le Bang, V., Spinrad, J.P.: Graph Classes: A Survey, SIAM Monographs on Discrete Mathematics and Applications. Philadelphia (1999)
6. Cai, L., Corneil, D.G.: Tree spanners. SIAM J. Disc. Math. 8, 359–387 (1995)
7. Charikar, M., Chekuri, C., Goel, A., Guha, S., Plotkin, S.: Approximating a Finite Metric by a Small Number of Tree Metrics. In: FOCS 1998, pp. 379–388 (1998)
8. Corneil, D.G., Dragan, F.F., Köhler, E., Yan, C.: Collective tree 1-spanners for interval graphs. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 151–162. Springer, Heidelberg (2005)
9. Dragan, F.F., Nicolai, F.: r-Domination Problems on Homogeneously Orderable Graphs. Networks 30, 121–131 (1997)
10. Dragan, F.F., Yan, C.: Collective Tree Spanners in Graphs with Bounded Genus, Chordality, Tree-width, or Clique-width. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 583–592. Springer, Heidelberg (2005)
11. Dragan, F.F., Yan, C., Corneil, D.G.: Collective Tree Spanners and Routing in AT-free Related Graphs. J. of Graph Algorithms and Applications 10, 97–122 (2006)
12. Dragan, F.F., Yan, C., Lomonosov, I.: Collective tree spanners of graphs. SIAM J. Discrete Math. 20, 241–260 (2006)

13. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. In: STOC 2003, pp. 448–455 (2003)
14. Fraigniaud, P., Gavoille, C.: Routing in Trees. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 757–772. Springer, Heidelberg (2001)
15. Gilbert, J.R., Rose, D.J., Edenbrandt, A.: A separator theorem for chordal graphs. SIAM J. Alg. Discrete Meth. 5, 306–313 (1984)
16. Gupta, A., Kumar, A., Rastogi, R.: Traveling with a Pez Dispenser (or, Routing Issues in MPLS). SIAM J. Comput. 34, 453–474 (Also in FOCS 2001) (2005)
17. Liestman, A.L., Shermer, T.: Additive graph spanners. Networks 23, 343–364 (1993)
18. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM Monographs on Discrete Math. Appl. (2000)
19. Prisner, E.: Distance approximating spanning trees. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 499–510. Springer, Heidelberg (1997)
20. Thorup, M., Zwick, U.: Compact routing schemes. In: SPAA 2001, pp. 1–10 (2001)

The Generalized Median Stable Matchings: Finding Them Is Not That Easy

Christine T. Cheng

Department of Computer Science
University of Wisconsin–Milwaukee, Milwaukee, WI 53211, USA
ccheng@cs.uwm.edu

Abstract. Let I be a stable matching instance with N stable matchings. For each man m, order his N stable partners from his most preferred to his least preferred. Denote the ith woman in his sorted list as $p_i(m)$. Let α_i consist of the man-woman pairs where each man m is matched to $p_i(m)$. Teo and Sethuraman proved this surprising result: for $i = 1$ to N, not only is α_i a matching, it is also stable. The α_i's are called the *generalized median stable matchings* of I.

In this paper, we present a new characterization of these stable matchings that is solely based on I's rotation poset. We then prove the following: when $i = O(\log n)$, where n is the number of men, α_i can be found efficiently; but when i is a constant fraction of N, finding α_i is NP-hard. We also consider what it means to approximate the median stable matching of I, and present results for this problem.

1 Introduction

In the *stable marriage problem* (SM), there are n men and n women, each of whom has a list that ranks all individuals of the opposite sex. A *matching* is a set of man-woman pairs where each individual appears in at most one pair. The objective of the problem is to find a matching μ that has n pairs and has no *blocking pairs* – i.e., a man and a woman who prefer each other over their partners in μ. The rationale behind the stability condition is that if a blocking pair exists, then the man and the woman will likely leave their partners and thereby compromise the integrity of the matching μ. A celebrated result by Gale and Shapley states that every instance of SM has a stable matching that can be found in $O(n^2)$ time [5]. Today, centralized stable matching algorithms match medical residents to hospitals [14], students to schools [1,2], etc.

To find a stable matching for an arbitrary instance, Gale and Shapley presented the deferred-acceptance (DA) algorithm, where the men ask and the women accept or reject offers. The result is the *man-optimal/woman-pessimal* stable matching – every man is matched to the woman he prefers the most among all his partners in a stable matching and every woman is matched to the man she prefers the least among all her partners in a stable matching. On the other hand, when the men and women switch roles, the result is the *woman-optimal/man-pessimal* stable matching and is defined accordingly. Thus, while the DA algorithm produces a stable matching, one may not want to use the matching because it is biased towards one side of the matching. This motivates the problem of finding "fair" stable matchings.

Different notions of fair stable matchings have been considered in the past. Suppose a person's happiness in a stable matching is based on his/her partner's rank in his/her preference list. Selkow [12] and, later, Gusfield [6] studied the *minimum-regret* stable matching, which maximizes the happiness of the unhappiest person in the matching. Irving et al. [9], on the other hand, considered the *egalitarian* stable matching, which maximizes the sum of the happiness of all the participants. In both cases, the proposed stable matchings can be found in polynomial time. However, by using global measures, these types of stable matchings may sacrifice the happiness of some individuals as they aim for the greater good. In another direction, Klaus and Klijn [11] suggested designing probabilistic matching mechanisms that are *procedurally fair*. That is, a stable matching is considered fair if the procedure used to arrive at the outcome is equitable to all the participants. They studied three different random mechanisms and the probability distributions they induce on the stable matchings of an instance.

Yet another notion of fair stable matchings is due to Teo and Sethuraman [15]. Let I be a stable matching instance, $\mathcal{M}(I)$ its set of stable matchings, and $N = |\mathcal{M}(I)|$. Let $p_\mu(a)$ denote the partner of a in stable matching μ. For each man m, sort the multiset of women $\{p_\mu(m), \mu \in \mathcal{M}(I)\}$ from m's most preferred to least preferred woman. Let $p_i(m)$ denote the ith woman in this sorted list for $i = 1, \ldots, N$. Do the same for each woman w. By applying linear programming tools, they showed that the following family of stable matchings exists:

Lemma 1. [Teo and Sethuraman] *Let α_i consist of man-woman pairs where each man m is matched to $p_i(m)$. Similarly, let β_i consist of all man-woman pairs where each woman w is matched to $p_i(w)$. For $i = 1, \ldots, N$, α_i and β_i are stable matchings; moreover, $\alpha_i = \beta_{N-i+1}$.*

The most remarkable of these stable matchings are the ones in the "middle" – $\alpha_{(N+1)/2}$ when N is odd, and $\alpha_{N/2}$ and $\alpha_{(N+2)/2}$ when N is even – called the *median stable matching* of I.[1] It matches *every* participant to his/her (lower or upper) median stable partner and, thus, is fair at the individual level in a very strong sense. The stable matchings α_i and β_{N-i+1} are called the ith generalized median stable matching of I, for $i = 1, \ldots, N$.

Since Teo et al.'s work [15], two other sets of authors [10,4] have proven the existence of these matchings using different tools. No one, however, has addressed the complexity of finding an instance's median stable matching. Simply using the definition can be inefficient because there are instances whose number of stable matchings is exponential in the input size. Our goal is to fill this gap.

To find an instance's median stable matching, we take a combinatorial approach and use its *rotation poset*. Algorithmically, the rotation poset is a very

[1] Throughout the paper, we shall refer to an instance's median stable matching in singular form even though there can be two such matchings.

useful structure because it encodes all the stable matchings of the instance and, yet, is polynomial in the input size. Our main results are as follows:

- First, we present a new characterization of the generalized median stable matchings that is solely based on rotation posets. It implies that to find an instance's median stable matching, enumerating all of its stable matchings can be avoided; instead, the task can be accomplished by *counting* certain subsets of its rotation poset. Additionally, the characterization also provides interesting insights into the generalized median stable matchings that are not evident from their definitions.
- We prove that finding α_i and α_{N-i+1} can be done efficiently when $i = O(\log n)$, but that it is NP-hard when i is a constant fraction of N. Hence, finding an instance's median stable matching is NP-hard.
- Finally, we consider what it means to approximate the median stable matching of an instance, and present results for this problem.

The outline for the rest of the paper is as follows: in Section 2, we define rotation posets and describe their properties. We present the new characterization for the generalized median stable matchings in Section 3, and prove the easy cases and the hardness result for finding α_i in Section 4. We consider the problem of approximating the median stable matching in Section 5.

2 Some Background: Distributive Lattices and Rotation Posets

Let I be a stable matching instance, and μ and μ' be two of its stable matchings. An individual a *prefers* μ to μ' if a prefers his/her partner in μ over his/her partner in μ'; otherwise, a prefers μ' to μ or is *indifferent* between them if a has the same partner in both matchings. The stable matching μ *dominates* μ', denoted as $\mu \preceq \mu'$, if every man prefers μ to μ' or is indifferent between them. It turns out that the dominance relation $\preceq$ induces a nice structure on $\mathcal{M}(I)$.

Theorem 1. *[12] $(\mathcal{M}(I), \preceq)$ is a distributive lattice.*

The *top* and *bottom* elements of $\mathcal{M}(I)$[2] are the man-optimal stable matching μ_M and the woman-optimal stable matching μ_W of I, respectively since μ_M dominates every stable matching of I which, in turn, dominates μ_W.

Associated with the distributive lattice $\mathcal{M}(I)$ is the *rotation poset* of I. *Rotations* are the incremental changes that need to be made so that a stable matching μ can be transformed into another stable matching μ' that it dominates. We define them formally next; we refer readers to [7] for a thorough discussion of the subject.

When $\mu \neq \mu_W$, there is a man m so that $p_\mu(m) \neq p_{\mu_W}(m)$. For each such man m, define his *successor woman*, $s_\mu(m)$, as the first woman on his preference

[2] We sometimes use $\mathcal{M}(I)$ to refer to the set as well as the distributive lattice. The context will indicate which one we are referring to.

list that follows $p_\mu(m)$, and prefers him over her current partner in μ. For example, $p_{\mu_W}(m)$ is a possible candidate for $s_\mu(m)$ but there may be other eligible women ahead of her in m's preference list. A *rotation ρ exposed in μ* is a cyclic sequence of man-woman pairs $\rho = ((m_0, w_0), (m_1, w_1), \ldots, (m_{r-1}, w_{r-1}))$ such that $(m_i, w_i) \in \mu$ and $s_\mu(m_i) = w_{i+1}$ for all i where the addition in the subscript is modulo r. To *eliminate* ρ from μ, each man m_i in ρ is matched to w_{i+1} while the rest of the pairs not in ρ are kept the same. The result is another stable matching denoted as μ/ρ which μ dominates.

Let I be an SM instance of size n (i.e., it has n men and n women), and let $R(I)$ denote the set of all rotations that are exposed in the stable matchings of I. We note the following properties of rotations. First, a man-woman pair can be part of at most one rotation of I. Thus, $|R(I)| \leq n^2/2$ because every rotation consists of at least two pairs, and the rotations in $R(I)$ together contain at most n^2 pairs. Second, if ρ and ρ' are two rotations exposed in μ, ρ and ρ' do not have any pairs in common. This implies that ρ' remains exposed in μ/ρ. Third, whenever $\mu \neq \mu_W$, there will always be at least one rotation ρ exposed in μ. Furthermore, there is no stable matching μ' such that $\mu \prec \mu' \prec \mu/\rho$. Thus, in the Hasse diagram of $\mathcal{M}(I)$, every edge between two stable matchings can be labeled by the rotation whose elimination from the dominant stable matching results in the dominated stable matching.

A rotation ρ' *precedes* rotation ρ, $\rho' \leq \rho$, if in order to obtain a stable matching in which ρ is exposed, ρ' must be eliminated first. The pair $(R(I), \leq)$ is called the *rotation poset of I*. Gusfield has shown that $R(I)$ can be constructed in an efficient manner [6]; more precisely, a *rotation digraph $G(I)$* whose vertices correspond to the rotations of $R(I)$ and whose transitive closure contains all the precedence relations of $R(I)$ can be built in $O(n^2)$ time.

A subset S of $R(I)$ is *closed* if whenever a rotation is in S, all rotations that precede it are also in S. There is a very nice correspondence between the stable matchings of I and the closed subsets of its rotation poset:

Theorem 2. *[8] There is a one-to-one correspondence between the elements of $\mathcal{M}(I)$ and the closed subsets of $R(I)$. In particular, if $\mu \in \mathcal{M}(I)$ corresponds to the closed subset S of $R(I)$, then μ can be obtained by starting at μ_M and eliminating all the rotations in S.*

For example, the empty subset of $R(I)$ corresponds to μ_M while $R(I)$ itself corresponds to μ_W. Given $R(I)$ and one of its closed subsets S, finding the stable matching that corresponds to S takes $O(n^2)$ time since constructing μ_M takes $O(n^2)$ time, and the total number of pairs in the rotations of S cannot exceed n^2. Conversely, finding the closed subset of $R(I)$ that corresponds to a stable matching μ of I also takes $O(n^2)$ time. This can be done by starting at μ_M, and then eliminating rotations until the partner of every man m is $p_\mu(m)$. The set containing all the eliminated rotations corresponds to μ.

When P is a poset, let $c(P)$ denote the number of closed subsets of P. According to Theorem 2, $c(R(I)) = |\mathcal{M}(I)|$. Later, we will use the fact that $c(P)$ is also equal to the number of antichains of P. The correspondence is as follows: if A is an antichain, let S_A be the closed subset that contains A and all the

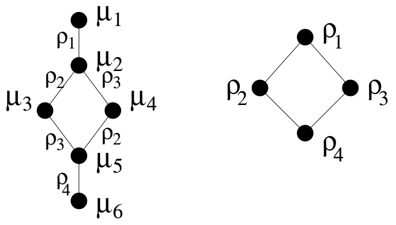


Fig. 1. The Hasse diagrams for the lattice of stable matchings and rotation poset of the example in Section 2

elements that precede a for each $a \in A$; if S is a closed subset, let A_S be the antichain that contains all the bottom elements of S.

Interestingly, the rotation posets of SM instances do not have any special structure in the following sense – Blair showed that for every poset P, there is an SM instance $I(P)$ whose rotation poset is isomorphic to P [3]. In [8], Irving and Leather presented an algorithm that given P will construct $I(P)$ whose size and construction time is $O(|P|^2)$. Moreover, defining the isomorphism between the elements of P and the rotations of $I(P)$ is straightforward.

Before we end this section, we present an example to illustrate the distributive lattice and rotation poset of the following SM instance I:

m_1: $w_1\ w_2\ w_3\ w_4\ w_5\ w_6$	w_1: $m_4\ m_5\ m_3\ m_1\ m_2\ m_4$
m_2: $w_2\ w_3\ w_4\ w_1\ w_5\ w_6$	w_2: $m_1\ m_2\ m_3\ m_4\ m_5\ m_6$
m_3: $w_3\ w_1\ w_5\ w_2\ w_4\ w_6$	w_3: $m_6\ m_4\ m_2\ m_3\ m_1\ m_5$
m_4: $w_4\ w_3\ w_1\ w_2\ w_5\ w_6$	w_4: $m_2\ m_4\ m_1\ m_3\ m_5\ m_6$
m_5: $w_5\ w_1\ w_6\ w_2\ w_3\ w_4$	w_5: $m_3\ m_5\ m_1\ m_2\ m_4\ m_6$
m_6: $w_6\ w_3\ w_1\ w_2\ w_4\ w_5$	w_6: $m_5\ m_6\ m_1\ m_2\ m_3\ m_4$

It has six stable matchings, where μ_1 is the man-optimal stable matching and μ_6 is the woman-optimal stable matching:

$$\begin{aligned}
\mu_1 &: \{(m_1, w_1), (m_2, w_2), (m_3, w_3), (m_4, w_4), (m_5, w_5), (m_6, w_6)\}\\
\mu_2 &: \{(m_1, w_2), (m_2, w_3), (m_3, w_1), (m_4, w_4), (m_5, w_5), (m_6, w_6)\}\\
\mu_3 &: \{(m_1, w_2), (m_2, w_4), (m_3, w_1), (m_4, w_3), (m_5, w_5), (m_6, w_6)\}\\
\mu_4 &: \{(m_1, w_2), (m_2, w_3), (m_3, w_5), (m_4, w_4), (m_5, w_1), (m_6, w_6)\}\\
\mu_5 &: \{(m_1, w_2), (m_2, w_4), (m_3, w_5), (m_4, w_3), (m_5, w_1), (m_6, w_6)\}\\
\mu_6 &: \{(m_1, w_2), (m_2, w_4), (m_3, w_5), (m_4, w_1), (m_5, w_6), (m_6, w_3)\}
\end{aligned}$$

It has four rotations: $\rho_1 : ((m_1, w_1), (m_2, w_2), (m_3, w_3))$, $\rho_2 : ((m_2, w_3), (m_4, w_4))$, $\rho_3 : ((m_3, w_1), (m_5, w_5))$ $\rho_4 : ((m_4, w_3), (m_5, w_1), (m_6, w_6))$. Figure 1 shows the Hasse diagrams of $(\mathcal{M}(I), \preceq)$ and $(R(I), \leq)$. The correspondence between the stable matchings of I and the closed subsets of $R(I)$ are: μ_1 and $\emptyset$, μ_2 and $\{\rho_1\}$, μ_3 and $\{\rho_1, \rho_2\}$, μ_4 and $\{\rho_1, \rho_3\}$, μ_5 and $\{\rho_1, \rho_2, \rho_3\}$, μ_6 and $\{\rho_1, \rho_2, \rho_3, \rho_4\}$.

3 A New Characterization

In this section, we present a characterization of the generalized median stable matchings by describing the closed subsets of $R(I)$ they correspond to. Not only does the characterization provide an alternate way to compute the matchings, it also leads to some interesting insights that are not evident from their definitons.

Theorem 3. *For each $\rho \in R(I)$, let n_ρ denote the number of closed subsets of $R(I)$ that do not contain ρ. For $i = 1, \ldots, N$, $S(i) = \{\rho : n_\rho < i\}$ is a closed subset of $R(I)$, and α_i is the stable matching obtained by starting at μ_M and eliminating all the rotations in $S(i)$.*

Proof. By the definition of closed subsets, the function n_ρ is an increasing function; i.e., if ρ' precedes ρ in $R(I)$, $n_{\rho'} < n_\rho$. Therefore, if $\rho \in S(i)$, every rotation that precedes ρ also belongs to $S(i)$.

For $i = 1$ to N, let $S(\alpha_i)$ denote the closed subset of $R(I)$ that corresponds to α_i. The second part of the theorem states that $S(\alpha_i) = S(i)$. To prove it, we shall show that each man m satisfies the following condition: the set of rotations in $S(\alpha_i)$ that m appears in is exactly the set of rotations in $S(i)$ that m appears in. This is sufficent because if $S(\alpha_i) \neq S(i)$, there is a rotation $\rho \in (S(\alpha_i) - S(i)) \cup (S(\alpha_i) - S(i))$, and every man that appears in ρ will not satisfy the condition.

When m does not appear in any rotations in $R(I)$ (i.e., m's partner in μ_M is never replaced in any of the stable matchings of I), the above condition is clearly satisfied. So suppose m does appear in some rotations in $R(I)$: $\rho_1, \rho_2, \ldots, \rho_k$ where m appears with w_j in ρ_j. Additionally, suppose m prefers w_1 the most, followed by w_2, then w_3, etc. This means that $\rho_1, \rho_2, \ldots, \rho_k$ form a chain in the rotation poset because ρ_i must be eliminated before m can be matched to w_{i+1} for $i = 1, \ldots, k-1$. Let x_j denote the number of stable matchings that match m to w_j. By applying the definition of α_i, m's partner in α_i is w_{j^*} where j^* is the index that satisfies the inequality:

$$x_1 + x_2 + \ldots + x_{j^*-1} < i \leq x_1 + x_2 + \ldots + x_{j^*-1} + x_{j^*}. \tag{1}$$

In order for m to be matched to w_{j^*}, rotations $\rho_1, \ldots, \rho_{j^*-1}$ have to be eliminated but $\rho_{j^*}, \rho_{j^*+1}, \ldots, \rho_k$ are not; i.e., of the k rotations that m appears in, the first $j^* - 1$ are the only ones that lie in $S(\alpha_i)$.

Next, notice that if a stable matching corresponds to a closed subset of $R(I)$ that does not contain ρ_j, m must be matched to one of the following women: $w_1, w_2, \ldots, w_j$. This implies that $n_{\rho_j} = x_1 + x_2 + \ldots + x_j$. Applying inequality (1), $n_{\rho_j} < i$ if and only if $j < j^*$. Thus, $\rho_1, \ldots, \rho_{j^*-1}$ belong to $S(i)$ but $\rho_{j^*}, \rho_{j^*+1}, \ldots, \rho_k$ do not. We have now shown that, for every man m, the rotations that m appears in in $S(\alpha_i)$ are exactly the same ones in $S(i)$. Thus, α_i is the stable matching obtained by eliminating the rotations in $S(i)$. □

In the example in Section 2, $n_{\rho_1} = 1$, $n_{\rho_2} = n_{\rho_3} = 3$, and $n_{\rho_4} = 5$. Since $N = 6$, the median stable matching will consist of two stable matchings – the two that correspond to $S(3) = \{\rho_1\}$ and $S(4) = \{\rho_1, \rho_2, \rho_3\}$, which are μ_2 and μ_5

respectively. Next, we note some unexpected observations about the generalized median stable matchings.

1. *Location of the median stable matching.* Does an instance's median stable matching lie in the "middle" of its lattice of stable matchings or can it be somewhere else? Consider an instance I whose rotation poset consists of k rotations which are pairwise incomparable; i.e., $R(I)$ is an antichain of size k. Every subset of $R(I)$ is a closed subset so I has 2^k stable matchings. Moreover, for each rotation ρ, $n_\rho = 2^{k-1}$. Thus, for $1 \le i \le 2^{k-1}$, $S(i) = \emptyset$, and for $2^{k-1}+1 \le i \le 2^k$, $S(i) = R(I)$. In other words, the lower median stable matching of I is the man-optimal stable matching, which is the top element of $\mathcal{M}(I)$, while the upper median stable matching of I is the woman-optimal stable matching, which is the bottom element of $\mathcal{M}(I)$ – exactly the two stable matchings we least expected for the median stable matching! Indeed, it is not difficult to construct examples that show that the median stable matching can lie "anywhere" in the distributive lattice of stable matchings.
2. *Number of distinct generalized median stable matchings.* Teo and Sethuraman [15] had already observed that an instance's generalized median stable matchings need not be all distinct. In our example above, the instance has 2^k different stable matchings and, yet, its generalized median stable matchings consisted of only two types. In general, how many could there be? By Theorem 3, it is equal to the number of distinct $S(i)$ sets. But this number is simply one more than the number of distinct n_ρ values; the additional one accounts for the fact that none of the rotations belong to $S(1)$. Thus, if I has size n, it can have at most $n^2/2+1$ distinct types of generalized median stable matchings.
3. *Instances with isomorphic rotation posets.* Finally, Theorem 3 also implies that an instance's generalized median stable matchings are completely dependent on the structure of its rotation poset. To see this, consider two instances I and I' whose rotation posets are isomorphic. Let f be an isomorphism from $R(I)$ to $R(I')$. For each $\rho \in R(I)$, $n_\rho = n_{f(\rho)}$. Thus, if $S(i)$ is the closed subset of $R(I)$ that corresponds to the ith generalized median stable matching of I, then $f(S(i)) = \{f(\rho) : \rho \in S(i)\}$ is the closed subset of $R(I')$ that corresponds to the ith generalized median stable matching of I'. Interestingly, the median stable matching is the only "fair" stable matching we know of that has this property.

4 Finding the Generalized Median Stable Matchings

Suppose we find the ith generalized median stable matching of I using its definition, which we call the *direct* method. It requires (i) enumerating all the stable matchings of I to construct the multiset of stable partners for each man, (ii) sorting each man's multiset of stable partners from his most preferred to his least preferred woman, and (iii) matching each man to the ith woman in his sorted multiset of stable partners. Clearly, the bottleneck is in steps (i) and (ii); these steps take $O(n^2 + nN)$ time by using the enumeration algorithm in [6] for step (i) and bucket sort for step (ii).

Lemma 2. *Let I be a stable matching instance of size n with N stable matchings. When N is polynomial in n, the direct method for finding α_i runs in time polynomial in n, for $i = 1, \ldots, N$.*

Thus, when N is no longer polynomial in n, can α_i still be computed efficiently? We shall show next that there are indeed some easy cases but that, in general, the the answer is "no" unless P=NP. Theorem 3 will play a key role in the derivation of our results.

Theorem 4. *Let I be an SM instance of size n with N stable matchings. When $i = O(\log n)$, computing α_i and α_{N-i+1} can be done in polynomial time.*

Proof. Let $\rho \in R(I)$. Let $R(I)^\rho$ denote the poset obtained from $R(I)$ by deleting ρ and every rotation it precedes. Notice that $n_\rho = c(R(I)^\rho)$, the number of closed subsets of $R(I)^\rho$. Moreover, (i) when $R(I)^\rho$ has at least $i-1$ elements, $\rho \notin S(i)$ because $c(R(I)^\rho) \geq i$, and (ii) when $R(I)^\rho$ has at most $i-2$ elements, we can simply check which subsets of $R(I)^\rho$ are closed subsets to determine the exact value of $c(R(I)^\rho)$. Using these observations, it is not difficult to show that determining $S(i)$ exactly and then eliminating all its rotations from μ_M to construct α_i can be done in $O(i2^i n^4)$ time. Thus, if $i = O(\log n)$, finding α_i can be done in polynomial time. Since $\alpha_{N-i+1} = \beta_i$, the same result also holds for finding α_{N-i+1}. □

4.1 The Hardness Result

Suppose P and Q are posets on disjoint sets. Their *ordinal sum* is the poset $P +_O Q$ on the set $P \cup Q$ such that $x \leq y$ in $P +_O Q$ if (i) $x, y \in P$ and $x \leq y$, or (ii) $x, y \in Q$ and $x \leq y$, or (iii) $x \in P$ and $y \in Q$. The following is easy to establish:

Lemma 3. *Let P and Q be posets on disjoint sets and $c(\cdot)$ be a function that counts the number of closed subsets of a poset. Then $c(P +_O Q) = c(P) + c(Q) - 1$.*

Here is a technical lemma we need for our hardness result.

Lemma 4. *Let M be a positive integer. There is a poset Q_M so that $|Q_M| = O(\log^2 M)$, the number of edges in its Hasse diagram is $O(\log^3 M)$, and $c(Q_M) = M$.*

Proof. Consider the binary representation of M: $b_0 \cdot 2^0 + b_1 \cdot 2^1 + \ldots + b_k 2^k$. Let A_i and C_i denote an antichain of size i and a chain of size i, respectively. When $b_0 = 0$ (i.e., M is even), let $Q_M = A_{b_1 \times 1} +_O A_{b_2 \times 2} +_O \ldots +_O A_{b_k \times k} +_O C_{nz-1}$ where nz is the number of non-zero bits in the binary representation of M. Thus, $|Q_M| \leq 1 + 2 + \ldots + k + k = O(k^2)$, which is $O(\log^2 M)$; the number of edges in

Q_M's Hasse diagram is at most $1 \times 2 + 2 \times 3 + \ldots + (k-1) \times k + k + k - 1 = O(k^3)$, which is $O(\log^3 M)$. From Lemma 3, it is straightforward to check that $c(A_{b_1 \times 1} +_O A_{b_2 \times 2} +_O \ldots +_O A_{b_k \times k}) = M - (nz - 1)$ and $c(C_{nz-1}) = nz$, so $c(Q_M) = M - (nz - 1) + nz - 1 = M$. When $b_0 = 1$ (i.e., M is odd), let $Q_M = Q_{M-1} +_O C_1$. By applying the same analysis when $b_0 = 0$, Q_M satisfies the three properties in the lemma as well. □

Theorem 5. *Let I be an SM instance with N stable matchings. Suppose p and q are positive integers such that $p < q$. When $i = \lceil pN/q \rceil$, computing α_i and α_{N-i+1} is NP-hard.*

Proof. Since $\alpha_{N-i+1} = \beta_i$, it is sufficient to prove the theorem for α_i. Let us first consider the case when $p = 1$. Let `ComputeMedian-`q be an algorithm whose input is an SM instance I and whose output is I's $\lceil |\mathcal{M}(I)|/q \rceil$-generalized median stable matching and its corresponding closed subset. Its runtime is $f(|I|)$, where $|I|$ denotes the size of I. Let P be a poset. Recall that $c(P)$ denotes the number of closed subsets of P. The main idea behind our proof is that `ComputeMedian-`q can be used to answer queries of the form "Is $c(P) \leq m$?". We describe how this can be done next.

`Query(`P, m`)`

1. Construct the poset $P_0 = qP +_O \mathbf{x} +_O Q_{q(q-1)m+q-1}$ where qP denotes the ordinal sum of q copies of P, $\mathbf{x}$ is the singleton poset containing x and Q_i is the poset described in Lemma 4. Note that $c(P_0) = qc(P) + 2 + q(q-1)m + q - 1 - (q-1) - 2 = qc(P) + q(q-1)m$.
2. Construct an SM instance $I(P_0)$ whose rotation poset is isomorphic to P_0. Find the rotation ρ_x in $R(I(P_0))$ that corresponds to x in P_0.
3. Use `ComputeMedian-`q to find $I(P_0)$'s $\lceil |\mathcal{M}(I(P_0))|/q \rceil$-generalized median stable matching and its corresponding closed subset S.
4. If $\rho_x \in S$ return "yes"; else, return "no".

By the construction of P_0, $n_x = qc(P) - (q-1)$. Since $n_{\rho_x} = n_x$ and $S = S(c(P) + (q-1)m)$, $\rho_x \in S$ implies that $c(P) < m+1$ or $c(P) \leq m$ because $c(P)$ is an integer. Similarly, when $\rho_x \notin S$, $c(P) \geq m + 1$. The correctness of `Query` follows. It is straightforward to verify that since q is a constant, the sizes of P_0, $I(P_0)$, and $R(I(P_0))$ are all polynomial in $|P|$ and $\log m$. Thus, the runtime of `Query` is polynomial in $|P|$, $\log m$ and $f(|P| + \log m)$.

Now, $c(P) \leq 2^{|P|}$. To determine $c(P)$, we simply do a binary search over the range $[1, 2^{|P|}]$ using `Query` as a subroutine. Clearly, $O(|P|)$ queries are sufficient where the value of m in each query is at most $2^{|P|}$. If `Compute-Median-`q also runs in time polynomial in its input size, the algorithm we have just described computes $c(P)$ in time polynomial in $|P|$. In [13], it was shown, however, that computing the number of antichains of a poset is #P-complete. Since there is a one-to-one correspondence between the antichains of a poset and its closed subsets, computing $c(P)$ is also #P-complete. It follows that finding the $\lceil |\mathcal{M}(I)|/q \rceil$-generalized median stable matching of an instance I is NP-hard.

Let us now prove that the theorem is also true when $p > 1$. Without loss of generality, assume that p and q are relatively prime. Suppose ComputeMedian-(p, q) is an algorithm like ComputeMedian-q except that it computes the $\lceil p|\mathcal{M}(I)|/q\rceil$-generalized median stable matching of I and its corresponding closed subset in $g(|I|)$ time. This time we shall use ComputeMedian-(p, q) to find the $\lceil|\mathcal{M}(I)|/q\rceil$-generalized median stable matching of I.

```
ConstructMedian-$q$($I$)

1. Construct $R(I)$. Find positive integers $k$ and $r$ so that $pk = qr + 1$. (Since
   $p$ and $q$ are relatively prime, these integers exist. Also, $1 \leq k \leq q$ and
   $1 \leq r \leq p$.)
2. Create the poset $P_0 = R(I) +_O \mathbf{x} +_O R(I) +_O \mathbf{x} +_O R(I) +_O \ldots +_O \mathbf{x} +_O R(I)$
   so that there are $k$ copies of $R(I)$ and $k - 1$ copies of $\mathbf{x}$ in $P_0$. For each
   $\rho \in R(I)$, mark as $\rho^{(j)}$ the element that corresponds to $\rho$ in the $j$th copy of
   $R(I)$. Similarly, mark as $x^{(j)}$ the $j$th copy of $x$. Construct an SM instance
   $I(P_0)$ whose rotation poset is isomorphic to $P_0$.
3. Use ComputeMedian-$(p, q)$ to find the $\lceil p|\mathcal{M}(I(P_0))|/q\rceil$-generalized median
   stable matching of $I(P_0)$ and its corresponding closed subset $S$.
4. Let $S'$ consist of the rotations in $P_0$ that have corresponding rotations in $S$.
   (Find these rotations using the isomorphism from $P_0$ to the rotation poset
   of $I(P_0)$.) Let $S'' = \{\rho : \rho^{(r+1)} \in S'\}$.
5. Find the man-optimal matching of $I$, $\mu_M$. Create the $\lceil|\mathcal{M}(I)|/q\rceil$-generalized
   median stable matching of $I$ by eliminating $S''$ from $\mu_M$ and return the stable
   matching.
```

It is straightforward to verify the following: $c(P_0) = k \times c(R(I))$, $n_{x^{(j)}} = j \times c(R(I))$ for $1 \leq j \leq k-1$, and $n_{\rho^{(j)}} = (j-1)c(R(I)) + n_\rho$ for each $\rho \in R(I)$, $1 \leq j \leq k-1$. By our choice of k and r, we know that $pk/q = r + 1/q$. Thus, in step 3, $\lceil p|\mathcal{M}(I(P_0))|/q\rceil = \lceil pk \times c(R(I))/q\rceil = r \times c(R(I)) + \lceil c(R(I))/q\rceil$. Consequently, in step 4, $S' = \cup_{j=1}^{r}\{\rho^{(j)} : \rho \in R(I)\} \cup \{\rho^{(r+1)} : n_\rho < \lceil c(R(I))/q\rceil\} \cup \{x^{(j)} : j \leq r\}$. Thus, $S'' = \{\rho : n_\rho < \lceil c(R(I))/q\rceil\}$. Since S'' is the closed subset that corresponds to the $\lceil|\mathcal{M}(I)|/q\rceil$-generalized stable matching of I, step 5 outputs the correct stable matching.

Since p and q are constants, $|P_0| = O(|R(I)|)$. Also, $|I(P_0)| = O(|R(I)|^2)$ so $|I(P_0)| = O(|I|^4)$. It is easy to verify that steps 1, 2, 4 and 5 can be accomplished in time polynomial in $|I|$. If ComputeMedian-(p, q) runs in time polynomial in its input size, step 4 will also run in time that is polynomial in $|I|$; i.e., ConstructMedian-q is an efficient algorithm. But we just showed that finding the $\lceil|\mathcal{M}(I)|/q\rceil$-generalized stable matching of I is an NP-hard problem. It follows that computing $\lceil p|\mathcal{M}(I)|/q\rceil$-generalized stable matching of I when $p > 1$ is also NP-hard. □

5 Approximating the Median Stable Matching

While the median stable matching is arguably the most fair stable matching that has been proposed in the literature so far, our result in the previous section shows that finding it is computationally hard unless P=NP. A natural direction to take is to find a stable matching that is "close" to the median stable matching.

When N is an even number, the median stable matching of I consists of $\alpha_{N/2}$ and $\alpha_{(N+2)/2}$. Notice though that I may have stable matchings the lie *between* $\alpha_{N/2}$ and $\alpha_{(N+2)/2}$ in $\mathcal{M}(I)$. (Recall our example in Section 3.) These stable matchings have the property that at least one (but not all) men are matched to their upper median stable partners and at least one (but not all) women are matched to their lower median stable partners. If we say that $\alpha_{N/2}$ and $\alpha_{(N+2)/2}$ are fair because every individual is matched to his/her median stable partner, surely stable matchings that lie between these two matchings must also be fair. By the same reasoning, if we say that $\alpha_{i'}$ and $\alpha_{i''}$ are good approximations of the median stable matching where $i' < \lceil N/2 \rceil < i''$, every stable matching that lies between the two matchings must also be a good approximation of the median stable matching. This leads us to the following definition:

Definition 1. *Let I be an SM instance with N stable matchings. Let μ be one of its stable matchings and S_μ be its corresponding closed subset in $R(I)$. We say that μ is an ϵ-approximation of the median stable matching of I if μ lies between $\alpha_{\lfloor N/2 \rfloor - \epsilon}$ and $\alpha_{\lceil N/2 \rceil + \epsilon}$ in the lattice of stable matchings of I. That is, $S(\lfloor N/2 \rfloor - \epsilon) \subseteq S_\mu \subseteq S(\lceil N/2 \rceil + \epsilon)$.*

Since an SM instance of size n can have at most $2^{n^2/2}$ stable matchings, Theorem 4 implies that finding an $(N/2 - O(\log\log N))$-approximation to the median stable matching of an SM instance can be found efficiently. In the next theorem, we present a slight improvement over this result.

Theorem 6. *Let I be an SM instance of size n with N stable matchings. Finding a stable matching μ of I such that μ lies between $\alpha_{\lceil \log N/2 \rceil}$ and $\alpha_{N - \lfloor \log N/2 \rfloor}$ can be done in $O(n^2)$ time. That is, finding an $(N/2 - O(\log N))$-approximation to the median stable matching of an SM instance can be found efficiently.*

Proof. Suppose $R(I)$ has m rotations. Let $\rho_1, \rho_2, \ldots, \rho_m$ be a topological ordering of its rotations. Let $J_r = \{\rho_1, \rho_2, \ldots, \rho_{r-1}\}$ for $r = 1, \ldots, m+1$.

Claim: For $r = 1, \ldots, m+1$, J_r is a closed subset, and $S(r) \subseteq J_r \subseteq S(N-m+r)$.
Proof of Claim: Since the rotations are ordered topologically, when $\rho \in J_r$, every rotation that precedes ρ also belongs to J_r. That is, J_r is a closed subset.

Suppose ρ does not belong to J_r. Then $n_\rho \geq r$ because $J_1, J_2, \ldots, J_r$ are all closed subsets that do not contain ρ. Hence, $\rho \notin S(r)$. In other words, $S(r) \subseteq J_r$. On the other hand, when $\rho \in J_r$, there are at least $m - r + 1$ closed subsets that contain ρ: J_{i+1} for $i = r, \ldots, m$. Thus, $N - n_\rho \geq m - r + 1$ or $N - m + r - 1 \geq n_\rho$, so $\rho \in S(N - m + r)$.

When $r = \lceil m/2 \rceil$, $S(\lceil m/2 \rceil) \subseteq J_{\lceil m/2 \rceil} \subseteq S(N - \lfloor m/2 \rfloor)$. But because $R(I)$ has m rotations, $m + 1 \leq N \leq 2^m$. Hence, $m \geq \log N$ so $S(\lceil \log N/2 \rceil) \subseteq J_{\lceil m/2 \rceil} \subseteq S(N - \lfloor \log N/2 \rfloor)$. Thus, the stable matching of I that corresponds to the closed subset $J_{\lceil m/2 \rceil}$ lies between $\alpha_{\lceil \log N/2 \rceil}$ and $\alpha_{N - \lfloor \log N/2 \rfloor}$.

Now, constructing $R(I)$ and its accompanying digraph $G(I)$ takes $O(n^2)$ time. Topologically sorting its rotations also take $O(n^2)$ time. Finding μ_M, and then eliminating all rotations of $J_{\lceil m/2 \rceil}$ from μ_M also takes $O(n^2)$ time. Thus, finding the stable matching in the theorem takes $O(n^2)$ time. □

In contrast, we have the next result whose proof is like that of Theorem 5.

Theorem 7. *Let ϵ be a constant. Let I be an SM instance with N stable matchings. Finding a stable matching μ of I such that μ lies between $\alpha_{\lfloor N/2 \rfloor - \epsilon}$ and $\alpha_{\lceil N/2 \rceil + \epsilon}$ is NP-hard. That is, finding an $O(1)$-approximation to the median stable matching of an SM instance is NP-hard.*

Interestingly, this leaves us with the following problem:

Open Problem: Is there an efficient algorithm for finding an ϵ-approximation to the median stable matching of an SM instance where ϵ is $\omega(1)$ but at most $N/2 - \Omega(\log N)$?

References

1. Abdulkadiroglu, A., Pathak, P., Roth, A.: The New York City high school match. American Economic Review, Papers and Proceedings 95, 364–367 (2005)
2. Abdulkadiroglu, A., Pathak, P., Roth, A., Sönmez, T.: The Boston public school match. American Economic Review, Papers and Proceedings 95, 368–371 (2005)
3. Blair, C.: Every finite distributive lattice is a set of stable matchings. Journal of Combinatorial Theory A 37, 353–356 (1984)
4. Fleiner, T.: A fixed-point approach to stable matchings and some applications. Mathematics of Operations Research 28, 103–126 (2003)
5. Gale, D., Shapley, L.: College admissions and the stability of marriage. American Mathematical Monthly 69, 9–15 (1962)
6. Gusfield, D.: Three fast algorithms for four problems in stable marriage. SIAM Journal on Computing 16, 111–128 (1987)
7. Gusfield, D., Irving, R.: The Stable Marriage Problem: Structure and Algorithms. The MIT Press, Cambridge (1989)
8. Irving, R., Leather, P.: The complexity of counting stable marriages. SIAM Journal on Computing 15, 655–667 (1986)
9. Irving, R., Leather, P., Gusfield, D.: An efficient algorithm for the optimal stable marriage. Journal of the ACM 34, 532–544 (1987)
10. Klaus, B., Klijn, F.: Median stable matchings for college admissions. International Journal of Game Theory 34, 1–11 (2006)
11. Klaus, B., Klijn, F.: Procedurally fair and stable matching. Economic Theory 27, 431–447 (2006)
12. Knuth, D.: Mariages Stables. Les Presses de l'Université de Montréal (1976)
13. Provan, J., Ball, M.: The complexity of counting cuts and of computing the probability that a graph is connected. SIAM Journal on Computing 12, 777–788 (1983)
14. Roth, A., Peranson, E.: The redesign of the matching market of American physicians: Some engineering aspects of economic design. American Economic Review 89, 748–780 (1999)
15. Teo, C.-P., Sethuraman, J.: The geometry of fractional stable matchings and its applications. Mathematics of Operations Research 23, 874–891 (1998)

Stateless Near Optimal Flow Control with Poly-logarithmic Convergence

Baruch Awerbuch[1,⋆] and Rohit Khandekar[2]

[1] Johns Hopkins University
baruch@cs.jhu.edu
[2] IBM T.J.Watson Research Center
rkhandekar@gmail.com

Abstract. We design completely local, stateless, and self-stabilizing flow control mechanism to be executed by "greedy" agents associated with individual flow paths. Our mechanism is very natural and can be described in a single line:

> If a path has many "congested" edges, decrease the flow on the path by a small multiplicative factor, otherwise increase its flow by a small multiplicative factor.

The mechanism does not require any initialization or coordination between the agents. We show that starting from an *arbitrary* feasible flow, the mechanism always maintains feasibility and reaches, after poly-logarithmic number of rounds, a $1 + \epsilon$ approximation of the maximum throughput multicommodity flow. Moreover, the total number of rounds in which the solution is *not* $1 + \epsilon$ approximate is also poly-logarithmic. Previous distributed solutions in our model either required a state since they used a primal-dual approach or had very slow (polynomial) convergence.

1 Introduction

The goal of this paper is to optimize resource allocation in a decentralized network architecture where different applications compete for shared network resources in a "greedy" manner, without explicit coordination with each other, while being subjected to some regulatory constraints that limit their behavior.

In this paper, we focus on the Flow Control version of the multi-commodity flow problem in a distributed environment. The essence of Flow Control is to decide how much flow of a commodity is *admitted* (rest is *rejected*), assuming infinite flow demand for each commodity, and assuming routing is pre-determined to go over a single path. Flow control is used by TCP and is considered a classical problem in the theory of networking, with numerous articles dedicated to this topic.

The flow control problem is a variant of the multicommodity flow problem in a directed capacitated graph, with a collection of *commodities*, each characterized by the following: source (where the flow is originated), sink (where the flow ends up), benefit (the utility of this flow), and a fixed path from source and the sink that must be used

⋆ Partially supported by NSF grants CCF 0515080, ANIR-0240551, CCR-0311795, and CNS-0617883.

by this commodity. The collection of all the flows must satisfy capacity constraints, namely the total flow on each edge cannot exceed its capacity.

Intuitively, it is clear that the flow control needs to be accomplished by requiring congested paths to decrease their flows and non-congested paths to increase their flows. However, there is an inherent instability problem, in this concurrent decision making environment — some paths may fluctuate between being congested or non-congested.

Stateless algorithms. "Statelessness" is often a desirable feature of an optimization in a distributed environment [3]. In a stateless and "local" solution, one desires that the flows make their routing decisions in a cooperative but uncoordinated manner, without having access to a global clock and without being able to properly initialize and synchronize their individual executions. The flows only observe the current network congestion, without being able to pin-point the individual contributions of other commodities, and without keeping any memory about the past. As pointed out in [3], statelessness implies a number of other features desirable in networks and distributed systems with unreliable components. *Self-stabilization:* it is a classical notion in the theory of robust distributed systems [8,13,9,6,5] means that the solution can withstand adversarial sequence of "hard reset" events, namely, crashes accompanied with loss of all memory contents. *Incremental and local adjustment:* Even if changes occur in the network topology or demand pattern, the algorithm does not need to be restarted. Rather, the algorithm adjusts the flows in a local and incremental manner, without disrupting the flows that are not affected. *No global clock:* Algorithms should not be driven by a global clock; however we assume that local clocks generate perfectly synchronized (but un-numbered) *rounds*, without maintaining a global time.

2 Greedy Stateless Maximum Benefit Flow Framework and Our Results

Consider a directed capacitated graph $G = (V, E, c)$ where $c : E \to R^+$ is the capacity assignment to edges. We consider the set $\mathcal{P}$ of commodities, each associated with a path p along which it flows and benefit b_p per unit of flow. We identify the commodity with its path p. Let $f(p)$ denote the flow of commodity p. For each edge e, total flow on e is $f(e) = \sum_{p:e\in p} f(p)$. Given a flow f, the *load* of an edge, maximum network load, and total network benefit (flow value) are respectively

$$\ell_f(e) = \frac{f(e)}{c(e)}, \qquad |\ell_f| = \max_{e\in E} \ell_f(e), \qquad \text{and} \qquad U(f) = \sum_{p\in\mathcal{P}} b_p \cdot f(p).$$

The objective is to compute a flow assignment f to commodities satisfying the capacity constraint: $|\ell_f| \le 1$, and *maximizing* the throughput $U(f)$.

Greedy framework. In this paper, we focus on a specific framework for stateless flow control, that we call a "greedy" framework. We imagine an "agent" associated with each commodity p. The agent has benefit b_p associated with sending per unit flow. The restrictions on the agents are as follows:

- *Penalty:* The network imposes at all times a cost on an agent p determined by a "penalty" function $cost(p) = \sum_{e \in p} \phi'_e(\ell_f(e)) = \sum_{p:e \in p} \phi'_e(f(p)/c(e))$ where $\phi_e(\ell_f)$ is a certain function of the congestion on edge e, and ϕ'_e is its derivative.
- *Inertia:* if $cost(p)$ is within $1 \pm \alpha$ factor of its benefit b_p, then the agent p cannot change its flow $f(p)$,
- *Speed limit:* the agent p can only modify (increase or decrease) a β fraction of its flow, namely can change its flow by $(1 \pm \beta)$.

Assuming that the agents act *greedily* and *selfishly* and try to maximize their profit, an agent p will maximally increase the flow if it is *significantly profitable*, namely $cost(p) < (1 - \alpha)b_p$, and will maximally decrease the flow if it is *significantly unprofitable*, i.e., $cost(p) > (1+\alpha)b_p$. Our flow control mechanism specifies cost function ϕ', the inertia parameter α and speed limit β.

Theorem 1. *Our mechanism that is presented in Fig. 2 guarantees that starting from an* arbitrary *feasible flow, the flow always remains feasible, and reaches a $1+\epsilon$ approximate solution in time upper-bounded by*

$$\tilde{O}\left(\frac{\log k \cdot \log^7(mCB)}{\epsilon^6}\right).$$

Moreover, the total number of rounds in which the solution is not $1 + \epsilon$ *approximate is also* $\tilde{O}\left(\frac{\log k \cdot \log^{O(1)}(mCB)}{\epsilon^{O(1)}}\right)$.

Here k and m denote the number of commodities and edges in the network respectively, C is the ratio of maximum to minimum edge capacity, and B is the ratio of maximum to minimum benefit. $\tilde{O}$ hides $\log(\frac{\ln m}{\epsilon})$ factors. We remark that we have not attempted to get the best possible powers of $\log m$ and ϵ in the convergence time. The emphasis is on a stateless distributed solution with poly-logarithmic convergence.

2.1 Previous Work

Stateful algorithms. In centralized or distributed setting, efficient "primal-dual" algorithms in the stateful model have been widely studied for network flows [11,10,18,4,2]. Most of these algorithms share features like exponential dual variables with our algorithms. However, these algorithms crucially depend on maintaining a state, e.g., proper initialization or some global information about the current solution, and perform globally optimum updates in each round. Many of these algorithms initialize the flows to zero. Thus they have to be restarted whenever the instance changes due to change in the network or commodities; and do not satisfy the *incremental* and *local adjustment* property. The packing/covering LP algorithm of Plotkin-Shmoys-Tardos [17] or the multi-commodity flow algorithm of Awerbuch and Khandekar [3] fall short of being stateless since they have to keep track of the maximum violation in a constraint or the global maximum congestion of the current solution. The algorithm of [3] converges in time *linear* in the maximum path-length.

Stateless	Distributed	Problem [citation]	Convergence time
no	no	packing/covering LP [17,11,10,18,14]	$m \cdot [\log(m)/\epsilon]^{O(1)}$
no	yes	packing/covering LP [16,7,18,15]	$[\log(mn)/\epsilon]^{O(1)}$
no	yes	multi-comm. flow routing [4,3]	$L \cdot [\log(m \cdot C)/\epsilon]^{O(1)}$
yes	yes	multi-comm. flow control [12]	$B \cdot [\log(C)/\epsilon]^{O(1)}$
yes	yes	bipartite load balancing [1]	$[\log(m)/\epsilon]^{O(1)}$
yes	yes	multi-comm. flow control [*this paper*]	$[\log(m \cdot CB)/\epsilon]^{O(1)}$

Fig. 1. A comparison of some combinatorial $(1+\epsilon)$-approximation algorithms for multicommodity flows. Here n denotes the number of variables and L denotes the maximum-path-length.

Stateless algorithms. Garg and Young [12] presented a stateless flow control algorithm. While their algorithm resembles ours in the case of flow control, the convergence time of their algorithm depends *linearly* on the ratio B of the maximum and minimum benefit of the flows. This linear dependence is inherent to their algorithm due to a severe limit on how much flow of a commodity can increase in a single round. Their algorithm is based on the packet drop-rates at various routers/links. Recently, Awerbuch-Azar-Khandekar [1] presented a stateless algorithm for a special case of load balancing in bipartite graphs. Their algorithm and techniques, which do not use any exponential duals, do not appear to generalize to arbitrary LPs and hence new techniques are required.

3 The Mechanism

By scaling, we assume without loss of generality, that $\max_p b_p = 1$, $\min_p b_p = 1/B$, $\min_e c(e) = 1$, and $\max_e c(e) = C$. The algorithm to be executed by an agent controlling the flow on path p is given in Figure 2. The main procedure executed by each agent is given in Figure 2a. It calls procedures ROUTEMETRIC, CONTROL, FLOWUPDATE in each round. In the rest of the paper, we use round and time interchangeably.

The ROUTEMETRIC **procedure.** Procedure ROUTEMETRIC in Figure 2b sets the most basic parameters. Let $f(e) = \sum_{p:e\in p} f(p)$ be the current flow and $\ell_f(e) = \frac{f(e)}{c(e)}$ be the congestion of an edge e. We define $\Phi_e(f(e)) = c(e)\cdot\exp[\mu(\frac{f(e)}{c(e)} - 1)]$ to be a "penalty" function where $\mu = \epsilon^{-1}\ln(mCB)$ and its derivative $\Phi'_e(f(e)) = \mu\cdot\exp[\mu(\frac{f(e)}{c(e)} - 1)]$ to be the "cost"of edge e. The cost of a path is simply the total cost of the edges on that path.

The CONTROL **procedure.** Procedure CONTROL in Fig. 2c sets the relevant parameters of flow control, namely of the amount of flow that one decreases or increases on a link.

We choose $\alpha = \frac{1}{4\mu}$. Intuitively, this is the accuracy of our cost comparisons, costs within $1+\alpha$ factor will be considered essentially equal. Note that if the flow of a certain commodity over an edge is small (e.g., zero), the multiplicative increase by factor $1+\beta$ is ineffective. To bootstrap the increase in the flow, we allow an additive increase $\ddot{f}$ in the flow. The parameters β and $\ddot{f}(e)$ are defined so that the cost $\Phi'(e)$ of any edge e

```
Repeat in each round:
  1. Input (read) $f(p)$ and $f(e)$ for each $e \in p$
  2. Call RouteMetric, Control, FlowUpdate
```

(a) Procedure Main for commodity p.

```
Procedure RouteMetric
  1. Let $\mu \leftarrow \epsilon^{-1} \ln(mCB)$.
  2. Define $\Phi_e(f(e)) = c(e) \cdot \exp[\mu(\frac{f(e)}{c(e)} - 1)]$ for all $e \in p$
  3. Define $\Phi'(p) = \sum_{e \in p} \Phi'_e(f(e)) = \sum_{e \in p} \mu \cdot \exp[\mu(\frac{f(e)}{c(e)} - 1)]$
```

(b) Procedure RouteMetric for commodity p.

```
Procedure Control
  1. Define parameters: $\alpha \leftarrow \frac{1}{4\mu}$ ; $\beta \leftarrow \Theta(\alpha \cdot \frac{1}{\mu})$ and $\ddot{f}(e) \leftarrow \frac{1}{\mu} \cdot \frac{c(e)}{k} \cdot \ln(1 + \frac{\alpha}{8})$
```

(c) Procedure Control for commodity p.

```
Procedure FlowUpdate
  1. if $\Phi'(p) < b_p \cdot (1 - \alpha)$, then $f(p) \leftarrow \max\{f(p)(1 + \beta), \min_{e \in p} \ddot{f}(e)\}$.
  2. if $\Phi'(p) > b_p \cdot (1 + \alpha)$, then $f(p) \leftarrow f(p)(1 - \beta)$.
```

(d) Procedure FlowUpdate for commodity p.

Fig. 2. The maximum benefit flow mechanism

changes by a factor of at most $\frac{\alpha}{4}$ in any round. Note that the flow $f(e)$ increases in one round by at most $\beta f(e) + k\ddot{f}(e)$. Here k denotes the number of agents or paths $|\mathcal{P}|$. Note that in the procedure FlowUpdate, the flow on e is increased only when $f(e) < c(e)$. Thus to ensure that the cost of e does not increase by more than a $1 + \frac{\alpha}{4}$ factor, it is enough to make sure that $\exp[\mu(\beta c(e) + k\ddot{f}(e))/c(e)] \leq 1 + \frac{\alpha}{4}$. We in fact set β and $\ddot{f}(e)$ so that $\exp[\mu \cdot \beta] \leq (1 + \frac{\alpha}{4})/(1 + \frac{\alpha}{8})$ and $\exp[\mu \cdot (k\ddot{f}(e)/c(e))] \leq 1 + \frac{\alpha}{8}$.

We set $\beta = \Theta(\alpha \cdot \frac{1}{\mu})$. We set $\ddot{f}(e) = \frac{1}{\mu} \cdot \frac{c(e)}{k} \cdot \ln(1 + \frac{\alpha}{8})$. For all k commodities, the total additive increase in the flow is at most $c(e) \cdot \frac{1}{\mu} \cdot \ln(1 + \frac{\alpha}{8})$ yielding a multiplicative increase in the cost of at most $1 + \frac{\alpha}{8}$.

The FlowUpdate **procedure.** The crux of our algorithm lies in procedure FlowUpdate (Fig 2d). Each agent p locally compares the cost $\Phi'(p)$ under the current "routing metric" with its benefit b_p. If the cost is significantly lower, i.e., less than $b_p \cdot (1 - \alpha)$, it increases the flow by a factor $1 + \beta$. In case the current flow on p is *very* small, the flow is instead increased *additively* by an amount $\min_{e \in p} \ddot{f}(e)$. If on the other hand, the cost is significantly higher, i.e., more than $b_p \cdot (1+\alpha)$, the flow is decreased by a factor of $1-\beta$. If neither of the above conditions hold, the flow on p is kept unchanged.

We note that the initial values of the flows are completely arbitrary, as long as they satisfy the capacity constraints; there is absolutely no coordination between flows in terms of how quickly they act, except that there is at most one action of each flow in each round.

4 Analysis

Notations. Let $U = \sum_{p \in \mathcal{P}} f_p \cdot b_p$ be the overall flow in the network at any given point. Let $\ell_f(e) = f(e)/c(e)$ and $|\ell_f| = \max_e \ell(e)$. Let $\Phi(e) = \Phi_e(f(e)), \Phi'(e) = \Phi'_e(f(e))$, and $\Phi''(e) = \Phi''_e(f(e))$. Let $\Phi(p) = \sum_{e \in p} \Phi(e)$, $\Phi'(p) = \sum_{e \in p} \Phi'(e)$, $\Phi = \sum_{e \in E} \Phi(e)$, and $\Phi' = \sum_{e \in E} \Phi'(e)$. Let $g(p)$ denote the *optimum* flow on path $p \in \mathcal{P}$, let $g(e)$ denote the optimum flow on edge $e \in E$. Let $\Gamma = \sum_{e \in E} f(e) \cdot \Phi'(e) = \sum_{p \in \mathcal{P}} f(p) \cdot \Phi'(p)$ be the cost of the entire flow f. Let $\Lambda = \sum_{e \in E} g(e) \cdot \Phi'(e) = \sum_{p \in \mathcal{P}} g(p) \cdot \Phi'(p)$ be the cost of the optimum flow under the current cost metric. Let $\Gamma(e) = f(e) \cdot \Phi'(e)$ and $\Lambda(e) = g(e) \cdot \Phi'(e)$. Let $\Gamma'(e) = f(e) \cdot \Phi''(e) + \Phi'(e)$. For a path $p \in \mathcal{P}$, $\Lambda'(p) = \sum_{e \in p} \Lambda'(e) = \sum_{e \in p} \Phi''(f(e)) \cdot g(e)$ be the derivative of Λ w.r.t. the flow $f(p)$ on path p. Similarly let $\Psi'(p) = b_p - \sum_{e \in p} \Phi'(e)$ be the derivative of Ψ w.r.t. the flow $f(p)$ on path p.

Definition 1. *We introduce the auxiliary potential function: $\Psi = U - \Phi$.*

Definition 2. *We call the network*

- unsaturated *if $\Phi' < \frac{1}{B} \cdot (1-\alpha)$, i.e., the overall cost of the edges is small.*
- saturated *if $\Phi' \geq \frac{1}{B} \cdot (1-\alpha)$, i.e., the overall cost of the edges is large.*
- cheap *if $\Gamma < U \cdot (1-2\alpha)$, i.e., the average cost of a path is small.*
- reasonable *if $U \cdot (1-2\alpha) \leq \Gamma \leq U \cdot (1+2\alpha)$, i.e., the average cost of a path is about right.*
- expensive *if $\Gamma > U \cdot (1+2\alpha)$, i.e., the average cost of a path is large.*

4.1 Preliminary Observations

The following lemma states how well a convex function is approximated by its first order linear approximation.

Lemma 1. *For a differentiable convex function $\Upsilon : \Re^k \to \Re$, for any $x, y \in \Re^k$ we have*

$$\Upsilon'(x) \cdot (y-x) \leq \Upsilon(y) - \Upsilon(x) \leq \Upsilon'(y) \cdot (y-x)$$

where $\Upsilon'(\cdot)$ denotes the gradient evaluated at a given point.

Lemma 2. *The potential Ψ does not decrease during the course of the algorithm.*

Proof. The negative potential $-\Psi$ is a differentiable convex function of the flow vector $f = (f(p))_{p\in\mathcal{P}}$. Let f_0 and Ψ_0 (resp. f_1 and Ψ_1) denote the values of f and Ψ in the beginning (resp. in the end) of a round. Using Lemma 1, we conclude that

$$\begin{aligned}\Psi_1 - \Psi_0 = -\Psi_0 - (-\Psi_1) &\geq -\Psi'(f_1)\cdot(f_0 - f_1)\\ &= \sum_{p\in\mathcal{P}}(f_1(p) - f_0(p))\cdot(b_p - \Phi'_{f_1}(p))\end{aligned} \tag{1}$$

where $\Phi'_{f_1}(p)$ denotes the cost of p under flow f_1. Since the cost of a path does not increase by a factor more than $1+\frac{\alpha}{4}$ in a single round, we conclude that $\Phi'_{f_1}(p) \leq (1+\frac{\alpha}{4})\Phi'_{f_0}(p)$. From the algorithm we conclude that $f_1(p) - f_0(p) > 0$ implies $b_p - \Phi'_{f_1}(p) > b_p - \Phi'_{f_0}(p)/(1-\alpha) > 0$ and $f_1(p) - f_0(p) < 0$ implies $b_p - \Phi'_{f_1}(p) < b_p - \Phi'_{f_0}(p)/(1+\alpha) < 0$. Combining these observations, we conclude that the potential Ψ does not decrease. □

Lemma 3. *The flow f always remains feasible, i.e., at all times $f(e) \leq c(e)$ and $\Phi(e) \leq c(e)$ for all edges e.*

Proof. We prove that $\Phi(e) \leq c(e)$ by induction on the number of rounds. Initially, since the flow satisfies the capacity constraints, we have $f(e) \leq c(e)$. Consider any round in which the flow $f(e)$ (or equivalently $\Phi'(e)$) increases, which is possible only when $f(p)$ increases for some path $p \ni e$. Since $f(p)$ is increased only when $\Phi'(p) < b_p(1-\alpha)$, we conclude that $\Phi'(e) < b_p(1-\alpha)$ in the beginning of this round. Since the cost $\Phi'(e)$ increases by at most a factor of $1+\frac{\alpha}{4}$ in any single round, the cost $\Phi'(e)$, whenever it increases, increases to at most $b_p \leq 1$. Since $\Phi(e) = \frac{c(e)}{\mu}\Phi'(e) < c(e)\cdot\Phi'(e)$, we conclude that $\Phi(e) \leq c(e)$ after the round. Thus the induction is complete and $\Phi(e) \leq c(e)$ always holds. From the definition of $\Phi(e)$, this implies that $f(e) \leq c(e)$ also holds always. □

Lemma 4. *If the network is unsaturated, then in $O(\tau_0)$ rounds it becomes saturated where*

$$\tau_0 = \max_{e\in E}\log_{1+\beta}\frac{c(e)}{\ddot{f}(e)} = \tilde{O}\left(\frac{\log(k\mu)}{\beta}\right). \tag{2}$$

Furthermore, once the network becomes saturated, it always remains saturated.

Proof. While the network is unsaturated, the cost of any path p satisfies $\Phi'(p) \leq \Phi' < \frac{1}{B}\cdot(1-\alpha) \leq b_p\cdot(1-\alpha)$. Thus all the flows increase by a factor of $(1+\beta)$. Since after one round, the flow on any edge e is at least $\ddot{f}(e)$ and it never exceeds $c(e)$, the network has to become saturated in $O(\tau_0)$ rounds.

Now assume that the network becomes unsaturated again. Consider the round during which the network changes from being saturated to being unsaturated. In this round, the flow on some path p must decrease. Just before decrease, it must be true that $\Phi' \geq \Phi'(p) > b_p\cdot(1+\alpha)$. However since single round reduces Φ' by at most $\frac{\alpha}{4}$ factor, we have $\Phi' > b_p\cdot(1+\alpha)/(1+\frac{\alpha}{4}) > \frac{1}{B}$ after the round. This is a contradiction. Thus we conclude that the network never becomes unsaturated again. □

The proofs of following two lemmas are omitted due to lack of space.

Lemma 5. *At all times,* $(1-2\epsilon)\cdot\left(1-\frac{1}{mCB}\right)\cdot|\ell_f|\cdot\Phi\cdot\mu \le \Gamma \le |\ell_f|\cdot\Phi\cdot\mu$.

Lemma 6. *If the network is saturated, then* $|\ell_f| \ge 1-2\epsilon$.

Theorem 2. *Suppose that the network is either* cheap *or* expensive. *Then, in these cases* Ψ *increases by at least* $\Delta\Psi = \Omega(\alpha\beta)\cdot(U+\Gamma)$ *in a single round.*

Proof. From equation (1), we recall that the increase in Ψ in a single round is

$$\Delta\Psi = \Psi_1 - \Psi_0 \ge \sum_{p\in\mathcal{P}}(f_1(p)-f_0(p))\cdot(b_p - \Phi'_{f_1}(p))$$

where the subscripts 0 and 1 indicate the values in the beginning and at the end of a round respectively. Denote $\mathcal{P}_x = \{p\in\mathcal{P} \mid \Phi'_{f_0}(p) \ge x\cdot b_p\}$ and $\mathcal{P}^y = \{p\in\mathcal{P} \mid \Phi'_{f_0}(p) \le y\cdot b_p\}$ in the beginning of a round.

For every cheap path $p\in\mathcal{P}^{(1-\alpha)}$, we have $f_1(p)-f_0(p)\ge\beta f_0(p)$. For every expensive path $p\in\mathcal{P}_{(1+\alpha)}$, we have $f_1(p)-f_0(p)\le-\beta f_0(p)$. Therefore

$$\begin{aligned}
\Delta\Psi &\ge \sum_{p\in\mathcal{P}^{(1-\alpha)}\cup\mathcal{P}_{(1+\alpha)}}(f_1(p)-f_0(p))\cdot(b_p-\Phi'_{f_1}(p))\\
&\ge \sum_{p\in\mathcal{P}^{(1-\alpha)}}\beta f_0(p)\cdot\left(b_p-\Phi'_{f_0}(p)\cdot\left(1+\frac{\alpha}{4}\right)\right)\\
&\qquad+\sum_{p\in\mathcal{P}_{(1+\alpha)}}-\beta f_0(p)\cdot\left(b_p-\Phi'_{f_0}(p)/\left(1+\frac{\alpha}{4}\right)\right) \qquad (3)\\
&\ge \frac{1}{1-\alpha}\sum_{p\in\mathcal{P}^{(1-\alpha)}}\beta f_0(p)\cdot((1-\alpha)b_p-\Phi'_{f_0}(p))\\
&\qquad+\frac{1}{1+\alpha}\sum_{p\in\mathcal{P}_{(1+\alpha)}}-\beta f_0(p)\cdot((1+\alpha)b_p-\Phi'_{f_0}(p)). \qquad (4)
\end{aligned}$$

The inequality (3) follows from the fact that the cost of any edge (or path) changes in one round by at most $1+\frac{\alpha}{4}$ factor. Note that both the terms in (4) are non-negative.

Cheap network: $\Gamma<(1-2\alpha)\cdot U$. From the definition of $\mathcal{P}^{(1-\alpha)}$, the 1st term in (4) is at least $\frac{1}{1-\alpha}\sum_{p\in\mathcal{P}}\beta f_0(p)\cdot((1-\alpha)b_p-\Phi'_{f_0}(p)) = \frac{\beta}{1-\alpha}((1-\alpha)U-\Gamma)\ge\frac{\alpha\beta}{1-\alpha}\cdot U\ge \Omega(\alpha\beta)\cdot\Gamma$.

Expensive network: $\Gamma>(1+2\alpha)\cdot U$. From the definition of $\mathcal{P}_{(1+\alpha)}$, the 2nd term in (4) is at least $\frac{1}{1+\alpha}\sum_{p\in\mathcal{P}}-\beta f_0(p)\cdot((1+\alpha)b_p-\Phi'_{f_0}(p)) = \frac{-\beta}{1+\alpha}((1+\alpha)U-\Gamma)\ge \Omega(\alpha\beta)\cdot\Gamma\ge\Omega(\alpha\beta)\cdot U$.

Putting things together, the proof is complete. □

Definition 3 (Reasonable interval). *We call an interval* $\mathcal{T}=[t_0,t_1]$ *of rounds* reasonable *if the network is reasonable at each round* $t\in\mathcal{T}$. *Otherwise, we call the interval* unreasonable.

4.2 Mileage Definitions

We define the *offline mileage* $\nabla^{\Lambda}(e,t)$, the *derivative mileage* $\nabla^{\Phi'}(e,t)$, and benefit mileage $\nabla^{U}(t)$, as the absolute value of the change, that takes place in round t, in the offline potential, derivative cost function, and flow on edge e respectively. The second parameter t indicates that these values are in round t.

$$\nabla^{\Lambda}(e,t) = |\Lambda(e,t) - \Lambda(e,t-1)|, \qquad \nabla^{\Phi'}(e,t) = |\Phi'(e,t) - \Phi'(e,t-1)|.$$

For an interval $\mathcal{T} = [t_0, t_1]$ of rounds, we define $\nabla^{\Lambda}(e,\mathcal{T}) = \sum_{t\in\mathcal{T}} \nabla^{\Lambda}(e,t)$ and $\nabla^{\Phi'}(P,\mathcal{T}) = \sum_{e\in P}\sum_{t\in\mathcal{T}} \nabla^{\Phi'}(e,t)$. We also define total mileage for the whole network: $\nabla^{\Lambda}(t) = \sum_{e\in E} \nabla^{\Lambda}(e,t)$ and $\nabla^{U}(t) = |U(t) - U(t-1)|$. Also $\nabla^{\Lambda}(\mathcal{T}) = \sum_{t\in\mathcal{T}} \nabla^{\Lambda}(t)$ and $\nabla^{U}(\mathcal{T}) = \sum_{t\in\mathcal{T}} \nabla^{U}(t)$.

Theorem 3 lower bounds the increase $\Delta\Psi$ in the potential Ψ in an interval of rounds in terms of mileages of offline potential and flow respectively. This key theorem is proved in Section 5.

Theorem 3 (Potential Increase). *In any interval $\mathcal{T} = [t_0, t_1]$ the increase in potential is at least $\Delta\Psi > \Omega(\nabla^{\Lambda}(\mathcal{T}) \cdot \frac{\alpha}{\mu})$ and $\Delta\Psi > \Omega(\nabla^{U}(\mathcal{T}) \cdot \alpha)$.*

Definition 4 (Stationary interval). *We call an interval of time $\mathcal{T} = [t_0, t_1]$* stationary *if the mileage for offline potential and flow are small as compared to the flow volume at time t_0. Specifically, $\nabla^{\Lambda}(\mathcal{T}) \leq \frac{\alpha^2}{64} \cdot U(t_0)$, and $\nabla^{U}(\mathcal{T}) \leq \frac{\beta}{2} \cdot U(t_0)$. Otherwise, we call the interval* unstationary.

The corollary below follows directly from Theorem 3 and Definition 4.

Corollary 1. *Once the network is saturated, each* unstationary *interval $\mathcal{T} = [t_0, t_1]$, leads to a large increase in the potential, i.e., $\Delta\Psi > \Omega(\min\{\frac{\alpha^3}{\mu}, \alpha\cdot\beta\}) \cdot U(t_0)$.*

4.3 Main Theorems

Let U^* be the optimal flow value. The following theorem states that any sufficiently long stationary interval leads to near optimality. This key theorem is proved in Section 6.

Theorem 4 (Optimality). *Assume that the network is saturated. Consider a reasonable and stationary interval $\mathcal{T} = [t_0, t_1]$ of length at least τ_0 as in (2). At some round $t \in \mathcal{T}$, we get near optimality: $U(t) \cdot (1 + O(\beta + \epsilon)) \geq U^*$.*

Corollary 2 (Main). *After $O(\tau_\ddagger = \tau_0 \cdot \tau_\dagger)$ time, where τ_0 is as in (2) and*

$$\tau_\dagger = O(\epsilon \cdot \mu^4 \cdot \ln(mC)) = O(\epsilon \cdot \mu^5), \tag{5}$$

or, after $\tau_\ddagger = O(\epsilon \cdot \mu^7 \cdot \ln(k\mu)) = \tilde{O}\left(\frac{\log k \cdot \log^7(mCB)}{\epsilon^6}\right)$ time, the flow becomes near optimal, namely $U \cdot (1 + O(\beta + \epsilon)) \geq U^*$. Moreover, the total number of rounds in which the solution is not *$1 + O(\beta + \epsilon)$ approximate is also $\tilde{O}\left(\frac{\log k \cdot \log^{O(1)}(mCB)}{\epsilon^{O(1)}}\right)$.*

The proof of the above corollary follows from Corollary 1, Theorem 4, and the fact that the potential is bounded above by $m \cdot C$. The proof is omitted due to lack of space.

5 Potential Increase Proof of Theorem 3

Theorem 5 (Offline mileage). *If the offline mileage in an interval $\mathcal{T}$ is $\nabla^{\Lambda}(\mathcal{T})$, then Ψ increases in $\mathcal{T}$ by at least $\Omega(\nabla^{\Lambda}(\mathcal{T}) \cdot \frac{\alpha}{\mu})$.*

Proof. To simplify the intuition behind the proof, assume that the augmentation or withdrawal of flows on different paths happens *sequentially* (see the remark at the end of the proof).

Consider δ units of flow sent on path p. The offline mileage ∇^{Λ} due to this can be upper bounded by the increase $\Delta\Psi$ in the potential due to this as follows.

$$\frac{1}{\delta} \cdot \nabla^{\Lambda} = \Lambda'(p) = \sum_{e \in p} \Lambda'(f(e)) = \sum_{e \in p} \Phi''_e(f(e)) \cdot g(e) = \sum_{e \in p} \Phi'_e(f(e)) \cdot \frac{\mu}{c(e)} \cdot g(e)$$
$$\leq \sum_{e \in p} \Phi'_e(f(e)) \cdot \mu = \Phi'(p) \cdot \mu \leq O\left(\frac{\mu}{\alpha} \cdot \Psi'(p)\right) = O\left(\frac{1}{\delta} \cdot \frac{\mu}{\alpha} \cdot \Delta\Psi\right).$$

The first inequality holds since $g(e) \leq c(e)$. The second inequality above follows from the fact that $\Psi'(p) = b_p - \Phi'(p)$ and that the flow on p is increased only if $\Phi'(p) < (1 - \alpha) \cdot b_p$, which in turn implies that $\Psi'(p) \geq \frac{1}{1-\alpha}\Phi'(p) - \Phi'(p) = \Omega(\alpha) \cdot \Phi'(p)$. A similar argument holds when δ units of flow is reduced on path p when $\Phi'(p) > (1 + \alpha) \cdot b_p$. The overall offline mileage in a round is at most the sum of such mileage contributions over all the paths. Since each unit of offline mileage contributes to an increase of $\Omega(\frac{\alpha}{\mu})$ units of Ψ, the proof is complete.

Remark: The proof also works without the assumption of sequential execution, since the cost $\Phi'(f(e))$ of any edge changes by a factor of at most $1 + \frac{\alpha}{4}$ in a single round and $\Psi'(p) \geq \Omega(\alpha) \cdot \Phi'(p)$ still holds. □

Theorem 6 (Benefit mileage). *If the benefit mileage in an interval $\mathcal{T}$ is $\nabla^{U}(\mathcal{T})$, then Ψ increases in $\mathcal{T}$ by at least $\Omega(\nabla^{U}(\mathcal{T}) \cdot \alpha)$.*

Proof. Note that $\Psi'(p) = b_p - \Phi'(p)$ denotes the increase in Ψ per unit flow increase in $f(p)$. Observe that flow $f(p)$ is increased only along paths with $\Phi'(p) < (1 - \alpha) \cdot b_p$ and is decreased only along paths p with $\Phi'(p) > (1 + \alpha) \cdot b_p$. Thus, the net contribution of "positive" terms (namely, b_p for $\Delta f(p) > 0$ and $-\Phi'(p)$ for $\Delta f(p) < 0$) is at least $1 + \Omega(\alpha)$ factor higher that the net contribution of "negative" terms (namely, $-\Phi'(p)$ for $\Delta f(p) > 0$ and b_p for $\Delta f(p) < 0$). The net effect is at least α fraction of either change in the positive or negative terms. Thus any unit change in $f(p)$ contributes to $\alpha \cdot b_p$ increase in Ψ. Since the overall benefit mileage is at most the contributions of mileages of all the paths, the proof is complete. □

6 Optimality Proof of Theorem 4

Suppose the network is saturated and the interval $\mathcal{T} = [t_0, t_1]$ of length at least τ_0 is both reasonable and stationary.

Lemma 7. *For all $t \in \mathcal{T}$, we have $\frac{\Gamma(t) - U(t)}{\Gamma(t)} \leq \frac{1}{2\mu}$.*

Proof. If $\Gamma(t) \leq U(t)$, then the above inequality trivially holds. On the other hand, since the network is *not* expensive at time t, we have $\Gamma(t) \leq U(t) \cdot (1+2\alpha)$. Therefore $\Gamma(t) - U(t) \leq 2\alpha \cdot U(t) \leq 2\alpha \cdot \Gamma(t)$. Since $\alpha = \frac{1}{4\mu}$, the lemma follows. □

Recall that g denotes the optimum flow that achieves the maximum throughput U^*. Let $h = g \cdot \frac{U(t_0)}{U^*}$ be the scaled optimum flow that achieves the throughput $U(t_0)$, the throughput of the solution f at time t_0. It is easy to see that if $|\ell_f| \leq (1+O(\epsilon)) \cdot |\ell_h|$ at time t_0, where $|\ell_h|$ is the maximum edge load under flow h, then we have near optimality: $U(t_0) \geq (1-O(\epsilon))U^*$.

Let $\hat{\Lambda}(t) = \sum_{e\in E} h(e) \cdot \Phi'(e) = \sum_{p\in\mathcal{P}} h(p) \cdot \Phi'(p)$ be the cost of flow h under the current cost metric. The next theorem shows that if $\Gamma(t)$ is a good approximation of $\hat{\Lambda}(t)$, then we have near optimality.

Theorem 7. *If at some time $t \in \mathcal{T}$, we have $\frac{\Gamma(t)-\hat{\Lambda}(t)}{\Gamma(t)} \leq \frac{7}{8\mu}$, then we have near optimality: $U(t) \cdot (1+O(\beta+\epsilon)) \geq U^*$.*

Proof. Let $\Phi(h) = \sum_{e\in E} c(e) \cdot \exp[\mu(\frac{h(e)}{c(e)} - 1)]$ be the potential of h. From Lemma 1, we know that $\Phi(f(t)) - \Phi(h) \leq \Phi'(f(t)) \cdot (f(t) - h) = \Gamma(t) - \hat{\Lambda}(t)$. Since $\Gamma(t) \leq \Phi(t) \cdot \mu$, we conclude

$$\frac{\Phi(f(t)) - \Phi(h)}{\Phi(f(t))} \leq \frac{\Gamma(t) - \hat{\Lambda}(t)}{\Gamma(t)/\mu} \leq \frac{7}{8\mu} \cdot \mu = \frac{7}{8}.$$

This combined with the definition of Φ in turn implies that we have achieved additive ϵ approximation: $|\ell_f| \leq |\ell_h| + \epsilon$. Since the network is saturated, which implies $|\ell_f| \geq 1 - 2\epsilon$, we have multiplicative approximation $|\ell_f| \leq (1+O(\epsilon)) \cdot |\ell_h|$.

Recall that the flow h has throughput $U(t_0)$, the throughput at the beginning of the interval. Since the benefit mileage is small: $\nabla^U(\mathcal{T}) \leq \frac{\beta}{2} \cdot U(t_0)$, we have $U(t) \geq (1-\frac{\beta}{2}) \cdot U(t_0)$. Putting all things together we have near optimality: $U(t) \geq (1-O(\epsilon+\beta)) \cdot U^*$. □

In light of Lemma 7 and Theorem 7, we can assume, without loss of generality, that for all $t \in \mathcal{T}$, $\frac{\hat{\Lambda}(t)-U(t)}{\Gamma(t)} = \frac{\hat{\Lambda}(t)-\Gamma(t)}{\Gamma(t)} + \frac{\Gamma(t)-U(t)}{\Gamma(t)} \leq \frac{-7}{8\mu} + \frac{1}{2\mu} = \frac{-3}{8\mu}$. Since the network is reasonable, $\Gamma(t) \geq (1-2\alpha) \cdot U(t)$. Therefore after simplifying, we get $\hat{\Lambda}(t) \leq U(t) \cdot \left(1 - \frac{3\alpha}{2} + 3\alpha^2\right)$. Since low benefit mileage implies $U(t) \leq U(t_0) \cdot (1+\frac{\beta}{2})$, we conclude that the average cost of the flow path in h is small: $\frac{\hat{\Lambda}(t)}{U(t_0)} \leq \left(1 - \frac{3\alpha}{2} + 3\alpha^2\right)\left(1+\frac{\beta}{2}\right)$.

Since $h = g \cdot \frac{U(t_0)}{U^*}$, we have that the mileage of h in the interval $\mathcal{T}$ (defined similarly and denoted by $\nabla^{\hat{\Lambda}}(\mathcal{T})$) is small: $\nabla^{\hat{\Lambda}}(\mathcal{T}) \leq \frac{\alpha^2}{64} \cdot U(t_0)$.

Lemma 8. *Consider a probability space $(\Omega, \pi : \Omega \rightarrow \Re^+)$ and two non-negative random variables χ and ψ with expectations $\bar{\chi}$ and $\bar{\psi}$ respectively. Then, for every $\kappa < 1$ there exists $\omega^* \in \Omega$ such that $\chi(\omega^*) \leq (1+\kappa) \cdot \bar{\chi}$ and $\psi(\omega^*) \leq \frac{2}{\kappa} \cdot \bar{\psi}$.*

The above lemma applied to the set of paths $\mathcal{P}$ with a probability function π given by $\pi(p) = h(p) \cdot b_p/U(t_0)$, the two random variables given by $\chi(p) = \Phi'(p, t_0)/b_p$ and $\psi(p) = \nabla^{\Phi'}(p)/b_p$, and $\kappa = \alpha/4$, we get the following theorem.

Theorem 8 (Anchor theorem). *There exists a path* $\mathbf{D} \in \mathcal{P}$ *(referred to as "anchor" path) such that*

$$\frac{\Phi'(\mathbf{D}, t_0)}{b_{\mathbf{D}}} \le (1 + \frac{\alpha}{4}) \cdot \frac{\hat{\Lambda}(t_0)}{U(t_0)} \quad \textit{and} \quad \frac{\nabla^{\Phi'}(\mathbf{D}, \mathcal{T})}{b_{\mathbf{D}}} \le \frac{8}{\alpha} \cdot \frac{\nabla^{\Lambda}(\mathcal{T})}{U(t_0)}.$$

Using the above theorem, we get $\frac{\Phi'(\mathbf{D},t_0)}{b_{\mathbf{D}}} \le \left(1 + \frac{\alpha}{4}\right) \cdot \left(1 - \frac{3\alpha}{2} + 3\alpha^2\right) \left(1 + \frac{\beta}{2}\right)$ and $\frac{\nabla^{\Phi'}(\mathbf{D},\mathcal{T})}{b_{\mathbf{D}}} \le \frac{8}{\alpha} \cdot \frac{\alpha^2}{64} = \frac{\alpha}{8}$. Therefore we conclude that the cost $\Phi'(\mathbf{D})$ at any time $t \in \mathcal{T}$ is at most

$$\begin{aligned}\Phi'(\mathbf{D}, t) &\le b_{\mathbf{D}} \cdot (\Phi'(\mathbf{D}, t_0) + \nabla^{\Phi'}(\mathbf{D}, \mathcal{T})) \\ &\le b_{\mathbf{D}} \cdot \left[\left(1 - \frac{5\alpha}{4} + O(\alpha^2)\right) \cdot (1 + O(\alpha^2)) + \frac{\alpha}{8}\right] < b_{\mathbf{D}} \cdot (1 - \alpha).\end{aligned}$$

This leads to a contradiction as follows. Since the cost of the anchor path $\mathbf{D}$ is *consistently* lower than $b_{\mathbf{D}} \cdot (1-\alpha)$, its flow $f(\mathbf{D})$ is increased by a factor $1+\beta$ in every single round. Thus after τ_0 rounds, the flow on some edge in $\mathbf{D}$ must exceed its capacity. This is a contradiction and thus the proof of Theorem 4 is complete. □

References

1. Awerbuch, B., Azar, Y., Khandekar, R.: Fast load balancing via bounded best response. In: SODA (2008)
2. Awerbuch, B., Khandekar, R.: Distributed network monitoring and multicommodity flows:primal-dual approach. In: PODC (2007)
3. Awerbuch, B., Khandekar, R.: Greedy distributed optimization of multi-commodity flows. In: PODC (2007)
4. Awerbuch, B., Khandekar, R., Rao, S.: Distributed algorithms for multicommodity flow problems via approximate steepest descent framework. In: SODA (2007)
5. Awerbuch, B., Patt-Shamir, B., Varghese, G.: Self-stabilization by local checking and correction. In: FOCS (1991)
6. Awerbuch, B., Varghese, G.: Distributed program checking: a paradigm for building self-stabilizing distributed protocols. In: FOCS (1991)
7. Bartal, Y., Byers, J.W., Raz, D.: Global optimization using local information with applications to flow control. In: FOCS (1997)
8. Dijkstra, E.: Self stabilizing systems in spite of distributed control. CACM 17, 643–644 (1974)
9. Dolev, S., Israeli, A., Moran, S.: Self-stabilization of dynamic systems assuming only read/write atomicity. In: PODC (1990)
10. Fleischer, L.: Approximating fractional multicommodity flow independent of the number of commodities. SIAM Journal on Discrete Mathematics 13, 505–520 (2000)
11. Garg, N., Könemann, J.: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In: FOCS (1998)
12. Garg, N., Young, N.E.: On-line, end-to-end congestion control. In: FOCS (2002)
13. Gouda, M.G., Multari, N.J.: Stabilizing communication protocols. Technical Report TR-90-20, Dept. of Computer Science, University of Texas at Austin (June 1990)

14. Koufogiannakis, C., Young, N.E.: Beating simplex for fractional packing and covering linear programs. In: FOCS (2007)
15. Kuhn, F.: The price of locality: exploring the complexity of distributed coordination primitives. PhD Thesis, ETH Zurich, Diss. ETH No. 16213, (December 2005)
16. Luby, M., Nisan, N.: A parallel approximation algorithm for positive linear programming. In: STOC (1993)
17. Plotkin, S., Shmoys, D., Tardos, E.: Fast approximation algorithms for fractional packing and covering problems. Math of Oper. Research 20(2), 257–301 (1994)
18. Young, N.E.: Sequential and parallel algorithms for mixed packing and covering. In: FOCS (2001)

The Least-Unpopularity-Factor and Least-Unpopularity-Margin Criteria for Matching Problems with One-Sided Preferences

Richard Matthew McCutchen

Department of Computer Science, University of Maryland, College Park, MD 20742
rmccutch@umd.edu

Abstract. We consider the problem of choosing the best matching of people to positions based on preferences expressed by the people, for which many different optimality criteria have been proposed. A matching is *popular* if no other matching beats it in a majority vote of the people. The popularity criterion has a manipulation-resistance property, but unfortunately, some sets of preferences admit no popular matching. In this paper, we introduce the *least-unpopularity-factor* and *least-unpopularity-margin* criteria, two generalizations of popularity that preserve the manipulation-resistance property but give an optimal matching for every set of preferences. Under each of these generalizations, we show that the "badness" of a given matching can be calculated efficiently but it is NP-hard to find an optimal matching.

Keywords: matching, one-sided preferences, algorithms, NP-hardness, popular matching, voting.

1 Introduction

One of the most common administrative tasks that many organizations perform is assigning people to positions of some kind based on preferences expressed by the people, the positions, or both. For example, the University of Maryland Department of Computer Science assigns teaching assistants to classes according to the teaching assistants' preferences. The National Resident Matching Program (see [10]) assigns residents to hospitals based on the preferences of both residents and hospitals. Gale and Shapley [4] even suggested that students could be assigned to colleges by a central organization that observes the preferences of both sides.

All of these organizations face the problem of choosing a matching of people to positions that gives fair consideration to everyone's preferences. In this paper, we consider the problem in which the people express preferences for the positions, but not vice versa: the preferences are *one-sided*. An *instance* of this problem consists of a set of people, a set of positions, and a *preference list* for each person giving his preference ordering of the subset of the positions that he would be willing to occupy; these orderings may contain ties. The problem is to find the matching that is best overall in light of the preferences. In a valid matching, each person p is either matched to a position on her preference list or left without a position, which we represent by matching her to a *last resort* position $LR(p)$ available only to her as her last choice.

What makes this problem interesting is that it is not clear what it means for a matching to be "best": since it is almost never possible to give everyone her first choice simultaneously, any matching will inevitably please some people at the expense of others. If we can change a matching to make someone better off without making anyone else worse off (a *Pareto improvement*), of course we should do so, but there are still many *Pareto efficient* matchings that admit no such improvement. To decide among these, we need an *optimality criterion* that, for each instance, designates one or more matchings as optimal. Then, to apply the criterion in practice, we need an efficient algorithm to compute an optimal matching for a given instance. Ideally, our criterion should be "fair" in some sense and should resist attempts by the people to obtain better positions by lying about their preferences.

Many different optimality criteria have been proposed, studied, and used. Some are based on minimizing functions of the rank numbers that the people give to the positions they receive, but such criteria tend to be easy to manipulate. For example, MIT once assigned incoming students to residence halls by minimizing the sum of the cubes of the rank numbers, and a student could improve her chances of being assigned to her true first-choice residence hall by inserting other highly desirable residence halls near the top of her preference list [9].

One criterion that does not use rank numbers and is therefore less susceptible to this kind of manipulation is *popularity.* A matching M is *popular* if no other matching N beats it by majority vote, where each person votes for the matching that gives him the position he likes better or abstains if he likes the two positions equally well. For example, matching M in Figure 1 is not popular because N beats it by majority vote (B and C outvote A in favor of N), while N is popular. (We display an instance as a table that gives the rank number (– if unwilling) for each person (row) and non-last-resort position (column) and a matching by parenthesizing the rank numbers of the matched pairs.)

Abraham et al. [2] gave an efficient algorithm to compute a popular matching for a given instance when one exists. Unfortunately, some instances have no popular matching because of nontransitivity in the voting. For example, no matching for I_2 is popular because we can obtain a matching that beats it by a vote of 2 to 1 by promoting the occupant of y to x and the occupant of x to w and demoting the occupant of w to y.

$$M = \begin{array}{c} \\ A \\ B \\ C \end{array}\!\!\begin{array}{c} \begin{array}{ccc} w & x & y \end{array} \\ \begin{pmatrix} (\mathbf{1}) & - & 2 \\ 1 & (\mathbf{2}) & - \\ - & 1 & (\mathbf{2}) \end{pmatrix} \end{array}, \qquad N = \begin{array}{c} \\ A \\ B \\ C \end{array}\!\!\begin{array}{c} \begin{array}{ccc} w & x & y \end{array} \\ \begin{pmatrix} 1 & - & (\mathbf{2}) \\ (\mathbf{1}) & 2 & - \\ - & (\mathbf{1}) & 2 \end{pmatrix} \end{array}, \qquad I_2 = \begin{array}{c} \\ A \\ B \\ C \end{array}\!\!\begin{array}{c} \begin{array}{ccc} w & x & y \end{array} \\ \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} \end{array}$$

Fig. 1. Example instances and matchings

In general, when no matching meets the standard of popularity, we would still like to choose a merely "poor" matching over a "terrible" one. To this end, we will propose two numerical measures of a matching's "badness": its *unpopularity factor* and its *unpopularity margin*. The *least-unpopularity-factor* and *least-unpopularity-margin* optimality criteria, respectively, minimize these two measures. We will show that a given matching's unpopularity factor and margin can be calculated efficiently using algorithms based on shortest paths and minimum-cost flow, respectively. However, surprisingly, it is possible to encode the structure of a 3-satisfiability instance in the people's preferences so as to prove that finding an optimal matching under either criterion is NP-hard; we will present this rather unusual reduction in detail.

The inability to find optimal matchings severely limits the immediate practical applicability of these two criteria. Nevertheless, by ruling them out, this paper makes a step toward the goal of finding a criterion that has all the desired properties. Such a criterion would make it possible for organizations to solve their matching problems easily, fairly, and objectively by computer.

2 Related Work

The notion of popularity was first proposed by Gärdenfors [5] in the context of two-sided preferences. More recently, Abraham et al. [2] discussed it for one-sided preferences and gave the first polynomial-time algorithm to find popular matchings. On an instance with n people and a total of m preference-list entries, the algorithm runs in $O(m\sqrt{n})$ if there are ties and $O(n+m)$ if not. Abraham and Kavitha suggested at the end of [3] that, when no popular matching exists, a matching could be chosen based on the graph induced by the "beats by majority vote" relation among matchings; in this paper, we take a different approach by generalizing the "beats" relation itself.

A matching M is *rank-maximal* if it has the lexicographically maximum tuple $(n_1, n_2, \ldots)$, where n_i is the number of people assigned to positions they respectively ranked ith. Irving et al. [7] found an algorithm to compute a rank-maximal matching in $O(\min(n+C, C\sqrt{n})m)$ time, where C is the worst numerical rank to which any person is assigned in the result. Unfortunately, just as MIT's criterion induced people to artificially increase the rank numbers of the positions they desire, the rank-maximality criterion induces people to *decrease* the rank numbers (by omitting positions they are unlikely to receive) because it gives lower rank numbers priority over higher ones. The least-unpopularity criteria, which do not consider rank numbers at all, may be fairer.

If each person specifies utilities for the positions instead of ranking them, the most natural rule, known as *weighted matching* [8], is to maximize the total utility of all pairs. The most commonly used criterion for instances with two-sided preferences (we call the two sides applicants and recruiters) is *stability*. A matching is *stable* unless some currently unpaired applicant and recruiter both prefer each other to their current partners. Gale and Shapley [4] proved that every instance has a stable matching and gave an algorithm to find one.

3 The Unpopularity Factor

Definition 3.1. *If M and N are two matchings for the same instance, N* dominates *M* by a factor of *u/v, where u is the number of people who strictly prefer N to M and v is the number of people who strictly prefer M to N. The* unpopularity factor *of a matching M is the maximum factor by which it is dominated by any other matching (ignoring matchings that give $u = v = 0$).*

Note that a matching N dominates a matching M by a factor of ∞ if and only if N is a Pareto improvement over M; thus M has a finite unpopularity factor if and only if it is Pareto efficient. Furthermore, a matching is popular as defined by Abraham et al. [2] if and only if its unpopularity factor is at most 1.

The *least unpopularity factor* of an instance is the minimum unpopularity factor of all of its matchings, and the matching(s) that achieve this minimum are considered optimal. Thus, different instances have different numbers of optimal matchings, but every instance has at least one.

Matching M from the Introduction has unpopularity factor 2 because N dominates it by a factor of 2. However, N has unpopularity factor 1: the only person who might favor a different matching is A, and to promote A we must demote B, achieving a dominance factor of 1. In I_2, the six matchings that fill all three positions all have unpopularity factor 2 because the people in x and y can improve at the expense of the person in w; these six matchings are optimal.

To determine the unpopularity factor of a matching directly from the definition, we would have to consider all possible alternative matchings and calculate the factor by which each dominates M, which would take exponential time. Fortunately, there is an efficient way to calculate the unpopularity factor using the concept of *pressures*, which we will develop next.

3.1 Differences Between Matchings: Reassignments, Paths and Cycles

We can think of an instance as a bipartite graph whose vertex sets are the people and the positions and whose edges are the preference-list entries. A matching of the instance is then just a matching of the graph that uses all the people (because every person has a position, though it might just be her last resort).

In what ways can two matchings M and N differ? Their symmetric difference, which we will denote $M \oplus N$, consists of vertex-disjoint paths and/or cycles that are alternating in the sense that their edges come alternately from M and N. Furthermore, the alternating paths stop at positions, because if a path stopped at a person, she would lack a position in one matching, which is not allowed. We can think of an alternating path as a sequence of reassignments: one person moves to a different position, ejecting its original occupant; the occupant takes another position, ejecting *its* occupant; and so forth until someone takes a previously unoccupied position. Each reassignment may constitute a *promotion* or *demotion* of the reassigned person according to his preferences. An alternating cycle is similar.

Conversely, we say that a path or cycle X is *applicable* to a matching M if $M \oplus X$ is a valid matching. If so, *applying* X to M gives $M \oplus X$; X represents the change from M to $M \oplus X$.

With this background, we can define pressures.

3.2 Pressures

Definition 3.2. *Let M be a Pareto efficient matching. The* pressure *of a filled position p in M is the largest k for which there exists an alternating path applicable to M that promotes k people without demoting anyone and then ends with the demotion of the occupant of p to her last resort. Note that the demotion by itself constitutes such a path for $k = 0$. (The term "pressure" comes from the idea of k people stacked up behind p, wishing that its occupant will leave so they can all become better off.)*

Theorem 3.3. *The unpopularity factor of a Pareto efficient matching M is the greatest pressure of any of its filled positions.*

Proof. Recall that the unpopularity factor of M is the greatest pressure by which any other matching dominates M. To establish that the two maxima are equal, we'll show that each is at least as great as the other, i.e., (a) if M has a position p of pressure k, then there exists a matching N that dominates M by a factor of at least k and (b) if a matching N dominates M by a factor of f, then M has a position of pressure at least $\lceil f \rceil$.

(a): Simply let N be the result of applying to M the path that determines the pressure of p. The path promotes k people and demotes one person, so N dominates M by a factor of k.

(b): Let u and v be the total number of people better and worse off in N than in M, so that $f = u/v$. First modify N so that everyone who is worse off in N than in M is demoted all the way to his last resort in N; this does not change u or v. Decompose $M \oplus N$ into a collection of paths and cycles $X_1, \ldots, X_c$, discarding those that neither promote nor demote anyone. Each X_i must demote at least one person so that it is not a Pareto improvement over M. That person is demoted to his last resort, and since there is no one else to leave the last resort, the demotion must be the last reassignment in X_i; thus X_i is a path (not a cycle). A path has only one last reassignment, so each X_i demotes *exactly* one person to his last resort, and $c = v$.

For each i, let u_i be the number of people promoted by X_i; of course, $\sum_i u_i = u$. Choose an i such that $u_i \geq \lceil u/v \rceil = \lceil f \rceil$; by the Pigeonhole Principle, one must exist. Let p be the position whose occupant is demoted by X_i. Observe that X_i has exactly the form considered in Definition 3.2 for the pressure of p, and it makes at least $\lceil f \rceil$ people better off; thus the pressure of p is at least $\lceil f \rceil$.

Corollary 3.4. *The unpopularity factor of a matching, if finite, is an integer.*

3.3 Computing the Unpopularity Factor

We can reduce computation of the pressures of a matching to a shortest path problem. Let n' and n be the numbers of people and positions, and let m be the total number of entries in the preference lists.

Algorithm 3.5. *In $O(m\sqrt{n})$ time, determines whether a matching M is Pareto efficient and, if so, finds the pressure of each filled position in M.*

Method. Construct a graph G whose vertices are the positions of M. A pair of vertices (p_1, p_2) is connected in G by an edge of length -1 if p_1 is filled by a person who strictly prefers p_2, an edge of length 0 if p_1 is filled by a person indifferent to p_2, or no edge otherwise. Run Goldberg's shortest-path algorithm [6] on G, using all positions as sources so that the algorithm finds the shortest path ending at each position. If Goldberg's algorithm finds a negative cycle in G or a negative-length path ending at an unfilled position, conclude that M is Pareto inefficient. Otherwise, the pressure of each position is the negative of the length of the shortest path ending at it.

A path or cycle X in G represents a sequence of reassignments that demotes no one and is applicable to M after the ending position (in the case of a path) is vacated; furthermore, the length of X in G is the negative of the number of people it promotes. In light of this, Pareto-improving cycles and paths applicable to M correspond to negative cycles and negative-length paths ending at unfilled positions, respectively, in G. Thus, the algorithm correctly determines whether M is Pareto efficient. If it is, then paths in G ending at a position p represent paths applicable to M of the form considered in Definition 3.2, with the negative of a path's length corresponding to k in that definition. Thus, the negative of the shortest length of a path ending at p gives the pressure of p, as desired.

Each preference-list entry of the instance accounts for at most one edge of G, so G has at most m edges. The running time is dominated by that of Goldberg's algorithm, which is $O(m\sqrt{n})$ since edge lengths are at least -1.

To find the unpopularity factor of a matching, we use this algorithm to find the pressures and then simply take the highest pressure. This algorithm can be seen as a generalization of the algorithm given by Abraham et al. [1] to determine in $O(m)$ time only whether M is Pareto efficient; both algorithms are based on the same graph G.

4 The Unpopularity Margin

The unpopularity margin of a matching is defined the same way as the unpopularity factor, except we subtract the numbers of votes instead of dividing them:

Definition 4.1. *If M and N are two matchings for the same instance, N* dominates *M* by a margin of *$u - v$, where u is the number of people who strictly prefer N to M and v is the number of people who strictly prefer M to N. The* unpopularity margin *of a matching M is the maximum margin by which it is dominated by any other matching.*

The unpopularity margin of a matching M is an integer; it is 0 if M is popular (because M dominates itself by a margin of 0) or otherwise positive. We can reduce calculating the unpopularity margin to a min-cost max-flow problem. Since we use integer edge capacities, we assume that edge flows are also integers.

Algorithm 4.2. *Finds the unpopularity margin u of a matching M in $O((u+1)m\sqrt{n'+n})$ time.*

Method. Construct a flow graph G having as vertices a source, a sink, and the people and positions of M. Add an edge of unit capacity and zero cost from the source to each person and from each position to the sink. For each preference-list entry submitted by a person A for a position p, add a unit-capacity edge from A to p whose cost is -1, 0 or 1 as A likes p better than, the same as, or worse than her position in M.

A max-flow of G must put one unit of flow through each person, and those units must reach the sink via different positions, so the max-flows of G correspond exactly to the possible matchings of the instance. Furthermore, the cost of a max-flow is the negative of the margin by which the corresponding matching dominates M. Thus, we find the min-cost max-flow by starting from the max-flow representing M itself and augmenting negative cycles found using Goldberg's shortest-path algorithm [6]. The negative of the cost of this flow gives the unpopularity margin of M.

To find each negative cycle, we run Goldberg's algorithm on a graph with $n'+n+2$ vertices and m edges, taking $O(m\sqrt{n'+n})$ time since edge lengths are at least -1. Each cycle decreases the cost by at least 1 until we reach cost $-u$, so we find at most u cycles and then perform one more failed search for a cycle. The running time bound follows.

5 NP-Hardness of Finding Least-Unpopularity Matchings

We now use a reduction from 3-satisfiability (3SAT) to prove that it is NP-hard to find the least unpopularity factor or margin of a given set of preferences; it happens that the same reduction works for both problems. Abraham et al. [2] analyze preference sets with no ties separately from the general case of ties. We have had no reason to make this distinction so far, but the reduction will always generate preference lists with no ties in order to prove that even the no-ties versions of the problems are NP-hard.

The reduction converts an instance S of 3SAT to a polynomial-size preference set P and an *ideal* unpopularity factor. We will show that any tuple of truth values that satisfies S can be converted to a matching of P whose unpopularity factor does not exceed the ideal value, and vice versa; thus, the least unpopularity factor of P is at most the ideal value if and only if S is satisfiable. Therefore, computing the least unpopularity factor of P is NP-hard because an algorithm to do that could be used to determine whether S is satisfiable. This paragraph applies equally to unpopularity margins, and henceforth "unpopularity" will refer to either the factor or the margin.

5.1 Overview of the Reduction Design

The reduction, like most, builds P from *gadgets* that represent pieces of S. Each gadget will contain some *internal* people and positions and some *linking people*; there will also be *linking positions* that do not belong to any gadget. An internal person is willing to occupy only internal positions of her own gadget, but a linking person is also open to exactly one linking position, which is always her first choice. Any reassignment of the occupant of a linking position is a demotion, which (from the perspective of voting) could just as well be to his last resort as into a gadget; thus, the dominance that can be achieved by replacing him depends only on the identity and state of the gadget providing the replacement, not on the states of any other gadgets. In other words, gadgets are isolated from one another unpopularity-wise; their only interactions are in *which gadget gets to fill each linking position.* Thus, we can analyze each gadget's contribution to the unpopularity of the matching separately as a function of which linking positions the gadget gets.

Motivated by this idea, we introduce three types of gadget, each of which is designed to enforce a certain constraint on which linking positions it must get by producing a low unpopularity if the constraint is satisfied or a higher one if it is not. To represent S, we start with a set of key linking positions representing its variables; the choice of which gadget gets each of these positions represents a tuple of truth values for the variables. We then add gadgets so that satisfaction of all of the gadget constraints is equivalent to satisfaction of S, and we let the ideal unpopularity of P be the low unpopularity that would result if every gadget's constraint were satisfied. Note that the unpopularity factor of the matching is the highest pressure produced by any gadget, while the unpopularity margins of separate gadgets roughly add.

5.2 The Gadgets

A *box* consists of four internal positions, three internal people (i_1, i_2, and i_3), and three linking people (w, n_1, and n_2). Figure 2(a) shows its structure, including the linking positions. w is known as the *wide* person, and n_1 and n_2 are the *narrow* people. A box is satisfied, and produces a pressure of 2 and a margin of 1, if either the wide person or both narrow people get their linking positions; however, if both the wide person and at least one narrow person are denied their linking positions, a pressure of 3 and a margin of 2 result.

A *peg* (Figure 2(b)) consists of one internal position available to three linking people, all of whom prefer the same linking position. Its purpose is very simple: to always produce a pressure of 2 on l and provide a way to replace its occupant at margin 1.

A *pool* (Figure 2(c)) consists of two internal positions and three linking people. If k of the people are denied their linking positions, the pool has one linking position with a pressure of k and can replace its occupant at a margin of $\max(k-1, 0)$. We want to use the pool to distinguish between two and three people being denied linking positions. To this end, we attach a peg to each linking position;

then, all positions have pressure 2 and replacement margin 1, except when all three positions are taken by people from other gadgets, one of them develops a pressure of 3 and a replacement margin of 2.

$$\begin{array}{c|ccccccc} & x & y & z & u & l_w & l_{n1} & l_{n2} \\ \hline w & 2 & 3 & 5 & 4 & 1 & - & - \\ i_1 & 1 & 2 & 3 & 4 & - & - & - \\ i_2 & 1 & 2 & 3 & 4 & - & - & - \\ i_3 & 1 & 2 & 3 & 4 & - & - & - \\ n_1 & - & - & - & 2 & - & 1 & - \\ n_2 & - & - & - & 2 & - & - & 1 \end{array}$$

(a)

$$\begin{array}{c|cc} & x & l \\ \hline f_1 & 2 & 1 \\ f_2 & 2 & 1 \\ f_3 & 2 & 1 \end{array}$$

(b)

$$\begin{array}{c|ccccc} & x & y & l_{f1} & l_{f2} & l_{f3} \\ \hline f_1 & 2 & 3 & 1 & - & - \\ f_2 & 2 & 3 & - & 1 & - \\ f_3 & 2 & 3 & - & - & 1 \end{array}$$

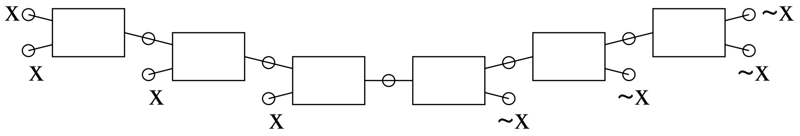
(c)

Fig. 2. The three gadgets: (a) box, (b) peg, and (c) pool

5.3 Constructing the Preference Set

A box is a "two-for-one" device: if another gadget takes its wide linking position, it demands both narrow linking positions. For any k, we can construct a k-for-one device from $k-1$ boxes by identifying the wide position of each box after the first with a narrow position of the previous box. If we identify the ultimate wide positions of two such devices, we can get a u-for-v device for any desired u and v.

For each variable x_i of the 3SAT instance S, we generate a many-for-many device whose two sets of narrow positions represent the references to x_i and the references to $\neg x_i$ in the clauses of S, respectively. The device for a variable x with four ordinary references and four negated ones could be drawn like this:



(Boxes represent boxes, circles represent linking positions, and lines represent linking people. Internal people and positions are not shown.) In a matching that obeys all the gadget constraints, we may assign all the linking people either to the right, filling the $\neg x$ positions and leaving the x positions open, or to the left, filling the x positions and leaving the $\neg x$ positions open. These two possibilities correspond to making x true or false, respectively. Either way, a linking position is left open if and only if the reference it represents evaluates to true.

Now, we add a pool for each clause of S and identify its three linking positions with linking positions of the variable devices according to the clause's three references. The pool demands that at least one of its linking people receive a linking position. This is possible if and only if the clause is satisfied, so we can see that a matching that obeys all the constraints represents a solution to S.

5.4 Correctness of the Reduction

We will now give the details of the argument that S is satisfiable if and only if P has a matching of the ideal unpopularity factor (margin) and, in doing so, specify the ideal unpopularities.

Suppose the tuple of truth values $(t_1, \ldots, t_v)$ satisfies S; we construct a matching M as follows. We match the device for each variable x_i in a manner that depends on t_i. If t_i is true, we match each box on the x_i side according to the first table below, filling its wide linking position, and each box on the $\neg x_i$ side according to the second, filling its two narrow linking positions. In the first table, we let n_1 be a/the person whose linking position is shared with a pool (rather than a box) so that the pressure of 2 on l_{n1} from the box is subsumed by that from the attached peg; we then assign n_2 to his last resort. Each table's superscripts give the pressures generated by the box shown; other gadgets may account for higher pressures on some linking positions.

$$\begin{array}{c} \\ w \\ i_1 \\ i_2 \\ i_3 \\ n_1 \\ n_2 \end{array}\begin{array}{c} \begin{array}{ccccccc} x^2 & y^1 & z^0 & u^1 & l_w^0 & l_{n1}^2 & l_{n2}^1 \end{array} \\ \begin{pmatrix} 2 & 3 & 5 & 4 & (\mathbf{1}) & - & - \\ (\mathbf{1}) & 2 & 3 & 4 & - & - & - \\ 1 & (\mathbf{2}) & 3 & 4 & - & - & - \\ 1 & 2 & (\mathbf{3}) & 4 & - & - & - \\ - & - & - & (\mathbf{2}) & - & 1 & - \\ - & - & - & 2 & - & - & 1 \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ w \\ i_1 \\ i_2 \\ i_3 \\ n_1 \\ n_2 \end{array}\begin{array}{c} \begin{array}{ccccccc} x^2 & y^1 & z^0 & u^0 & l_w^1 & l_{n1}^0 & l_{n2}^0 \end{array} \\ \begin{pmatrix} 2 & 3 & 5 & (\mathbf{4}) & 1 & - & - \\ (\mathbf{1}) & 2 & 3 & 4 & - & - & - \\ 1 & (\mathbf{2}) & 3 & 4 & - & - & - \\ 1 & 2 & (\mathbf{3}) & 4 & - & - & - \\ - & - & - & 2 & - & (\mathbf{1}) & - \\ - & - & - & 2 & - & - & (\mathbf{1}) \end{pmatrix} \end{array}$$

Each of the device's linking positions is filled exactly once except for those representing references to x_i. If t_i is instead false, we use the same construction but with the two sides of the device switched. Either way, exactly those linking positions that represent references evaluating to true are left open, and no pressure exceeds 2.

Now we assign each pool linking person to her linking position if it is available or otherwise to the best available position in her pool. Since the t_i satisfy S, at least one linking person from each pool will get a linking position, so each pool only needs to accommodate at most two people. A pool can hold two people as in the table below (the pressure superscripts consider the attached pegs as well as the pool itself), and additional people can be moved to linking positions without increasing the pressures.

$$\begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \end{array}\begin{array}{c} \begin{array}{ccccc} x^1 & y^0 & l_{f1}^2 & l_{f2}^2 & l_{f3}^2 \end{array} \\ \begin{pmatrix} (\mathbf{2}) & 3 & 1 & - & - \\ 2 & (\mathbf{3}) & - & 1 & - \\ 2 & 3 & - & - & (\mathbf{1}) \end{pmatrix} \end{array}$$

Nowhere did M incur a pressure exceeding 2, so it has unpopularity factor 2, which we designate as ideal. To bound its unpopularity margin, we use the following lemma, whose proof we omit due to space constraints:

Lemma 5.1. *The unpopularity margin of a Pareto efficient matching M does not exceed the number of filled positions in M that have pressure* 2 *or greater.*

M has exactly $6c - 2v$ positions of pressure 2, namely the $3c$ positions bearing pegs and the internal position x of each of the $3c - 2v$ boxes (one per variable reference minus $2v$ at the ends of the variable devices), so M has unpopularity margin at most $6c - 2v$, which we designate as ideal.

For the other direction of the proof, suppose that S is unsatisfiable and let M be an arbitrary Pareto efficient matching of P; we will show that M achieves neither the ideal unpopularity factor nor the ideal unpopularity margin. It should be clear that M cannot satisfy all the gadget constraints, but we must show that non-ideal unpopularities result.

A peg may or may not get its linking position in M, but either way, x must be filled for Pareto efficiency, and at least one person is left at her last resort. Starting from M, we "cycle" each peg by promoting a last-resort person to x, promoting the occupant of x to the linking position, and demoting the occupant of the linking position to his last resort. In each box, the three people i_1, i_2, and i_3 are all eager to fill the three positions x, y, z, so M must fill those positions for Pareto efficiency. We cycle the box by promoting z's occupant (who could be w rather than an i_j) to y and y's occupant to x, demoting the occupant of x. Let N be the resulting matching. We have performed two demotions and one promotion for each of the $3c$ pegs and $3c - 2v$ boxes, so N dominates M by the ideal margin of $6c - 2v$.

Now we modify N to exploit the gadget dissatisfaction in M. Suppose M dissatisfies a box, i.e., both the wide person w and at least one narrow person (say n_1) are denied linking positions. The four people w, i_1, i_2, and i_3 are eager to fill the four positions x, y, z, and u, so M must fill those positions. Instead of cycling this box, we do the following. If either w or an i_j is at her last resort, we promote her to z and z's occupant to y. Otherwise, n_1 must be at his last resort; we promote him to u and u's occupant (who must be w or an i_j) to y. Either way, we then promote y's occupant to x and demote the occupant of x. We now have 3 promotions in the box instead of 2, so N dominates M by a margin of $6c - 2v + 1$. Furthermore, the chain of promotions exerts a pressure of 3 on x.

On the other hand, if M dissatisfies a pool (by denying all three of its people their linking positions), then one of the people must be in x, one must be in y, and the third (call him p) must be at his last resort. Let l_{fi} be the linking position of the occupant of x. Instead of cycling the peg attached to l_{fi}, we promote p to y, y's occupant to x, and x's occupant to l_{fi}, demoting the occupant of l_{fi}. This strategy similarly raises the dominance margin to $6c - 2v + 1$ and reveals a pressure of 3 on l_{fi}.

In either case, the pressure of 3 shows that M fails to achieve the ideal unpopularity factor of 2 and N's dominance margin of $6c - 2v + 1$ shows that M fails to achieve the ideal unpopularity margin of $6c - 2v$, so the proof is complete.

By means of the reduction, we have proved the following result:

Theorem 5.2. *It is NP-hard to calculate the least unpopularity factor or margin of a given preference set. Thus, it is also NP-hard to compute an optimal matching.*

How well the least unpopularity factor and margin of an instance can be approximated is an open problem. The above proof shows that it is NP-hard to approximate the least unpopularity factor within a factor better than 3/2. The author examined three heuristic algorithms (see the supplementary material) that often find good matchings but could not prove an approximation bound for any of them; a search for a simple construction to increase the additive gap in the least unpopularity factor was also unsuccessful. Of course, a more important goal for future work is to find a good manipulation-resistant criterion under which *optimal* matchings always exist and can be found efficiently.

Acknowledgments. I would like to thank Samir Khuller, my advisor, for introducing me to previous work on matching with one-sided preferences; Brian Dean for suggesting the use of Goldberg's algorithm in Algorithm 3.5; and Dr. Khuller, Bobby Bhattacharjee, Glenda Torrence, Nancy Zheng, my parents, and others for suggesting improvements to this paper and its precursors.

Supplementary materials for this work, including Java implementations of some of the algorithms, are available at `http://mattmccutchen.net/lumc/`.

References

1. Abraham, D., Cechlárová, K., Manlove, D., Mehlhorn, K.: Pareto Optimality in House Allocation Problems. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 3–15. Springer, Heidelberg (2004)
2. Abraham, D., Irving, R., Kavitha, T., Mehlhorn, K.: Popular matchings. In: Proceedings of SODA 2005: The 16th ACM-SIAM Symposium on Discrete Algorithms, pp. 424–432 (2005)
3. Abraham, D., Kavitha, T.: Dynamic matching markets and voting paths. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 65–76. Springer, Heidelberg (2006)
4. Gale, D., Shapley, L.: College admissions and the stability of marriage. American Mathematical Monthly 16, 9–15 (1962)
5. Gärdenfors, P.: Match Making: assignments based on bilateral preferences. Behavioural Sciences 20, 166–173 (1975)
6. Goldberg, A.: Scaling algorithms for the shortest paths problem. In: Proceedings of SODA 1993: The 4th ACM-SIAM Symposium on Discrete Algorithms, pp. 222–231 (1993)
7. Irving, R., Kavitha, T., Mehlhorn, K., Michail, D., Paluch, K.: Rank-maximal matchings. In: Proceedings of SODA 2004: the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 68–75 (2004)
8. Papadimitriou, C., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity, ch. 11. In: Weighted Matching. Prentice-Hall, Englewood Cliffs (1982)
9. Price, E.: Personal communication (August 2005)
10. Roth, A.: The evolution of the labor market for medical interns and residents: A case study in game theory. Journal of Political Economy 92, 991–1016 (1984)

Randomized Rendez-Vous with Limited Memory

Evangelos Kranakis[1], Danny Krizanc[2], and Pat Morin[1]

[1] School of Computer Science, Carleton University
[2] Department of Mathematics and Computer Science, Wesleyan University

Abstract. We present a tradeoff between the expected time for two identical agents to rendez-vous on a synchronous, anonymous, oriented ring and the memory requirements of the agents. In particular, we show that there exists a $2t$ state agent, which can achieve rendez-vous on an n node ring in expected time $O(n^2/2^t + 2^t)$ and that any $t/2$ state agent requires expected time $\Omega(n^2/2^t)$. As a corollary we observe that $\Theta(\log \log n)$ bits of memory are necessary and sufficient to achieve rendez-vous in linear time.

1 Introduction

The problem of *rendez-vous* (the gathering of agents widely dispersed in some domain at a common place and time) has been studied under many guises and in many settings [2,15,4,5,7,6,8,10,9,14,12,18,20,21,22]. (See Reference [13] for a survey of recent results.) In this paper we consider the problem of autonomous mobile software agents gathering in a distributed network. This is a fundamental operation useful in such applications as web-crawling, distributed search, meeting scheduling, etc. In particular, we study the problem of two identical agents attempting to rendez-vous on a synchronous anonymous ring.

We consider the standard model of a synchronous anonymous oriented n-node ring [19]. The nodes are assumed to have no identities, the computation proceeds in synchronous steps and the edges of the ring are labelled `clockwise` and `counterclockwise` in a consistent fashion. We model the agents as identical probabilistic finite automata $A = \langle S, \delta, s_0 \rangle$ where S is the set of states of the automata including s_0 the initial state and the special state `halt`, and $\delta : S \times C \times P \rightarrow S \times M$ where $C = \{H, T\}$ represents a random coin flip, $P = \{\texttt{present}, \texttt{notpresent}\}$ represents a predicate indicating the presence of the other agent at a node, and $M = \{-1, 0, +1\}$ represents the potential moves the agent may make, $+1$ representing clockwise, -1 counterclockwise and 0 stay at the current node. During each synchronous step, depending upon its current state, the answer to a query for the presence of the other agent, and the value of an independent random coin flip with probability of `heads` equal to $1/2$, the agent uses δ in order to change its state and either move across the edge labelled `clockwise`, move across the edge labelled `counterclockwise` or stay at the current node. We assume that the agent halts once it detects the presence of the other agent at a node. Rendezvous occurs when both agents halt on the same node. The complexity measures we are interested in are the expected time (the number of synchronous steps) to rendez-vous (where the expectation is taken over all sequences of coin flips of the two agents) and the size ($|S|$) or memory requirement ($\log_2 |S|$) of the agents.

A number of researchers have observed that using random walks one can design $O(1)$ state agents that will rendez-vous in polynomial number steps on any network [3]. For the ring the expected time for two random walks to meet is easily shown to be $O(n^2)$. (See Reference [11] for an example proof of this fact.) This expected time bound can be improved by considering the following strategy. Repeat the following until rendez-vous is achieved: flip a fair coin and walk $n/2$ steps clockwise if the result is heads, $n/2$ steps counterclockwise if the result is tails. If the two agents choose different directions (which they do with probability 1/2) then they will rendez-vous (at least in the case where they start at an even distance apart). It is easy to see that the expected time until rendez-vous in this case is $O(n)$. Alpern refers to this strategy as Coin Half Tour and studies it in detail [1]. Note that the agents are required to count up to $n/2$ and thus require $\Omega(n)$ states or $\Omega(\log n)$ bits of memory to perform this algorithm. The main result of this paper is that this memory requirement can be reduced to $O(\log \log n)$ bits while still achieving rendez-vous in $O(n)$ expected time, and this is optimal.

Below we show a tradeoff between the size of the agents and the time required for them to rendez-vous. We prove there exists a $2t$ state algorithm, which can achieve rendez-vous on an n node ring in expected time $O(n^2/2^t + 2^t)$ and that any $t/2$ state algorithm requires expected time $\Omega(n^2/2^t)$. As a corollary we observe that $\Theta(\log \log n)$ bits of memory are necessary and sufficient to achieve rendez-vous in linear time. Section 2 contains some preliminary results, section 3 our upper bound and section 4 the lower bound.

2 Preliminaries

2.1 Martingales, Stopping Times, and Wald's Equations

In this section, we review some results on stochastic processes that are used several times in our proofs. The material in this section is based on the presentation in Ross' textbook [17, Chapter 6]. Let $X = X_1, X_2, X_3, \ldots$ be a sequence of random variables and let $Q = Q_1, Q_2, Q_3 \ldots$ be a sequence of random variables in which Q_i is a function of $X_1, \ldots, X_i$. Then we say that Q *is a martingale with respect to* X if, for all i, $\mathrm{E}[|Q_i|] < \infty$ and $\mathrm{E}[Q_{i+1} \mid X_1, \ldots, X_i] = Q_i$.

A positive integer-valued random variable T is a *stopping time* for the sequence $X_1, X_2, X_3, \ldots$ if the event $T = i$ is determined by the values $X_1, \ldots, X_i$. In particular, the event $T = i$ is independent of the values $X_{i+1}, X_{i+2}, \ldots$. Some of our results rely on the Martingale Stopping Theorem:

Theorem 1 (Martingale Stopping Theorem). *If $Q_1, Q_2, Q_3, \ldots$ is a martingale with respect to $X_1, X_2, X_3, \ldots$ and T is a stopping time for $X_1, X_2, X_3, \ldots$ then*

$$\mathrm{E}[Q_T] = \mathrm{E}[Q_1]$$

provided that at least one of the following holds:

1. *Q_i is uniformly bounded for all $i \leq T$,*
2. *T is bounded, or*
3. *$\mathrm{E}[T] < \infty$ and there exists an $M < \infty$ such that*

$$\mathrm{E}\left[|Q_{i+1} - Q_i| \mid X_1, \ldots, X_i\right] < M \ .$$

If $X_1, X_2, X_3, \ldots$ is a sequence of i.i.d. random variables with expected value $\mathrm{E}[X] < \infty$ and variance $\mathrm{var}(X) < \infty$ then by applying Theorem 1 on the sequence $Q_i = \sum_{j=1}^{i}(X_j - \mathrm{E}[X])$ we obtain Wald's Equation:

$$\mathrm{E}\left[\sum_{i=1}^{T} X_i\right] = \mathrm{E}[T] \cdot \mathrm{E}[X] \tag{1}$$

whenever T is a stopping time for $X_1, X_2, X_3, \ldots$ Similarly, we can derive a version of Wald's Equation for the variance by considering the martingale $Q_i = \left(\sum_{j=1}^{i}(X_j - \mathrm{E}[X]\right)^2 - i \cdot \mathrm{var}(X)$ to obtain

$$\mathrm{var}\left(\sum_{i=1}^{T} X_i\right) = \mathrm{E}\left[\left(\sum_{i=1}^{T}(X_i - \mathrm{E}[X_i])\right)^2\right] = \mathrm{E}[T] \cdot \mathrm{var}(X) \ . \tag{2}$$

2.2 A Lemma on Random Walks

Let $X_1, X_2, X_3, \ldots \in \{-1, +1\}$ be independent random variables with

$$\Pr\{X_i = -1\} = \Pr\{X_i = +1\} = 1/2$$

and let $S_i = \sum_{j=1}^{i} X_j$. The sequence $S_1, S_2, S_3, \ldots$ is a *simple random walk* on the line, where each X_i represents a step to the left ($X_i = -1$) or a step to the right ($X_i = +1$). Define the *hitting time* h_m as

$$h_m = \min\{i : |S_i| = m\} \ ,$$

which is the number of steps in a simple random walk before it travels a distance of m from its starting location. The following result is well-known (see, e.g., Reference [16]):

Lemma 1. $\mathrm{E}[h_m] = m^2$.

Applying Markov's Inequality with Lemma 1 yields the following useful corollary

Corollary 1. $\Pr\{\max\{|S_i| : i \in \{1, \ldots, 2m^2\}\} \geq m\} \geq 1/2$.

In other words, Corollary 1 says that, at least half the time, at some point during the first $2m^2$ steps of a simple random walk, the walk is at distance m from its starting location.

Let $Y_1, \ldots, Y_m$ be i.i.d. non-negative random variables with finite expectation $r = \mathrm{E}[Y_i]$, independent of $X_1, \ldots, X_m$, and with the property that

$$\Pr\{Y_i \geq \alpha r\} \geq 1/2$$

for some constant $\alpha > 0$. The following lemma considers a modified random walk in which the ith step is of length Y_i:

Lemma 2. *Let $X_1, \ldots, X_m$ and $Y_1, \ldots, Y_m$ be defined as above. Then there exists constants $\beta, \kappa > 0$ such that*

$$\Pr\left\{ \max\left\{ \left| \sum_{i=1}^{m'} X_i Y_i \right| : m' \in \{1, \ldots, m\} \right\} \geq \beta r \sqrt{m} \right\} \geq \kappa \ .$$

Proof. We will define 3 events E_1, E_2, E_3 such that $\Pr\{E_1 \cap E_2 \cap E_3\} \geq 1/8$ and, if E_1, E_2, and E_3 all occur, then there exists a value $m' \in \{1, \ldots, m\}$ such that $\left|\sum_{i=1}^{m'} X_i Y_i\right| \geq \alpha r \sqrt{m}/2^{3/2}$. This will prove the lemma for $\kappa = 1/8$ and $\beta = \alpha/2^{3/2}$.

Let E_1 be the event that there exists a value $m' \in \{1, \ldots, m\}$ such that

$$\left| \sum_{i=1}^{m'} X_i \right| \geq \sqrt{m/2} \ .$$

By Corollary 1, $\Pr\{E_1\} \geq 1/2$. Assume E_1 occurs and, without loss of generality, assume $\sum_{i=1}^{m'} X_i > 0$.

Let $I^+ = \{i \in \{1, \ldots, m'\} : X_i = +1\}$ and $I^- = \{1, \ldots, m'\} \setminus I^+$. We further partition I^+ into two sets I_1^+ and I_2^+ where I_1^+ contains the smallest $|I^-|$ elements of I^+ and I_2^+ contains the remaining elements. Note that, with these definitions, $|I_1^+| = |I^-|$ and that $|I_2^+| = \sum_{i=1}^{m'} X_i$. Let E_2 be the event that

$$\sum_{i \in I_1^+} X_i Y_i + \sum_{i \in I^-} X_i Y_i \geq 0$$

which is equivalent to $\sum_{i \in I_1^+} Y_i \geq \sum_{i \in I^-} Y_i$ and observe that, by symmetry, $\Pr\{E_2 | E_1\} \geq 1/2$.

Finally, let E_3 be the event

$$\sum_{i \in I_2^+} X_i Y_i \geq \alpha r |I_2^+|/2$$

To bound $\Pr\{E_3 | E_1 \cap E_2\}$, let $T = \left|\{i \in I_2^+ : Y_i \geq \alpha r\}\right|$ and observe that $T \geq |I_2^+|/2$ implies E_3. Now, T is a binomial$(|I_2^+|, p)$ random variable for $p \geq 1/2$ so its median value is at least $p|I_2^+| \geq |I_2^+|/2$ and therefore $\Pr\{E_3 | E_1 \cap E_2\} \geq \Pr\{T \geq |I_2^+|/2\} \geq 1/2$.

We have just shown that $\Pr\{E_1 \cap E_2 \cap E_3\} \geq 1/8$. To complete the proof we observe that, if E_1, E_2 and E_3 occur then

$$\begin{aligned}\sum_{i=1}^{m'} X_i Y_i &= \sum_{i \in I_1^+} X_i Y_i + \sum_{i \in I^-} X_i Y_i + \sum_{i \in I_2^+} X_i Y_i \\ &\geq \sum_{i \in I_2^+} X_i Y_i \\ &\geq \alpha r |I_2^+|/2 \\ &\geq \alpha r \sqrt{m}/2^{3/2} \ .\end{aligned}$$

□

2.3 An Approximate Counter

In the previous section we have shown that, if we can generate random variables Y_i that are frequently large, then we can speed up the rate at which a random walk moves away from its starting location. In this section we consider how to generate these frequently-large random variables. Consider a random variable Y generated by the following algorithm:

```
BigRand($t$)
1: $Y \leftarrow C \leftarrow 0$
2: while $C < t$ do
3:    $Y \leftarrow Y + 1$
4:    if a coin toss comes up heads then
5:       $C \leftarrow C + 1$
6:    else
7:       $C \leftarrow 0$
8: return $Y$
```

Lemma 3. *Let Y be the output of Algorithm* BigRand(t). *Then*

1. $\mathrm{E}[Y] = 2^t(2 - 1/2^{t-1})$ *and*
2. $\Pr\{Y \geq \mathrm{E}[Y]/2\} \geq 1/2$.

Proof. To compute the expected value of Y we observe that the algorithm begins by tossing a sequence of $i-1$ heads and then either (a) returning to the initial state if the ith coin toss is a tail or (b) terminating if $i = 2^t$. The first case occurs with probability $1/2^i$ and the second case occurs with probability $1/2^t$. In this way, we obtain the equation

$$\mathrm{E}[Y] = \sum_{i=1}^{t} \frac{1}{2^i}(i + \mathrm{E}[Y]) + \frac{t}{2^t} \ .$$

Rearranging terms and multiplying by 2^t, we obtain

$$\mathrm{E}[Y] = 2^t(2 - 1/2^{t-1}) \ .$$

To prove the second part of the lemma, consider the number of times the counter C is reset to 0 in Line 7 of the algorithm. This number is a geometric $(1/2^t)$ random variable and its expected value is therefore $2^t \geq \mathrm{E}[Y]/2$. Since the number of times Line 7 executes is a lower bound on the number of times the value of Y is incremented (Line 3), this completes the proof. $\square$

3 The Rendez-Vous Algorithm

Consider the following algorithm used by an agent to make a random walk on a ring. The agent repeatedly performs the following steps: (1) toss a coin to determine a direction $d \in \{\texttt{clockwise}, \texttt{counterclockwise}\}$ then (2) run algorithm BigRand(t) replacing each increment of the variable Y with a step in direction d. By using t states for a clockwise counter and t states for a counterclockwise counter this algorithm can be implemented by a $2t$ state finite automata. (Or using one bit to remember the direction d and $\log t$ bits to keep track of the counter C in the BigRand algorithm, it can be implemented by an agent having only $1 + \log_2 t$ bits of memory.)

We call m iterations of the above algorithm a *round*. Together, Lemma 2 and Lemma 3 imply that, during a round, with probability at least κ, an agent will travel a distance of at least $\beta 2^t \sqrt{m}$ from its starting location. Set

$$m = \left\lceil \frac{n^2}{\beta^2 2^{2t}} \right\rceil \leq 1 + \frac{n^2}{\beta^2 2^{2t}}$$

and consider what happens when two agents A and B both execute this rendez-vous algorithm. During the first round of A's execution, with probability at least κ, agent A will have visited agent B's starting location. Furthermore, with probability at least $1/2$ agent B will not have moved away from A when this occurs, so the paths of agents A and B will cross, and a rendez-vous will occur, with probability at least $\kappa/2$.

By Lemma 3, the expected number of steps taken for A to execute the ith round is at most

$$\mathrm{E}[M_i] \leq m2^t \ .$$

The variables $M_1, M_2, \cdots$ are independent and the algorithm terminates when A and B rendez-vous. If we define T as the round in which agents A and B rendez-vous then the time to rendez-vous is bounded by

$$\sum_{i=1}^{T} M_i \ .$$

Note that the event $T = j$ is independent of $M_{j+1}, M_{j+2}, \ldots$ so T is a stopping time for the sequence $M_1, M_2, \ldots$ so, by Wald's Equation

$$E\left[\sum_{i=1}^{T} M_i\right] = E[T] \cdot E[M_1] \leq \frac{2}{\kappa} \cdot m2^t \ .$$

This completes the proof of our first theorem.

Theorem 2. *There exists a rendez-vous algorithm in which each agent has at most $2t$ states and whose expected rendez-vous time is $O(n^2/2^t + 2^t)$.*

4 The Lower Bound

Next we show that the algorithm in Section 3 is optimal.

The model of computation for the lower bound represents a rendez-vous algorithm $\mathcal{A}$ as a probablistic finite automata having t states. Each vertex of the automata has two outgoing edges representing the two possible results of a coin toss and each edge e is labelled with a real number $\ell(e) \in [-1, +1]$. The edge label of e is represented as a step of length $|\ell(e)|$ with this step being counterclockwise if $\ell(e) < 0$ and clockwise if $\ell(e) > 0$. As before, both agents use identical automata and start in the same state. The rendez-vous process is complete once the distance between the two agents is at most 1. This model is stronger than the model used for upper bound, since the edge labels are no longer restricted to be in the discrete set $\{-1, 0, +1\}$ and the definition of a rendez-vous has been slightly relaxed.

4.1 Well-Behaved Algorithms and Reset Times

We say that an algorithm is *well-behaved* if the directed graph of the state machine has only one strongly connected component that contains all nodes. We are particularly interested in intervals between consecutive visits to the start state, which we will call *rounds*.

Lemma 4. *Let R be the number of steps during a round. Then* $\mathrm{E}[R] \le 2^t$ *and* $\mathrm{E}[R^2] \le c2^{2t}$.

Proof. For each state v of $\mathcal{A}$'s automata fix a shortest path (a sequence of edges) leading from v to the start state. For an automata that is currently at v we say that the next step is a *success* if it traverses the first edge of this path, otherwise we say that the next step is a *failure*.

Each round can be further refined into *phases*, where every phase consists of 0 or more successes followed by either a failure or by reaching the start vertex. Let X_i denote the length of the ith phase and note that X_i is dominated[1] by a geometric(1/2) random variable X'_i, so $\mathrm{E}[X_i] \le \mathrm{E}[X'_i] \le 2$. On the other hand, if a phase lasts $t-1$ steps then the start vertex is reached. Therefore, the probability of reaching the start vertex during any particular phase is at least $1/2^{t-1}$ and the number T of phases is dominated by a geometric$(1/2^{t-1})$ random variable T', so $\mathrm{E}[T] \le \mathrm{E}[T'] \le 2^{t-1}$. Therefore, by Wald's Equation

$$\mathrm{E}[R] = \mathrm{E}\left[\sum_{i=1}^{T} X_i\right] \le \mathrm{E}\left[\sum_{i=1}^{T'} X'_i\right] = \mathrm{E}[T'] \cdot \mathrm{E}[X'_1] \le 2^t \enspace .$$

[1] A random variable X dominates a random variable Y if $\Pr\{X > x\} \ge \Pr\{Y > x\}$ for all $x \in \mathbb{R}$.

For the second part of the lemma, we can apply Wald's Equation for the variance ([2]) to obtain

$$\begin{aligned}
\mathrm{E}[R^2] &= \mathrm{E}\left[\left(\sum_{i=1}^{T} X_i\right)^2\right] \\
&\le \mathrm{E}\left[\left(\sum_{i=1}^{T'} X'_i\right)^2\right] \\
&= \mathrm{var}\left(\sum_{i=1}^{T'} X'_i\right) + (\mathrm{E}[T'] \cdot \mathrm{E}[X'_1])^2 \\
&= \mathrm{E}[T'] \cdot \mathrm{var}(X_1) + (\mathrm{E}[T'] \cdot \mathrm{E}[X'_1])^2 \\
&\le 2^{t-1} \cdot 4 + (2^{2t-1} \cdot 8) \\
&\le 5 \cdot 2^{2t}
\end{aligned}$$

as required. □

4.2 Unbiasing Algorithms

Note that $\mathrm{E}[R]$ can be expressed another way: For an edge e of the state machine, let $f(e)$ be the expected number of times the edge e is traversed during a round. The *reset time* of algorithm $\mathcal{A}$ is then defined as

$$\mathrm{reset}(\mathcal{A}) = \sum_e f(e) = \mathrm{E}[R] \enspace .$$

The *bias* of a well-behaved algorithm $\mathcal{A}$ is defined as

$$\mathrm{bias}(\mathcal{A}) = \sum_e f(e) \cdot \ell(e) \enspace ,$$

which is the expected sum of the edge labels encoutered during a round. We say that $\mathcal{A}$ is *unbiased* if $\mathrm{bias}(\mathcal{A}) = 0$, otherwise we say that $\mathcal{A}$ is *biased.*

Biased algorithms are somewhat more difficult to study. However, observe that, for any algorithm $\mathcal{A}$ we can replace every edge label $\ell(e)$ with the value $\ell(e) - x$ for any real number x and obtain an equivalent algorithm in the sense that, if two agents A and B execute the modified algorithm following the same sequence of state transitions then A and B will rendez-vous after exactly the same number of steps. In particular, if we replace each edge label $\ell(e)$ with the value

$$\ell'(e) = \ell(e) - \frac{\mathrm{bias}(\mathcal{A})}{\mathrm{reset}(\mathcal{A})}$$

then we obtain an algorithm $\mathcal{A}'$ with $\mathrm{bias}(\mathcal{A}') = 0$. Furthermore, since $|\,\mathrm{bias}(\mathcal{A})| \le \mathrm{reset}(\mathcal{A})$, every edge label $\ell'(e)$ has $-2 \le \ell'(e) \le 2$. This gives the following relation between biased and unbiased algorithms:

Lemma 5. *Let $\mathcal{A}$ be a well-behaved t-state algorithm with expected rendez-vous time R. Then there exists a well-behaved unbiased t-state algorithm $\mathcal{A}'$ with expected rendez-vous time at most $2R$.*

4.3 The Lower Bound for Well-Behaved Algorithms

We now have all the tools in place to prove the lower bound for the case of well-behaved algorithms.

Lemma 6. *Let $\mathcal{A}$ be a well-behaved t-state algorithm. Then the expected rendez-vous time of $\mathcal{A}$ is $\Omega(n^2/2^{2t})$.*

Proof. Suppose the agents are placed at antipodal locations on an n node ring, so that the distance between them is $n/2$. We will show that there exists constants $c > 0$ and $p > 0$ such that, after $cn^2/2^t$ steps, with probability at least p neither agent will have travelled a distance greater than $n/4$ from their starting location. Thus, the expected rendez-vous time is at least $pcn^2/2^t = \Omega(n^2/2^t)$.

By Lemma 5 we can assume that $\mathcal{A}$ is unbiased. Consider the actions of a single agent starting at location 0. The actions of the agent proceed in rounds where, during the ith round, the agent takes R_i steps and the sum of edge labels encountered during these steps is X_i. Note that the random variables $X_1, X_2, \ldots$ are i.i.d. with expectation $\mathrm{E}[X] = 0$ and variance $\mathrm{E}[X^2]$. Since the absolute value of X_i is bounded from above by R_i, we have the inequalities $\mathrm{E}[|X_i|] \leq \mathrm{E}[R_i]$ and $\mathrm{E}[X_i^2] \leq \mathrm{E}[R_i^2]$.

Let $S_i = \left|\sum_{j=1}^{i} X_j\right|$, for $i = 0, 1, \ldots$ be the agent's distance from their starting location at the end of the ith round. Let $Q_i = S_i^2 - i\mathrm{E}[X^2]$ and observe that the sequence $Q_1, Q_2, \ldots$ is a martingale with respect to the sequence $X_1, X_2, \ldots$ [17, Example 6.1d]. Define

$$T = \min\{i : S_i \geq m\} \ ,$$

and observe that this is equivalent to

$$T = \min\{i : Q_i \geq m^2 - i\mathrm{E}[X^2]\} \ .$$

The random variable T is a *stopping time* for the martingale $Q_1, Q_2, \ldots$ so, by the Theorem 1

$$\mathrm{E}[Q_T] = \mathrm{E}[Q_1] = \mathrm{E}[(X_1)^2 - \mathrm{E}[X^2]] = 0 \ . \tag{3}$$

However, by definition $Q_T \geq m^2 - T \cdot \mathrm{E}[X^2]$, so

$$\mathrm{E}[Q_T] \geq \mathrm{E}[m^2 - T \cdot \mathrm{E}[X^2]] = m^2 - \mathrm{E}[T] \cdot \mathrm{E}[X^2] \ . \tag{4}$$

Equating the right hand sides of (3) and (4) gives

$$\mathrm{E}[T] \geq \frac{m^2}{\mathrm{E}[X^2]} \ .$$

Furthermore, the expected number of steps taken by the agent during these T rounds is, by Wald's Equation,

$$\mathrm{E}\left[\sum_{i=1}^{T} R_i\right] = \mathrm{E}[T] \cdot \mathrm{E}[R_1] \geq \frac{m^2\mathrm{E}[R]}{\mathrm{E}[R^2]} \geq \frac{m^2\mathrm{E}[R]}{c2^{2t}} \geq \frac{m^2}{c2^{2t}} \ ,$$

where the last two inequalities follow from Lemma 4 and the fact that $R \geq 1$. □

4.4 Badly-Behaved Algorithms

Finally, we consider the case where the algorithm $\mathcal{A}$ is not well-behaved. In this case, $\mathcal{A}$'s automata contains a set of *terminal components*. These are disjoint sets of vertices of the automata that are strongly connected and that have no outgoing edges (edges with source in the component and target outside the component). From each terminal component, select an arbitrary vertex and call it the *terminal start state* for that terminal component. An argument similar to that given in Lemma 4 proves:

Lemma 7. *The expected time to reach some terminal start state is at most* 2^t.

Observe that each terminal component defines a well-behaved algorithm. Let c be the number of terminal components and let $t_1, \ldots, t_c$ be the sizes of these terminal components. When two agents execute the same algorithm $\mathcal{A}$, Lemma 7 and Markov's Inequality imply that the probability that both agents reach the same terminal component after at most 2^{t+2} steps is at least $1/2c$. By applying Lemma 6 to each component, we can therefore lower bound the expected rendez-vous time by

$$\frac{1}{2c}\Omega(n^2/2^{t-c}) \geq \Omega(n^2/2^{2t}) \ ,$$

Substituting $t' = t/2$ into the above completes the proof of our second theorem:

Theorem 3. *Any $t/2$-state rendez-vous algorithm has expected rendez-vous time* $\Omega(n^2/2^t)$.

4.5 Linear Time Rendez-vous

We observe that Theorems 2 and 3 immediately imply:

Theorem 4. $\Theta(\log\log n)$ *bits of memory are necessary and sufficient to achieve rendez-vous in linear time on an n node ring.*

Note that the tradeoff between the time and the number of states, t, required for rendez-vous presented in Theorems 2 and 3 is tight to within a factor of 4 for $t \leq 2\log n$. For $t > 2\log n$ the upper bound diverges. It would be interesting to know if there exists a modified version of our algorithm that is optimal for all t. Also, investigating similar tradeoffs for other networks would be of interest.

Acknowledgments

Research of the first and third authors was supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada) and MITACS (Mathematics of Information Technology and Complex Systems) grants.

References

1. Alpern, S.: The rendezvous search problem. SIAM Journal of Control and Optimization 33, 673–683 (1995)
2. Alpern, S., Gal, S.: The Theory of Search Games and Rendezvous. Kluwer Academic Publishers, Norwell, Massachusetts (2003)
3. Coppersmith, D., Tetali, P., Winkler, P.: Collisions among random walks on a graph. SIAM Journal of Discrete Mathematics 6, 363–374 (1993)
4. Dessmark, A., Fraigniaud, P., Pelc, A.: Deterministic rendezvous in graphs. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 184–195. Springer, Heidelberg (2003)
5. Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Multiple agents rendezvous in a ring in spite of a black hole. In: Papatriantafilou, M., Hunel, P. (eds.) OPODIS 2003. LNCS, vol. 3144, pp. 34–46. Springer, Heidelberg (2004)
6. Flocchini, P., Kranakis, E., Krizanc, D., Luccio, F., Santoro, N., Sawchuk, C.: Mobile agent rendezvous when tokens fail. In: Kralovic, R., Sýkora, O. (eds.) SIROCCO 2004. LNCS, vol. 3104, pp. 161–172. Springer, Heidelberg (2004)
7. Flocchini, P., Kranakis, E., Krizanc, D., Santoro, N., Sawchuk, C.: Multiple mobile agent rendezvous in the ring. In: Farach-Colton, M. (ed.) LATIN 2004. LNCS, vol. 2976, pp. 599–608. Springer, Heidelberg (2004)
8. Gasieniec, L., Kranakis, E., Krizanc, D., Zhang, X.: Optimal memory rendezvous of anonymous mobile agents in a uni-directional ring. In: Wiedermann, J., Tel, G., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2006. LNCS, vol. 3831, pp. 282–292. Springer, Heidelberg (2006)
9. Kowalski, D., Malinowski, A.: How to meet in an anonymous network. In: Flocchini, P., Gasieniec, L. (eds.) SIROCCO 2006. LNCS, vol. 4056, pp. 44–58. Springer, Heidelberg (2006)
10. Kowalski, D., Pelc, A.: Polynomial deterministic rendezvous in arbitrary graphs. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 644–656. Springer, Heidelberg (2004)
11. Kranakis, E., Krizanc, D.: An algorithmic theory of mobile agents. In: Symposium on Trustworthy Global Computing (2006)
12. Kranakis, E., Krizanc, D., Markou, E.: Mobile agent rendezvous in a synchronous torus. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 653–664. Springer, Heidelberg (2006)
13. Kranakis, E., Krizanc, D., Rajasbaum, S.: Mobile agent rendezvous: A survey. In: Flocchini, P., Gasieniec, L. (eds.) SIROCCO 2006. LNCS, vol. 4056, pp. 1–9. Springer, Heidelberg (2006)
14. Kranakis, E., Krizanc, D., Santoro, N., Sawchuk, C.: Mobile agent rendezvous search problem in the ring. In: Proc. International Conference on Distributed Computing Systems (ICDCS), pp. 592–599 (2003)
15. De Marco, G., Gargano, L., Kranakis, E., Krizanc, D., Pelc, A., Vacaro, U.: Asynchronous deterministic rendezvous in graphs. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 271–282. Springer, Heidelberg (2005)

16. Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press, Cambridge (2005)
17. Ross, S.M.: Probability Models for Computer Science. Harcourt Academic Press, London (2002)
18. Roy, N., Dudek, G.: Collaborative robot exploration and rendezvous: Algorithms, performance bounds and observations. Autonomous Robots 11, 117–136 (2001)
19. Santoro, N.: Design and Analysis of Distributed Algorithms. Wiley, Hoboken (2006)
20. Sawchuk, C.: Mobile Agent Rendezvous in the Ring. PhD thesis, Carleton University, School of Computer Science, Ottawa, Canada (2004)
21. Suzuki, I., Yamashita, M.: Distributed anonymous mobile robots: Formation of geometric patterns. SIAM Journal of Computing 28, 1347–1363 (1999)
22. Yu, X., Yung, M.: Agent rendezvous: A dynamic symmetry-breaking problem. In: Meyer auf der Heide, F., Monien, B. (eds.) ICALP 1996. LNCS, vol. 1099, pp. 610–621. Springer, Heidelberg (1996)

Origami Embedding of Piecewise-Linear Two-Manifolds

Marshall Bern[1] and Barry Hayes[2]

[1] Palo Alto Research Center, 3333 Coyote Hill Rd., Palo Alto CA 94304, USA
bern@parc.com
[2] Google, Inc., 1600 Amphitheatre Parkway, Mountain View CA 94043, USA
bhayes@cs.stanford.edu

Abstract. We show that any compact, orientable, piecewise-linear two-manifold with Euclidean metric can be realized as a flat origami, meaning a set of non-crossing polygons in Euclidean 2-space “plus layers”. This result implies a weak form of a theorem of Burago and Zalgaller: any orientable, piecewise-linear two-manifold can be embedded into Euclidean 3-space “nearly” isometrically. We also correct a mistake in our previously published construction for cutting any polygon out of a folded sheet of paper with one straight cut.

1 Introduction

Inspired by various examples (see Figure 1), Erik Demaine [8] asked the following question: can any polyhedron be “crushed”? In other words, can any polyhedral surface in Euclidean three-space be realized as a flat folding, that is, as a finite set of planar faces in the Euclidean plane “plus layers”? This question asks only for the existence of a flat folding, and not for the existence of a continuous motion that transforms the polyhedral surface to a flattened state. Indeed if faces are assumed rigid, with edges as hinges, then there exists no such continuous motion; certain special polyhedra can flex [6], but none can change volume [7].

In this paper, we answer Demaine’s question affirmatively by adapting a construction of Bern, Demaine, Eppstein, and Hayes [3], that solves the problem of folding a sheet of paper so that any polygon can be realized by one straight cut. In fact, we prove a more general result, by showing that a flat folding exists even for a polyhedral surface without an embedding, that is, an orientable piecewise-linear (PL) 2-manifold given as a cell complex of Euclidean polygons glued together at edges. This result extends a theorem of Zalgaller [18]: any PL 2-manifold has an isometric submersion into $\mathbb{E}^2$. (In an isometric submersion the surface may pass through itself.)

Our result also relates to a theorem of Burago and Zalgaller [5], which states that any *submetric* (or “short”) embedding of a PL 2-manifold into Euclidean 3-space $\mathbb{E}^3$ can be approximated by an isometric embedding. (A submetric embedding is one in which the geodesic distances between corresponding points are non-increasing.) In other words, a submetric embedding (proved to exist in [4]) can be “wrinkled up” so that it remains within any small ϵ of its original position,

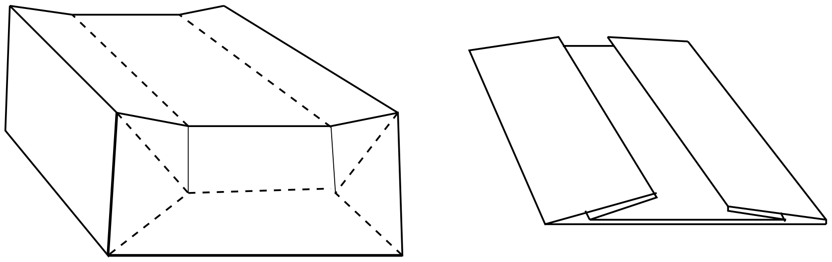


Fig. 1. This rectangular parallelopiped can be creased like a paper bag at two ends, giving the flat folding shown on the right. Dashed lines show valley folds and solid lines show mountain folds.

yet exactly recovers all geodesic distances of the original PL manifold. This result is the two-dimensional, piecewise-linear analogue to John Nash's celebrated embedding theorems for differentiable manifolds. In fact, it is the analogue for both the C^∞ Nash embedding theorem [16] and the C^1 Nash-Kuiper embedding theorem [13,15], because it preserves curvature as in the C^∞ theorem, but embeds in the lowest possible dimension as in the C^1 theorem. Our result can be used to give a "nearly isometric" embedding: given any $\epsilon > 0$, any orientable PL 2-manifold can be embedded into $\mathbb{E}^3$ such that the distance between pairs of points changes (increases or decreases) by no more than ϵ.

The techniques used in this paper also relate to the techniques used by Burago and Zalgaller. In constructing their isometric embedding into $\mathbb{E}^3$, Burago and Zalgaller use a very finely subdivided all-acute-angle triangulation of the PL 2-manifold. For our isometric embedding into layered $\mathbb{E}^2$, we use a coarser mix of triangles and "origami-friendly" quadrangles. If the 2-manifold M is a topological sphere, and all its faces are triangles of bounded aspect ratio, our construction has complexity $O(n)$, where n is the number of faces of M. In fact, the construction is simple enough that one can imagine using it to fold inflatable sculptures.

Recently, Krat, Burago, and Petrunin [12] showed that Zalgaller's submersion theorem generalizes to any dimension d. Moreover, Krat *et al.* removed the self-crossings for the case of 2-dimensional PL manifolds, and independently obtained the same origami embedding result as our own. However, the origami embedding of Krat *et al.* uses the techniques of Burago and Zalgaller, and hence has much higher complexity than our construction.

2 Definitions and Main Results

The notion of flat folding has been formalized several times in various ways [2,9,11]. In the following definition we require a total order on the faces of the folding; this rules out certain flat foldings such as weavings [2] that might be perfectly acceptable for real paper.

Definition 1. *A* **flat folding** *consists of a set of open polygons $P_1, P_2, \ldots, P_n$ in the Euclidean plane. The polygons are ordered so that P_i lies "above" P_j if $i < j$. Two polygons P_i and P_k, $i < k$, may be* **joined** *at an edge e if (1) e is a boundary edge of each of them, (2) no P_j with $i < j < k$ intersects e, and (3) no P_j with $i < j < k$ is joined to a polygon P_ℓ, with $\ell \leq i$ or $\ell \geq k$ at any point of the relative interior of e.*

Definition 2. *A* **metric piecewise-linear (PL) 2-manifold** *is a finite complex of Euclidean polygons with the topology of a 2-manifold (possibly with boundary). (Thus we disallow three faces meeting at an edge, dangling edges, pinch points, and so forth.)*

Definition 3. *A metric PL 2-manifold H has a* **flat-folded realization** *if there is a subdivision of H, with faces subdivided by extra vertices and edges ("creases"), and a continuous, bijective mapping from H to a flat folding, taking each subface of H to an isometric copy in the flat folding.*

Conditions (2) and (3) of Definition 1 guarantee that the complex of polygons behaves like paper, meaning that it cannot pass through itself. In our arguments, we will sometimes use informal terms such as "folds flat" instead of "has a flat-folded realization". Our first result is the following.

Theorem 1. *Every compact, orientable, metric PL 2-manifold has a flat-folded realization.*

Our second result is a new proof of the existence of "nearly isometric" embeddings into $\mathbb{E}^3$. Theorem 2 follows almost immediately from Theorem 1, and thus we obtain Zalgaller's submersion theorem and a weak form of the Burago-Zalgaller theorem with one basic construction.

Theorem 2 (Weak form of Burago-Zalgaller). *Let M be a compact, orientable, metric PL 2-manifold. Given any $\epsilon > 0$, M has a piecewise-linear embedding into $\mathbb{E}^3$, such that the distance between any pair of points increases or decreases by an additive factor of at most ϵ.*

3 One Straight Cut Revisited

The proof of Theorem 1 reuses the construction we used for the one-straight-cut problem [3]. As shown in Figure 2, the construction packs non-overlapping disks into a polygon so that a disk is centered at each vertex of the polygon, each edge of the polygon is a union of disk radii, and each gap between disks has either three or four sides. Any gap with more than four sides can be broken into gaps with fewer sides [1]; but 4-sided gaps give new 4-sided gaps. The number of disks needed to pack an n-sided polygon P is bounded by a constant times the sum of the aspect ratios of the triangles in a triangulation of P. The aspect ratio of a triangle can be defined in various ways; for specificity, let us define the aspect ratio to be the length of the longest side divided by the radius of the inscribed circle.

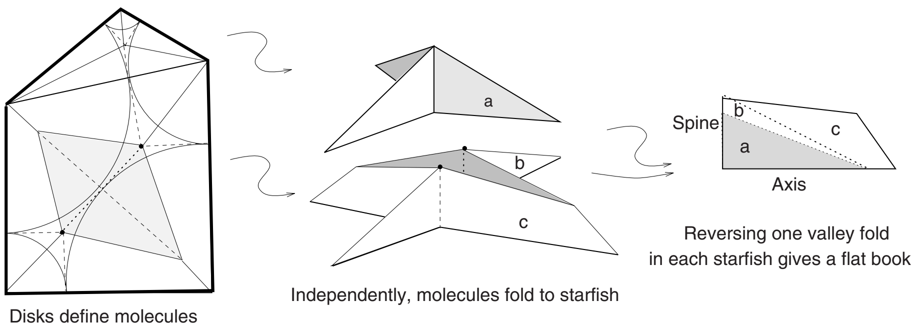

Disks define molecules
Independently, molecules fold to starfish
a
b
c
Spine
Axis
Reversing one valley fold in each starfish gives a flat book

Fig. 2. The construction from our previous paper [3] breaks a polygon (bold pentagon) into triangles and quadrangles ("molecules") by packing disks, so that each gap between disks has either 3 or 4 sides. Each polygon can then be folded into a shape called a "starfish". After reversing one valley fold to a mountain fold, starfish form book foldings that fit together spine to "armpit".

Connecting the centers of tangent disks by new edges now breaks the polygon into triangles and quandrangles. The triangles and quandrangles can be folded independently with known origami patterns [14], called *rabbit-ear* and *gusset molecules*. Perpendicular creases exit the triangle and quadrangle molecules at the disk tangency points; this property helps provide compatibility between adjacent molecules. Burago and Zalgaller use acute triangles as their basic units, with folds exiting at edge midpoints [5]. The number of acute triangles can be much larger than the number of molecules in our construction, because it depends upon uniformly good approximation of a set of real numbers (edge lengths) by rational numbers with large common denominator [4,5]. In our construction, the number of molecules depends upon aspect ratios; for example, if we start with a triangulated topological sphere with n vertices and no small face angles, then we use only $O(n)$ molecules.

Crucial to our use of the gusset molecule is the specialness of our quadrangles. Two of the vertices of the gusset, shown by dots in Figure 3(a), are fixed by the requirement that valley folds extend perpendicularly from the points of tangency. We refer to these vertices as the *perpendicular points*. The other two vertices of the gusset are not completely constrained. They must, however, lie on the angle bisectors of the quadrangle in order for the boundary of the quadrangle to fold to a common plane. One way to locate the the unconstrained vertices—p and r in Figure 3(a)—is to place them at the vertices of an *inset quadrangle*, a quadrangle inside the overall quadrangle, with sides parallel and equidistant to the sides of the original quadrangle. In Figure 3(a) the original quadrangle is $abcd$ and the inset quadrangle is $pqrs$. The gusset folding restricted to $pqrs$ is just two rabbit-ear molecules, as shown in Figure 3(b). Hence, the perpendicular points must lie at the in-centers of triangles pqr and prs, and this requirement determines the size of $pqrs$.

We now argue that all quadrangles induced by 4-sided gaps can be folded with the gusset molecule. What we must show is that the triangles pqr and prs with

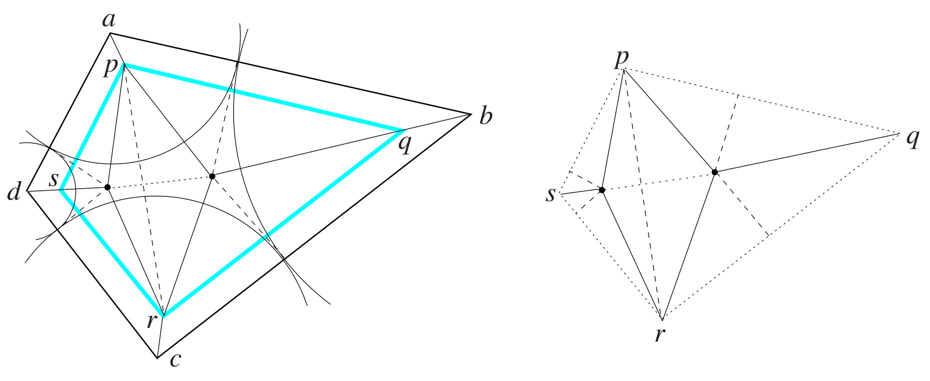


Fig. 3. (a) Two interior vertices of the gusset molecule, shown as dots, are fixed by the requirement that valley folds extend perpendicularly from disk-tangency points. The other two interior vertices, p and q, lie along an inset quadrangle, with the inset distance determined by the requirement that the dotted points must be the in-centers of triangles pqr and prs. (b) The inset quadrangle $pqrs$ is folded with two rabbit-ear molecules, so the boundary of $pqrs$ folds to a single axis, and the boundary of the original quadrangle $abcd$ does as well. One can think of a gusset as two rabbit-ear molecules surrounded by a ribbon (the region between $abcd$ and $pqrs$).

in-centers at the perpendicular points do indeed lie within $abcd$, in other words, that the requirements of the gusset are not in conflict with each other.

First assume that the perpendicular points are distinct, and consider the line L through the perpendicular points. Line L is the line of equal power distance[1] from the disks centered at a and c, and hence passes between these disks. The bisector of the angle between L and the valley fold perpendicular to bc fixes the point r. Since L passes above the disk at c, r lies above c along the angle bisector at c. Thus $pqrs$ does indeed lie within $abcd$. In the extreme case that the disks at a and c touch each other, $pqrs$ equals $abcd$ and the gusset molecule reduces to two rabbit-ear molecules.

What if the perpendicular points coincide? For this extreme case, we use a special property [1] of 4-gaps: the points of tangency of four disks, tangent in a cycle, are cocircular. This property implies that the angle bisectors of the quadrangle all meet at a common point o, namely the center of the circle through the tangencies. So in the extreme case that the perpendicular points coincide, $pqrs$ shrinks to point o, and the valleys from the points of tangency and the mountains along the angle bisectors all meet at one flat-foldable vertex.

The molecules fold into shapes we call *starfish*. By reversing exactly one tangency-point valley fold into a mountain, a starfish becomes a flat folding that we call a *book folding*.

[1] The power distance from a point to a circle is the square of the usual distance minus the radius of the circle squared. For points outside the circle it is the same as the tangential distance to the circle squared.

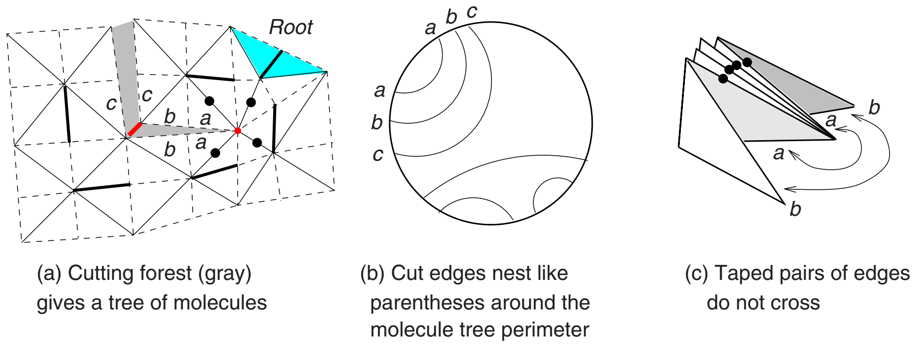

Root
c
c
b
a
b
a
a
b
c
a
b
c
b
a
a
b
(a) Cutting forest (gray) gives a tree of molecules
(b) Cut edges nest like parentheses around the molecule tree perimeter
(c) Taped pairs of edges do not cross

Fig. 4. The algorithm for folding a patch (topological disk) of molecules into a book uses a cutting forest (a), spanning interior molecule corners (red), to define a rooted tree of adjoining molecules. Tangency-point edges from child to parent (bold line segments) are reversed from valley to mountain so that each starfish turns into a book folding, tucked into an "armpit" of its parent starfish. The cutting-forest edges are joined ("taped") back together (c) after folding. There is no pair of crossed tapings, because tapings nest like parentheses in a tour (b) around the boundary of the tree of molecules.

Definition 4. *A* **book folding** *is a flat folding of a metric PL topological disk. The boundary edges of the topological disk are embedded on a common line called the* **axis**, *and all share a common endpoint on the axis. All polygons of the topological disk lie on the same side (half-plane) of the axis. Each polygon has one edge along another common line called the* **spine**, *perpendicular to the axis, and all polygons lie on the same side of the spine.*

Lemma 1 (Bern et al). *Any simple polygon can be realized by a book folding.*

Proof. As shown in Figure 4(a), we cut along a forest of molecule-boundary edges, in order to form a tree of molecules. The cutting forest F spans the molecule corners interior to the polygon, shown red in 4(a). Each connected component of F must also touch the boundary of the polygon, so that after cutting along the forest edges, all molecule corners touch the exterior face, and molecules form a tree. We root the tree arbitrarily and perform a pre-order traversal. As we traverse the tree, we reverse each tangency-point edge from child to parent from valley to mountain, so that each starfish closes up to become a book folding, nested spine to spine within its parent. The blue faces in 4(a) form the "cover" of the book. Finally the cutting-forest edges are rejoined along the "bottoms of pages" in post-order traversals of the cutting-forest components. □

Our previous paper [3] also claimed that any polygon with holes can be realized by a book folding, but the proof given there only works for the case of simple polygons. The one-straight-cut problem for a simple polygon strictly interior to the sheet of paper resembles the book-folding problem for a polygon with a hole, because we (somewhat unnecessarily) treat the boundary of the paper as a connected component of the polygon. We form molecules both interior and exterior to the polygon, and we must be sure that the tapings of exterior molecules do not cross the tapings of interior molecules. The previous paper positioned all the molecule boundaries on the same axis, with interior molecules pointing down and exterior molecules pointing up; without further restrictions this construction may force crossed tapings.

To repair the construction, we first require that the tree of molecules respect the containment relations of boundary components. Thus the molecules interior to a simple polygon, strictly interior to the sheet of paper, must form a proper subtree of the tree of molecules. And if the polygon contains a hole within an island within a hole, then each successive level of molecules must form a proper subtree within the tree above it. We surround each boundary component with a ribbon of width ϵ as in [3] or Figure 5. The deepest molecules then fold to books as in Figure 4(c). The second-deepest molecules fold to books containing the deepest molecules as "chapters", contiguous set of flaps within the larger books. The ribbons offset the shared axis of the deepest molecules so that it is ϵ above (and parallel to) the axis of the second-deepest molecules. We continue in this manner, building books within books, with each level of containment offset from the previous one by ϵ, something like a layered wedding cake.

Now in order to cut all boundary components at once—and nothing else—we must fold the interiors of the molecules out of the way of the axes. To do this, we reduce the heights of the molecules so that they are all smaller than ϵ, using the technique shown in Figure 6. Now the ribbons determine the height of the entire folding (which will be about ϵ for a simple polygon, 2ϵ for a polygon with simple-polygon holes, and so forth). Finally, for the one-straight-cut problem, we can fold the entire book using folds parallel to the axes in order to bring all the axes to a single cutting line.

This repaired construction also gives a (theoretical) way to perform an origami salami magic trick. In this trick, the first straight cut produces a hole in the shape of a silhouette of George Washington, the second straight cut (salami thickness ϵ away) produces a small John Adams, the third cut a still smaller Thomas Jefferson, and so forth, until we reach the tiniest president of all.

4 Extending the Construction

Lemma 1 generalizes almost immediately to the case of metric PL topological disks. The flat disk-packing disks are replaced by geodesic disks, which in the case of metric PL manifolds are simply unions of interior-disjoint sectors of Euclidean disks with common center and radius. A geodesic disk centered at a vertex of the PL manifold may have nonzero curvature, meaning that the sum of the sector angles may be more or less than 360°, but this makes no difference to the construction, as the angles appear only at the tips of starfish arms.

The case of a metric PL topological sphere is similarly easy. We puncture the sphere by cutting an edge or path e into a slit, so that the sphere becomes a topological disk D. We root all components of the cutting forest at e so that

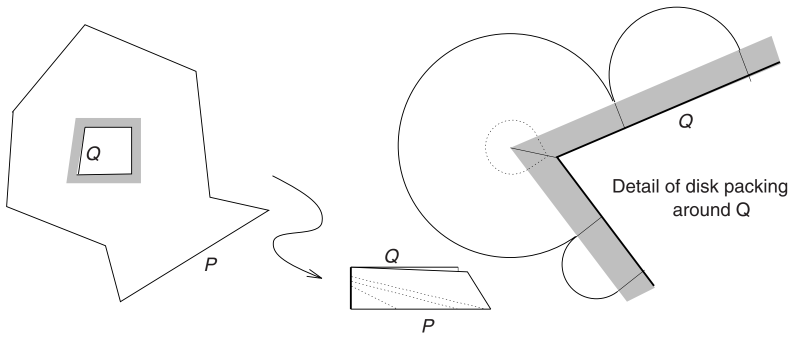


Fig. 5. A polygon P with a hole Q can be folded so that P forms one axis and Q forms another, and the interior of P lies between the two axes. The distance between the axes is the width of the ribbon (gray) around Q. Disk radii can be smaller than the ribbon width, as shown by the dotted disk on the right.

the two sides of e are outermost (that is, first and last "chapters") in the book folding of D. Path e will be spread out over a number of flaps (starfish arms), as many flaps as there are disks along e in the disk packing. A final sequence of tapings, as in Figure 4(c), joins the two sides of e and completes the embedding of the topological sphere. The complexity of this construction (that is, the number of vertices, edges, and faces) is bounded by a constant times the number of molecules, so in the case of a topological sphere with n triangular faces of bounded aspect ratio, our flat folding has complexity $O(n)$.

When we move up to a torus, we hit a snag. Cutting the torus into a tree of molecules requires a cycle of cuts, rather than just a forest of cuts, and no matter how we cut, the book folding of the resulting molecules always requires a crossed pair of tapings along the axis in order to recover the topology of the torus. (This is because any polygonal schema for the torus, including our tree of molecules, requires a crossed pair around its boundary.) We can, however, easily embed a flat torus as a flat folding: fold a square of paper in half, tape it into a flat tube, fold the tube perpendicularly to the first fold, and tape the two inner free edges together and the two outer free edges together. This simple experiment reveals the key to solving the general case: we need a new axis in order to form a handle. (This experiment also reveals what goes wrong with non-orientable manifolds: a Klein bottle would require crossed tapings, each inner free edge to the opposite outer free edge.) With this observation, we are now ready to argue our main result. The following lemma is a restatement of classical results; algorithms were given by Erickson and Har-Peled [10].

Lemma 2. *Any compact, orientable, metric, PL 2-manifold M of genus g admits a set of g disjoint cycles, such that cutting along these cycles gives a PL 2-manifold with boundary, homeomorphic to a sphere with $2g$ holes. The holes form g pairs, each pair consisting of two oppositely-oriented congruent polygons.*

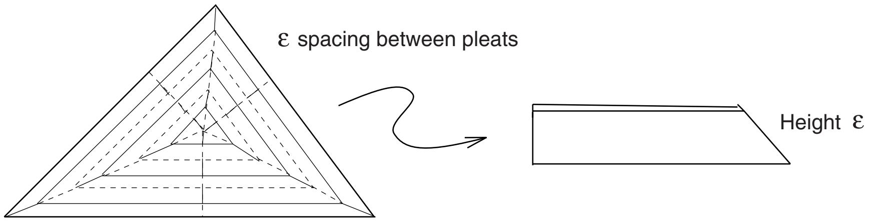


Fig. 6. The height of a book folding can be made arbitrarily small with pleat folds

The next lemma folds a polygon with holes into a book with two axes. The complexity of the folding increases to the area of the polygon divided by its minimum feature size.

Lemma 3. *A polygon with holes can be embedded as a book folding so that the outer boundary embeds to one axis, the hole boundaries all embed to a parallel axis, and the interior of the polygon embeds between the two parallel axes.*

Proof. As shown in Figure 5, we surround each hole with a thin "ribbon" (offset polygon) of width ϵ, where ϵ is smaller than polygon's minimum feature size (the minimum distance between non-adjacent edges). In the disk packing step of the construction, we pack the polygon minus the ribbons, and then project the tangency-point folds from the molecules perpendicularly across the ribbons. Thus the interior boundaries of the ribbons fold to the axis, and the boundaries of the holes fold to a parallel line, a new axis, distance ϵ from the original axis. By pleating down all the molecules (Figure 6) to have height less than ϵ, we can ensure that the interior of P embeds between the two parallel axes in the book folding. Alternatively we can use small disks, with radius less than $\epsilon/2$, to ensure that all molecules have height less than ϵ. □

Proof Sketch for Theorem 1. Let M be a compact, orientable, genus-g, PL 2-manifold without boundary. If necessary, we finely subdivide the faces of M in order to enable all subsequent steps. Using Lemma 2, we cut M's handles. Then as in the case of a topological sphere, we open one more path e to serve as the outer boundary, thereby obtaining a PL manifold M' homeomorphic to a disk with g pairs of holes, one pair for each handle loop. See Figure 7.

We surround each hole by a ribbon of suitable width $\epsilon > 0$. We use the disk-packing algorithm to break M' into triangle and quadrangle molecules, with molecules bordering the ribbons sending folds perpendicularly across the ribbons as in Figure 5. We impose some further requirements on the disk packing. (1) Each hole must be connected to its paired hole by a path of tiny molecules, meaning ones that fold to starfish of height less than ϵ, and these paths of tiny molecules must be disjoint. (2) We surround each ribbon by a ring of tiny molecules. This step requires disks smaller than ϵ. At corners of holes, folds from such tiny disks meet inside the ribbon at angle bisectors rather than crossing the ribbon. (See the dotted disk in Figure 5.) (3) Each hole and its paired hole must

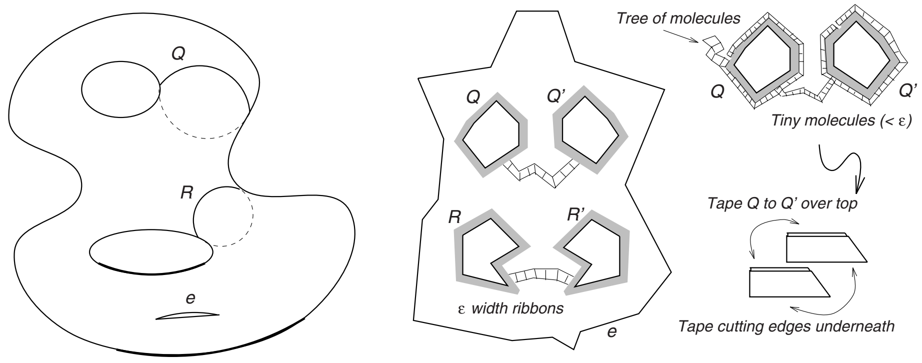


Fig. 7. A metric PL 2-manifold of genus g can be cut into a PL topological disk with g pairs of mirror-image holes. In our flat folding algorithm, an initial step surrounds the holes with ribbons (gray), and then the disk packing step surrounds and connects paired ribbons with tiny molecules. The cutting forest does not cross the paths of tiny molecules, except to break the rings around the holes. Thus paired holes appear adjacent to each other ("successive chapters") in the book folding of Figure 4(c), and the taping of the Q, Q' pair does not block the taping of the R, R' pair.

have matching disk packings, so that the hole and its paired hole are (mirror-image) congruent polygons, even after the tangency points of disks are projected perpendicularly across the ribbons.

We also constrain the cutting forest. The cutting forest does not cut the paths of tiny molecules, and cuts each ring of tiny molecules only once, so that each pair of holes appears within its own proper subtree of the tree of molecules. Thus a tour around the perimeter of the tree of molecules as in Figure 4(b) visits each hole and its paired hole in successsion, so no pairs cross (or even nest). Lemma 3 now folds M' so that cutting-forest edges all appear along one axis, and hole boundaries all appear along a second parallel axis. The construction is such that cutting-forest edges can be taped across the bottom of the book folding and holes can be taped across the top of the book folding, without any crossed pairs of tapings. Now a final taping closes the puncture path e. $\square$

5 Warping the Flat Folding into an Embedding

In this section, we show how to transform a flat folding of a PL 2-manifold into a "nearly isometric" PL embedding of the PL 2-manifold in $\mathbb{E}^3$. As elsewhere in this paper, we do not give a continuous deformation, but simply show how to construct the embedding given a flat folding.

Proof Sketch for Theorem 2: We fatten each edge of the flat folding into a narrow channel, of width less than ϵ/n, where $\epsilon > 0$ is the amount of distance stretching we are allowed, and n is the number of edges in the flat folding. We conceptually cut out a small disk around each vertex. We bend the channels vertically

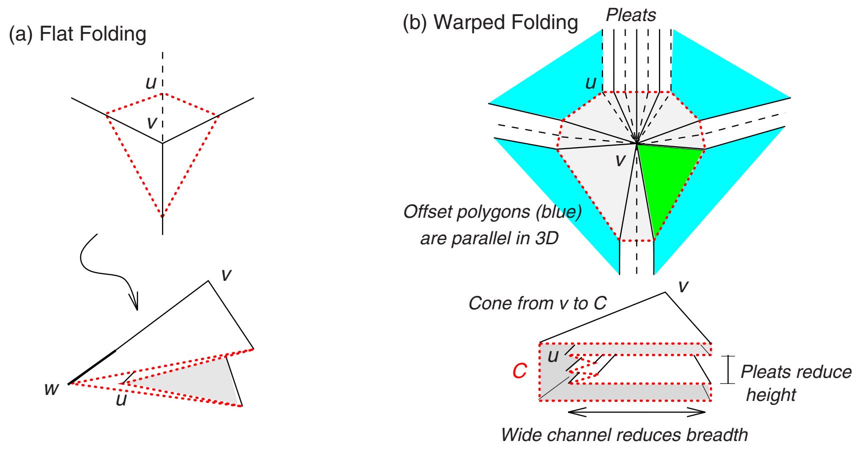


Fig. 8. To warp the flat folding shown in (a) into an embedding in $\mathbb{E}^3$, we shrink the faces and fatten the edges so that outside the neighborhood of vertices, faces are parallel and closely spaced. Pleats within the channels (fattened edges) control the spacing between parallel faces. The width of channels controls the breadth of faces to avoid interpenetration. We attach vertices to the parallel faces and channels with a cone as shown in (b).

(perpendicular to the plane of the flat folding) to separate the faces of the flat folding. We pleat the channels to control the spacing between parallel faces, taking care to avoid interpenetration, as shown in Figure 8(b). (See Pak [17] for a similar construction.) Finally we attach each vertex to the embedding with a cone of triangular faces (shown green in Figure 8(b)).

There is one more possible interpenetration not shown in Figure 8. A vertex tucked inside another vertex in the flat folding may try to cross to the outside in the embedding. We can solve this problem with more pleats, in annular rings around the inner vertex as in Figure 6, to move the inner vertex back inside.

The embedding just sketched is not isometric, because the attachment of the vertices to the rest of the embedding warps distances slightly. For example, vertex u in Figure 8 where the cone joins to rest of the embedding has angle sum less than 360°, but it would necessarily have curvature zero in an isometric embedding. (The edge channels need not warp distances; these can be created through an isometric deformation as in [17].) □

6 Discussion

Theorem 1 generalizes in various directions. For example, essentially the same proof holds for the case of PL 2-manifolds with boundary. Krat *et al.* generalize the result still further, stating their theorem for the more general case of "polyhedral spaces", which allows non-manifold topology, such as three triangles meeting at an edge. Our proof should also generalize to this case in a straightforward way. Another generalization would be to polyhedral knots and links in $\mathbb{E}^3$. We believe that the techniques given here are sufficient to prove that such inputs can be isometrically re-embedded as flat foldings, preserving all topology.

We close with some open questions. Can we extend the techniques of Section 5 to give isometry at vertices too? This would give an alternate proof of a simple form of the theorem of Burago and Zalgaller: any PL 2-manifold can be isometrically embedded in $\mathbb{E}^3$. Assuming we can do this, can we then further extend our proof to the stronger statement of Burago-Zalgaller, showing isometric approximation of any submetric embedding? Or to show that any flat folding can be continuously and isometrically "opened up" into an embedding in $\mathbb{E}^3$? Or in the reverse direction, and requiring more than an infinitesimal change in volume, does every polyhedron admit a bending [17,19] that continuously and isometrically reduces its volume to zero? Finally and most importantly, does the Burago-Zalgaller theorem generalize to higher dimensions?

References

1. Bern, M., Mitchell, S., Ruppert, J.: Linear-size nonobtuse triangulation of polygons. Disc. Comput. Geom. 14, 411–428 (1995)
2. Bern, M., Hayes, B.: The complexity of flat origami. In: Proc. 7th ACM-SIAM Symp. Disc. Algorithms, pp. 175–183 (1996)
3. Bern, M., Demaine, E., Eppstein, D., Hayes, B.: A disk-packing algorithm for an origami magic trick. In: E. Lodi, L. Pagli, N. Santoro, (eds.) Preliminary version: Fun with Algorithms, pp. 32–42, Carleton Scientific (1999); Also: Hull, T., Peters, A.K. (ed.) Origami^3, pp. 17–28 (2002)
4. Burago, Y.D., Zalgaller, V.A.: Polyhedral realizations of developments (Russian). Vestnik Leningrad. Univ. 15, 66–80 (1960)
5. Burago, Y.D., Zalgaller, V.A.: Isometric piecewise linear embedding of two-dimensional manifolds with a polyhedral metric in $\mathbb{R}^3$. St. Petersburg Math. Journal 7, 369–385 (1996)
6. Connelly, R.: A flexible sphere. Math. Intelligencer 1, 130–131 (1978)
7. Connelly, R., Sabitov, I., Walz, A.: The bellows conjecture. Contributions to Algebra and Geometry 38, 1–10 (1997)
8. Demaine, E., Demaine, M., Lubiw, A.: Flattening polyhedra. (Manuscript 2001)
9. Demaine, E., O'Rourke, J.: Geometric Folding Algorithms: Linkages, Origami, and Polyhedra. Cambridge University Press, Cambridge (2007)
10. Erickson, J., Har-Peled, S.: Optimally cutting a surface into a disk. In: Symp. Comp. Geometry (2002)
11. Hull, T.: On the mathematics of flat origamis. Congressus Numerantium 100, 215–224 (1994)
12. Krat, S., Burago, Y.D., Petrunin, A.: Approximating short maps by PL-isometries and Arnold's "Can you make your dollar bigger" problem. In: Fourth International Meeting of Origami Science, Mathematics, and Education, Pasadena (2006)
13. Kuiper, N.H.: On C^1-isometric imbeddings I. Proc. Nederl. Akad. Wetensch. Ser. A 58, 545–556 (1955)

14. Lang, R.J.: Origami Design Secrets: Mathematical Methods for an Ancient Art, A.K. Peters (2003)
15. Nash, J.F.: C^1-isometric imbeddings. Annals of Mathematics 60, 383–396 (1954)
16. Nash, J.F.: The imbedding problem for Riemannian manifolds. Annals of Mathematics 63, 20–63 (1956)
17. Pak, I.: Inflating polyhedral surfaces. Department of Mathematics. MIT Press, Cambridge (2006)
18. Zalgaller, V.A.: Isometric immersions of polyhedra. Dokladi Akademii, Nauk USSR, 123(4) (1958)
19. Zalgaller, V.A.: Some bendings of a long cylinder. J. Math. Soc. 100, 2228–2238 (2000)

Simplifying 3D Polygonal Chains Under the Discrete Fréchet Distance*

Sergey Bereg[1], Minghui Jiang[2], Wencheng Wang[3], Boting Yang[4], and Binhai Zhu[5]

[1] Department of Computer Science, University of Texas at Dallas, Richardson, TX 75083, USA
besp@utdallas.edu

[2] Department of Computer Science, Utah State University, Logan, UT 84322-4205, USA
mjiang@cc.usu.edu

[3] Institute of Software, Chinese Academy of Sciences, Beijing 100080, China
whn@ios.ac.cn

[4] Department of Computer Science, University of Regina, Regina, Saskatchewan, S4S 0A2, Canada
boting@cs.uregina.ca

[5] Department of Computer Science, Montana State University, Bozeman, MT 59717-3880, USA
bhz@cs.montana.edu

Abstract. A well-known measure to characterize the similarity of two polygonal chains is the famous Fréchet distance. In this paper, for the first time, we consider the problem of simplifying 3D polygonal chains under the discrete Fréchet distance. We present efficient polynomial time algorithms for simplifying a single chain, including the first near-linear $O(n \log n)$ time exact algorithm for the continuous min-# fitting problem. Our algorithms generalize to any fixed dimension $d > 3$. Motivated by the ridge-based model simplification we also consider simplifying a pair of chains simultaneously and we show that one version of the general problem is NP-complete.

1 Introduction

Simplifying polygonal chains is a well-studied problem, especially in the plane (and occasionally in 3D and higher dimensional spaces). In short, the problem is to simplify a given chain A with n vertices into A' such that A and A' are close and $|A'| \ll n$. For instance, in 3D we face the problem of simplifying optic nerves in medical studies and simplifying river networks in GIS [21]. Most of the previous researches are focused on simplifying 2D polygonal chains [6,7,10,14,15,16,20,22], with the notable exception of [10,4]. Readers are referred to [4] for a list of complete references on simplifying polygonal chains in all dimensions. In this paper, we first follow the traditional work on simplifying a polygonal chain (a polyline or simply a chain) in 3D, but under a relatively new measure — the discrete Fréchet distance.

Fréchet distance was first defined by Maurice Fréchet in 1906 [11]. While known as a famous distance measure in the field of mathematics (more specifically, abstract

* This research is supported by the NSERC grant 261290-03 and grant A13501 at Utah State University.

spaces), it was Alt and Godau who first applied it in measuring the similarity of polygonal curves in early 1990s [2,3].

In 1994, Eiter and Mannila defined the *discrete Fréchet distance* between two polygonal chains A and B (in any fixed dimensions) [9]. Recently, Jiang, Xu and Zhu applied the discrete Fréchet distance in aligning the backbones of proteins (which is called the *protein structure-structure alignment* problem) [17]. In fact, in this application the discrete Fréchet distance makes more sense as the backbone of a protein is simply a polygonal chain in 3D, with each vertex being the alpha-carbon atom of a residue. So if the (continuous) Fréchet distance is realized by an alpha-carbon atom and some other point which does not represent an atom, it is not meaningful biologically. Jiang, *et al.* showed that given two planar polygonal chains the minimum discrete Fréchet distance between them, under both translation and rotation, can be computed in polynomial time. They also applied some ideas therein to design an efficient heuristic for the original protein structure-structure alignment problem in 3D.

Very recently, the discrete Fréchet distance was used to align protein backbones locally. It was shown that given many proteins finding such a local alignment is NP-complete, but when a constant number of chains are given then the problem is polynomially solvable [24]. Notice that finding local alignment between two proteins (or 3D chains) A, B is different from simplifying them. Loosely speaking, a local alignment is to find a subsequence A' of A and a subsequence B' of B such that A' and B' are very close. But A' and A (hence B' and B) could have a huge difference.

While one can claim that the discrete Fréchet distance is a special case of the (continuous) Fréchet distance, the use of discrete Fréchet distance, in many situations, makes more sense. Firstly, the discrete Fréchet distance is more efficient to compute. For instance, Godau used the Fréchet distance to approximate polygonal chains using vertices of the original curve [12] (i.e., *discrete* fitting in our terminology). The running time of his algorithms are $O(n^3)$ for min-# fitting and $O(n^4 \log n)$ for min-ϵ fitting, while using the discrete Fréchet distance these bounds are $O(n^2)$ and $O(n^3)$ respectively. Secondly, as we just mentioned, in many biological applications (continuous) Fréchet distance does not make any sense.

Now coming back to the second motivation of our research — ridge-based geometric model simplification. A ridge is a critical 3D polygonal chain on a surface whose projection on the XY-plane is a simple (planar) polygonal chain. Ridge simplification and approximation is an interesting problem in geometric modeling, approximation and 3D geometric compression. We refer to Fig. 1 for an example. We have identified two ridges P and Q and wish to simplify them into P' and Q' so as to have a simplified surface between P' and Q'. In this case, however, we not only want P and P' (Q and Q') to be close, but also want that P' and Q' are close. Otherwise, as can be seen from Fig. 1 (II), the large discrete Fréchet distance between P', Q' induces some long skinny triangle anchored at the vertex y. On the other hand, when we simplify P into P'' such that P'' and Q' have a smaller discrete Fréchet distance then the long skinny triangle disappears (Fig. 1 (III) and (IV)).

It turns out that this problem of simultaneously simplifying a pair of chains is much more difficult than the protein local alignment problem. We show that a special case, where we measure the similarity between P, P' (and between Q, Q') using the

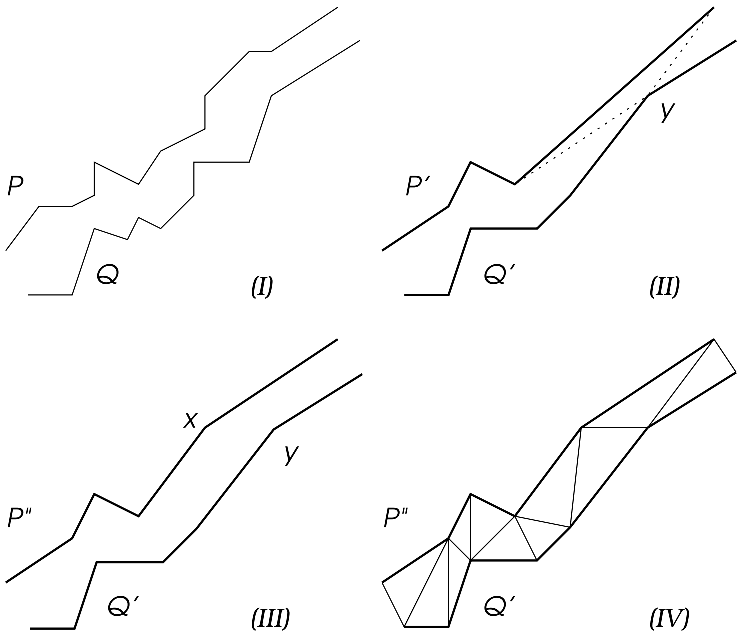


Fig. 1. Large discrete Fréchet distance implies long skinny triangles

Hausdroff distance between vertices while measuring the similarity between P', Q' using the discrete Fréchet distance, is NP-complete. This implies that we should better add Steiner points in this application, which is a popular way to improve the quality of a mesh.

2 Preliminaries

Given two polygonal chains A, B with $|A| = k$ and $|B| = l$ respectively, we aim at aligning the similarity of A and B (sometimes under translation and rotation) such that their distance is minimized under certain measure. Among the various distance measures, the Hausdorff distance is known to be better suited for matching two point sets than for matching two polygonal chains; the (continuous) Fréchet distance is a superior measure for matching two polygonal chains, but it is not quite easy to compute especially when translation/rotation are allowed.

Let X be the Euclidean space R^3; let $d(a, b)$ denote the Euclidean distance between two points $a, b \in X$. The (continuous) Fréchet distance between two parametric curves $f : [0, 1] \rightarrow X$ and $g : [0, 1] \rightarrow X$ is

$$\delta_{\mathcal{F}}(f, g) = \inf_{\alpha,\beta} \max_{s\in[0,1]} d(f(\alpha(s)), g(\beta(s))),$$

where α and β range over all continuous non-decreasing real functions with $\alpha(0) = \beta(0) = 0$ and $\alpha(1) = \beta(1) = 1$.

Imagine that a person and a dog walk along two different paths while connected by a leash; they always move forward, though at different paces. The minimum possible length of the leash is the Fréchet distance between the two paths. To compute the Fréchet distance between two polygonal curves A and B (in the Euclidean plane) of $|A|$ and $|B|$ vertices, respectively, Alt and Godau [2] presented an $O(|A||B|\log^2(|A||B|))$ time algorithm. Later this bound was reduced to $O(|A||B|\log(|A||B|))$ time [3].

We now define the discrete Fréchet distance following [9].

Definition 1. *Given a polygonal chain (polyline) in 3D $P = \langle p_1, \ldots, p_k\rangle$ of k vertices, an m-**walk** along P partitions the path into m (disjoint) non-empty subchains $\{\mathcal{P}_i\}_{i=1..m}$ such that $\mathcal{P}_i = \langle p_{k_{i-1}+1}, \ldots, p_{k_i}\rangle$ and $0 = k_0 < k_1 < \cdots < k_m = k$.*

*Given two 3D polylines $A = \langle a_1, \ldots, a_k\rangle$ and $B = \langle b_1, \ldots, b_l\rangle$, a **paired walk** along A and B is an m-walk $\{\mathcal{A}_i\}_{i=1..m}$ along A and an m-walk $\{\mathcal{B}_i\}_{i=1..m}$ along B for some m, such that, for $1 \le i \le m$, either $|\mathcal{A}_i| = 1$ or $|\mathcal{B}_i| = 1$ (that is, either $\mathcal{A}_i$ or $\mathcal{B}_i$ contains exactly one vertex). The **cost** of a paired walk $W = \{(\mathcal{A}_i, \mathcal{B}_i)\}$ along two paths A and B is*

$$d_F^W(A,B) = \max_i \max_{(a,b)\in\mathcal{A}_i\times\mathcal{B}_i} d(a,b).$$

*The **discrete Fréchet distance** between two polylines A and B is*

$$d_F(A,B) = \min_W d_F^W(A,B).$$

*The paired walk that achieves the discrete Fréchet distance between two paths A and B is also called the **Fréchet alignment** of A and B.*

Consider the scenario in which the person walks along A and the dog along B. Intuitively, the definition of the paired walk is based on three cases:

1. $|\mathcal{B}_i| > |\mathcal{A}_i| = 1$: the person stays and the dog moves forward;
2. $|\mathcal{A}_i| > |\mathcal{B}_i| = 1$: the person moves forward and the dog stays;
3. $|\mathcal{A}_i| = |\mathcal{B}_i| = 1$: both the person and the dog move forward.

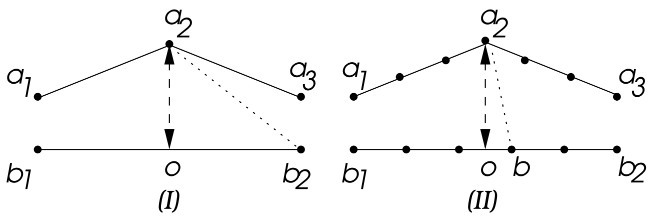


Fig. 2. The relationship between the discrete and continuous Fréchet distances

Eiter and Mannila presented a simple dynamic programming algorithm to compute $d_F(A,B)$ in $O(|A||B|) = O(kl)$ time [9]. The recent result of Jiang, *et al.* shows that in 3D the minimum discrete Fréchet distance between A and B under translation can be computed in $O(k^4 l^4 \log(k+l))$ time, and under both translation and rotation it can be computed in $O(k^7 l^7 \log(k+l))$ time using the ideas presented in [23,17]. They are significantly faster than the corresponding bounds for the continuous Fréchet distance. In 3D, Wenk showed that given two chains with sum of length $N = k + l$, the minimum Fréchet distance between them can be computed in $O(N^{3f+2} \log N)$ time, where f is the degree of freedom for moving the chains [23]. So with translation alone this minimum Fréchet distance can be computed in $O(N^{11} \log N)$ time, and when both translation and rotation are allowed the corresponding minimum Fréchet distance can be computed in $O(N^{20} \log N)$ time [23].

We comment that while the discrete Fréchet distance could be arbitrarily larger than the corresponding continuous Fréchet distance (e.g., in Fig. 2 (I), they are $d(a_2, b_2)$ and $d(a_2, o)$ respectively), by adding sample points on the polylines, one can easily obtain a close approximation of the continuous Fréchet distance using the discrete Fréchet distance (e.g., one can use $d(a_2, b)$ in Fig. 2 (II) to approximate $d(a_2, o)$). This fact was also pointed out in [9]. Moreover, the discrete Fréchet distance is a more natural measure for matching the geometric shapes of biological sequences such as proteins. As we mentioned in the introduction, in such applications, the continuous Fréchet distance does not make much sense to biologists.

3 Min-# Fitting with a Given Error Bound

In this section, we discuss min-# fitting (simplification) with a given error bound; namely, given a chain A and an error bound δ, we want to simplify A into another chain C with the minimum number of vertices such that $d_F(A, C) \leq \delta$. This is a traditional problem on polygonal chain simplification, except that almost all the previous work are all focused on different measures, for instance, the *ϵ-tolerance zone* error measure [4]. With the (continuous) Fréchet error measure, for the 2D problem, Guibas, *et al.* obtained an $O(n^2 \log^2 n)$ time algorithm [13]. We show that in 3D this problem can be solved in $O(n \log n)$ time using the discrete Fréchet error measure.

Let $A = A[1..n]$ be the given chain A of n vertices. Let $A[i..j]$ be the (contiguous) subchain of A starting from the index i to the index j. We call $A[1..i]$ a prefix of A. Let $A \circ B$ be the concatenation of two chains A and B (by connecting the last vertex of A and the first vertex of B).

Given a discrete Fréchet distance (error) δ, we wish to simplify $A = A[1..n]$ using a simple chain C such that $d_F(A, C) \leq \delta$ and the size of C is minimized. Apparently, we have two cases; the vertices of C could be arbitrary or could only be the vertices of A. We call them the *continuous* and *discrete* cases respectively. It turns out that the two cases can be solved differently, with the greedy method and dynamic programming respectively. We cover the continuous case first.

For the continuous case, we can see that following the definition of the discrete Fréchet distance, the paired walk between A and C, $\mathcal{A}_i$ and $\mathcal{C}_i$, must satisfy the property that $|\mathcal{A}_i| \geq |\mathcal{C}_i| = 1$ for all i (otherwise, we can simply delete some vertices in C to obtain a better simplification). Then, if $|\mathcal{A}_i| \geq |\mathcal{C}_i| = 1$ for all i but $\mathcal{A}_i$ is not maximal, we can merge $\mathcal{A}_i$ with a prefix y of $\mathcal{A}_{i+1}$ which is at most distance δ away from $\mathcal{C}_i$ to obtain a new $\mathcal{A}'_i = \mathcal{A}_i \circ y$, without affecting the size of C.

So we can use a greedy method to find the first *breakpoint* (the largest index j) such that (all the vertices on) $A[1..j]$ can be covered by a ball centered at a point b_j with radius δ, $\mathcal{B}(b_j, \delta)$, but $A[1..j+1]$ cannot be covered by any ball of radius δ. Given $A[1..j]$, we can decide whether it can be covered by $\mathcal{B}(b_j, \delta)$ in $O(n)$ time. In fact, one can simply compute the smallest enclosing ball for the vertices of $A[1..j]$ to locate the point b_j in linear time [19]. So b_j will be the first vertex on the simplified chain C. Repeating this greedy process at most $m^* = O(n)$ times, where m^* is the optimal solution value for the problem, we can obtain a chain C with m^* vertices. Because we are in 3D, a simple perturbation on the vertices of A can easily eliminate any self-intersection in C. It is easy to see that this greedy method solves the continuous min-# fitting problem in $O(n^2)$ time.

We can use binary search to repeatedly find the breakpoints, so the problem can in fact be solved in $O(m^* n \log n)$ time (which seems tough to beat at the first sight). However, we present the following Algorithm $CMN(A[1..n], \delta)$ which solves the problem in $O(n \log n)$ time. We use the *doubling search* method, which has been used before in [18,5,1].

Algorithm. $CMN(A[1..n], \delta)$

(1) Search with $t = 1, 2, 3, \ldots$ the first t such that $A[1..2^{t-1}]$ can be covered by a ball with radius δ but $A[1..2^t]$ cannot. Then find the first breakpoint k_1 in $A[2^{t-1}..2^t]$ using binary search.

(2) Repeat the above process on $A[k_1+1..n]$ to compute all of the m^*-1 breakpoints.

Theorem 1. *The CMN procedure solves the continuous min-# fitting problem for a 3D polyline under the discrete Fréchet distance in $O(n \log n)$ time and $O(n)$ space.*

Proof. Clearly k_1 can be found in $O(k_1 \log k_1)$ time. This is due to that when k_1 is in $A[2^{t-1}..2^t]$, then 2^t is at most $2k_1$. Let $n_1, n_2, \ldots, n_{m^*}$ be the sizes of the subchains determined by the $m^* - 1$ breakpoints (note that $n_1 = k_1$). The overall running time of CMN is

$$O(n_1 \log n_1) + O(n_2 \log n_2) + \cdots + O(n_{m^*} \log n_{m^*}),$$

which is $O(n \log n)$, due to $n_1 + n_2 + \cdots + n_{m^*} = n$. □

We remark that the greedy method in Theorem 1 is similar to that in [4,1]. In [1], Agarwal, *et al.* considered the similar discrete problem of min-# fitting (simplification) with a given continuous Fréchet error. An approximation algorithm, which returns at most twice the size of an optimal simplified curve within half of the error, was presented in [1]. An open question on a better near-linear time approximation was also raised in [1]. The above theorem shows that, using the discrete Fréchet distance, the continuous version of the problem can be solved exactly with a near-linear time algorithm. For the same discrete problem, we will use the discrete Fréchet measure and present an $O(n^2)$ time solution to solve it exactly.

Regarding the discrete min-# fitting problem under the discrete Fréchet distance, i.e., when the vertices of the simplified chain, C', must come from A, it turns out that there is a dramatic difference compared with the continuous case. We refer to Fig. 3, in which we have a chain A with five vertices, and when $\delta = 1$ the optimal simplified chain

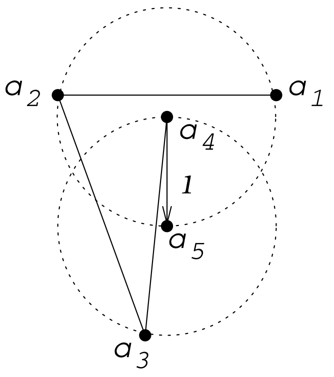


Fig. 3. Given a chain A, when $\delta = 1$, the optimal simplified chain is $\langle a_4, a_5 \rangle$

is $C' = \langle a_4, a_5 \rangle$. In other words, $A[1..2]$ is not covered by a vertex on the subchain $A[1..2]$. This is also completely different from the situation in [4].

Due to the unfavorable non-local property of the discrete problem, we solve the problem using a dynamic programming method. Without loss of generality, we only show how to compute the optimal size of C'. The actual chain C' can be constructed easily by modifying the algorithm.

Define $T[i, s]$ as the maximum index $j, j \geq i$, such that the ball with radius δ and centered at $A[s]$ covers $A[i..j]$. $T[i, s] = i-1$ if $d(A[i], A[s]) > \delta$; otherwise, $T[i, s] = T[i+1, s]$. For each s, $T[-, s]$ can be computed in $O(n)$ time. So $T[-, -]$ can be computed in $O(n^2)$ time.

Define $N[i, s]$ as the minimum number of balls with radius δ and centered at $A[s..n]$ that cover $A[i..n]$. We have

$$N[i, s] = \min\{N[i, s+1], N[j+1, s+1] + 1\},$$

where $j = T[i, s]$. The boundary cases when $i = n$ or $s = n$ can be handled easily. So $N[-, -]$ can be computed in $O(n^2)$ time and space.

Theorem 2. *The discrete min-# fitting problem for a 3D polyline under the discrete Fréchet distance can be solved in $O(n^2)$ time and $O(n^2)$ space.*

We remark that using the continuous Fréchet distance the discrete min-# fitting problem can be solved in $O(n^3)$ time [12]. In [1] whether this bound can be reduced was listed as an open problem. Our above theorem shows again that using the discrete Fréchet distance the problem can be solved more efficiently, in fact, in quadratic time.

4 Min-ϵ Fitting with m-chains

The *min-ϵ fitting with m-chains* problem is defined as follows. Given a 3D chain $A = \langle a_1, a_2, \ldots, a_n \rangle$ and a positive integer m, we wish to simplify A into a polyline $B = \langle b_1, b_2, \ldots, b_m \rangle$ such that $d_F(A, B)$ is minimized. Again, we have two cases: the *continuous* case (when the vertices of B are arbitrary ones) and the *discrete* case (when the vertices of B must come from A). We first show how to solve the continuous version of the problem using CMN as a subroutine.

Be reminded that in the continuous case, the vertices of B do not have to come from A. We first design a procedure $CME(A[i..j], m)$ which covers $A[i..j]$ with m balls of the smallest radius. Let $\delta(i, j)$ be the radius of the smallest ball covering $A[i..j]$.

Algorithm. $CME(A[i..j], m)$

```
    (1) If $i \geq j$ then return 0.
    (2) If $m = 1$ then return $\delta(i, j)$.
    (3) Find $k$ such that $CMN(A[i..j], \delta(i, k)) > m \geq CMN(A[i..j], \delta(i, k+1))$
and return $\min\{\delta(i, k+1), CME(A[k+1..j], m-1)\}$.
```

Apparently CME is a recursive procedure and we call $CME(A[1..n], m)$ the first time. We have the following theorem.

Theorem 3. *The continuous min-ϵ fitting problem under the discrete Fréchet distance can be solved in $O(mn \log n \log(n/m))$ time.*

Proof. We first sketch the correctness proof of CME. Recall that $\delta(i, j)$ is the radius of the smallest enclosing ball of $A[i, j]$. Denote by $\delta(i, j, k)$ the minimum radius of k uniform balls covering $A[i, j]$. The following properties are not difficult to prove.

Let x be the smallest index such that $A[x+1, n]$ can be covered by $m-1$ balls of radius $\delta(1, x)$; that is, $\delta(x, n, m-1) > \delta(1, x-1)$ and $\delta(x+1, n, m-1) \leq \delta(1, x)$. Then we have the recursion

$$\delta(1, n, m) = \min\{\delta(1, x), \delta(x, n, m-1)\}.$$

This corresponds to two cases in CMN: (1) Cover $A[1, x]$ with a ball of radius $\delta(1, x)$ and $A[x+1, n]$ with $m-1$ balls of radius at most $\delta(1, x)$; and (2) cover $A[1, x-1]$ with a ball of radius $\delta(1, x-1)$ and $A[x, n]$ with $m-1$ balls of radius $\delta(x, n, m-1)$.

Note that as we use CMN, which takes $O(n \log n)$ time, as a subroutine and we have to recurse CME m times, the crucial question is how to find k quickly at each recursion. A naive binary search would find each k in $O(\log n)$ time hence giving us a total running time of $m \times O(\log n) \times O(n \log n) = O(nm \log^2 n)$ time. However, we can use the same doubling search idea in Theorem 1 so that at the i-th recursion the corresponding k_i can be found in $O(\log n_i \times n \log n)$ time, for $i = 1, 2, ..., m$, where n_i is the size of the subchain covered by the i-th vertex of B (with optimal radius/error). So the running time of the algorithm is

$$\sum_{1 \leq i \leq m} O(\log n_i \times n \log n),$$

which is $O(mn \log n \log(n/m))$, due to that $\sum_{1 \leq i \leq m} n_i = n$. □

We now consider the discrete case, i.e., the vertices of B must come from A. In this problem, following Fig. 3 (when $m = 2$) we can again see that in the optimal solution $A[1..i]$ is not necessarily covered by a ball centered at a vertex on $A[1..i]$. Similar to the discrete min-# fitting problem, we again follow the dynamic programming method.

Define $R[i,j,s]$ as the minimum radius of a ball centered at $A[s]$ that covers $A[i..j]$. We have $R[i,j,s] = \max\{d(A[i], A[s]), R[i+1,j,s]\}$. So $R[-,j,s]$ can be computed in $O(n)$ time and the whole table $R[-,-,-]$ can be computed in $O(n^3)$ time.

Define $E[i,s,z]$ as the minimum radius of z uniform balls centered at $A[s..n]$ that cover $A[i..n]$. Define $J[i,s,z]$ as the minimum index $j \geq i$ such that $R[i,j,s] \geq E[j+1, s+1, z-1]$. $E[i,s,z]$ can be updated in two cases: (1) s is used as a center for a uniform ball, and (2) s is not used as a center for a uniform ball. Therefore,

$$E[i,s,z] = \min\{E[i,s+1,z], R[i,j,s], E[j,s+1,z-1]\},$$

where $j = J[i,s,z]$. Again, the boundary cases when $i = n$ or $s = n$ or $z = 0$ can be handled easily.

$E[-,-,-]$ can be computed in $O(mn^2)$ time given $J[-,-,-]$. Note that $E[-,-,z]$ depends on $J[-,-,z]$, and that $J[-,-,z]$ depends on $E[-,-,z-1]$. Therefore, for each z from 1 to m, we need to compute $J[-,-,z]$ before $E[-,-,z]$.

To compute $J[i-1,s,z]$, compare $d(A[i-1], A[s])$ with $R[i,j,s]$, where $j = J[i,s,z]$. If $d(A[i-1], A[s]) < R[i,j,s]$, then set $J[i-1,s,z]$ to j. Otherwise, use a sequential search to find the minimum $j' \leq j$ such that $R[i-1,j',s] \geq E[j'+1, s+1, z-1]$, then set $J[i-1,s,z]$ to j'. The time is $O(j-j'+1)$ for filling each $J[i-1,s,z]$, which adds up to $O(n)$ for $J[-,s,z]$. So $J[-,-,-]$ can be computed in $O(mn^2)$ time. The total running time for constructing $E[-,-,-]$ is $O(n^3) + O(mn^2) = O(n^3)$.

Theorem 4. *The discrete min-ϵ fitting problem under the discrete Fréchet distance can be solved in $O(n^3)$ time and $O(n^3)$ space.*

We comment that the running times in Theorem 1, Theorem 2 (when $m = o(n)$), Theorem 3 are all much faster than the corresponding ones for the ϵ-tolerance zone metric [4]. This might due to the strong 'ordering' property of the discrete Fréchet distance. However, we show in the next section that when we have to simplify a pair of chains simultaneously under the discrete Fréchet distance, one version of the general problem is even NP-complete. No such negative result is known, on any distance measure, in the previous research on chain simplification.

5 Simplifying a Pair of Chains Under the Discrete Fréchet Distance

As we have discussed in the introduction, in this section we investigate the problem of simplifying a pair of chains A, B into A', B' such that the vertices of A', B' must come from A, B respectively, $d'(A,A'), d'(B,B'), d_F(A',B')$ are all bounded. We will show that when $d'(-,-)$ is the Hausdorff distance between the vertices of two chains (denoted as $d_H(-,-)$ henceforth) then the problem for general 3D chains is NP-complete. This indicates that for ridge-based model simplification, we should use Steiner points to ensure the quality of the simplified surface.

Formally, the Chain Pair Simplification (CPS) problem is defined as follows.

Instance: Given a pair of 3D chains A and B in 3D, each with length $O(n)$, an integer K, and three real numbers $\delta_1, \delta_2, \delta_3$.

Problem: Does there exist a pair of chains A', B' each of at most K vertices such that the vertices of A', B' are from A, B respectively, and $d_1(A, A') \leq \delta_1, d_2(B, B') \leq \delta_2, d_F(A', B') \leq \delta_3$?

When $d_1 = d_2 = d_H$, we call the corresponding problem CPS-2H and when $d_1 = d_2 = d_F$, we call the corresponding problem CPS-3F. We have the following theorem.

Theorem 5. *The CPS-2H problem is NP-complete.*

Proof. It is easy to see that CPS-2H belongs to NP. We now reduce 3SAT to the CPS-2H problem. The idea of this reduction is from [8], even though over here we are handling a geometric problem.

Let $\phi = F_1 \bigwedge F_2 \bigwedge \cdots \bigwedge F_m$ be a conjunctive normal form, where each sub-formula F_i is a 3-disjunctive clause like $(x_2 \bigvee x_5 \bigvee \neg x_7)$. Assume that $x_1, x_2, \cdots, x_n$ are the boolean variables in the formula ϕ and each F_i cannot contain both x_k and $\neg x_k$ (otherwise F_i is already true and can be discarded). We construct a triple of points for each F_i as $p_{i1} = (i, i^2, 0), p_{i2} = (i, i^2, \epsilon), p_{i3} = (i, i^2, 2\epsilon)$, for some $0 < \epsilon < 0.1$. We then construct two chains A and B each with $4n-1$ vertices such that ϕ is satisfiable iff A and B can be simplified into A', B' each with $K = 2n-1$ vertices such that $d_H(A, A') \leq 2\epsilon$, $d_H(B, B') \leq 2\epsilon$ and $d_F(A', B') = 0$ (i.e., $\delta_3 = 0$ in our construction).

First we construct $n-1$ points $q_j = (j, 0, 0), 1 \leq j \leq n-1$. For each variable x_i in ϕ, we construct two sequences S_i and S_i^*. Let $F_{i_1}, \cdots, F_{i_u}$ be the clauses in ϕ that contain x_i, and let $F_{j_1}, \cdots, F_{j_v}$ be the clauses of ϕ that contain $\neg x_i$. Let $S_i = F_{i_1} \cdots F_{i_u} F_{j_1} \cdots F_{j_v}$ and $S_i^* = F_{j_1} \cdots F_{j_v} F_{i_1} \cdots F_{i_u}$. We next convert S_i (S_i^*) into a sequence of 3D points T_i (T_i^*), where each occurrence of $F_k (1 \leq k \leq m)$ in S_i or S_i^* corresponds to a unique point in $\{p_{kj} | j \leq 3\}$. Note that since F_k contains 3 literals, it appears in all S_i and S_i^* exactly three times. So from now on we assume that the three occurrences of p_{kj}'s are always in the order p_{k1}, p_{k2} and p_{k3} and with this in mind we will use p_k to simplify the presentation.

Let $A = \langle T_1, q_1, T_2, q_2, \cdots, q_{n-1}, T_n \rangle$ and $B = \langle T_1^*, q_1, T_2^*, q_2, \cdots, q_{n-1}, T_n^* \rangle$.

Assume that $x_1 = b_1, \cdots, x_n = b_n$ are assignments that make ϕ true. If $b_i = 1$, simplify both T_i and T_i^* to $T_i' = p_{i_1}, \cdots, p_{i_u}$ and $T_i^{*'} = p_{i_1}, \cdots, p_{i_u}$, respectively. If $b_i = 0$, simplify both T_i and T_i^* to $T_i' = p_{j_1}, \cdots, p_{j_v}$ and $T_i^{*'} = p_{j_1}, \cdots, p_{j_v}$, respectively. It is easy to see that $A' = \langle T_1', q_1, T_2', \cdots, T_{n-1}', q_{n-1}, T_n' \rangle$ is the same as $B' = \langle T_1^{*'}, q_1, T_2^{*'}, \cdots, T_{n-1}^{*'}, q_{n-1}, T_n^{*'} \rangle$ except that some p_j in T_i' are at most 2ϵ distance away. It is easy to see that $d_H(A, A') \leq 2\epsilon, d_H(B, B') \leq 2\epsilon$ and $d_F(A', B') = 0$; moreover, $K = 2n-1$.

Assume that A is simplified into A'' and B is simplified into B'' via removing some points in $\{p_{ij} | 1 \leq i \leq n, 1 \leq j \leq 3\}$ such that $d_H(A, A'') \leq 2\epsilon$, $d_H(B, B'') \leq 2\epsilon$, and $d_F(A'', B'') = 0$. Notice that the distance between p_{ij} and p_{kl} and the distance between q_i and q_k are at least one, as long as $i \neq k$. The condition that $d_H(A, A'') \leq 2\epsilon$, $d_H(B, B'') \leq 2\epsilon$ implies that we can only remove points in $\{p_{ij} | 1 \leq i \leq n, 1 \leq j \leq 3\}$ and we must leave at least one point in A'', B'' for each $p_{ij}, 1 \leq j \leq 3$. As F_i cannot contain both x_k and $\neg x_k$, on the subchain between q_r and q_{r+1} on A or B there is exactly one point p_z, for some z. Finally, as $K = 2n-1$, to make $d_F(A'', B'') \leq 2\epsilon$, we must leave all q_s's and leave exactly one point in A'', B'' for each $p_{ij}, 1 \leq j \leq 3$.

Let $T_i^{''}$ and $T_i^{*''}$ be the subchains in A'' and B'' which are obtained from simplifying T_i and T_i^* in A and B respectively. If $T_i^{''}$ is empty then we can assign a value to x_i arbitrarily. Now we focus on the case when $T_i^{''}$ is not empty, which implies that $T_i^{''}$ and $T_i^{*''}$ have the same size and $d_F(T_i^{''}, T_i^{*''}) \leq 2\epsilon$. If $T_i^{''}$ is not empty and it is a subsequence of $p_{i_1}, \cdots, p_{i_u}$ then we assign $x_i = 1$. If $T_i^{''}$ is not empty and it is a subsequence of $p_{j_1}, \cdots, p_{j_v}$ then we assign $x_i = 0$. It is easy to see that ϕ is true by the assignments to those variables $x_1, \cdots, x_n$.

To conclude the proof of this theorem, notice that the reduction takes linear (in the length of ϕ) time. □

We comment that for several *optimization* versions of the problem the proof still holds. For instance, when all the other conditions hold and we try to minimize K, then the problem is still NP-complete. Moreover, as in the above proof deciding whether $d_F(A', B') = 0$ is NP-complete, when all other conditions hold, there is no polynomial time algorithm for approximating $d_F(A', B')$ unless P=NP. The above theorem certainly implies that it is better to add Steiner points when we simplify a pair of (adjacent) ridges in ridge-based geometric model simplification.

6 Concluding Remarks

In this paper, for the first time, we study the problem of simplifying/approximating polylines in 3D under the discrete Fréchet distance. There are many open questions. (1) Our algorithms also work for any fixed dimension $d > 3$. However, when applied on 2D chains our algorithms might return self-intersecting approximating chains. This is also a problem for previous chain simplification algorithms using (continuous) Fréchet distance. In fact, this was listed as an open problem in [1]. How can we handle this problem? (2) In Theorem 4, the running time of the algorithm is dominated by the computation of the smallest enclosing balls in table $R[-,-,-]$. Is there a way to improve the $O(n^3)$ bound? Also, regardless of the running time it might be possible to reduce the space complexity in Theorem 4 (and Theorem 2). (3) The proof of the NP-completeness of CPS-2H uses general 3D polylines, not really ridges. Can we use ridges to finish the proof? (4) What is the complexity of the CPS-3F problem? We conjecture that it is also NP-complete.

References

1. Agarwal, P., Har-Peled, S., Mustafa, N., Wang, Y.: Near-linear time approximation algorithms for curve simplification. Algorithmica 42, 203–219 (2005)
2. Alt, H., Godau, M.: Measuring the resemblance of polygonal curves. In: Proceedings of the 8th Annual Symposium on Computational Geometry (SoCG 1992), pp. 102–109 (1992)
3. Alt, H., Godau, M.: Computing the Fréchet distance between two polygonal curves. Intl. J. Computational Geometry and Applications 5, 75–91 (1995)
4. Barequet, G., Chen, D.Z., Daescu, O., Goodrich, M., Snoeyink, J.: Efficiently approximating polygonal paths in three and higher dimensions. Algorithmica 33, 150–167 (2002)
5. Chan, T.: Optimal output-sensitive convex hull algorithms in two and three dimensions. Discrete and Computational Geometry 16, 361–368 (1996)

6. Chan, S., Chin, F.: Approximation of polygonal curves with minimum number of line segments or minimum error. Intl. J. Computational Geometry and Applications 6, 59–77 (1996)
7. Chen, D.Z., Daescu, O.: Space-efficient algorithms for approximating polygonal curves in two-dimensional space. Intl. J. Computational Geometry and Applications 13, 95–111 (2003)
8. Chen, Z., Fu, B., Zhu, B.: The approximability of the exemplar breakpoint distance problem. In: Cheng, S.-W., Poon, C.K. (eds.) AAIM 2006. LNCS, vol. 4041, pp. 291–302. Springer, Heidelberg (2006)
9. Eiter, T., Mannila, H.: Computing discrete Fréchet distance. Tech. Report CD-TR 94/64, Information Systems Department, Technical University of Vienna (1994)
10. Eu, D., Toussaint, G.: On approximating polygonal curves in two and three dimensions. CVGIP: Graphical Models and Image Processing 56, 231–246 (1994)
11. Fréchet, M.: Sur quelques points du calcul fonctionnel. Rendiconti del Circolo Mathematico di Palermo 22, 1–74 (1906)
12. Godau, M.: A natural metric for curves — computing the distance for polygonal chains and approximation algorithms. In: Jantzen, M., Choffrut, C. (eds.) STACS 1991. LNCS, vol. 480, pp. 127–136. Springer, Heidelberg (1991)
13. Guibas, L., Hershberger, J., Mitchell, J., Snoeyink, J.: Approximating polygons and subdivisions with minimum-link paths. Intl. J. Computational Geometry and Applications 3, 383–415 (1993)
14. Imai, H., Iri, M.: Computational-geometric methods for polygonal approximation. CVGIP 36, 31–41 (1986)
15. Imai, H., Iri, M.: An optimal algorithm for approximating a piecewise linear function. J. of Information Processing 9, 159–162 (1986)
16. Imai, H., Iri, M.: Polygonal approximation of a curve — formulations and algorithms. In: Toussaint, G. (ed.) Computational Morphology, pp. 71–86 (1988)
17. Jiang, M., Xu, Y., Zhu, B.: Protein structure-structure alignment with discrete Fréchet distance. In: Proceedings of the 5th Asia-Pacific Bioinformatics Conf (APBC'07), pp. 131–141 (2007)
18. Kenyon-Mathieu, C., King, V.: Verifying partial orders. In: Proceedings of the 21st Annual Symposium on Theory of Computing (STOC'89), pp. 367–374 (1989)
19. Megiddo, N.: Linear programming in linear time when the dimension is fixed. J. ACM 31(1), 114–127 (1984)
20. Melkman, A., O'Rourke, I.: On polygonal chain approximation. In: Toussaint, G. (ed.) Computational Morphology, pp. 87–95 (1988)
21. McAllister, M., Snoeyink, J.: Medial axis generalisation of hydrology networks. In: AutoCarto 13: ACSM/ASPRS Ann. Convention Technical Papers, Seattle, WA, pp. 164–173. (1997)
22. Varadarajan, K.: Approximating monotone polygonal curves using the uniform metric. In: Proceedings of the 12th Annual Symposium on Computational Geometry (SoCG 1996), pp. 311–318 (1996)
23. Wenk, C.: Shape Matching in Higher Dimensions. PhD thesis, Freie Universitaet Berlin (2002)
24. Zhu, B.: Protein local structure alignment under the discrete Fréchet distance. J. Computational Biology 14(10), 1343–1351 (2007)

Weighted Rectilinear Approximation of Points in the Plane

Mario A. Lopez[1] and Yan Mayster[2]

[1] University of Denver, Department of Mathematics, 2360 S. Gaylord St., Denver, CO 80208, USA
mlopez@du.edu

[2] University of Denver, Department of Computer Science, 2360 S. Gaylord St., Denver, CO 80208, USA
ymayster@cs.du.edu

Abstract. We consider the problem of weighted rectilinear approximation on the plane and offer both exact algorithms and heuristics with provable performance bounds. Let $S = \{(p_i, w_i)\}$ be a set of n points p_i in the plane, with associated distance-modifying weights $w_i > 0$. We present algorithms for finding the best fit to S among x-monotone rectilinear polylines $\mathcal{R}$ with a given number $k < n$ of horizontal segments. We measure the quality of the fit by the greatest weighted vertical distance, i.e., the approximation error is $\max_{1 \leq i \leq n} w_i d_v(p_i, \mathcal{R})$, where $d_v(p_i, \mathcal{R})$ is the vertical distance from p_i to $\mathcal{R}$. We can solve for arbitrary k optimally in $O(n^2)$ or approximately in $O(n \log^2 n)$ time. We also describe a randomized algorithm with an $O(n \log^2 n)$ expected running time for the unweighted case and describe how to modify it to handle the weighted case in $O(n \log^3 n)$ expected time. All algorithms require $O(n)$ space.

1 Introduction

The approximation of points in the plane using piecewise linear functions has drawn much interest from researchers in computational geometry and other fields. Many variants exist as a result of different constraints on the nature of the approximating curve, its complexity, error metric or the quality of the approximation. For a sample of recent results as well as references to other relevant work see [1,4,5,6,9,12]. For these variants, two subclasses of problems can be considered. The first, min-#, calls for a solution curve with the least number of line segments (in the rectilinear case, only horizontal segments are counted) given a target error ε. The second, min-ε, specifies a number k and asks for a curve with no more than k segments that achieves least possible error ε. Finally, we can also add an additional restriction in the form of weights attached to individual points, which modify the distances from the points to the approximating line (usually, as multiplicative constants). This restriction creates many new versions of the problem (see [8,16]). This paper addresses the weighted min-ε problem of approximating a set of points by a rectilinear curve using the min-max vertical distance metric.

In defining the problem we use much of the same notation as in [3], which was the first to tackle the unweighted case. Let $S = \{p_i = (x_i, y_i), i = 1, \ldots, n\}, x_1 < x_2 < \ldots < x_n$, be a set of n points in the plane. For $1 \leq i \leq j \leq n$, define $S_{ij} := \{p_i, p_{i+1}, \ldots, p_j\}$. A curve $\mathcal{R}$ is rectilinear if it consists only of alternating horizontal and vertical segments and is x-monotone if the x-domains of any two consecutive horizontal segments meet in a single value. From now on, when we speak of approximation curves they are both rectilinear and x-monotone.

We now define *vertical distance*, the error function used in our method. If a horizontal segment s has y-coordinate y_s and x-range $[x_s, x'_s]$, then the weighted vertical distance from s to a point p_i with associated weight w_i is

$$d_v^W(p_i, s) = \begin{cases} w_i|y_i - y_s| \text{ if } & x_i \in [x_s, x'_s], \\ \quad \infty & \text{otherwise.} \end{cases}$$

Then, the weighted vertical distance between a point p_i and a curve $\mathcal{R}$ is defined as

$$d_v^W(p_i, \mathcal{R}) = \min_{s \in \mathcal{R}} d_v^W(p_i, s).$$

In the spirit of [3], the eccentricity of $\mathcal{R}$ with respect to S is the maximum vertical error between the points of S and $\mathcal{R}$, i.e.,

$$e(S, \mathcal{R}) = \max_{1 \leq i \leq n} d_v^W(p_i, \mathcal{R}).$$

A point p_i of S is said to be "covered" by a horizontal segment s of $\mathcal{R}$ if x_i is in the x-domain of s. We can see that every point of S is covered by some horizontal segment of $\mathcal{R}$ and the set of all points covered by s is some S_{ij} (which, as in [3], we call the *allocation set* of s). All allocation sets can be assumed nonempty as, otherwise, we unnecessarily increase the complexity of $\mathcal{R}$. Furthermore, the boundaries between adjacent horizontal segments can be fixed arbitrarily in the intervals between adjacent allocation sets.

Díaz-Báñez and Mesa [3] provide an $O(n^2 \log n)$ algorithm to solve the unweighted min-ε problem using their $O(n)$ solution for the min-# problem. They solve the min-# problem by sweeping the points from left to right and extending the current segment while the y-span of the points it covers is at most twice the allowed eccentricity. Thereafter, they solve min-ε by reducing it to a binary search on the "candidate" eccentricities (which number $O(n^2)$, one for each possible pair of points p_i, p_j, $i \leq j$). Later, Wang [15] reduces the time for min-ε to $O(n^2)$ by carefully generating a set of at most $2n - 2$ candidate errors which includes the optimal one and running the linear min-# algorithm on each of these errors.

Mayster and Lopez [10] improve over Wang with a min-ε algorithm that runs in $O(\min\{n^2, nk \log n\})$ time. Their algorithm uses Wang's $O(n)$ candidate eccentricities coupled with an auxiliary tree structure that cuts the time for each min-# instance down to $O(k \log n)$. The second result of [10] is a greedy heuristic (GCSA) that runs in $O(n \log n)$ time. It can generate curves with $2k-1$ segments with eccentricity no worse than that for an optimal curve consisting of

k segments as well as produce curves with k segments with eccentricity within a factor of 3 from k-optimal.

The rest of the paper is organized as follows. In the next section we introduce the dual perspective to modelling weighted distances from points to an approximating segment. In Section 3, we describe an exact algorithm that runs in $O(n^2)$ time. In Section 4 we discuss a modified GCSA heuristic that utilizes the dual perspective to maintain the costs efficiently. It runs in $O(n \log^2 n)$ time and has the same error bounds with proofs carrying over from [10]. Finally, in Section 5 we describe a randomized algorithm that solves the unweighted (resp. weighted) version of the problem in $O(n \log^2 n)$ (resp. $O(n \log^3 n)$) expected time.

2 Preliminaries

First, we consider the optimal placement of a horizontal segment with respect to its (fixed) allocation set. In the unweighted case the error is minimized when the segment is centered with respect to the y-range of the points. Thus, the optimal location of the horizontal segment is unique and can be determined from two points in the allocation set. This is still true in the weighted scenario, but the optimal location may not correspond to the midpoint of the y-range. However, it must still be equidistant (under weighted distance) from the furthest points above and below it, as otherwise a small shift in its position would decrease the error.

There cannot be two different locations for the optimal segment because of the semi-monotonicity of the distance function. If two distinct segments s and s' were both optimal, then the distance from s' to one of the two points that define s would be greater than the distance from s to that point, in violation of the optimality of s'. If the two points that define the y-coordinate y_s of the best approximating segment s have coordinates $(x_i, y_i), (x_j, y_j)$ and corresponding weights w_i, w_j, then y_s is given by

$$(y_i - y_s)w_i = (y_s - y_j)w_j \Rightarrow y_s = \frac{y_i w_i + y_j w_j}{w_i + w_j}.$$

Therefore, the solution to the problem is the intersection of two lines $c = -w_i y + y_i w_i$ and $c = w_j y - y_j w_j$, where c stands for the cost of approximating the point by a segment located at y. This leads us to consider a "cost-location" space composed of such lines, each point in S giving rise to one upward and one downward sloping line with the absolute values of the slopes equal to the weight of the point. Let us suppose that all points in S are located in the first quadrant, i.e. $x_i, y_i > 0 \, \forall 1 \leq i \leq n$. We map each point p_i with the corresponding weight w_i to the pair of lines in the "cost-location" plane $\ell_{i0} = w_i y_i - w_i y$ and $\ell_{i1} = -w_i y_i + w_i y$ and restrict their domain to the first quadrant. Thus, for each point we have a linear transformation ℓ_i of the absolute value metric function restricted to the nonnegative domain. Each such wedge shaped function ℓ_i computes the distance from p_i to the approximating segment as we hypothetically sweep it upward starting from $y = 0$ and consists of a finite down-sloping segment (recording the

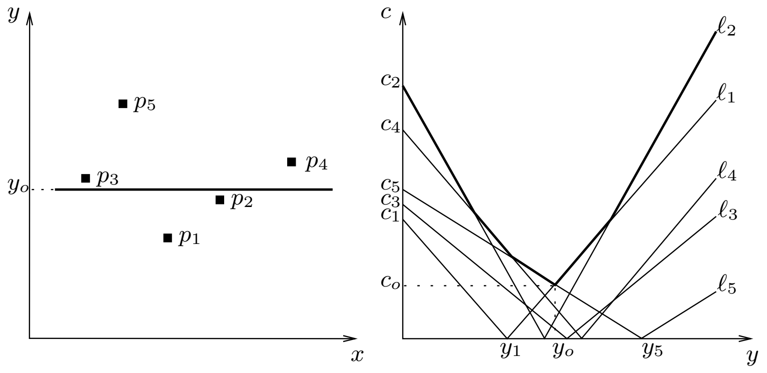


Fig. 1. (a) A set of points $p_i = (x_i, y_i)$, numbered by increasing y-coordinate, having respective weights w_i such that $w_2 > w_4 > w_1 > w_3 > w_2$, and the best fit segment. (b) The corresponding lines in the cost-location plane with the slopes w_i and the vertical axis intercepts c_i. The lowest point of the envelope is identified with the cost and y-coordinate of the best fit segment.

cost for $y < y_i$) and an infinite up-sloping ray (for the cost when $y > y_i$). Which portion of this arrangement of $2n$ cost lines keeps track of the greatest distance (i.e., the furthest point) to the approximating segment for any segment position y? The answer is quite obvious - the upper envelope of the arrangement is made up of the segments of the cost lines of those points that at some y are furthest from the approximating segment. The optimal location is given by the lowest point, which is also the lowest vertex, of the upper envelope.

We observe that in the case when S is known and fixed the above-mentioned problem of finding the lowest vertex of the upper envelope has been tackled successfully before, as it is nothing other than finding the optimal solution to a linear program in 2D. The best known deterministic algorithm for this has been developed by [11] and runs in $O(n)$ time. In addition, a very simple randomized algorithm [14] exists that has $O(n)$ expected running time. In our optimal algorithm we shall need to solve this problem repeatedly for each new subset of S, which differs in a single point from the previous subset, in order to compute the eccentricities of candidate curves and, therefore, using the $O(n)$ algorithm as a subroutine is an overkill.

However, there are more efficient algorithms to dynamically maintain common intersections of half-planes. In particular, a clever dynamization technique by Overmars and van Leeuwen [13] can be exploited to maintain the upper envelope in $O(\log^2 n)$ time per update (insertion or deletion of a line) and enables us to query for the lowest point on the boundary in just $O(\log n)$ time. The essence of their approach is to store the "left" half-planes (i.e., those that contain the left ray of any horizontal line) and the "right" half-planes in two separate augmented binary search trees. The lines bounding the half-planes are stored at the leaves and ordered by slope. In our case, since each point contributes an entire wedge with both bounding lines having the same (in absolute value) slope, it makes sense to have just one tree and store the points themselves at the leaves sorted by weight. Then, the bounding lines of the left half-planes are sorted in descending order and the bounding lines of the right half-planes are sorted in ascending order (without explicitly storing these lines). As per [13], each internal node is augmented with a pointer to the parent and the largest slope value (largest point weight) of the lines in its left subtree (needed for concatenation). Most importantly, the portion of the upper envelope of the left half-plane lines in its subtree that does not contribute to the upper envelope of the left half-planes of its parent is stored in a concatenable queue along with the number of lines on its envelope that belong to the upper "left" envelope of the parent. The "right" upper envelope is handled similarly, so each internal node has two concatenated queues associated with it.

Then, the overall "left" upper envelope is stored at the root of the "left" tree (and, similarly, the "right" upper envelope is stored at the root of the "right" tree). Using the procedures DOWN and UP described in [13] one can insert and delete lines and maintain the queue structures as well as the balance of the tree. Then, the intersection of the left and right envelopes can be found efficiently in $O(\log n)$ time as is also proven in the original paper.

Finally, we note that there are other dynamic half-plane intersection algorithms that outperform the above-mentioned algorithm by Overmars and Leeuwen and run in $O(n \log n)$ amortized time, such as [7] and [2].

3 An Exact Algorithm

As observed in the previous section, the error of each approximating segment in its best position is determined by two points and, therefore, so is the eccentricity of the curve. It is still valid to use Wang's choice of candidate eccentricities and then it remains to describe how to compute these and the candidate curves that they give rise to. In Wang's algorithm, when one of the two pointers (called sweep lines in the original paper) is advanced, the error of the best approximating segment for the set of points between the two pointers is computed. This error computation in the weighted distance case corresponds to finding the lowest point on the upper envelope of the wedge lines in the cost-location plane as these lines are added or deleted one at a time. As mentioned in the previous section, computing the candidate eccentricities can be done using the $O(\log^2 n)$ dynamic half-plane intersection algorithm of [13].

We now turn to the question of how to compute a candidate curve itself once the target eccentricity ε has been found. This can be done with a slightly modified min-# algorithm of [3]. In this new version, each point (x_i, y_i) with the weight w_i is represented by a vertical line segment $v_i = (x_i, y_i - \frac{\varepsilon}{w_i})(x_i, y_i + \frac{\varepsilon}{w_i})$. Then, the algorithm proceeds in essentially the same way as described in [3]. We build horizontal segments of the curve by piercing consecutive vertical segments v_i. At first, we initialize the allocation set of the first horizontal segment to the single point (x_1, y_1) and define its corridor to be $(y_{min} = y_1 - \frac{\varepsilon}{w_1}, y_{max} = y_1 + \frac{\varepsilon}{w_1})$. Then, adding each additional point p_i to the allocation set causes the segment's corridor to be updated to $y'_{min} = \max\{y_{min}, y_i - \frac{\varepsilon}{w_i}\}, y'_{max} = \min\{y_{max}, y_i + \frac{\varepsilon}{w_i}\}$. We keep extending the current horizontal segment of the curve for as long as adding new points does not cause the corridor to become empty, i.e. until further expansion of the allocation set would make $y'_{min} > y'_{max}$. Therefore, computing both the candidate eccentricity and curve takes $O(n)$ time leading to the following result.

Theorem 1. *The weighted rectilinear approximation problem can be solved in $O(n^2)$ time.*

This time bound becomes considerably reduced if the number of distance weights associated with the points of S is equal to a constant. In this case, the line wedges in the cost-location plane only have a constant number of distinct slopes. It is easy to see that for any given slope only one line wedge with that slope may contribute to the downward (and, similarly, upward) portion of the upper envelope. Furthermore, in our case, it is obvious that only the line wedge that contributes the first segment to the downward portion may also contribute a segment to the upward portion (due to the fact that all other line wedges that are part of the downward portion have smaller slope and a further x-intercept than the first one). All other line wedges may contribute only to one of the two portions. This means that the upper envelope consists of no more than $n + 1$ segments. In the case of a constant number of slopes c, we have no more than $c+1$ segments on the envelope and, therefore, the above algorithm runs in linear time. This is summarized in the next theorem.

Theorem 2. *The weighted rectilinear approximation problem with a constant number c of distance-modifying weights can be solved in $O(cn)$ time.*

4 A Heuristic with Provable Bounds

In [10], the authors describe a simple yet in practice quite accurate GCSA approximation algorithm for the problem of rectilinear curve fitting. The algorithm begins by building a curve consisting of n singleton segments and computes the costs that would result from merging the allocation sets of each adjacent pair of such segments. These costs are prioritized by storing them in a min-heap and, subsequently, at each iteration the minimum cost is extracted and the pair of associated segments is merged. The algorithm then updates the structure and the costs that involve the newly created enlarged segment and its neighbors.

We now modify this algorithm to be able to solve the weighted version of the same problem. While the overall structure of the algorithm shall remain unchanged, we have to supply new details for the merge step and analyze how these affect the overall running time. Now merging two allocation sets can no longer be accomplished in constant time as the points responsible for the error of the new larger segment are not necessarily a subset of the points defining the placement of the old segments. Recall that the y-coordinate of the new longer segment s is determined by a pair of points whose so-called cost lines in the cost-location plane define the lowermost point of the upper envelope of all such cost lines coming from the points in the allocation set of s. Clearly, the cost lines that define this point come from the upper envelopes of the old segments' cost lines. Hence, the placement of the new longer segment can be determined by any two points whose cost lines were on the upper envelopes of their respective segments. We, therefore, have to keep track of the points defining these upper envelopes for each allocation set (upper envelope points).

Each of these points contributes at most two edges to the upper envelope and no two edges on the same envelope have overlapping x-ranges except at the boundaries. We can, therefore, store these in a binary tree ordered by x-range with pointers going to the original points. We also note that ordering the edges by x-range also has the effect of sorting them by slope as well as inducing a semi-sorted order on their y-ranges, since these decrease until the lowest point on the envelope and then monotonically increase. Furthermore, all upper envelopes are necessarily concave down, an important property that will be of use later.

When "merging" the allocation sets of two curve segments, their upper envelopes S (for *small*) and B (for *big*) need to be "merged" to produce the upper envelope of the new segment. Suppose that $|B| = n, |S| = m$ and $n > m$ (where the cardinality of an envelope is equal to the number of lines contributing segments to it). When we merge S and B, we always traverse S sequentially and B sometimes sequentially (when B is below S) and sometimes logarithmically (when B is above S). Clearly, the segments that survive (either partially or in their entirety) are on the upper envelope of $S \cup B$. Therefore, we need to find all points of intersection between S and B (for this is where they switch roles, one going below the other) and stitch together those portions that contribute to the overall upper envelope. Hence, when B is below S, we remove segments from B one by one (in $O(\log n)$ time per segment) and replace them by segments from S. Once removed, these lines (i.e., points in the allocation set) will never contribute to the upper envelope. When B is above S, that portion of B needs to be preserved and traversing it sequentially in order to find the next intersection between S and B would lead to a linear amortized time per merge and, thus, to the total quadratic time for the entire algorithm (consisting of $O(n)$ merges).

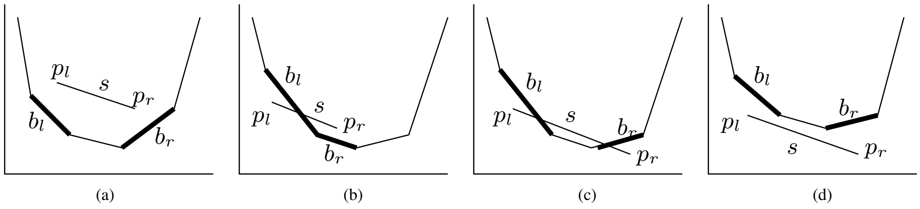


Fig. 2. (a) Case I: both endpoints of s are above the bigger envelope B. (b) Case II: p_l is below a segment b_l of B while p_r is above B (same as p_l above B and p_r below). (c) Case III: both endpoints of s are below B and an intersection exists. (d) Case IV: endpoints of s are as in Case III but there is no intersection.

We begin with the leftmost segments of S and B. As we move along S, for each of its segments s with endpoints p_l, p_r we locate (via a binary search) the segments b_l, b_r (potentially, $b_l = b_r$) in B whose x-ranges contain the x-coordinates x_l, x_r of those endpoints. In the case of ties, when the endpoints of two segments of B have x-coordinate x_l or x_r, we always pick the segment of B that begins at x_l and ends at x_r. We then test if p_l is above or below b_l and, similarly, whether p_r is above or below b_r. If p_l or p_r coincide with the endpoints of b_l or b_r, we test whether s itself is below or above b_l or b_r. If both p_l and p_r (or s itself in the case of coinciding endpoints) are above the segments of B, then because of the concavity of upper envelopes we know that s is completely above B (Figure 2a) and, therefore, it must be added to the upper envelope of $S \cup B$ and all segments of B from b_l to b_r (except, perhaps, b_r itself if its x-range is not completely covered by the x-range of s) can be removed from consideration. As a way to simplify and speed up the process, we create the upper envelope of $S \cup B$ completely inside of the data structure for B. Therefore, all deletions of segments of B and insertions of the segments of S are carried out straight on the binary tree containing B with the result that after the merge is complete B contains the final "merged" envelope.

If one of the endpoints of s (again, in the case of endpoints coinciding, s itself) is above B and the other is below B, then an intersection exists (Figure 2b) and can be found in logarithmic time by simply doing a binary search on the segments of B and testing them as being above or below s, or simply walking along B starting from the segment which is below s and deleting segments from B until we arrive at the intersection at which point we link up with s. Thus, all segments of B below s are removed (again, except perhaps for b_r even if it is below s) and a portion of s is added to B (starting or ending at the intersection point, depending on which part of s is above B).

Finally, we come to the case when both endpoints of s are below B, which leads to the two possibilities illustrated in Figures 2c and 2d. In this case, there may or may not be an intersection and some extra work needs to be done to determine this. Namely, we certainly do not have an intersection when s belongs to the downsloping part of S and p_l is below the upsloping part of B or vice versa, when s has an upward slope and p_r is below the downsloping part of B. However, this is not sufficient to decide whether there is an intersection between s and B. These cases, then, are subsumed by the following simple check. First, we determine whether the slope of s is between the slope of b_l and that of b_r (remember, that slopes uniformly increase from b_l to b_r). Only if it is, there may be an intersection. We then find, via a binary search on the slopes of lines between b_l and b_r, the line b of B that has slope closest to that of s. If this line is not above s (Figure 2c), then we have two intersections which can be found by walking from b in opposite directions, while deleting segments from B. Otherwise, there is still no intersection (Figure 2d). To see that this is indeed a correct strategy, we remember that if s were to pierce B it must either intersect or "obscure" the line with the closest slope since in the resulting envelope lines must appear in the order from smallest to largest slope.

To complete the description of the modified GCSA heuristic, we need to address one more problem and that is the computation of the merge cost, i.e. the eccentricity of the resulting curve if the two allocation sets were merged. This, however, can be achieved with the same algorithm as above except that no changes should be made to B (i.e., we "simulate" a merge) and we can stop once the lowest point on the envelope has been found (note that this technique cannot be used for the exact algorithm in the previous section for it only handles envelopes obtained by merges and does not handle those obtained by deleting lines).

Let's analyze now the running time of this new GCSA algorithm. We first look at the operation of a single merge step involving the smaller allocation set S with $|S| = n_S$ and the bigger allocation set B with $|B| = n_B$. How many times can a segment of S's envelope intersect B's envelope? The answer is at most twice, since envelopes have parabolic shape. Therefore, only one part of a segment of S or that segment in its entirety can be inserted into B's envelope and since the number of segments in the envelope is at most one more than the size of the allocation set, no more than $n_S + 1$ insertions take place. Therefore, the total cost of insertions per merge step is $O(n_S \log n_B)$. It remains to sum the cardinalities of all such smaller allocation sets S participating in merge steps. This question can be approached from the point of view of how many times, at the most, the same point can belong to the smaller set over the course of all merge steps. This is very similar to the analysis of the disjoint data set union operation and we know that the same point can be merged from a smaller set at most $\log n$ times, for the sizes of the smaller sets it is part of will in the worst case increase as the sequence $1, 2, 4, 8, \ldots$ So, the number of insertions over all merge steps is at most $O(n \log n)$ and with each insertion taking $O(\log n)$ time, the total time is $O(n \log^2 n)$. We still need to remember to account for the deletions taking place during merging, but this is easy for once a line has been removed from an envelope, the point responsible for it will no longer be considered. Hence, the total cost of deletions is only $O(n \log n)$. Also, "simulating" a merge to compute the prospective eccentricity has the same cost as an ordinary merge and we know that only at most two such simulations are needed for every real merge step. Thus, we can perform all $O(n)$ merges in $O(n \log^2 n)$ time. This gives us the following result.

Theorem 3. *The modified GCSA algorithm runs in $O(n \log^2 n)$ time and guarantees the error bounds proven for the original GCSA. Namely that for $n \geq 2k$, the GCSA algorithm with $m = 2k - 1$ produces a curve C with eccentricity $\epsilon \leq \epsilon^*$ and with $m = k$ segments achieves eccentricity at most $3\epsilon^*$.*

The above claims regarding the error bounds follow directly from the proofs given in [10], as they carry over verbatim to this modified version of GCSA.

5 A Randomized Algorithm

The main idea of this algorithm is to perform an efficient search on the set of $O(n^2)$ possible eccentricities but, unlike [3], the entire set of eccentricities is not generated explicitly. Instead, only those for which a candidate curve is constructed are computed. This results in $O(\log n)$ candidates on average and $O(n \log^2 n)$ expected running time. We begin by describing the unweighted version of the algorithm and then show how to extend it to handle weights.

The algorithm starts by picking a random pair of points p_i and p_j and computing the eccentricity of the allocation set S_{ij}. This can be done in $O(n)$ time (e.g., using the linear programming algorithm in [11]). Then, using the min-# algorithm of [3], the first candidate curve R_{ij} of size k_{ij} is constructed and compared against the target k. The result of this comparison is to be used to decide about the bounds on the achievable eccentricity. The algorithm, therefore, keeps track of the feasible eccentricity window $\mathcal{E}_f = [\varepsilon_{min}, \varepsilon_{max}]$, which is updated after investigating each candidate curve. This window is initialized to $[0, \infty)$. Now, if $k_{ij} \leq k$, we update the window to $[0, \varepsilon_{ij}]$. While, in the opposite case of $k_{ij} > k$, we know that the eccentricity has to be increased, and so the feasible window becomes $[\varepsilon_{ij}, \infty)$.

Now, to discard all allocation sets whose errors are outside of the feasible eccentricity window, we create a data structure that records for each point p_i of S the number of allocation sets that start at p_i and end at some p_j with errors still in the current feasible window as well as the smallest index l_i and the largest index r_i such that $i \leq l_i \leq j \leq r_i$. For each p_i and a given feasible eccentricity window $\mathcal{E}_f$, we thus have the set $S_i^{\mathcal{E}_f}$ of possible values of j. In this set, $j = l_i$ specifies the index of the closest (in x-direction) point to p_i such that the error of the allocation set $\{p_i, \ldots, p_{l_i}\}$ is at least ε_{min} and, similarly, $j = r_i$ gives the furthest point from p_i with the error of the allocation set $\{p_i, \ldots, p_{r_i}\}$ at most ε_{max}. To see that p_j runs across a contiguous subset of S, we note that the eccentricity of an allocation set S_{ij} is monotonically non-decreasing as i is kept fixed and j is advanced.

Let us now explain how to compute for each p_i the cardinality and bounds l_i, r_i of $S_i^{\mathcal{E}_f}$ as well as how to maintain this information as $\mathcal{E}_f$ changes. After the first candidate curve is generated and $\mathcal{E}_f$ is initialized, we compute $|S_1^{\mathcal{E}_f}|$ and l_1, r_1 by scanning S. We then note that $l_i \leq l_k, r_i \leq r_k$ whenever $i \leq k$. This holds because the y-range of the set $\{p_k, \ldots, p_{l_i}\}$ (possibly empty if $k > l_i$) is subsumed by the y-range of the set $\{p_i, \ldots, p_{l_i}\}$ forcing l_k to be no less than l_i and, similarly, for r_k and r_i. Therefore, it seems that l_2 and r_2 can be found by simply moving ahead the pointers from l_1 and r_1, respectively, if needed. Unfortunately, in order to know when to stop for l_2 and r_2 we need to know the error of the allocation sets that begin at p_2 rather than p_1 for it could be that p_1, which is now removed from consideration, was one of the two points determining the error of $L_1 = \{p_1, \ldots, p_{l_1}\}$ or the two points determining the error of $R_1 = \{p_1, \ldots, p_{r_1}\}$. This necessitates the creation of two priority queues, such as min-max heaps, to keep track of the lowest and highest points in the two allocation sets L_i, R_i as they are being determined for each p_i. Then, in the case of p_2, we set $L_2 = L_1, R_2 = R_1$ and then remove p_1 from the heaps for each of the sets. Then, we start adding points to L_2 beginning with p_{l_1+1} and stop after having added the first point that had caused the error of L_2 to exceed or become equal to ε_{min}. Similarly, we add points to R_2 until adding the next point would make the error of R_2 become greater than ε_{max}. We remember the index of the last point added to L_2 as l_2 and that of the last point added to R_2 as r_2. All subsequent bounds for $S_i^{\mathcal{E}_f}, 2 \leq i \leq n$, can be found by advancing these two pointers, each making at most one full pass through S. Finally, computing the size of $S_i^{\mathcal{E}_f}$ is trivial as it is just $|r_i - l_i + 1|$. We note that every point is added to each of the two queues exactly once and is removed at most once as we compute all values of i_l, i_r for a given eccentricity window $\mathcal{E}_f$.

Thus, every time $\mathcal{E}_f$ changes, recomputing the bounds and cardinality information takes only $O(n \log n)$ time since it only involves $O(n)$ heap operations. Hence, the key to good performance becomes reducing the (expected) number of changes to the feasible eccentricity window that are necessary to process before the optimal eccentricity is found. This goal we achieve through randomization as we shall describe next.

After the initial step has determined $\mathcal{E}_f$ and the bounds and cardinalities of each of the sets $S_i^{\mathcal{E}_f}$ have been computed, we pick a pair of points that enclose an allocation set with error in the feasible eccentricity window at random from the set of all such possible pairs. In order to do this, and have a uniform distribution of probabilities, for each $2 \leq i \leq n$ we sum up the cardinalities of the sets $S_k^{\mathcal{E}_f}$ for all $k \leq i$ and store this number for p_i, i.e. we have

$$K_i = \sum_{k=1}^{i} \left| S_k^{\mathcal{E}_f} \right| .$$

Clearly, these can be computed in one scan of the array since K_i is just the sum of K_{i-1} and the cardinality of $S_i^{\mathcal{E}_f}$. Then, we can generate a pseudo-random number x between 1 and K_n and identify the unique pair (p_i, p_j) corresponding to this index (we just search for x in the array of K_i's, find the smallest i_0 such that $K_{i_0} \geq x$, and then find the unique j from $S_{i_0}^{\mathcal{E}_f}$.

This way we make sure that each pair is selected with the same probability but only from the set of those pairs that already fulfill the criteria for the error of its allocation set. Thus, we can expect that on average picking a new pair will reduce the number of pairs in the feasible eccentricity window roughly in half and so, our search has an expected logarithmic number of steps in the size of the set of possible eccentricities, that is, $O(\log(n^2)) = O(\log n)$. Since after each pair is picked an $O(n \log n)$ time is spent updating the auxiliary arrays described above and constructing a candidate curve, the total expected running time of this randomized algorithm is $O(n \log^2 n)$.

Now, notice that even though the discussion so far focused on the unweighted case only, our algorithm can be easily adapted to the weighted case. First, we observe that it is still true in the presence of weights that $l_i \leq l_k, r_i \leq r_k$ for any two points p_i, p_k such that $i \leq k$. Clearly, not just the y-range but the weighted error range of the set $\{p_k, \ldots, p_{l_i}\}$ is subsumed by the weighted error range of $\{p_i, \ldots, p_{l_i}\}$ because the lowest point on the upper envelope of a set of cost lines can be no lower than the lowest point on the upper envelope of its subset. Consequently, instead of min-max heaps to keep track of the errors of L_i and R_i we would have to maintain upper envelopes and we can do so again using the algorithm from [13]. Each individual update of that structure takes $O(\log^2 n)$ and so one full pass through the array to update the pointers l_i, r_i for all i takes $O(n \log^2 n)$. Hence, the total time is $O(n \log^3 n)$ as there are still $O(\log n)$ candidate curves to construct.

References

1. Aronov, B., Asano, T., Katoh, N., Mehlhorn, K., Tokuyama, T.: Polyline fitting of planar points under min-sum criteria. International Journal of Computational Geometry and Applications 16, 97–116 (2006)
2. Brodal, G., Jacob, R.: Dynamic planar convex hull. In: Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science, pp. 617–626 (2002)
3. Díaz-Báñez, J.M., Mesa, J.A.: Fitting rectilinear polygonal curves to a set of points in the plane. European Journal of Operations Research 130, 214–222 (2001)
4. Eu, D., Toussaint, G.: On approximating polygonal curves in two and three dimensions. CVGIP: Graphical Models and Image Processing 56(3), 231–246 (1994)
5. Goodrich, M.: Efficient piecewise-linear function approximation using the uniform metric. Discrete and Computational Geometry 14, 445–462 (1995)
6. Hakimi, S.L., Schmeichel, E.F.: Fitting polygonal functions to a set of points in the plane. CVGIP: Graphical Models and Image Processing 53(2), 132–136 (1991)
7. Hershberger, J., Suri, S.: Off-line maintenance of planar configurations. Journal of Algorithms 21, 453–475 (1996)
8. Houle, M., Imai, H., Imai, K., Robert, J.-M., Yamamoto, P.: Orthogonal weighted linear L_1 and L_∞ approximation and applications. Discrete Applied Mathematics 43(3), 217–232 (1993)
9. Imai, H., Iri, M.: Computational-geometric methods for polygonal approximations of a curve. Computer Vision, Graphics and Image Processing 36(1), 31–41 (1986)
10. Mayster, Y., Lopez, M.A.: Rectilinear approximation of a set of points in the plane. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 715–726. Springer, Heidelberg (2006)
11. Megiddo, N.: Linear programming in linear time when the dimension is fixed. Journal of ACM 31(1), 114–127 (1984)
12. Melkman, A., O'Rourke, J.: On polygonal chain approximation. In: Toussaint, G.T. (ed.) Computational Morphology, pp. 87–95, North-Holland, Amsterdam, Netherlands (1988)
13. Overmars, M.H., van Leeuwen, J.: Maintenance of configurations in the plane. Journal of Computer and System Sciences 23(2), 166–204 (1981)
14. Seidel, R.: Linear programming and convex hulls made easy. In: SCG 1990: Proceedings of the Sixth Annual Symposium on Computational Geometry, pp. 211–215 (1990)
15. Wang, D.P.: A new algorithms for fitting a rectilinear x-monotone curve to a set of points in the plane. Pattern Recognition Letters 23, 329–334 (2002)
16. Yamamoto, P., Kato, K., Imai, K., Imai, H.: Algorithms for vertical and orthogonal L_1 linear approximation of points. In: Proceedings of the 4th Annual Symposium on Computational Geometry, pp. 352–361 (1988)

Paths with no Small Angles

Imre Bárány[1,2], Attila Pór[3], and Pavel Valtr[3]

[1] Rényi Institute of Mathematics, Hungarian Academy of Sciences, POBox 127, 1364 Budapest, Hungary
[2] Department of Mathematics, University College London, Gower Street, London WC1E 6BT, England
[3] Department of Applied Mathematics, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic

Abstract. Giving a partial solution to a problem of S. Fekete and G.J. Woeginger [3,4] we show that given a finite set X of points in the plane, it is possible to arrange them on a polygonal path (with the vertex set X) so that every angle on the polygonal path is at least $\pi/9$. According to a conjecture of Fekete and Woeginger, $\pi/9$ can be replaced by $\pi/6$. Previously, the result has not been known with any positive constant.

1 Introduction and Results

The aim of this paper is to answer a beautiful and inspiring question which appeared first in 1992 in S. Fekete's thesis [3], and later in the paper by Fekete and Woeginger [3,4] in 1997. The question is this. Given a finite set X of points in the plane, is it possible to arrange them on a polygonal path (with the vertex set X) so that every angle on the path is at least α, for some universal constant $\alpha > 0$? The answer is, as we shall see soon, yes. This might be a step towards proving a conjecture of S. Fekete and G.J. Woeginger [3,4] that this result holds with $\alpha = \pi/6$. We prove the result with the constant $\alpha = \pi/9$. First we introduce notation and terminology.

Let $A_0, \dots, A_n$ be $n+1$ distinct points in the plane. We denote the path consisting of the segments A_0A_1, A_1A_2, …, $A_{n-1}A_n$ by $A_0A_1 \dots A_n$. This is a polygonal path with vertices $A_0, A_1, \dots, A_n$. The angle of this path at A_i is the angle of the triangle $A_{i-1}A_iA_{i+1}$ at vertex A_i, $1 \le i < n$.

Definition. Let $\alpha > 0$. We call the path $A_0A_1 \dots A_n$ α-*good* if the angle at A_i is at least α for every $1 \le i < n$. A path is called *good*, if it is $\pi/9$-good.

The main result of this paper is the following

Theorem 1. *For every finite set of points X in the plane there exists a $\pi/9$-good path on the points of X (containing each point of X exactly once).*

We mention that $\pi/9$ in the theorem cannot be replaced by anything larger than $\pi/6$. This is shown when X consists of the center and the three vertices of a regular triangle (see Figure 1).

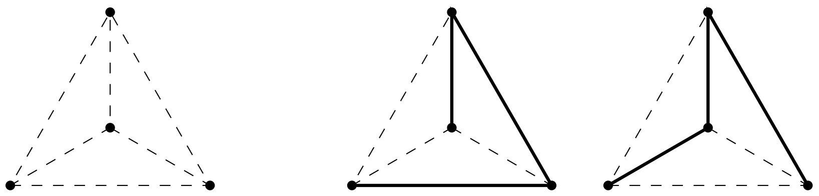


Fig. 1. A 4-point configuration and its two paths

Another example, depicted in Figure 2, shows that Theorem 1 cannot be strengthened to paths with no self-intersections. It also shows that paths minimizing various quantities (such as total length, total turning angle) may have an arbitrarily small angle.

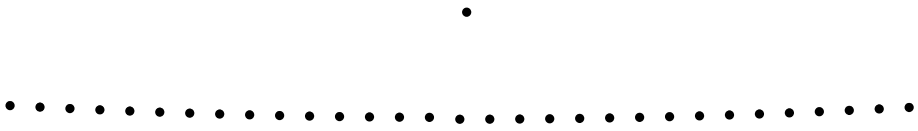


Fig. 2. Every good path on this point set is self–intersecting (the set consists of points on a huge circle and one extra point inside the circle)

For the rest of the paper we fix α as $\pi/9$. We will prove a slightly stronger statement which is more convenient for the induction argument. We will need two additional definitions.

Definition. We call the (oriented) directions of the vectors $\overline{A_1A_0}$ and $\overline{A_{n-1}A_n}$ the two *end directions* of the path $A_0 \dots A_n$. We identify the (oriented) directions with points of the unit circle S^1.

Definition. We call a subset R of the unit circle a *restriction* if it is the disjoint union of two intervals $R_1, R_2 \subset S^1$ such that both have length $4\alpha(= 4\pi/9)$ and their distance from each other (along the unit circle) is larger than $2\alpha(= 2\pi/9)$. We call the path $A_0 \dots A_n$ *R-avoiding* if the two end directions are not in the same R_i $(i = 1, 2)$ and the path is good (see Figure 3).

The following theorem is a strengthening of Theorem 1.

Theorem 2. *Let X be a finite set of points in the plane. For every restriction R there is an R-avoiding path on all the points of X.*

The proof of this theorem goes by induction on $|X|$ which gives a quadratic algorithm for finding a $\pi/9$-good path.

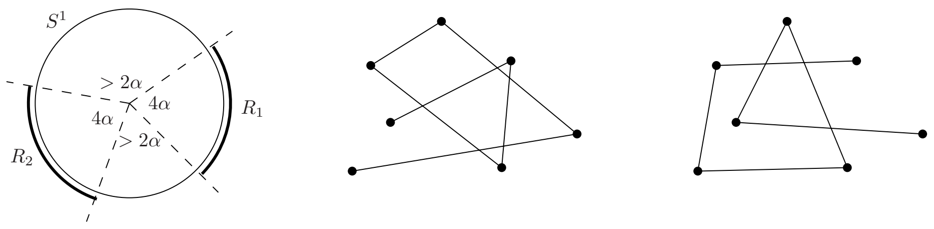


Fig. 3. A restriction $R = R_1 \cup R_2$ and two (good) paths that are not R-restricted

Theorem 3. *A $\pi/9$-good path on a set of n points in the plane can be found in time $O(n^2)$.*

We prove this theorem in the last section.

2 Further Questions and Extensions

The natural question is what happens in higher dimensions. In the journal version [2] of this paper we prove the following result.

Theorem 4. *For every $d \geq 2$ there is a positive α_d such that for every finite set of points X in d-dimensional space there exists a α_d-good path on X.*

Actually, the proof method of Theorem 2 works but some extra difficulties have to be overcome.

Another problem that we encountered while working on this paper seems interesting and nontrivial. Call a finite set X in the d-dimensional space α-flat if every triangle with vertices from X has an angle smaller than α. One example of an α-flat set is a finite set X_0 of collinear points. Each point of X_0 can be moved freely in a small enough neighbourhood so that the resulting set X_1 is still α-flat. Next, each point of X_1 can be replaced by a very small but otherwise arbitrary α-flat set, and the resulting set is still α-flat if the replacements are small enough. Perhaps all α-flat sets can be obtained this way.

Next, call the set X β-separable if it can be partitioned as $X = U \cup V$ with U, V disjoint and nonempty so that the angle between the line through u_1, v_1 and the line through u_2, v_2 is smaller than β for every $u_1, u_2 \in U$ and every $v_1, v_2 \in V$.

Conjecture. For every $d \geq 2$ and for every positive β there is a positive $\alpha_d(\beta)$ such that for every α-flat finite set X in d-dimensional space is β-separable.

We have a proof of this conjecture for $d = 2$.

We remark that the authors of the recent paper [1] investigate straight-line drawings of graphs with restricted sizes of angles spanned by pairs of incident edges.

3 Proof of Theorem 2

We prove the theorem by induction on the number of points in X.

If $|X| = 2$ then the two end directions are the opposite to each other. Since the length of R_i, $4\alpha = 4\pi/9$, is smaller than π the two end directions cannot be in the same R_i interval.

Assume $|X| > 2$. Let K be the convex hull of X and let V be the vertex set of K. Of course $V \subseteq X$. Next let $R = R_1 \cup R_2$ be a restriction. We distinguish two cases depending on the smallest angle of the polygon K.

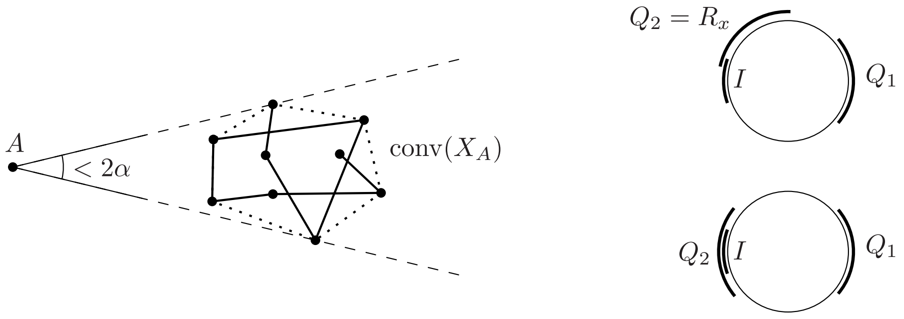


Fig. 4. Case 1

Case 1: The smallest angle of K is smaller than 2α. Let A be the vertex where that smallest angle occurs and let $X_A = X \setminus \{A\}$. See Figure 4. Without loss of generality we can assume that for each point $B \in X_A$ the direction $\overline{BA}$ is in the interval $I = (\pi - \alpha, \pi + \alpha) \subset S^1$. Since the length of I is 2α it can only intersect one of the two intervals R_1 and R_2. Let $Q_1 = [-2\alpha, 2\alpha] \subset S^1$. If one of the sets R_1 or R_2 intersects I, then let Q_2 be equal to the one that intersects I. Otherwise set $Q_2 = [\pi - 2\alpha, \pi + 2\alpha]$. It is easy to see that $Q = Q_1 \cup Q_2$ is a restriction; this is the point where $\alpha = 20^0 = \frac{\pi}{9}$ is being used. By induction on $|X_A|$ we find a Q-avoiding path $p = A_0 \dots A_n$ on X_A. If the end direction $\overline{A_1 A_0}$ is not in Q_1, then we can extend this path to the good path $Ap = AA_0 \dots A_n$ on X. Analogously, if the end direction $\overline{A_{n-1} A_n}$ is not in Q_1, then we can extend this path to the good path $pA = A_0 \dots A_n A$ on X. (We only have to take care of the angle $A_1 A_0 A$ or $A_{n-1} A_n A$.)

So at least one of the extended paths pA, Ap is α-good. The end direction at A is always in I. Therefore, if both end directions of Ap or of pA are in R_1 (or R_2), then both have to be in Q_2. In this case we can extend p at both ends. But only one of the end directions of p is in Q_2. So we extend p at the end which is in Q_2 and we get an R-avoiding path on X.

Case 2: Every angle of K is at least 2α. See Figure 5. Without loss of generality we can assume that R_1 and R_2 are symmetric to the horizontal line.

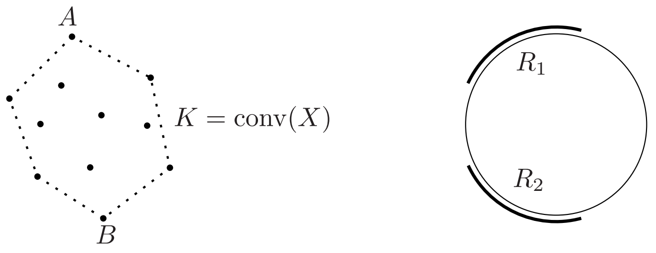


Fig. 5. Case 2

Let A and B be the vertices of K with the largest and smallest y-coordinate, respectively. We will distinguish three subcases depending on the size of $Y = X \setminus V$.

Case 2a: The set Y is empty. As a first attempt we try to find an R-avoiding path that contains only edges of K. Such a path can be identified by the missing edge of K. All these paths are clearly (α-)good. If there is an edge on the perimeter of K with a direction not in R_1 or R_2, then the path missing the next edge will have that direction as end direction. In this case we have found an R-avoiding path.

Now we assume that each edge of K is in R_1 in one direction and in R_2 in the other direction. If $|X| > 4$, then there is a path along the perimeter of K between A and B of length at least three. Take the path that misses an interior edge that is, an edge disjoint from A and B — see Figure 6 (left). One of the end directions will be in the interval $[0, \pi]$ (upwards) and the other one will be in $[\pi, 2\pi]$ (downwards). Therefore this path is R-avoiding since $R_1 \subset (0, \pi)$ and $R_2 \subset (\pi, 2\pi)$.

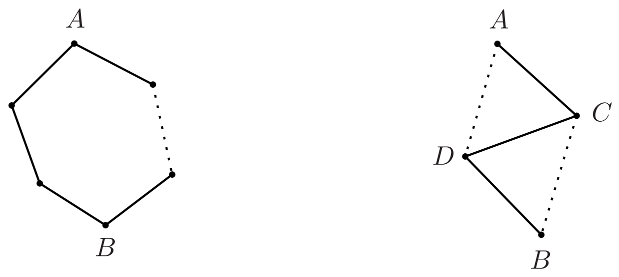


Fig. 6. Case 2a

If $|X| = 3$ then the path missing edge AB from K is R-avoiding since it has one upward and one downward end direction.

If $|X| = 4$ and AB is an edge of the convex hull, then the path missing this edge is R-avoiding. If A and B are opposite vertices of K (which is a quadrilateral now), then we connect the four vertices from top to bottom starting with A and ending with B. Let this path be $ACDB$ — see Figure 6 (right). We have CA in R_1 and CD is pointing downwards. That is $\overline{CA} \in [\alpha, \pi - \alpha]$ and $\overline{CD} \in [\pi, 2\pi]$ and therefore the angle at C is at least α. Similarly the angle at D is at least α as well which shows that this path is good. The end directions are again upward and downward therefore the path $ACDB$ is an R-avoiding path.

Case 2b. The set Y consists of one point: $Y = \{F\}$, say. Take a path that contains all edges of K except one and the segment from F to one of the endpoints of the missing edge. If the angle at the vertex which is connected to F is at least α we have a good path.

In this way every segment from F to a vertex of K can be extended to a good path since each angle of K is at least 2α and therefore the angle towards one of the neighbours along the perimeter of K has to be at least α.

Consider the extended path starting with FB — see Figure 7 (left). The end direction $\overline{BF}$ is upwards. If $\overline{BF}$ or the other end direction is not in R_1 we have an R-avoiding path.

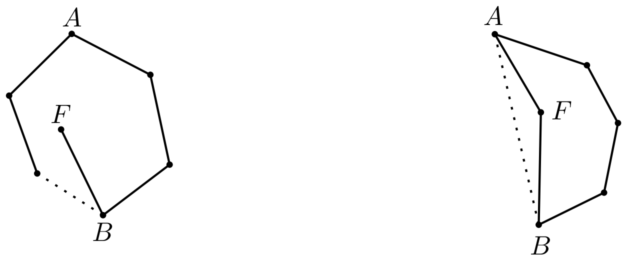


Fig. 7. Case 2b

If the other end direction is in R_1, then it directs upwards which can only occur if AB is an edge of the convex hull and the path extended from FB ends at A.

Similarly the path extended from FA will end in B so we found an α-good Hamiltonian cycle — see Figure 7 (right). If X has at least five elements, then there is an edge of K which is disjoint from A and B and we can use a previous argument. If X has four elements, then we take the path going from top to bottom starting at A and ending at B. In both cases the arguments are identical to the ones in **Case 2a**.

Case 2c. The set Y has at least two elements. We use induction on $|Y|$ and find an R-avoiding path $p = A_0 \dots A_n$ on Y. We will extend this path as follows. Let $F \in V$, that is, F a vertex of K. Connect A_0 (resp. A_n) to F and then connect F to one of its neighbours, G say, on the convex hull. Continue the path along the convex hull, we get a new path p^*. This path can be written as $p^* = ..GFp$ or $pFG..$, where the two dots represent the unique continuation of the path along

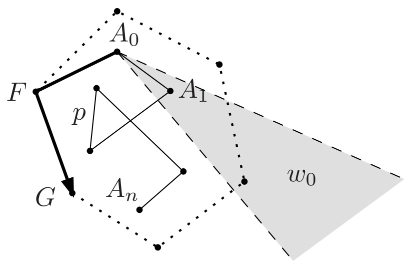


Fig. 8. The wedge w_o

the perimeter of K. The path p^* will be good if the angles at A_0 (resp. A_n) and at F are at least α.

Consider first the angle at A_0 and A_n. Let w_0 be the set of all points W for which the angle A_1A_0W is smaller then α — see Figure 8. Similarly let w_n be the set of all points W for which the angle $A_{n-1}A_nW$ is smaller then α. Both sets w_0 and w_n are wedges with an angle of 2α. The angle of p^* at A_0 (resp. A_n) is at least α if and only if F is not in the wedge w_0 (or w_n). Observe that V is not contained in w_0 as otherwise A_0 would be a vertex of K. Thus we can choose $F \in V$ so that the angle at A_0 is at least α — see Figure 8. Similarly, V is not contained in w_n, and we can choose F so that the angle at A_n is at least α.

Consider now the angle at F. To continue the path from F we have two choices for G to go along the perimeter of K. Since the angle at each vertex of K is at least 2α, one of the choices certainly yields a path whose angle at F is at least α. Consequently there is at least one good path p^* of the form $..GFp$ and one of the form $pFG..$.

One end of such a p^* is an edge of K and the other one is A_1A_0 or $A_{n-1}A_n$. If A_1A_0 or $A_{n-1}A_n$ is not in R, then we keep the end which is not in R and extend the path through the other end to get a good path on X which will be R-avoiding.

Thus we can assume that A_1A_0 is in R_1 and $A_{n-1}A_n$ is in R_2, say. This has the beneficial consequence that A is not in w_0 as the wedge w_0 lies completely below the horizontal line trough A, further denoted by l — see Figure 9 (left). Thus A can be taken for F and there is a good path of the form $p^* = ..GAp$. Similarly, $B \notin w_n$ and there is a good path $p^* = pBG..$.

Notice now that $p^* = ..GAp$ is R-avoiding unless both of its end directions are in R_2. This can only happen if AB is an edge of K and the angle A_0AB is smaller than α. Similarly, $p^* = pBG..$ is R-avoiding unless both of its end directions are in R_1. This can only happen if AB is an edge of K and the angle A_nBA is smaller than α.

Now let $A, C_1 \dots, C_k, B, A$ be the vertices of K in this order. It follows that all angles along the Hamiltonian cycle

$$A, C_1, \dots, C_k, B, A_n, A_{n-1} \dots, A_0, A$$

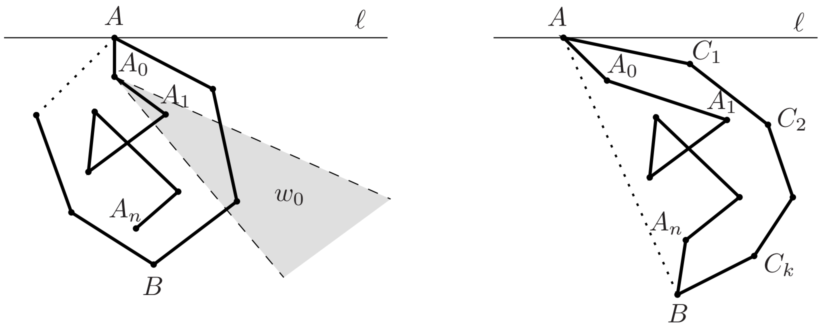


Fig. 9. Case 2c

are at least α. See Figure 9 (right). As we have seen in **Case 2b**, such a cycle produces an R-avoiding path unless $k = 1$.

The only remaining case is when $k = 1$, then K is the triangle ABC where we set $C = C_1$. Observe now that $|V \cap w_0| \leq 1$, since the angle at A of K is at least 2α and so w_0 cannot contain both B and C. Similarly, $|V \cap w_n| \leq 1$.

We assume next that the angle A_1A_0C is at least α, that is $C \notin w_0$. If the angle A_0CB is at least α, then the path $A_n \dots A_0CBA$ is R-avoiding — see Figure 10 (left). Otherwise the angle A_0CA is at least α and the path $BA_n \dots A_0CA$ is R-avoiding. From now on we can assume that $C \in w_0$.

Similarly we can find an R-avoiding path if the angle $A_{n-1}A_nC$ is at least α. From now on we can assume that $C \in w_n$.

This implies $V \cap w_0 = V \cap w_n = \{C\}$. Thus p can be extended to a good path p^* at both ends through both A and B.

The angle A_nAC has to be smaller than α as otherwise $A_0 \dots A_nACB$ is an R-avoiding path. Similarly the angle A_0BC is smaller than α. Wew have seen above that

$\angle A_0AB < \alpha$ and $\angle A_nBA < \alpha$.

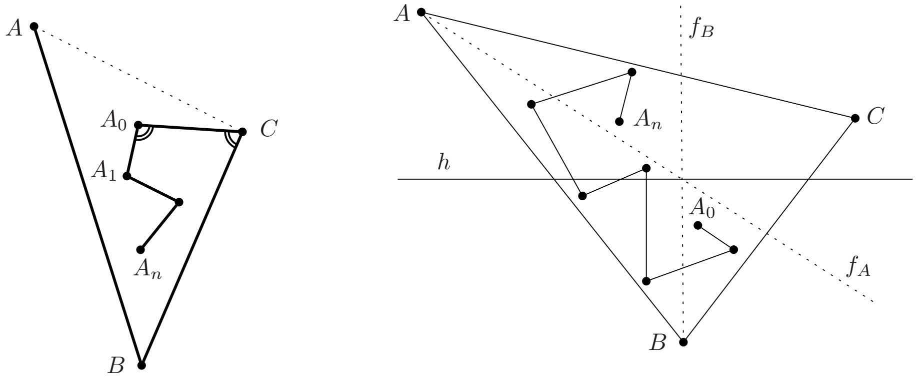


Fig. 10. Case 2c when the convex hull is a triangle

Now let f_A resp. f_B be lines through A and B halving the angle BAC and ABC. See Figure 10 (right). Let h be the horizontal line through the intersection of f_A and f_B. What we established so far implies that A_0 (resp. A_n) is in the triangle delimited by f_A, f_B, BC and by f_A, f_B, AC.

It follows then that A_0 is below and A_n is above h. Now w_0 lies entirely below h and w_n lies entirely above h, contradicting $C \in w_0 \cap w_n$.

4 Algorithm

Here we describe a quadratic algorithm for finding a good path on a given point set. Just as in the case of Theorem 1, we prove a stronger result.

Theorem 5. *Given a restriction R and a point set X in the plane with $|X| = n$, an R-avoiding path on X can be found in time $O(n^2)$.*

Proof. This is based on the proof of Theorem 2 so we only give a sketch. We are going to use the same notation. We assume by induction that there is an algorithm for the problem with $|X| = m < n$ that works in time at most cm^2. We assume now that X consists of n points.

It is not hard to see that the smallest angle of the convex polygon K (which is the convex hull of X) occurs at a vertex of K which has maximal or minimal x or y component among all points of X. There are at most 8 such vertices of K, and they can be found in time $4n$. Deciding whether the angle at such a vertex is smaller than 2α can be done in time n. Using this one can decide, in time $12n$, whether **Case 1** or **Case 2** occurs.

Assume first that **Case 1** occurs. Then the new restriction Q can be determined with 8 comparisons, the Q-avoiding path p on X_A can be found in time $c(n-1)^2$. Now either pA or Ap is an R-avoiding path, and a single comparison decides which one is. So we are finished with **Case 1** if $c \geq 7$ since then $cn^2 \geq 12n + 9 + c(n-1)^2$.

Assume next that **Case 2** occurs. The convex hull algorithm of Kirkpatrick and Seidel [5] finds K in time $O(n \log k)$ where k is the number of vertices of K. It is easy to see that in **Case 2a** and **Case 2b** we find an R-avoiding path in $O(n \log n)$ time. In **Case 2c**, we find an R-avoiding path p on $Y = X \setminus K$ in time $c(n-k)^2$ by induction (since $|Y| = n - k$). If neither $..GAp$ nor $pBG..$ is R-avoiding (which can be decided by checking their end directions), then AB is an edge of K and the Hamiltonian cycle $AC_1, \ldots, C_{k-2}B, A_n, \ldots, A_0, A$ gives an R-avoiding path unless $k = 3$. When $k > 3$ this path can be found immediately. So altogether this takes time $12n + O(n \log k) + c(n-k)^2$. This is smaller than cn^2 if c is large enough depending on the constant in the $O(n \log k)$ convex hull algorithm, as one can check easily.

Finally, the analysis of the case $k = 3$ shows that one of the paths $pCBA$, $ACpB$, $pACB$, or $CBpA$ is R-avoiding. Deciding which one takes four comparisons at most. In this case the algorithm takes time $12n + O(n \log 3) + c(n-3)^2 + 4$ which is, again, smaller than cn^2.

Acknowledgments. We thank Adrian Dumitrescu and Géza Tóth for bringing this problem to our attention and for stimulating discussions. We thank Ferran Hurtado for pointing out the reference [4] to us. Partial support for the first and second author from Hungarian National Foundation Grants No. 060427 and 062321 is also acknowledged. Research by the third author was partially supported by project 1M0545 of The Ministry of Education of the Czech Republic.

References

1. Aichholzer, O., Hackl, T., Hoffmann, M., Huemer, C., Pór, A., Santos, F., Speckmann, B., Vogtenhuber, B.: Maximizing Maximal Angles for Plane Straight-Line Graphs. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 458–469. Springer, Heidelberg (2007)
2. Bárány, I., Pór, A., Valtr, P.: Paths with no small angles (manuscript in preparation, 2007)
3. Fekete, S.: Geometry and the Travelling Salesman Problem, Ph.D. thesis, University of Waterloo (1992)
4. Fekete, S., Woeginger, G.J.: Angle-restricted tours in the plane. Comput. Geom.: Theory and Appl. 8, 195–218 (1997)
5. Kirkpatrick, Seidel, R.: The ultimate planar convex hull algorithm. SIAM J. Computing 15, 286–297 (1986)

Simpler Constant-Seed Condensers

Domingos Dellamonica[⋆]

Emory University – Department of Mathematics
ddellam@mathcs.emory.edu

Abstract. Condensers are functions which receive two inputs—a random string of bits chosen according to some unknown distribution and an independent uniform (short) seed—and output a string of bits which somehow preserves the randomness of the input. The parameters of interest here are the seed length, output length and how much randomness is preserved.

Here we present explicit algorithms for condensers which have constant seed size. Our constructions improve on previous constant-seed condensers of Barak et al (2005). When the input distribution has high min-entropy, we provide a condenser having optimal rate and seed chosen from $\{1, 2, 3\}$. The analysis of this construction is considerably simpler than those of previous constructions. For the low min-entropy regime, we provide a different construction which can be viewed as a pseudorandom coloring of hypergraphs. The analysis of this condenser involves a generalization of the celebrated Balog–Szemerédi–Gowers Theorem. As an example of the simplicity of the ideas behind this generalization, we improve Bourgain–Katz–Tao sum-product estimates in just a few lines.

1 Introduction

A *condenser* is a function that takes an input according to some unknown probability distribution, which may be far from uniform (a *weak source*), and outputs according to a distribution having a greater *randomness rate*. In the next section we shall formalize the notion of weak sources and explain how the concept of *min-entropy* is used to measure the amount of randomness of distributions. Informally, the min-entropy indicates how large is the probability of the most likely outcome in the underlying probability space.

This notion is particularly useful, for instance, if we want to use some distribution of random bits in a randomized algorithm. A randomized algorithm may be viewed as a deterministic algorithm that receives two inputs: one is an encoding of an instance and the other is a sequence of random bits, the *seed*, which is used by the algorithm for its internal decisions. An algorithm might give the wrong answers for a given instance when some particular seed is given; if that bad seed is sampled with high probability then the algorithm will fail (for that instance) with high probability.

[⋆] Supported by a CAPES/Fulbright scholarship.

There is a particularly strong interest in obtaining explicit condensers that can be computed in polynomial time (e.g., [1], [2], [3], [4], [5], [6], [7]). In this note we present two different constructions of constant-seed condensers. These constant-seed condensers have applications in randomness extraction, in particular, they may be composed with themselves recursively in order to convert a source with a small fraction of min-entropy into a (small) collection of blocks such that at least one of those blocks has high min-entropy. Our constructions, having greater condensing factor, may then be used to reduce the number of output blocks at the end of the recursion.

Our main contributions are: (1) an explicit construction of a condenser that achieves optimal condensing factor (and has only three output blocks); (2) a general upper bound on the condensing factor of condensers; (3) a condenser for any constant min-entropy rate that may be derived using only density results (generalizations of Gowers' Lemma on partial sum-sets); (4) sum-product estimates, that improve on Bourgain–Katz–Tao [8] with explicit bounds (although these bounds are only better than the current best bounds obtained by recent works [9], [10] for some range of the parameters). We remark that previous constructions rely on sum-product estimates (even if indirectly, as in Raz's condenser) and have condensing factor depending on some very small (and implicit) constant.

In (1), not only we obtain an explicit construction having optimal condensing factor, but the analysis is based on a new, quite simple, two-source extractor (Theorem 1). In (3), we dispense the use of much of the machinery used in previous constructions. Although at first the hypergraph lemmas we use for the analysis of the low-entropy condenser might seem overwhelming, the ideas behind the proof are simple tricks that nicely complement the celebrated Balog–Szemerédi–Gowers–Theorem. It also seems much easier to derive explicit bounds on the condensing factor using this method.

These hypergraph density theorems may be interesting on their own right and could find applications on proving sum-product estimates for more complicated arithmetic expressions (the celebrated sum-product theorem involves sum-sets like $A + A$ and product-sets like $A \cdot A$).

1.1 Organization of the Note

We formally state some definitions and introduce the (mostly standard) notation in Section 2. Some additive combinatorics results are stated in Section 3. Theorem 1 in that section may be of independent interest. These results are used to obtain our first construction, the `acond` asymmetric condenser, in Section 4.

Our low min-entropy condenser is obtained in Section 5. In Subsection 5.1 we show how some ideas used in the proof of the hypergraph density results can be applied to the problem of obtaining sum-product estimates for finite fields. In Section 6, we give upper bounds for the performance of any condenser (which may have slightly super-constant seed length).

2 Preliminaries

The objects we construct in this note are called condensers. A condenser is a function defined over a pair consisting of a binary string of some length n and a (usually) small seed. The output of the condenser is (usually) a smaller string and the aim is to get as much randomness from the string and seed as possible.

Here, we shall measure randomness using *min-entropy*, which became standard since Zuckerman [11] showed that several previous models of weak random distributions are special cases of distributions having some min-entropy lower bound.

Definition 1. *A* source *is a probability distribution on binary strings of some fixed length. Let X be a source over $\{0,1\}^n$. The* min-entropy *of X is defined as*[1]

$$H^{\infty}(X) = -\log\bigl(\max_{a\in\{0,1\}^n} \mathbf{P}[X=a]\bigr).$$

*Here, and throughout this note, logarithms have base 2. We shall say that X is a δ-*source *if its* min-entropy rate $\mathrm{r}(X) = H^{\infty}(X)/n$ *is at least δ. The* support *of a distribution, denoted by* $\mathrm{supp}(X)$ *is the set of elements having positive probability.*

Definition 2. *Let $\alpha_1,\ldots,\alpha_r$ be non-negative reals such that $\sum_{i=1}^r \alpha_i = 1$. The* convex combination *X of sources $X_1,\ldots,X_r \subseteq \Lambda = \{0,1\}^n$ having* weights *$\alpha_1,\ldots,\alpha_r$ is defined to be the distribution having $\mathbf{P}[X=x] = \sum_{i=1}^r \alpha_i \mathbf{P}[X_i = x]$ for all $x \in \Lambda$.*

Definition 3. *A source having uniform probability over its support is called a* flat source*.*

A very useful property of δ-sources is that they may be expressed as a convex combination of flat δ-sources. It is quite simple to show that if our constructions work for any flat source then they also work with any source having the same min-entropy. For this reason, we may safely assume that the sources are flat when convenient.

Definition 4. *The* statistical difference *between two sources $X, Y \subseteq \Lambda = \{0,1\}^n$, is defined as*[2]

$$\frac{1}{2}\|X-Y\|_1 = \frac{1}{2}\sum_{a\in\Lambda} \bigl|\mathbf{P}\bigl[X=a\bigr] - \mathbf{P}\bigl[Y=a\bigr]\bigr|.$$

*We say that X is α-*close *to Y if the statistical distance between X and Y is at most α.*

[1] For simplicity, we denote by the same symbol a distribution and a random variable having that same distribution assigned.

[2] The $1/2$ factor is just to keep the statistical distance in the range $[0,1]$.

We shall also make use of a concept that is directly linked to the min-entropy of distributions.

Definition 5. *Given a distribution $\mathcal{D}$. The* collision probability *of $\mathcal{D}$ is defined as*

$$\mathrm{cp}(\mathcal{D}) = \sum_{x \in \mathrm{supp}(\mathcal{D})} \mathcal{D}(x)^2,$$

where $\mathcal{D}(x)$ denotes the probability of x in the distribution.

When an element x is picked randomly from some source X, we write $x \in_R X$. If A is some set, we abuse the notation and write $x \in_R A$ when x is chosen uniformly from A.

3 Some Additive Combinatorics

In this section we shall prove a result, Corollary 2, bearing resemblance to [12, Lemma 3.1]. We remark that our results in this section are self-contained. In contrast, the analysis of the condensers of [3], [2] require deep results due to Plünnecke-Ruzsa (cf. [13], [14]), Gowers (cf. [15], [16]) and sum-product estimates for finite fields [8], [17].

Theorem 1. *Let X be a flat δ-source of length $2n$. Divide X into two blocks having n bits each, say $X = (A, B)$. Map*[3] *the elements of $\{0,1\}^n$ into elements of the finite field $\mathbb{F} = \mathrm{GF}(2^n)$. Let $\zeta \in \mathbb{F}$ be an arbitrary element. Denote by $\zeta \cdot A + B$ the distribution induced by taking $(a, b) \in_R (A, B) = X$ and calculating $\zeta a + b \in \mathbb{F}$. We have*

$$\mu = \mathbf{E}_{\zeta \in_R \mathbb{F}}\big[\mathrm{cp}(\zeta \cdot A + B)\big] \le 2^{-n} + 2^{-2\delta n}. \tag{1}$$

Furthermore, for every $\beta > 1$,

$$\mathbf{P}_\zeta\big[\mathrm{cp}(\zeta \cdot A + B) > \beta\mu\big] \le \frac{1}{\beta - 1} 2^{-n\,|1-2\delta|}. \tag{2}$$

Proof. Let $N = 2^n = |\mathbb{F}|$. Observe that $N\mu = \sum_{\zeta \in \mathbb{F}} \mathrm{cp}(\zeta \cdot A + B)$. Let $S = \mathrm{supp}(X)$ and $M = |S|$. For an arbitrary ζ, we have

$$\begin{aligned} M^2\,\mathrm{cp}(\zeta \cdot A + B) &= \#\big\{(a_1, b_1, a_2, b_2) \in S^2 \mid \zeta a_1 + b_1 = \zeta a_2 + b_2\big\} \\ &= M + \#\Big\{(a_1, b_1, a_2, b_2) \in S^2 \mid a_1 \ne a_2, \zeta = \frac{b_2 - b_1}{a_1 - a_2}\Big\}. \end{aligned} \tag{3}$$

Denote by S_ζ the set considered on the last row of equation (3). Since the sets in $\{S_\zeta\}_{\zeta \in \mathbb{F}}$ are pairwise disjoint, it follows that

$$\sum_\zeta |S_\zeta| = \Big|\bigcup_\zeta S_\zeta\Big| \le |S^2| = M^2,$$

[3] This map may be explicitly defined as follows. Find an irreducible polynomial $p(X) \in \mathrm{GF}(2)[X]$ of degree n (this can be done in time $\mathrm{poly}(n)$). Each n-bit binary string $b_0 b_1 \ldots b_{n-1}$ is mapped to $b_0 + b_1 X + \ldots b_{n-1} X^{n-1} \in \mathrm{GF}(2)[X]/\langle p(X)\rangle \cong \mathbb{F}$.

hence,

$$\begin{aligned} N\mu &= M^{-2}\sum_{\zeta}\big(M + |S_\zeta|\big) \\ &\le M^{-2}\big(NM + M^2\big) = 1 + N/M. \end{aligned} \tag{4}$$

Since X is a δ-source, we must have $M \ge 2^{2\delta n}$, and (1) follows directly from (4).

Let $T \subseteq \mathbb{F}$ be the set of elements ζ such that $\operatorname{cp}(\zeta \cdot A + B) \ge \beta\mu$. Define $\alpha = |T|/N$. Observe that any distribution over $\mathbb{F}$ has collision probability at least 2^{-n}, with equality holding if and only if the distribution is uniform. Therefore, we have

$$\mu \ge \alpha\beta\mu + (1-\alpha)\,2^{-n}.$$

We shall prove the case $\delta \ge 1/2$; the other case is similar, but instead of using the 2^{-n} lower bound for the collision probability, we use the fact that $\operatorname{cp}(\zeta \cdot A + B) \ge 2^{-2\delta n}$. It follows that $\mu(1-\alpha\beta) \ge (1-\alpha)\,2^{-n}$. By the upper bound (1) for μ, we get

$$(N^{-1} + 2^{-2\delta n})(1-\alpha\beta) \ge (1-\alpha)\,2^{-n}.$$

Hence,

$$1 + \alpha(\beta - 1) \le \frac{1-\alpha}{1-\alpha\beta} \le N(N^{-1} + 2^{-2\delta n}) = 1 + 2^{n(1-2\delta)},$$

proving that $\alpha(\beta-1) \le 2^{n(1-2\delta)}$ and that (2) holds.

From Theorem 1 we may get the following Corollary.

Corollary 1. *Let X_1 be a δ_1-source of n bits and let X_2 be an independent flat δ_2-source of $2n$ bits, with $\delta_2 > 1/2$. Assume $1 \gg \beta = \beta(n) > 2^{n(1-2\delta_2)}$. Then, interpreting X_2 as two n-bit blocks X_2^1 and X_2^2, we conclude that the distribution $X_1 \cdot X_2^1 + X_2^2$ is $\beta^{-1}2^{n(2-2\delta_2-\delta_1)}$-close to having collision probability at most $(1+4\beta)2^{-n}$.* □

Collision probability and min-entropy are tightly linked by Theorem 2. Hence, we may translate our previous results into min-entropy analogues effortlessly. Note that since min-entropy is a constrain over all elements in the support of the distribution it cannot be exactly equivalent to collision probability, which averages the effect of elements having high probability, hence they are equivalent under a small statistical distance.

Theorem 2. *Suppose X is a distribution over $\{0,1\}^n$ having $\operatorname{cp}(X) \le (1+\beta)2^{-n}$. Then X is $\sqrt{\beta}$-close to uniform.* □

From the above Theorem, we get the following corollary.[4]

[4] The differences between our result and that of [12, Lemma 3.1] are essentially: (a) their result requires three independent sources and, if one wishes for an absolute condensing factor, a translation of elements in $\{0,1\}^n$ into elements of a field $\mathrm{GF}(p)$ for some prime p must be explicitly known; (b) their conclusion is in terms of min-entropy rather than uniformity; (c) our result presupposes rather strong lower bounds on the min-entropy rates.

Corollary 2. *Let X_1 be a δ_1-source of n bits, X_2 be an independent δ_2-source of $2n$ bits, with $\delta_2 > 1/2$, and β as in Corollary 1. Then $X_1 \cdot X_2^1 + X_2^2$ is $(2\beta^{-1}2^{n(2-2\delta_2-\delta_1)} + 2\sqrt{\beta})$-close to uniform.* □

4 An Asymmetric Condenser

Our definition of condensers is a generalization of usual condensers since we allow possibly different output lengths. Formally, this is stated below.

Definition 6 ((Asymmetric) Condenser). *Let $r \in \mathbb{N}$, $\varepsilon > 0$ and $\rho > 1$. Assume $f_i \colon \{0,1\}^n \to \{0,1\}^{n_i}$, $i \in [r]$, are such that, for every δ-source X, there are sources $X_1, \ldots, X_r$ such that X is ε-close to being a convex combination of $\{X_i\}_{i=1}^r$ and, for each X_j, we have $\mathrm{r}(f_j(X_j)) \geq \rho\delta$. Then $f(X) = \big(f_1(X), \ldots, f_r(X)\big)$ is an r-*block (asymmetric) condenser *with* condensing factor ρ *and* error ε.

Now we introduce our construction. Let $n, n_1, n_2 \in \mathbb{N}$ be such that $n = n_1 + n_2$ and $n_1 \leq n_2 \leq 2n_1$. Given an n_2-bit string $x = x_1x_2 \ldots x_{n_2}$, define $A(x) = x_1x_2 \ldots x_{n_1}$ as the n_1 first bits of x and, similarly, define $B(x) = x_{n_2-n_1+1} \ldots x_{n_2}$ as the n_1 last bits of x. The function $\mathtt{acond}\colon \{0,1\}^n \to \{0,1\}^{n_1} \times \{0,1\}^{n_2} \times \{0,1\}^{n_1}$ is defined as

$$\mathtt{acond}(X_1, X_2) = \big(X_1, X_2, X_1 \cdot A(X_2) + B(X_2)\big),$$

where X_1 is an n_1-bit string and X_2 is a n_2-bit string. The arithmetic in the third block is done in $\mathrm{GF}(2^{n_1})$.

The `acond` condenser bears similarity to the condenser in [3]. Indeed, the latter divides the input into three equal parts (X_1, X_2, X_3) and outputs four blocks $(X_1, X_2, X_3, X_1 \cdot X_2 + X_3)$. We also remark that Zuckerman [4] described a condenser with only two output blocks which relies on the Line–Point Incidences Theorem. However, the condensing factor of such construction is proportional to some small constant associated with the Incidences Theorem. Raz's condenser [2], although having a neat and straightforward analysis, also uses a lot of additive number theory machinery, but this is hidden on the use of the extractor constructed by Barak, Impagliazzo and Wigderson [12], which also imposes a rather small condensing factor.

The formal statement of our construction is given in Theorem 3.

Theorem 3. *Let $5/9 < \delta \leq 1$, $\alpha \in (0,1)$ and $\gamma \in [1,2]$ be constants such that*

$$\delta > \frac{3+\gamma}{(\gamma+1)^2}, \text{ and } \alpha > \frac{1+\gamma}{2+\gamma}. \tag{5}$$

Let $n_1 = n/(1+\gamma)$ and $n_2 = n - n_1 = \gamma n_1$.[5] *There exists a polynomial time algorithm*

$$\mathtt{acond}\colon \{0,1\}^n \to \{0,1\}^{n_1} \times \{0,1\}^{n_2} \times \{0,1\}^{n_1}$$

[5] For simplicity we are assuming n_1 and n_2 are integers.

and $\varepsilon = 2^{-n^{\Omega(1)}}$ *such that* $\texttt{acond}(X)$ *is* ε*-close to being a convex combination of sources having min-entropy rate* $\alpha \cdot \delta + (1-\alpha) \cdot 1$ *in some block whenever* X *is a* δ*-source with* n *bits.*

4.1 Proof Strategy

The proof of the above theorem is quite similar to the analysis in [3]. First, let us restrict our attention to flat sources. Suppose that $X = (X_1, X_2)$ is a flat source. We show that, if there is a high probability that no output block has min-entropy rate as desired, there is a large number of elements in supp(X) that are mapped (in the third block $X_1 \cdot A(X_2) + B(X_2)$) into a small set. By the definition of the third block, this contradicts Corollary 2. The main subtlety here is that the blocks X_1 and X_2 are dependent and some care must be taken before applying Corollary 2.

Remark 1. If the asymmetry in the block sizes of `acond` is undesirable, the condenser can be composed with Raz's merger [2], [18] (a slight modification of that merger is needed to deal with different block sizes). This will slightly decrease the min-entropy and multiply the number of blocks by some constant but, at the same time, it will make the length of all output blocks the same and almost all of them will have high min-entropy.

5 Condensers for Low Min-Entropy

Since the simple analysis given in Section 4 does not apply for low min-entropies, we shall develop a different strategy for this setting. Our analysis for these condensers involve some hypergraph versions of the Balog–Szemerédi–Gowers theorem, which can be thought as density results of the following kind: suppose that we have some expression $f(x_1, \dots, x_l)$ such that, if $\mathcal{H}$ is some dense l-uniform hypergraph in which the cardinality $|f(\mathcal{H})|$ is small, then there are relatively large sets $A_1, \dots, A_l$ contained in the vertex-set of $\mathcal{H}$ such that $|f(A_1, \dots, A_l)|$ is also small.

In contrast, we shall show that some expressions f always produce large sets. Hence, a contradiction is derived if some dense $\mathcal{H}$ is supposed to have a small image under f. The first lower bound of this type is a slight modification of a bound for $|(A-A)/(A-A)|$ (which is given, for instance, in [12, Claim A4]) that employs Rusza's triangle inequality (cf. [19, p. 60]):

$$|X/Y| \leq \frac{|X/Z|\,|Z/Y|}{|Z|}. \tag{6}$$

Inequality (6) admits a one-line proof: just consider the representations of $x/y \in X/Y$ as $x/y = (x/z)(z/y) \in (X/Z) \cdot (Z/Y)$. Notice that each different $z \in Z$ produces a different pair $(x/z, z/y) \in X/Z \times Z/Y$.

Lemma 1. *Let* A *and* B *be subsets of a finite field* $\mathbb{F}$ *with* $|\mathbb{F}| = N$ *and* $|A|, |B| > N^{1/k}$ *for some integer* $k \geq 2$. *Then* $|(A-A)/(B-B)| \geq N^{1/(k-1)}$.

Using Rusza's triangle inequality (6) and Lemma 1 we obtain an analogous result for different sets.

Lemma 2. *Let A, B, C, D be subsets of a finite field $\mathbb{F}$ with $|\mathbb{F}| = N$ and $|A|$, $|B|$, $|C|$, $|D| > N^{1/k}$ for some integer $k \geq 2$. Then*

$$\left|\frac{A-B}{C-D}\right| \geq N^{\frac{2k-1}{2k(k-1)}}.$$

If $N = |\mathbb{F}| = p^r$ with p prime and r not a multiple of k then $N^{1/k}$ cannot be an integer and the conclusion of Lemma 1 holds with strict inequality for sets of cardinality larger than $N^{1/(k+1)}$. Hence, when $k \geq 2$, one may compose Lemmas 1 and 2 to prove that whenever $|X_1|, |X_2|, \ldots, |X_8| > N^{1/(k+1)}$, it holds that

$$\left|\frac{\frac{X_1-X_1}{X_2-X_2} - \frac{X_3-X_3}{X_4-X_4}}{\frac{X_5-X_5}{X_6-X_6} - \frac{X_7-X_7}{X_8-X_8}}\right| \geq N^{\frac{2k-1}{2k(k-1)}}. \tag{7}$$

Notice that $(2k-1)/\{2k(k-1)\} > 1/k$. This implies that an expression given by (7) always provides us with a large image relative to the lower bound known for the X_i's. In contrast, using Lemma 1 alone does not provide a good bound for the cardinality of $(A-A)/(B-B)$ when $|A| = |B| = \lfloor N^{1/k} \rfloor$.

In what follows, we shall use the following notation. Given some number $M \in \mathbb{N}$ and some $K = K(M)$, we say that $A \lessapprox B$ if there exists absolute constants $c_1 \neq 0$ and $c_2 \in \mathbb{R}$ such that $A \leq c_1 K^{c_2} B$. If both $A \lessapprox B$ and $B \lessapprox A$ we denote $A \approx B$. Observe that if we have a chain $A_1 \lessapprox A_2 \lessapprox A_3 \lessapprox \cdots \lessapprox A_l$, and l is an absolute constant, then $A_1 \lessapprox A_l$. If $A \lessapprox C$ and $B \lessapprox D$ then $A \cdot B \lessapprox C \cdot D$ (and, of course, this generalizes for any constant number of factors).

We are now ready to state a few analogs of the Balog–Szemerédi–Gowers Theorem that we use in our construction.

Lemma 3. *Let X_1, X_2, X_3 be sets of cardinality M. Suppose that $\mathcal{H} \subseteq X_1 \times X_2 \times X_3$ is such that $|\mathcal{H}| \approx M^3$ and $|f_1(\mathcal{H})|, |f_2(\mathcal{H})| \lessapprox M$, where we define $f_1(x_1, x_2, x_3) = x_1 - x_2$ and $f_2(x_1, x_2, x_3) = (x_1 - x_2)x_3$. Then, there are subsets $X_i' \subseteq X_i$ with $|X_i'| \approx M$ such that*

$$|f_1(X_1', X_2', X_3')|, |f_2(X_1', X_2', X_3')| \lessapprox M$$

and, furthermore, $|\mathcal{H} \cap X_1' \times X_2' \times X_3'| \gtrapprox M^3$.

Lemma 3 together with the Hypergraph Balog–Szemerédi–Gowers Theorem of [16] are used to prove a density result closer to our objective.

Lemma 4. *Let $A_1, \ldots, A_{10}$ be subsets of a finite field having $|A_i| \approx M$. Suppose $\mathcal{H} \subseteq \prod_{i=1}^{10} A_i$ is such that $|\mathcal{H}| \gtrapprox M^{10}$ and that $|f(\mathcal{H})| \lessapprox M$, $|f_i(\mathcal{H})| \lessapprox M$, for all $i \in [4]$, $|g(\mathcal{H})| \lessapprox M$ and $|h(\mathcal{H})| \lessapprox M$, where $f(\mathbf{a}) = \prod_{i=1}^{4} f_i(\mathbf{a})$, with $f_i(\mathbf{a}) = a_{2i-1} - a_{2i}$, $g(\mathbf{a}) = \{f(\mathbf{a}) - a_9\}/a_{10}$ and $h(\mathbf{a}) = f(\mathbf{a})/a_{10}$. Then, for all $i \in [10]$, there exists $A_i' \subseteq A_i$, with $|A_i'| \approx M$, such that*

$$\left|\{(A_1' - A_1')(A_3' - A_3')(A_5' - A_5')(A_7' - A_7') - A_9'\}/A_{10}'\right| \lessapprox M. \tag{8}$$

Furthermore, $\left|\mathcal{H} \cap \prod_{i=1}^{10} A_i'\right| \gtrapprox M^{10}$.

Finally, we use Lemma 4 to obtain the density result that will be used directly in the analysis of the condenser construction (Theorem 4).

Lemma 5. *Let $\mathcal{H} \subseteq A_1 \times \cdots \times A_{18}$, with $|A_i| \approx M$ and $\mathcal{H} \gtrapprox M^{18}$. Suppose that*

$$|f_i(\mathcal{H})|, |g_j(\mathcal{H})|, |h_k(\mathcal{H})|, |q_l(\mathcal{H})| \lessapprox M,$$

where

$$\begin{gathered} f_i(\mathbf{a}) = a_{2i-1} - a_{2i}, \text{ for } i \in [8], \\ g_1 = f_1 f_4 f_6 f_8, \quad g_2 = f_2 f_3 f_6 f_8, \quad g_3 = f_2 f_4 f_5 f_8, \quad g_4 = f_2 f_4 f_6 f_7, \\ h_k(\mathbf{a}) = (g_k(\mathbf{a}) - a_{17})/a_{18}, \text{ for } k \in [4], \\ q_l(\mathbf{a}) = g_l(\mathbf{a})/a_{18}, \text{ for } l \in [4]. \end{gathered}$$

Then there exists sets $A'_{2i-1} \subseteq A_{2i}$, for $i \in [8]$, with $|A'_{2i-1}| \approx M$, such that, letting $Y_i = A'_{2i-1} - A'_{2i-1}$, we have

$$\left| \frac{Y_1/Y_2 - Y_3/Y_4}{Y_5/Y_6 - Y_7/Y_8} \right| \le \left| \frac{Y_1 Y_4 Y_6 Y_8 - Y_2 Y_3 Y_6 Y_8}{Y_2 Y_4 Y_5 Y_8 - Y_2 Y_4 Y_6 Y_7} \right| \lessapprox M. \tag{9}$$

Using Lemma 5 we obtain our condenser for low min-entropy rates.

Theorem 4. *Let $0 < \delta < 2/3$ be fixed. There is $n_0 = n_0(\delta)$ and $\gamma = \Omega(\delta)$, such that for all $n \ge n_0$, and $\varepsilon = \varepsilon(n) = \Omega(2^{-\gamma\delta n})$, we have an explicit polynomial time algorithm*

$$\texttt{cond}\colon (n) \times [38] \to (m),$$

with $m \ge n/18$, such that for any δ-source X, there exists subsources $X_i \subseteq X$ with densities α_i, having $\|X - \sum_{i=1}^{38} \alpha_i X_i\|_1 \le \varepsilon$ and $H^\infty(\texttt{cond}(X_i, i)) \ge (1 + \gamma)\delta m$.

Proof (outline). We shall interpret strings of the input as being edges in a hypergraph with classes $A_i = \{0,1\}^m$, $i = 1, \ldots, 18$. In particular, if x is an n-bit string in the support of some flat source and π_i is the projection onto the coordinates $[(i-1)m + 1, im]$, we map x to the edge $(\pi_1(x), \ldots, \pi_{18}(x))$.

The proof of the theorem follows by contradiction using ideas borrowed from the condenser construction in [2]. We define the functions $\{\texttt{cond}(\cdot, i)\}_{i=1}^{38}$ to be

$$\{\pi_1, \ldots, \pi_{18}\} \cup \{f_i\}_{i=1}^{8} \cup \{g_j\}_{j=1}^{4} \cup \{h_k\}_{k=1}^{4} \cup \{q_l\}_{l=1}^{4},$$

where the functions above are those of Lemma 5.

Suppose that the condenser fails for some X. Then, there exists a subsource $Z = X - \sum_{i=1}^{38} \alpha_i X_i$ of density $> \varepsilon$ such that $|\operatorname{supp}(\texttt{cond}(Z, i))| < 2^{(1+\gamma)\delta m}$ for all i. Moreover, we may assume that Z is flat. We may now conveniently represent the support of Z as a hypergraph over $\prod_{i=1}^{18} \pi_i(Z)$ and apply Lemma 5. This yields sets satisfying equation (9) and hence, contradicting inequality (7).

5.1 Explicit Sum-Product Estimates for Finite Fields

One may use simple techniques as the ones described in the construction of our condenser to prove explicit sum-product estimates over finite fields. There has been many interesting improvements over the sum-product estimates of Bourgain–Katz–Tao [8], for instance, see [20], [21], for estimates dealing with small subsets of finite fields (where small means up to square root of the field size). For larger sets, explicit estimates have appeared recently in [9], [10]. Here, we provide a simple and explicit estimate, Theorem 5, for subsets larger than $\sqrt{|\mathbb{F}|}$. Our result beats recent estimates only for sets of size close to $\sqrt{|\mathbb{F}|}$, but has the advantage of being much simpler and shorter.

Theorem 5. *Let $\mathbb{F}$ be some finite field. Suppose that $A \subseteq \mathbb{F}$ is such that $|\mathbb{F}| = N$ and $M = |A| = 2N^\alpha$ for some $\alpha \geq 1/2$. Then* $\max\{|A \cdot A|, |A + A|\} = \Omega\big(M^{1+c}\big)$ *where $c = \frac{1-\alpha}{12\alpha}$.*

Proof. Let

$$S = \Big\{x \in \mathbb{F} \,:\, \#\big\{(a_1, a_2) \in A^2 \,:\, a_1 + a_2 = x\big\} \geq \frac{M^2}{100\,|A + A|}\Big\}.$$

Consider the graph $G' = \big(A, \big\{\{a_1, a_2\} \in \binom{A}{2} \,:\, a_1 + a_2 \in S\big\}\big)$. Clearly, $e(G') \geq 0.98\binom{M}{2}$. Roughly, the number of vertices having degree smaller than $0.8M$ is at most $0.1M$. Hence, removing those vertices, we obtain a graph G with minimum degree at least $0.7M$ over a set $A' \subseteq A$ with $|A'| \geq 0.9M$.

Given any element $(a_1 - a_2)a_3 \in (A' - A') \cdot A$, we can represent it in $\Omega\big(M^5/|A + A|^2\big)$ ways as a sum of four elements in $\pm A \cdot A$: (1) pick an arbitrary $a \in \Gamma_G(a_1) \cap \Gamma_G(a_2)$; (2) pick one of the $M^4/(10^4\,|A + A|^2)$ quadruples $b_1, b_2, b_3, b_4 \in A$ satisfying $a_1 + a = b_1 + b_2$ and $a_2 + a = b_3 + b_4$; (3) let $(a_1 - a_2)a_3 = b_1a_3 + b_2a_3 - b_3a_3 - b_4a_3 \in 2A \cdot A - 2A \cdot A$. It follows that $|(A' - A') \cdot A| = O\big(|A \cdot A|^4 |A + A|^2 M^{-5}\big)$.

By Lemma 1 we have $\tilde{A} = \frac{A' - A'}{A' - A'} = \mathbb{F}$. On the other hand, every $(a_1 - a_2)/(a_3 - a_4) \in \tilde{A}$ admits at least $\Omega(M)$ representations as xy^{-1} with $x, y \in (A' - A') \cdot A$: just take an arbitrary $0 \neq a \in A$ and set $x = (a_1 - a_2)a$ and $y = (a_3 - a_4)a$. It follows that $N = |\tilde{A}| = O\big(|A \cdot A|^8 |A + A|^4 M^{-11}\big)$.

6 Upper Bound for the Performance of Condensers

We now state a general upper bound on the condensing factor of condensers. This upper bound shows that the asymmetric condenser `acond` achieves essentially optimal condensing factor.

Theorem 6. *Let $r \in \mathbb{N}$, $\delta = \delta(n) \in (0, 1)$ and $\delta_i = \delta_i(n) \in (0, 1)$ for $i \in [r]$. Let $f_i \colon \{0, 1\}^n \to \{0, 1\}^{n_i}$, with $i \in [r]$, be arbitrary functions. Set $u = \sum_i \delta_i n_i$*

and $m = \sum_i n_i$. Suppose that we have $m - u < (1-\delta)n - r$ and $\delta_i n_i \geq c \log n$ for some absolute constant $c > 0$. Then, there exists a subset $S \subseteq \{0,1\}^n$ with $|S| \geq 2^{\delta n}$ such that the flat δ-source X over S satisfies

$$\left|\operatorname{supp}\bigl(f_i(X)\bigr)\right| < \frac{2}{3} 2^{\delta_i n_i} \tag{10}$$

for all i.

In particular, $f = (f_1, f_2, \ldots, f_r)$ cannot be a condenser with error smaller than $1/3$.

Let us consider the following class of convex combination of sources.

Definition 7. *Assume $\delta_1, \ldots, \delta_r \in (0,1)$ and $n_1, \ldots, n_r \in \mathbb{N}$ are given. Let $Z \subseteq \{0,1\}^{n_1 + \cdots + n_r}$ be a source such that, for some i, the ith block of Z has min-entropy $\delta_i n_i$. We say that i is a* good block *of Z. A convex combination of sources $\{Z^i\}_{i=1}^r$ such that i is a good block of Z^i is called a* good combination.

A corollary of equation (10) is that the output $f(X) = \bigl(f_1(X), \ldots, f_r(X)\bigr)$ is not even $\frac{1}{3}$-close to a good combination. Indeed, for any $Z = (Z_1, \ldots, Z_r)$ such that Z_i is a δ_i-source, we have

$$\begin{aligned} \mathbf{P}\bigl[Z \in \operatorname{supp}\bigl(f(X)\bigr)\bigr] &\leq \mathbf{P}\bigl[Z_i \in \operatorname{supp}\bigl(f_i(X)\bigr)\bigr] \\ &\leq \left|\operatorname{supp}\bigl(f_i(X)\bigr)\right| 2^{-\delta_i n_i} < \frac{2}{3}. \end{aligned} \tag{11}$$

This implies that, in any good combination, the probability mass associated to the set $\operatorname{supp}\bigl(f(X)\bigr)$ is less than $2/3$. Hence, the statistical distance from $f(X)$ to any good combination is greater than $1/3$.

In view of Theorem 6, we conclude that the condenser `acond` of Theorem 3 achieves asymptotically optimal condensing factor. Indeed, take $r = 3$ and $\delta_1 = \delta_2 = \delta_3 = \alpha\delta + 1 - \alpha$ with α and γ as in Theorem 3. Since $1 - \delta_i = \alpha(1-\delta)$, it follows that

$$\sum_{i=1}^{3} (1-\delta_i) n_i = \alpha \frac{2+\gamma}{1+\gamma}(1-\delta)n,$$

which is slightly greater than $(1-\delta)n - 3$, a necessary condition, given the result of Theorem 6.

Remark 2. The bound obtained in this section is essentially best possible for general condensers. A probabilistic condenser that asymptotically attains this upper bound can be shown to exist.

Acknowledgments. I would like to thank my Master's adviser, Prof. Yoshi Kohayakawa, for his support and for introducing me to the field of randomness extraction.

References

1. Ta-Shma, A., Umans, C., Zuckerman, D.: Loss-less condensers, unbalanced expanders, and extractors. In: STOC: ACM Symposium on Theory of Computing (STOC) (2001)
2. Raz, R.: Extractors with weak random seeds. In: Electronic Colloquium on Computational Complexity (ECCC), vol. 4(099) (2004)
3. Barak, B., Kindler, G., Shaltiel, R., Sudakov, B., Wigderson, A.: Simulating independence: new constructions of condensers, Ramsey graphs, dispersers, and extractors. In: STOC 2005: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, pp. 1–10. ACM Press, New York (2005)
4. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. In: Electronic Colloquium on Computational Complexity (ECCC) (2005)
5. Reingold, O., Shaltiel, R., Wigderson, A.: Extracting randomness via repeated condensing. SIAM J. Comput. 35(5), 1185–1209 (2006)
6. Ta-Shma, A., Umans, C.: Better lossless condensers through derandomized curve samplers (2006)
7. Guruswami, V., Umans, C., Vadhan, S.: Extractors and condensers from univariate polynomials. In: Electronic Colloquium on Computational Complexity (ECCC), vol. 6(134) (2006)
8. Bourgain, J., Katz, N., Tao, T.: A sum-product estimate in finite fields, and applications. Geometric And Functional Analysis 14, 27–57 (2004)
9. Hart, D., Iosevich, A., Solymosi, J.: Sum-product estimates in finite fields (2006)
10. Vu, V.: Sum-product estimates via directed expanders (2007)
11. Zuckerman: General weak random sources. In: FOCS: IEEE Symposium on Foundations of Computer Science (FOCS) (1990)
12. Barak, B., Impagliazzo, R., Wigderson, A.: Extracting randomness using few independent sources. In: FOCS, pp. 384–393 (2004)
13. Ruzsa, I.: An analog of Freiman's theorem in groups, Structure theory of set adition. Astérisque 258, 323–326 (1999)
14. Nathanson, M.B.: Additive Number Theory: Inverse Problems and the Geometry of Sumsets. Graduate Texts in Mathematics, vol. 165. Springer, New York (1996)
15. Gowers, W.T.: A new proof of Szemerédi's theorem for arithmetic progressions of length four. Geom. Funct. Anal. 8(3), 529–551 (1998)
16. Sudakov, B., Szemerédi, E., Vu, V.H.: On a question of Erdős and Moser. Duke Math. J. 129, 129–155 (2005)
17. Konyagin, S.V.: A sum-product estimate in fields of prime order (2003)
18. Dvir, Z., Raz, R.: Analyzing linear mergers. In: Electronic Colloquium on Computational Complexity (ECCC) (2005)
19. Tao, T., Vu, V.H.: Additive Combinatorics. In: Cambridge Studies in Advanced Mathematics (2006)
20. Garaev, M.Z.: An explicit sum-product estimate in $\mathbb{F}_p$. In: ArXiv Mathematics e-prints (February 2007)
21. Hawk Katz, N., Shen, C.Y.: Garaev's Inequality in finite fields not of prime order. In: ArXiv Mathematics e-prints (March 2007)

Parallel Repetition of the Odd Cycle Game

Kooshiar Azimian and Mario Szegedy[⋆]

Department of Computer Science, Rutgers, the State University of NJ
110 Frelinghuysen Road, Piscataway, NJ, USA 08854-8019
kooshiar@cs.rutgers.edu, szegedy@cs.rutgers.edu

Abstract. Higher powers of the Odd Cycle Game has been the focus of recent investigations [3,4]. In [4] it was shown that if we repeat the game d times in parallel, the winning probability is upper bounded by $1 - \Omega(\frac{\sqrt{d}}{n\sqrt{\log d}})$, when $d \leq n^2 \log n$. We

1. Determine the exact value of the square of the game;
2. Show that if Alice and Bob are allowed to communicate a few bits they have a strategy with greatly increased winning probability;
3. Develop a new metric conjectured to give the precise value of the game up-to second order precision in $1/n$ for constant d.
4. Show an $1 - \Omega(d/n \log n)$ upper bound for $d \leq n \log n$ if one player uses a "symmetric" strategy.

Keywords: parallel repetition, two prover games, CHSH.

1 Introduction

It is well known due to the famous parallel repetition theorem of Ran Raz, that higher powers of any two-prover game have exponentially decreasing values:

Theorem 1 (Parallel Repetition Theorem [10]). *For every fixed answer size c and for every $\epsilon > 0$ there is a $\delta > 0$ such that if $v(G) < 1 - \epsilon$ then $v(G^d) < (1 - \delta)^d$ for every $d \geq 1$. Moreover, for fixed c we have $\delta \in \Omega(\epsilon^{32})$.*

A simplification by Hollenstein improves on the dependence between ϵ and δ:

Theorem 2 (Hollenstein [7]). *In Theorem 1 $\delta \in \Omega(\epsilon^3)$ for fixed c.*

We will focus on the Odd Cycle Game, which is a special XOR game. XOR games are two-prover games for which the answers, a of Alice and b of Bob are binary, and to every question pair there is a $rel \in \{=, \neq\}$ such that the answers are good iff $a\ rel\ b$. For the n-cycle game, G_n (n is odd), $v(G_n) = 1 - 1/2n$. Since this value tends to 1, when $n \to \infty$, the game is a candidate for a counter-example to:

Conjecture 1 (Strong parallel repetition conjecture [4]). $\delta \in \Omega(\epsilon)$.

[⋆] Supported by NSF grant EMT-0523866.

The conjecture, among others, would imply that to prove the famous Unique Game Conjecture it would be sufficient to prove the NP-hardness of the gap problem $MAXLIN2(1-\epsilon, 1-\sqrt{\epsilon})$ [4]. Feige and Lovász [5], and later Cleve et.al. proved that $\delta \in \Omega(\epsilon^2)$ for XOR games [3]. Both proofs are based on the duality theorem for semidefinite programming and a clever product theorem.

Uri Feige, Guy Kindler and Ryan O'Donnell have recently shown that $v(G_n^d) = 1-\Omega(\frac{\sqrt{d}}{n\sqrt{\log d}})$ (for $d \leq n^2 \log n$) by a novel geometric intuition [4], thus improving on [5,3] for the odd cycle games for $d < n^{2-c}$ $(c > 0)$. They also showed that improving their bound requires improving lower bounds on the surface area of high-dimensional foams. In other words if someone wants to improve on the upper bound for the odd cycle games (in particular, if one wants to prove the Strong parallel repetition conjecture), she also needs to improve on the best lower bound to the following hard question: What is the least surface area of a cell that tiles R^d by the lattice Z^d? We need to note that the version of odd cycle game they discussed is sightly (but not essentially) different from our version. We list some further differences compared to [4]:

1. We have found tight bounds for the two rounds repetition of our version.
2. In [4] the connection between geometry and the odd cycle games is one-directional, while in our case, for $d = 2$ it is two-directional.
3. In Section 3, we give a meaning to the "2-cycle game," which turns out to be identical to the so-called CHSH game (Section 4). The connection might provide a hint to the exact general formula for $v(G_n)^d$, when d is small: Strategies for small powers of the CHSH game can be searched with computer. Conversely, for powers of the CHSH game the geometrical approach presented in this paper and in [4] may turn out to be the right approach.

We develop a topological machinery for our version of the odd cycle game and invent a new, interesting metric. Our approach, although does not represent an essential departure from [4], may provide additional insight. In particular, we believe, the connection between the topology and the game is more transparent in our discussion. More importantly, due to our precise metric, we are potentially able to obtain the exact constants for small powers of the game up to the second order term in n.

In the second half of the paper we extend our investigation to the case when the two players can communicate, and achieve winning probability 1 or close to 1 for linear number of repeats. We also show that if either Alice or Bob plays a *symmetric* strategy, then the value of the strategy almost meets the strong parallel repetition bound.

2 A Syntactic Aside

To avoid the nightmare of "onion-ized" expressions when dealing with iterated moduli of the type ((expr (mod a)) (mod b)), we introduce the following convention: $\tilde{+}$ is the operator that adds two integers mod n and returns an integer in $\{-\lfloor\frac{n-1}{2}\rfloor, \ldots, \lceil\frac{n-1}{2}\rceil\}$; $\oplus$ is the mod 2 addition. It takes two integers and returns their sum mod 2. The result is an integer in $\{0, 1\}$. If the left hand side of an equation is mod addition, the reader needs to reduce the the right hand side with the same modulus. Sometimes we need to forcefully reduce both sides of an equation by the mod 2 operator. We extend all operators to vectors, coordinate-wise. The $=$, $\geq$ and other relations hold to vectors if they hold for all coordinates.

3 The Odd Cycle Game and Its Powers

Let n be an odd number. The *n-cycle game*, G_n, starts with Alice and Bob picking a pair of colorings of the n-cycle, $\mathcal{S}_A, \mathcal{S}_B : [n] \to \{0, 1\}$, called strategies. The verifier then selects $0 \leq x < n$ and a type $t \in \{0, 1\}$, both randomly and accepts iff $\mathcal{S}_A(x) \oplus \mathcal{S}_B(x \tilde{+} t) = t$. The best strategy $\mathcal{T} = (\mathcal{S}_A, \mathcal{S}_B)$ is where $\mathcal{S}_A(x) = \mathcal{S}_B(x) = x \mod 2$. $v(G_n) = v(\mathcal{T}) = 1 - 1/2n$.

Remark 1. In [4] $t \in \{0, 1, -1\}$ randomly, and the test is the same.

The verifier of the dth power of the n-cycle game selects a tuple $\boldsymbol{x}$ from $[n]^d$ randomly and a type $\boldsymbol{t}$ from $[2]^d$ randomly. The strategies of Alice and Bob are: $\mathcal{S}_A, \mathcal{S}_B : [n]^d \to [2]^d$. The verifier then evaluates the following predicate:

$$\mathcal{S}_A(\boldsymbol{x}) \oplus \mathcal{S}_B(\boldsymbol{x} \tilde{+} \boldsymbol{t}) \stackrel{?}{=} \boldsymbol{t}. \tag{1}$$

By definition, the equality of the two sides means that the acceptance criterion has to hold for all coordinates.

4 Even Cycle Games

The so-called CHSH game has received considerable attention in quantum physics due to the Einstein, Podolsky, Rosen paradox. To single bit questions x and y the single bit answers $\mathcal{S}_A(x)$, $\mathcal{S}_B(y)$ are accepted, iff

$$\mathcal{S}_A(x) \oplus \mathcal{S}_B(y) = xy.$$

The value of this game is 0.75. The paradox arises in the quantum world, where Alice and Bob can win the game with probability 0.85 by communicating through a pair of entangled electrons. The communication is instant even when Alice and Bob are light years apart, and the evidence that communication has happened is the increased value of the game. This met Einstein's disapproval. Let us generalize the acceptance criterion of the n-cycle game, (n is odd) as:

$$\mathcal{S}_A(x) \oplus \mathcal{S}_B(x \tilde{+} t) \stackrel{?}{=} \ t \oplus \delta(x, t),$$

where $\delta : [n] \times \{0, 1\} \to \{0, 1\}$ is any function with $\sum_{n,t} \delta(n, t) = 0 \mod 2$. The transformation preserves all properties of the game, including its and its powers' values. To extend the notion to even ns we just replace the condition on δ with

$$\sum_{n,t} \delta(n, t) = n + 1 \mod 2. \tag{2}$$

Regardless of the parity of n, the value of the n cycle game is $1 - \frac{1}{2n}$. Although for simplicity we discuss only odd cycle games, all our arguments carry over to even cycle games. The CHSH game arises from $\delta(x,t) = xt - x - t \mod 2$ (one can check that (2) holds). The value of CHSH^2 is $5/8$ (this also follows from our results). By exhaustively searching all strategies, Aaronson and Toner [3] independently have found that $v(\mathrm{CHSH}^3) = 31/64$, while any strategy, where the first two rounds are independent of the third have value at most $5/8 * 3/4 = 30/64$. This shows that something interesting is happening for the third power too.

5 Local Consistency and Pearls

For fixed $\mathcal{S}_A$ let Bob optimize over his strategies: $v(\mathcal{S}_A) = \max_{\mathcal{S}_B} v(\mathcal{S}_A, \mathcal{S}_B)$. Bob now has to optimize only locally: For every $\boldsymbol{y} \in [n]^d$ Bob needs to set $\mathcal{S}_B(\boldsymbol{y})$ such that the number of $\boldsymbol{t}$s for which $\mathcal{S}_A(\boldsymbol{y} \tilde{-} \boldsymbol{t}) \oplus \mathcal{S}_B(\boldsymbol{y}) = \boldsymbol{t}$ is maximized. Putting it differently, Bob needs to pick the plurality value of $\mathcal{S}_A(\boldsymbol{y} \tilde{-} \boldsymbol{t}) \oplus \boldsymbol{t}$ on the cube $Q_{\boldsymbol{y}} = \{\boldsymbol{y} \tilde{-} \boldsymbol{t} \mid \boldsymbol{t} \in \{0,1\}^d\}$. We say that $\boldsymbol{x}, \boldsymbol{x'} \in Q_{\boldsymbol{y}}$ are consistent iff there exists an answer B for Bob to $\boldsymbol{y}$, which is consistent with both $\mathcal{S}_A(\boldsymbol{x})$ and $\mathcal{S}_A(\boldsymbol{x'})$. This is the case if and only if: $\mathcal{S}_A(\boldsymbol{x}) - \mathcal{S}_A(\boldsymbol{x'}) = \boldsymbol{x} \tilde{-} \boldsymbol{x'} \mod 2$. *Notice that consistency is independent of $\boldsymbol{y}$!!*

Definition 1 (consistency of a region). *A region $R \subseteq [n]^d$ is* consistent *(w.r.t. $\mathcal{S}_A$) if for every $\boldsymbol{x}, \boldsymbol{x'} \in R$ it holds that $\mathcal{S}_A(\boldsymbol{x}) - \mathcal{S}_A(\boldsymbol{x'}) = \boldsymbol{x} \tilde{-} \boldsymbol{x'} \mod 2$.*

Whether or not two points of $[n]^d$ are consistent, from the point of view of computing $v(\mathcal{S}_A)$ is interesting only locally. Define $R_{\boldsymbol{y}}$ as the maximal consistent sub-region of $Q_{\boldsymbol{y}}$ (if there are more than one, break the tie arbitrarily). Then:

$$v(\mathcal{S}_A) = \frac{1}{n^d} \sum_{\boldsymbol{y} \in [n]^d} \frac{|R_{\boldsymbol{y}}|}{2^d}. \qquad (3)$$

What prevents $R_{\boldsymbol{y}} = Q_{\boldsymbol{y}}$ for all $\boldsymbol{y} \in [n]^d$? We give a completely topological explanation. We call a sequence $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_k \in [n]^d$ *cycle* if $\boldsymbol{x}_k = \boldsymbol{x}_0$.

Definition 2. *Let n be odd. A cycle $C = \boldsymbol{x}_0, \ldots, \boldsymbol{x}_k \in [n]^d$; $\boldsymbol{x}_k = \boldsymbol{x}_0$ is*

$$\textbf{topologically non-trivial} \quad \textit{iff} \quad \sum_{i=0}^{k-1} \boldsymbol{x}_{i+1} \tilde{-} \boldsymbol{x}_i \neq \boldsymbol{0}.$$

$$\textbf{topologically odd} \quad \textit{iff} \quad \sum_{i=0}^{k-1} \boldsymbol{x}_{i+1} \tilde{-} \boldsymbol{x}_i \neq \boldsymbol{0} \mod 2$$

Definition 3 (Pearl, Consistent Pearl). *A* pearl $\wp$ *is a collection $\{R_{\boldsymbol{y}} | \boldsymbol{y} \in [n]^d\}$ such that $R_{\boldsymbol{y}} \subseteq Q_{\boldsymbol{y}}$ for all $\boldsymbol{y} \in [n]^d$. A pearl $\wp$ is* consistent *(with respect to $\mathcal{S}_A$) if all regions $R_{\boldsymbol{y}}$ are consistent (with respect to $\mathcal{S}_A$).*

The value of the n-cycle game is the maximum of $\frac{1}{n^d 2^d}\sum_{\boldsymbol{y}\in[n]^d}|R_{\boldsymbol{y}}|$, where $\{R_{\boldsymbol{y}}\}_{\boldsymbol{y}}$ is *some* consistent pearl for *some* $\mathcal{S}_A$. To translate the maximization problem to a purely combinatorial problem we characterize these pearls.

Definition 4. *A cycle $\boldsymbol{x}_0,\ldots,\boldsymbol{x}_k\in[n]^d$; $\boldsymbol{x}_0=\boldsymbol{x}_k$ is* contained *in $\wp=\{R_{\boldsymbol{y}}\}_{\boldsymbol{y}}$, if there are $R_0,\ldots,R_{k-1}\in\wp$ such that $\boldsymbol{x}_i,\boldsymbol{x}_{i+1}\in R_i$ for $0\le i\le k-1$.*

Lemma 1. *For fixed $\mathcal{S}_A$ let $C=\boldsymbol{x}_0,\ldots,\boldsymbol{x}_k$ ($\boldsymbol{x}_k=\boldsymbol{x}_0$) be such that $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i+1}$ are consistent w.r.t. $\mathcal{S}_A$ for $0\le i\le k-1$. Then C is a topologically even cycle.*

Lemma 2 (Main Lemma). *$\wp=\{R_{\boldsymbol{y}}\}_{\boldsymbol{y}}$ is a consistent pearl with respect to some $\mathcal{S}_A$, or shortly a* consistent pearl*, if and only if it does not contain a topologically odd cycle.*

6 The Topological Approach

Let $T_d=(0,1]^n\subseteq\mathbf{R}^n$ be the d dimensional unit torus and $n\times T_d=(0,n]^n$. Consider a cycle $C=\boldsymbol{x}_0,\ldots,\boldsymbol{x}_k$ ($\boldsymbol{x}_k=\boldsymbol{x}_0$) in $[n]^d$ (n is odd) . We can naturally embed $[n]^d$ into $n\times T_d$. If we connect each $\boldsymbol{x}_i$ with $\boldsymbol{x}_{i+1}$ via the geodesics Γ_i in $n\times T_d$, we get a closed curve, $\Gamma=\Gamma(C)=\cup_i\Gamma_i$. We can then study the group element $g(\Gamma)$ of the homotopy group $\pi_1(T_d)$, associated with Γ. It is well known that $\pi_1(T_d)=\mathbf{Z}^d$, where the i^{th} coordinate of $g\in\pi_1(T_d)$ tells how many times a curve wraps around the cycle in the i^{th} coordinate direction. The following lemma, which justifies the terms "topologically trivial" and "odd," is easy to prove:

Lemma 3. *Let C be a cycle in $[n]^d$ (n is odd). Then C is topologically trivial (even) if and only if $g(\Gamma(C))=\mathbf{0}$ ($g(\Gamma(C))\in(2\mathbf{Z})^d$).*

Corollary 1. *A pearl $\wp$ is consistent if and only if whenever C is a cycle of $\wp$, $g(\Gamma(C))\in(2\mathbf{Z})^d$.*

7 Blockers

Blockers are subsets of T_d that intersect with all cycles that are not in the homotopy class of $\mathbf{0}$. Blockers are called *foams* in [4]. *Odd blockers* are subsets of T_d that intersect with all cycles whose homotopy class is not in $(2\mathbf{Z})^d$. We can construct blockers and odd blockers from $d-1$ skeletons of cell complexes. For an intuition the reader can skip to Section 9, that discusses the case of $d=2$.

$\wp_n(B)$ (Odd blockers $\rightarrow$ consistent pearls): Let B be an odd blocker of T_d such that $(n\times B)\cap[n]^d=\emptyset$. For any $\boldsymbol{y}\in n\times T_d$ we define the solid cube $Q^*_{\boldsymbol{y}}=\boldsymbol{y}\hat{-}Q^*$, where Q^* is $[0,1)^n$ and $\hat{-}$ is the wrap-around subtraction inside $n\times T_d$. We then define an equivalence relation between the elements of $Q_{\boldsymbol{y}}$: $\boldsymbol{x},\boldsymbol{x'}\in Q_{\boldsymbol{y}}$ are equivalent if they can be connected inside $Q^*_{\boldsymbol{y}}$ without intersecting $n\times B$. Let $R_{\boldsymbol{y}}(B)$ be a maximum size equivalence class (we break ties in some arbitrary systematic manner). We define the pearl $\wp_n(B)=\{R_{\boldsymbol{y}}(B)\}_{\boldsymbol{y}}$.

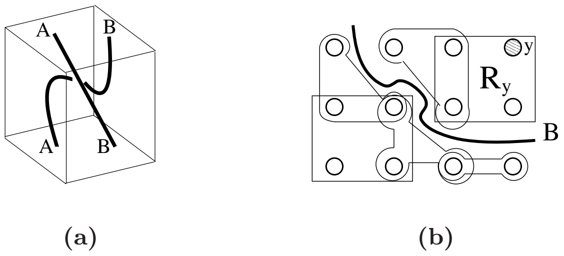


Fig. 1. **(a)** Even, but non-trivial cycle **(b)** Part of a pearl created by blocker B

Lemma 4. *Let $n \geq 3$, odd, B be an odd blocker in T_d, Then pearl $\wp_n(B)$ is consistent.*

Proof. Let $C = \boldsymbol{x}_0, \ldots, \boldsymbol{x}_k$ ($\boldsymbol{x}_k = \boldsymbol{x}_0$) be a cycle in $\wp_n(B)$. We need to prove that C is not topologically odd. Since C is *in* $\wp_n(B)$, there exist $\boldsymbol{y}_i \in [n]^d$ such that $\boldsymbol{x}_i, \boldsymbol{x}_{i+1} \in Q_{\boldsymbol{y}_i}$ (Definition 4). By the definition of $\wp_n(B)$ we can construct a curve Γ'_i that connects $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i+1}$, runs inside $Q^*_{\boldsymbol{y}_i}$, and which is not blocked by B. Since $\Gamma' = \cup_i \Gamma'_i$ is not blocked by B, we have that $g(\Gamma') \in (2\mathbf{Z})^d$. For Corollary 1, however, we need $g(\Gamma(C)) \in (2\mathbf{Z})^d$.

Definition 5. *For curves Γ and Γ' define $|\Gamma, \Gamma'|_\infty$ as $\inf_\phi |x, \phi(x)|_\infty$, where ϕ is a one-one continuous map between Γ and Γ'.*

Since $|\Gamma', \Gamma(C)|_\infty \leq 1$, we can apply the following lemma:

Lemma 5. *if Γ, Γ' are curves in $n \times T_d$ and $|\Gamma, \Gamma'|_\infty < \frac{n}{2}$, then $g(\Gamma) = g(\Gamma')$.*

8 A New Metric

Let S be a piece of a smooth $d-1$ dimensional surface (or a union of these) in T_d such that $(n \times S) \cap [n]^d = \emptyset$ for every $n \geq 1$. Create the pearl $\wp_n(S) = \{R_{\boldsymbol{y}}\}_{\boldsymbol{y}}$ in the same way as in Section 7 when S was an odd blocker. Although now $\wp_n(S)$ is not (necessarily) consistent, we can still associate the value $v_n(S) = \frac{1}{2^d n^d} \sum_{\boldsymbol{y} \in [n]^d} |R_{\boldsymbol{y}}|$ to it. It turns out that the measure

$$\lim_{n \to \infty} n(1 - v_n(S)) = \lambda(S)$$

exists, and it is additive in the sense that if S is a disjoint union of pieces $S_1, \ldots, S_m$ then $\lambda(S) = \sum_{i=1}^m \lambda(S_i)$. What is this measure? If S is a piece of a $d-1$ dimensional hyper-plane with normal vector $S = (s_1, \ldots, s_d)$ (where s_i is the projection size of S on the i^{th} coordinate plane), then

$$\lambda(S) = \frac{1}{2} E\left(|\sum_{i=1}^{d} s_i \chi_i| \right), \tag{4}$$

where χ_i are independent $\{1, -1\}$-valued uniform random variables.

Definition 6 (Diamond norm). *For vector $A = (a_1, \dots, a_n)$ define its diamond norm as $|A|_\diamondsuit = E(|\sum_{i=1}^{d} a_i \chi_i|)$, where χ_i are independent $\{1, -1\}$-valued uniform random variables.*

Lemma 6. *$|A|_\diamondsuit = |A|_\infty$, when $d = 2$, and $|A|_\diamondsuit \geq |A|_\infty$ otherwise. $\frac{|A|_2}{\sqrt{2}} \leq |A|_\diamondsuit \leq |A|_2$.*

Only the $\frac{|A|_2}{\sqrt{2}} \leq |A|_\diamondsuit$ relation is hard to show, which comes from the Khintchine inequality. Notice that the diamond norm is the same as the L_2 norm within a constant factor, which explains [4]. The additivity of λ and (4) gives:

Theorem 3. *Let odd blocker B be the $d-1$-skeleton, of a smooth cell complex. Assume that $(\forall n > 0)$ $(n \times B) \cap [n]^d = \emptyset$. Then for every $\epsilon > 0$ there exists an n_ϵ such that the strategy associated with $\wp_n(B)$ $(n \geq n_\epsilon)$ has value at least*

$$1 - \frac{1+\epsilon}{2n} \iint_B |dB|_\diamondsuit .$$

It is unclear to us if there is any strategy with greater value than the one that arises from the best odd blocker or even from the best blocker. In [4] it is conjectured that blockers give the best strategies. They also show that $v(G_n^d) = 1 - \Omega(\frac{\sqrt{d}}{n\sqrt{\log d}})$ for $d \leq n^2 \log n$.

9 The Case of $d = 2$

Two dimensional cell complexes on the torus are simply graphs drawn on the torus. A graph drawn on the torus is torical (i.e. "takes the full use of the torus") if its edges block all non-trivial cycles.

Theorem 4. *Every topologically non-trivial* simple cycle *in the two dimensional torus is also topologically odd.*

Corollary 2. *A graph on T_2 is torical iff its edges block all the odd cycles.*

If a torical graph creates more than one facets we can delete at least one of its edges and remain torical. For a torical graph the Euler's theorem gives:

$$-f + e - v = 0,$$

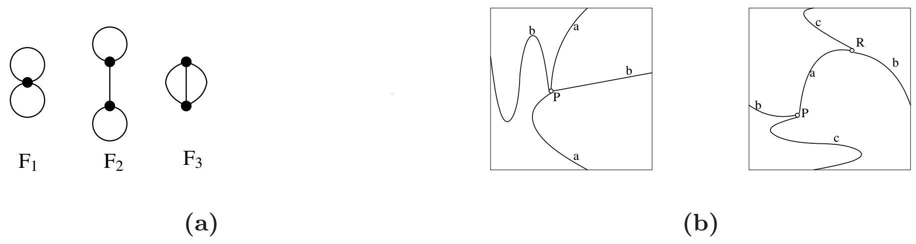
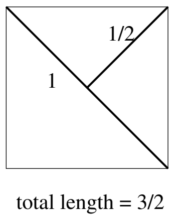


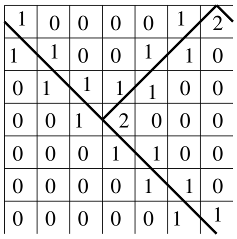
Fig. 2. (a) shows the three combinatorially possible graphs **(b)** shows the two of the three that are torical

where f is the number of faces, e is the number of edges and v is the number of vertices. We can assume that all vertices have degree at least 3. This gives $e \geq \frac{3}{2}v$. Thus if $f = 1$, the possible parameter combinations are $v = 1, e = 2$ and $v = 2, e = 3$. This gives us three graphs, F_1, F_2, and F_3 (see Figure 2). Only F_1 and F_3 have torical representations. Furthermore, F_1 can be viewed as a special case of F_3, where one of the edges is shrunk to a point.

Lemma 7. *The minimum total edge-length of any torical representation of F_3 on the unit torus is at least* 1.5. *Above all lengths are measured in the L_∞ norm (hence in the Diamond norm).*

Proof. As in Figure 2 **b** we denote the two nodes of F_3 by P and R. Let the length of the shorter horizontal projection of $\overline{PR}$ be $x \leq 0.5$ and the length of the shorter vertical projection be $y \leq 0.5$. Without loss of generality we can assume that the L_∞ length of one of the three edges is at most 0.5. This edge has lengths at least $\max\{x, y\}$ and then the other two have length at least $\max\{x, 1-y\}$ and $\max\{1-x, y\}$, respectively. Assume $x \geq y$. For the total L_∞ length of the graph we now get $x + (1-x) + (1-y) \geq 1.5$.





(a) **(b)**

Fig. 3. (a) An optimal blocker in the two-dimensional unit torus with respect to the L_∞ norm. **(b)** An optimal strategy for $n = 7$ arising from the blocker on the left. The torus is scaled up by a factor of n. Losses are shown in each square.

Figure 3 **a** shows that 1.5, is achievable and Figure 3 **b** shows how this gives rise to a strategy $\mathcal{S}_2$ with value exactly $1 - \frac{3}{4n}$. The number in each square denotes the "loss" $4 - |R_{\boldsymbol{y}}|$. One can easily see that the total loss is precisely $3n$.

Theorem 5. $v(\mathcal{S}_2) = 1 - \frac{3}{4n}$.

We also give a matching upper bound relying on Lemma 7.

Theorem 6. $v(G_n^2) = 1 - \frac{3}{4n}$.

Proof. We need to prove the upper bound. Consider a strategy $\mathcal{S}_A$ of Alice and associate a torical graph, G, with it. We define "portions" of G in each square $Q^*_{\boldsymbol{y}}$, using the consistency classes created by $\mathcal{S}_A$. Instead of a detailed explanation we refer the reader to Figure 4. In each Q^*_x the total L_∞ length of the portion of G is $\frac{1}{2}(4 - |R_x|)$ thus $\sum_{e \in E(G)} |e|_\infty = 2n^2(1 - v(\mathcal{S}_A))$ by (3) ($|e|_\infty$ is the L_∞ length of e).

Consider an arbitrary *simple* (i.e. not self-intersecting) cycle Γ' in T_2 that does not intersect G. We show that $g(\Gamma') = \mathbf{0}$. Pick an orientation for Γ', and a staring point P in it. We define a cycle $C = \boldsymbol{x}_0, \ldots, \boldsymbol{x}_k$ ($\boldsymbol{x}_k = \boldsymbol{x}_0$) by the following algorithm. We start with the empty sequence, and walk along Γ' from P. Whenever we leave the current $Q^*_{\boldsymbol{y}}$, we look for the grid point that is closest (in L_∞ norm) to the point we exit $Q^*_{\boldsymbol{y}}$, and we add this grid point to the sequence. We continue until we get back to P, and pass a little further until we add $\boldsymbol{x}_k$, which is $\boldsymbol{x}_0$. The main observation is that $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i+1}$ are consistent for $0 \le i \le k-1$. This comes from reviewing Figure 4 (a). By Lemma 1 this implies that C is topologically even. Now $|\Gamma', \Gamma(C)| < 1$ together with Lemmas 3, 5 and Corollary 2 imply that G is torical, which in turn, by Lemma 7 gives that $\sum_{e \in E(G)} |e|_\infty \ge 1.5n$. Hence $1.5n \le 2n^2(1 - v(\mathcal{S}_A))$, as needed.

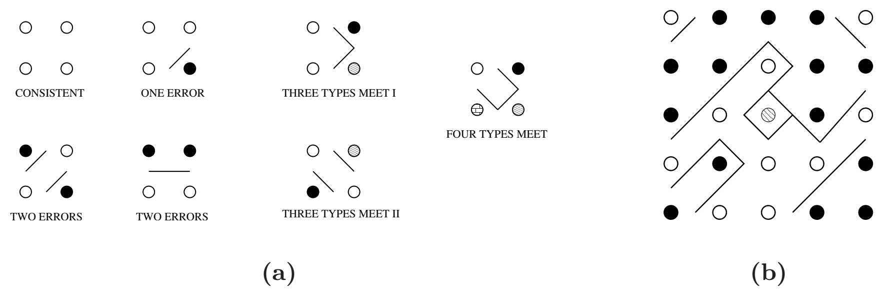


Fig. 4. **(a)** Turning a strategy into lines **(b)** Part of the emerging graph

10 Gap Commitment Problem

The starting point of this research was the following question of Gavinsky: Is it true that $v(G_n^d) = 1 - \Omega(1)$ for $d = n$? When $d \approx n$, non-topological type strategies could be promising. In the rest of the article we discuss such a "different" type of strategy.

Let us think of the question vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ to Alice and Bob as d-element multi-sets of Z_n. Let $\underline{\boldsymbol{x}} = \{x_i\}_{i=1}^d$, $\underline{\boldsymbol{y}} = \{y_i\}_{i=1}^d$ be their supporting sets. When $d = n$, there are typically points and short intervals missing from $\underline{\boldsymbol{x}}$ (and from $\underline{\boldsymbol{y}}$). We call such a interval *gap*. Since the provers receive different questions, they see different gaps in their multi-set, but since their questions correlate, so will the gaps. We say that $\alpha \in Z_n$ is in a gap of size l for a verifier's question $\underline{\boldsymbol{x}}$ to Alice, if the interval $\{\alpha, \alpha \pm 1, \ldots, \alpha \pm l\}$ is disjoint from $\underline{\boldsymbol{x}}$.

The main idea is that if Alice and Bob can agree with probability $1 - \epsilon$ over the verifier's question pair $(\boldsymbol{x}, \boldsymbol{y})$ in some $\alpha \in Z_n$ such that α is in some gap with respect to both $\boldsymbol{x}$ and $\boldsymbol{y}$, then they can win the game with probability $1 - \epsilon$. Indeed, to every x_i Alice can answer with $x_i \hat{-} \alpha \mod 2$ (Bob with $y_i \hat{-} \alpha$

mod 2), where $\hat{-}$ returns the mod n value of the difference in the non-negative representation. A refinement of this idea is that it is sufficient if Alice and Bob find gaps with respect to their inputs with non-empty intersection. Such agreement actually seems easy at first: both players just pick their largest gap. The catch is that their largest gaps will be totally different with constant probability. In the above algorithm Alice (Bob too) plays a *symmetric* strategy: Her answers depend only on the mlti-set of $\boldsymbol{x}$. For symmetric strategies we have found a bottleneck:

Theorem 7. *Assume that Alice plays a symmetric strategy and $d = \Omega(n \log n)$. Then regardless of Bob's strategy the winning probability is at most $1 - \Omega(1)$.*

The above strategy works, however, if we allow a small amount of communication. **Gap Commitment Problem (GCP):** The input to both Alice and Bob are multi-sets from Z_n of size d. How many bits of communication is required to find an x, which is in a gap for both Alice and Bob? We denote this problem by GCP_n^d. It seems that GCP is an interesting problem on its own right.

Let D denote the deterministic one-round communication complexity and let D_ϵ denote the one round (Distributional) communication complexity (by a deterministic protocol) which is allowed to fail on ϵ fraction of all inputs (see [8]), where the distribution of $(\boldsymbol{x}, \boldsymbol{y})$ is the same as in the odd cycle game.

Theorem 8. *Let $d < n/10$, $t \geq 1$. Then:*

$$D(GPC_n^d) \in O(\log n),$$
$$D_{\frac{1}{t}}(GPC_n^d) \in O(\log \log t).$$

Proof. In the first case, Alice can send the location of a point in a gap with size at least 2 to Bob. Since $d < n/10$, such a point exists. This point is in a gap with size at least 1 for Bob.

In the second case Alice looks at $\{1, \ldots, \lceil 5 \log t \rceil\}$ and selects a gap in this set with size of at least 2 with center g, if exists. She just needs $O(\log \log t)$ bits (one-round) communication to report the location of g to Bob.

Remark 2. Although the above strategies are simple, they demonstrate that no strong direct sum theorems hold for the communication version of the odd cycle game: $D(G_n) = 2 \gg \frac{10}{n} D(G_n^{n/10}) \in O(\frac{\log n}{n})$. Different amortized measures were studied by I. Parnafes, R. Raz, and A. Wigderson [9], and also, by T. Feder, E. Kushilevitz, M. Naor, and N. Nisan [6] and by Ambainis et. al. [1].

References

1. Ambainis, A., Buhrman, H., Gasarch, W., Kalyanasundaram, B., Torenvliet, L.: The Communication Complexity of Enumeration, Elimination, and Selection. Journal of Computer and System Science 63(2), 148–185 (2001)
2. Clauser, J., Horne, M.A., Shimony, A., Holt, R.A.: Phys. Rev. Lett. 23, 880 (1969)

3. Cleve, R., Slofstra, W., Unger, F., Upadhyay, S.: Perfect Parallel Repetition Theorem for Quantum XOR Proof Systems. In: IEEE Conference on Computational Complexity, pp. 109–114 (2007)
4. Feige, U., Kindler, G., O'Donnell, R.: Understanding parallel repetition requires understanding foams. In: IEEE Conference on Computational Complexity, pp. 179–192 (2007)
5. Feige, U.: Lovász: Two-prover one-round proof systems: their power and their problems. In: Proc. 24th ACM Symp. on Theory of Computing, pp. 733–744 (1992)
6. Feder, T., Kushilevitz, E., Naor, M., Nisan, N.: Amortized communication complexity. SIAM Journal of Computing, 239–248 (1995)
7. Holenstein, T.: Parallel repetition: simplifications and the no-signaling case. In: Proc. of 39th STOC (to appear, 2007)
8. Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge University Press, Cambridge (1997)
9. Parnafes, I., Raz, I., Wigderson, A.: Direct Product Results and the GCD Problem, in Old and New Communication Models. In: Proc. of the 29th STOC, pp. 363–372 (1997)
10. Raz, R.: A Parallel Repetition Theorem. Siam Journal of Computing 27(3), 763–803 (1998)

I/O-Efficient Point Location in a Set of Rectangles

Yakov Nekrich

Dept. of Computer Science, University of Bonn
yasha@cs.uni-bonn.de

Abstract. In this paper we present an external memory data structure for point location queries in a set of d-dimensional rectangles. Our data structure uses $O(N/B)$ blocks of space and supports point location queries in $O(\log_B^{d-1} N)$ I/Os, where N is the number of rectangles and B is the block size.

We also present a $O((N/B)\log_B N)$ space data structure that supports point location queries in a two-dimensional rectangular subdivision of a $U \times U$ grid in $O(\log_2 \log_B U + (\log_2 \log_B N)^2)$ I/Os and a $O((N/B)\log_B^2 N)$ space data structure that supports point location queries in a three-dimensional rectangular subdivision in $O(\log_B N)$ I/Os. As an application of our result, we describe a data structure for three-dimensional orthogonal range reporting queries on a grid of size U with $O(\log_2 \log_B U + (\log_2 \log_B N)^2 + T/B)$ I/O operations per query, where T is the number of points in the answer.

1 Introduction

In the point location problem a set of disjoint geometric objects is stored in a data structure, so that for an arbitrary point p an object that contains p can be found efficiently. In this paper we consider the special case when the objects stored in the data structure are rectangles and describe efficient external memory data structures for several variants of this problem.

A survey of results for the general point location problem in the RAM model is well beyond the scope of this paper. Instead we refer the reader to [21] and confine ourselves to point location queries for rectangles. The internal memory data structure of Edelsbrunner, Haring and Hilbert [12] supports point location queries in the set of N disjoint d-dimensional rectangles for $d \geq 2$ in $O(\log_2^d N)$ time. Smid [20] improved this result and presented a dynamic data structure that supports queries in $O(\log_2^{d-1} N \log_2 \log_2 N)$ time and updates in $O(\log_2^2 N)$ time. A *rectangular subdivision* of a d-dimensional space X is a set of disjoint rectangles $R_1, \ldots, R_N$ whose union covers X. An internal memory data structure for point location queries in a rectangular subdivision of two- and three-dimensional grid is described by de Berg, van Kreveld, and Snoeyink [9]. If all coordinates are integers bounded by a parameter U, the data structure of [9] supports queries in a two- and three-dimensional subdivision in $O((\log_2 \log_2 U)^2)$ and $O((\log_2 \log_2 U)^3)$ time respectively. Observe that the problems of searching in an arbitrary set of d-dimensional rectangles and searching in a rectangular subdivision of the d-dimensional space are not equivalent for $d \geq 3$: for instance, a set of N 3-dimensional rectangles can partition the 3-dimensional space into $\Theta(N^{3/2})$ rectangles.

In the external memory model the data is stored in *disk blocks* of size B, a block can be read into internal memory from disk (resp. written from internal memory into disk) with one I/O operation, and computation can only be performed on data stored in the internal memory. The space usage is measured in the number of blocks, and the time of computation is measured in the number of I/O operations. A more detailed description of the external memory model can be found in e.g. [24] or [3].

The data structure of Goodrich et.al [16] supports two-dimensional point location queries in $O(\log_B N)$ I/Os. There are also dynamic data structures for two-dimensional point location in a monotonous [1] and general [7] subdivision. Data structures for processing a batch of point location queries are described in e.g. [5], [6], [11]. There are no external memory data structures for general d-dimensional point location queries when $d \geq 3$. The stabbing-max data structure of Agarwal, Arge, and Yi [2] can be used to answer point location queries in a set of d-dimensional rectangles in $O(\log_B^d N)$ I/Os and $O((N/B)\log_B^{d-1} N)$ space. In the case when the only allowed operation is element comparison, $\Omega(\log_B N)$ is a natural lower bound for many data structure problems including point location. However, when other operations besides comparisons are allowed and all elements are bounded by an appropriate parameter U, there are external memory data structures for predecessor searching [19] and orthogonal range reporting in two dimensions [18] that support queries in $o(\log_B N)$ I/O operations.

In this paper we present external memory data structures for point location in a set of rectangles. We present a $O(N/B)$ space data structure that supports point location queries in a set of d-dimensional rectangles, $d \geq 2$, in $O(\log_B^{d-1} N)$ I/O operations. For the case of a two-dimensional rectangular subdivision of the grid of size U we present a data structure that uses $O((N/B)\log_B N)$ space and supports queries in $O(\log_2 \log_B U + (\log_2 \log_B N)^2)$ I/O operations. An application of this result is a data structure that supports three-dimensional orthogonal range reporting queries on a $U \times U \times U$ grid in $O(\log_2 \log_B U + (\log_2 \log_B N)^2 + T/B)$ I/Os and uses $O((N/B)\log_2^4 N)$ blocks of space. Thus we demonstrate that two-dimensional point location queries and three-dimensional range reporting queries can be supported with $o(\log_B N)$ I/O operations when the size of the universe is bounded by an appropriate parameter U. We also obtain a $O((N/B)\log_B^2 N)$ space data structure for a three-dimensional rectangular subdivision that supports point location queries in $O(\log_B N)$ I/Os.

In section 2 we describe data structures that support rectangular point location queries in a special case when all point coordinates but the first one are from the interval $[1, B^{1/d}]$. In section 3 we describe a data structure for point location in a set of rectangles. Data structures for point location in two- and three-dimensional rectangular subdivisions are given in section 4.

2 Point Location on the $(1, d)$ Grid

In this section we consider a special case of the point location problem in a set of d-dimensional rectangles. A d-dimensional (r, d) grid is defined as a set of d-dimensional points P, if the last $d - r$ coordinates of all points belong to the interval $[1, B^{1/d}]$. If the first r coordinates of points on an (r, d) grid are bounded by a parameter U, we denote it a (U, r, d) grid. In this section we consider two data structures for the case $r = 1$. In the next section we will extend our construction to an arbitrary integer constant r and thus obtain a data structure for the point location in a set of r-dimensional rectangles.

Lemma 1. *There exists a $O(N/B)$ space data structure that answers point location queries for a set of N d-dimensional rectangles on a $(1, d)$ grid in $O(\log_B N)$ I/Os.*

Proof. For convenience, we denote the first coordinate the x-coordinate throughout this paper. For a rectangle $R = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$, a_1 and b_1 are called respectively the opening x-bound and the closing x-bound of R. We construct a B-tree $\mathcal{T}_1$ with node degree $\rho = B^{1/d}$ on the set of x-bounds of rectangles in $\mathcal{R}$. In each leaf of $\mathcal{T}_1$, B different x-bounds are stored. We say that rectangle R contains a $(d-1)$-dimensional point p', if the projection of R on the last $d-1$ coordinates contains p'. Consider an arbitrary node v of $\mathcal{T}_1$. For every point p' in the $(d-1)$-dimensional rectangle $[1, B^{1/d}]^{d-1}$, $l_i(p')$ ($s_i(p')$) denotes the largest (smallest) x-bound stored in a descendant of the i-th child of v, such that $l_i(p')$ ($s_i(p')$) is the x-bound of a rectangle R that contains p'. All $l_i(p')$ and $s_i(p')$ for all possible $p' \in [1, B^{1/d}]^{d-1}$ are stored in the node v. For convenience, we assume that $s_0(p') = l_0(p') = -\infty$ and $s_{\rho+1}(p') = l_{\rho+1}(p') = +\infty$ for all possible p'.

The search for the rectangle R that contains the query point p starts at the root r of $\mathcal{T}_1$ and visits all nodes v on the path from r to a leaf of $\mathcal{T}_1$. Let $\pi(p)$ be the projection of p on the last $d-1$ coordinates, and let x_p be the first coordinate of p. Let i denote the largest index, such that $s_i(\pi(p)) \leq x_p$ and j denote the smallest index, such that $l_j(\pi(p)) \geq x_p$. If $i = j$ and $0 < i < \rho+1$, then x-bounds of the rectangle that contains p can be stored only in descendants of the i-th child of v. If $i = j = 0$ ($i = j = \rho + 1$), then the x-bounds of all rectangles stored in the descendants of v are smaller (greater) than x_p. It remains to consider the case $i < j$. In this case $s_j(\pi(p)) > x_p$ and $l_i(\pi(p)) < x_p$. If $l_i(\pi(p))$ is the opening x-bound of some rectangle R, then the closing x-bound of R is greater than x_p: Suppose that c'_p, the closing x-bound of R is smaller than or equal to x_p. Then, c'_p is stored in the k-th child of v for some $k > i$, and $s_k(\pi(p)) \leq x_p$. This contradicts to our choice of index i. Hence, if $l_i(\pi(p))$ is the opening x-bound of some rectangle R, then R contains p. If $l_i(\pi(p))$ is the closing x-bound, then $s_{i+1}(\pi(p)) > x_p$ and no rectangle R contains p.

Therefore the search procedure works as follows: In every visited internal node v we identify the largest index i such that $s_i(\pi(p)) \leq x_p$ and the smallest index j, such that $l_j(\pi(p)) \geq x_p$. If $i = j = 0$ or $i = j = \rho+1$, then there is no rectangle R that contains p. If $i = j$ and $0 < i < \rho + 1$, then the search continues in the

i-th child of v. If $i < j$ and $l_i(\pi(p))$ is the opening x-bound of some rectangle R, then p belongs to the rectangle R. If $i < j$ and $l_i(\pi(p))$ is the closing x-bound, then there is no rectangle that contains p. If the search reaches a leaf l of $\mathcal{T}_1$, then all rectangles stored in l can be examined in $O(1)$ I/Os. Since the search procedure spends $O(1)$ I/Os in every visited node, the query is answered with $O(\log_B N)$ I/Os.

If the x-bounds of all rectangles belong to interval $[1, N]$, the result of Lemma 1 can be further improved.

Lemma 2. *There exists a $O(N/B)$ space data structure that answers point location queries for a set of N d-dimensional rectangles on $(N, 1, d)$ grid in $O(1)$ I/Os.*

Proof. We say that a rectangle R is long if the length of its projection on the x-axis is greater than B, otherwise R is short. For every $p' \in [1, B^{1/d}]^{d-1}$ x-bounds of all long rectangles that contain p' are stored in a data structure $D(p')$. $D(p')$ is organized as an array with N/B elements: for every interval $I_s = [(s-1)B, sB]$, $s = 1, \ldots, N/B$, the entry $D(p')[s]$ contains pointers to long rectangles R, such that the projection of R on the x-axis intersects with I_s. An interval I_s of length B can intersect with projections of at most two long rectangles. Hence, the total number of rectangles in each $D(p')$ is $O(N/B)$, and all data structures $D(p')$ can be stored in $O(N/B)$ blocks of space. For every interval I_s, $s = 1, \ldots, N/B$, all short rectangles R whose x-projections intersect with I_s are stored in a data structure D_s. Since the x-bound of a short rectangle can intersect with at most two intervals, the total number of elements in all data structures D_s is $O(N)$. We implement D_s using Lemma 1; hence, all data structures D_s can be packed into $O(N/B)$ blocks of space. Observe that the maximum number of rectangles that can be stored in a data structure D_s is $O(B^2)$: if R is stored in D_s, then at least one point of R belongs to a rectangle $I_s \times [1, B^{1/d}]^{d-1}$ that consists of $O(B^2)$ points.

If the query point p belongs to a long rectangle R, then the x-bounds of R are stored in $D(\pi(p))$. Let $s = \lceil p_x/B \rceil$ where p_x is the x-coordinate of p. The entry $D(\pi(p))[s]$ contains pointers to at most two rectangles. We can check whether one of those rectangles contains p in constant time. If p belongs to a short rectangle R', then R' is stored in D_s. According to the result of Lemma 1, we can check whether p belongs to a rectangle stored in D_s in $O(\log_B B^2) = O(1)$ I/Os.

3 Point Location in a Set of Rectangles

In this section we describe data structures for point location in a set of r-dimensional rectangles:

Theorem 1. *There exists a $O(N/B)$ space data structure that answers point location queries for a set of N d-dimensional rectangles on an (r, d) grid, $r \geq 2$, in $O(\log_B^r N)$ I/Os.*

Clearly, if $d = r$ we obtain the data structure for point location in a set of r-dimensional rectangles.

Proof. Using the method of [9], a query on an (r, d) grid can be transformed into a query on (N, r, d) grid. For an element x and a set S, the predecessor of x in S is defined as $pred(x, S) = \max\{y \in S | y \leq x\}$. The rank of x in S is defined as $rank(x, S) = |\{y | y \leq x\}|$. If $a \in S$ and $b \in S$, then $a \leq x \leq b \Leftrightarrow rank(a, S) \leq rank(x, S) \leq rank(b, S)$. Let P_i be the set that contains the upper and lower bounds of projections of all rectangles in $\mathcal{R}$ on the i-th coordinate. The reduction of a query on an (r, d) grid to a query on an (N, r, d) grid works as follows. Every rectangle $R = [a_1, b_1] \times \ldots \times [a_r, b_r] \times \ldots \times [a_d, b_d]$ is replaced by $R' = [a'_1, b'_1] \times \ldots \times [a'_r, b'_r] \times \ldots \times [a_d, b_d]$, where $a'_i = rank(a_i, P_i)$ and $b'_i = rank(b_i, P_i)$ for $i = 1, \ldots, r$. A point $p = (p_1, p_2, \ldots p_r, p_{r+1} \ldots, p_d)$ belongs to a rectangle R if and only if $p' = (rank(p_1, P_1), \ldots, rank(p_r, P_r), p_{r+1} \ldots, p_d)$ belongs to the corresponding rectangle R'. The point p' can be computed from p in $O(r \log_B N)$ I/Os using the standard B-trees. Below we assume that the values of the first r coordinates of all points belong to the interval $[1, N]$ and construct the data structure for point location queries on (N, r, d) grid.

Our data structure uses the approach similar to the k-dimensional skewer trees of [12] modified for the external memory. The data structure for $r = 1$ was described in Lemma 2. If there exists a data structure that supports point location queries on the $(N, r-1, d)$ grid in $O(\log_B^{r-1} N)$ I/Os, then the data structure for the (N, r, d) grid can be constructed as follows. We construct a trie $\mathcal{T}_x$ on the set of possible x-coordinates. All nodes of $\mathcal{T}_x$ have degree $B^{1/d}$.We associate B consecutive values from the interval $[0, N-1]$ with each leaf of $\mathcal{T}_x$. The range of an internal node v is an interval $rng(v) = [v_{min}, v_{max}]$, where v_{min} and v_{max} are minimal and maximal values associated with leaf descendants of v. The x-interval of a rectangle R is the projection of R on the x-coordinate. A node v covers rectangle R with x-interval $[a, b]$, if $[a, b] \subset rng(v)$, but for any child v_i of v, $[a, b] \not\subset rng(v_i)$, where $[a, b]$ is the x-interval of R. We denote by $\pi(R)$ ($\pi(p)$) the projection of the rectangle R (point p) on the last $d-1$ coordinates (i.e., all coordinates except of the x-coordinate).

In every internal node v of $\mathcal{T}_x$, there is a data structure D_v: for each rectangle R covered by v, such that the ranges of children $v_i, v_{i+1} \ldots, v_j$ of v belong to the x-interval of R, but $rng(v_{i-1}) \not\subset [a, b]$ and $rng(v_{j+1}) \not\subset [a, b]$, D_v contains the rectangle $R' = [i, j] \times \pi(R)$, i.e, the x-interval of R is replaced by $[i, j]$ in R'. A query to a data structure D_v can be transformed into a query on the $(N, r-1, d)$ grid by changing the order of coordinates. Hence, D_v supports point location queries in $O(\log_B^{r-1} N)$ I/Os.

The search for the rectangle that contains the query points p visits all nodes on the path Π_x from the root of $\mathcal{T}_x$ to the leaf that contains the x-coordinate p_x of p. Suppose that a node $v \in \Pi_x$ is visited and the child v_i of v also belongs to Π_x. We use data structure D_v to answer three queries $(i, \pi(p))$, $(i-1, \pi(p))$, and $(i+1, \pi(p))$. Each of those queries can be answered in $O(\log_B^{r-1} N)$ I/Os. For every rectangle R' that contains $(i, \pi(p))$, $(i-1, \pi(p))$, or $(i+1, \pi(p))$, we check whether the corresponding rectangle R contains p. If some rectangle R contains p, the search is completed, otherwise the search continues in the child v_i of v. If the search achieves a leaf l of T_x, all B rectangles stored in l can be examined in $O(1)$ I/Os. The total number of queries on the $(r-1,d)$ grid answered during the search is $O(\log_B N)$; hence, a query is answered in $O(\log_B^r N)$ I/Os. Suppose that a rectangle R stored in a node v contains the point p and the child v_i of v belongs to Π_x. Then, the x-interval of R contains $rng(v_j)$ for at least one child v_j of v. Since R contains p, the x-interval of R contains either $rng(v_{i-1})$, or $rng(v_i)$, or $rng(v_{i+1})$. Hence, one of the three queries asked at the node v would return the rectangle R' that corresponds to R. This proves the correctness of the search procedure.

By a combination of the approach of Theorem 1 with the fractional cascading technique [10], we can obtain a two-dimensional data structure with $O(\log_B N)$ query time.

Lemma 3. *There exists a $O(N/B)$ space data structure that answers point location queries for a set of N rectangles on a $(2,d)$ grid in $O(\log_B N)$ I/O operations.*

Proof. The tree $\mathcal{T}_x$ and data structures D_v in the nodes of $\mathcal{T}_x$ are defined in the same way as in the proof of Theorem 1. Each D_v supports queries on $(N,1,d)$ grid. The search algorithm is also the same as in Theorem 1: To find the rectangle R that contains the query point p, we visit all nodes on the path Π_x from the root of $\mathcal{T}_x$ to the leaf that contains the x-coordinate p_x of p. In each visited node v queries $(i,\pi(p))$, $(i-1,\pi(p))$, and $(i+1,\pi(p))$, where i is the index of the child of v that belongs to Π_x, are answered by the data structure D_v. Using a variant of the fractional cascading technique, we can support queries in a data structure D_v for all $v \in \Pi_x$ except of the root in $O(1)$ I/Os.

Suppose that data structure D_v contains L rectangles. Let Y_v denote the set of y-bounds of all rectangles stored in D_v. For a rectangle R stored in D_v, we denote by R' the rectangle obtained from R by replacing the y-bounds of R with their ranks in Y_v. Data structure D'_v contains all such rectangles R'. That is, in a data structure D'_v the y-bounds of rectangles are replaced by their ranks in Y_v. All rectangles in D'_v belong to the $(L,1,d)$ grid. Hence, by Lemma 2 D'_v uses $O(L/B)$ blocks and supports point location queries in $O(1)$ I/Os. The point $p(v)$ is obtained from p by replacing the y-coordinate p_y of p with $rank(p_y, Y_v)$. A point p belongs to a rectangle R stored in D_v if and only if $p(v)$ belongs to the corresponding rectangle R' stored in D'_v. Therefore, if $rank(p_y, Y_v)$ is known, then the point location query for D_v can be answered in constant time. For every element $y \in Y_v$ we store $rank(y, Y_v)$. Since $rank(e, Y_v) = rank(pred(e, Y_v), Y_v)$, it suffices to find predecessors $pred(p_y, Y_v)$ for all nodes $v \in \Pi_x$. Using the fractional cascading technique [10] [6], predecessors of the same p_y in all sets Y_v can be found in $O(\log_B N)$ I/Os. This completes the proof of the Lemma.

Theorem 2. *There exists a $O(N/B)$ space data structure that answers point location queries for a set of N d-dimensional rectangles on an (r,d) grid, $r \geq 2$, in $O(\log_B^{r-1} N)$ I/Os.*

Proof. As shown in the proof of Theorem 1, given a data structure that answers point location queries on (N, r, d) grid in $q(N)$ I/Os, we can construct a data structure that answers point location queries on $(N, r+1, d)$ grid in $O(q(N) \log_B N)$ I/Os. Applying this construction $r-2$ times to the result of Lemma 3, we obtain a $O(N/B)$ space data structure that supports queries in $O(\log_B^{r-1} N)$ I/Os.

4 Point Location in Two- and Three-dimensional Rectangular Subdivisions

In this section we present a data structure for point location in two-dimensional and three-dimensional rectangular subdivisions. We will need the following Proposition [19]:

Proposition 1. *Given a set $S \subset [1, U]$, there exists a $O(N/B)$ space data structure for S that supports predecessor and successor queries in $O(\log_2 \log_B U)$ I/Os.*

The proof of Proposition 1 can be found in [19].

Theorem 3. *There exists a $O((N/B) \log_B N)$ space data structure that supports point location queries in a two-dimensional rectangular sub-division of a $U \times U$ grid in $O(\log_2 \log_B U + (\log_2 \log_B N)^2)$ I/Os.*

Proof. As follows from the description in section 3 and Proposition 1, a query on a $U \times U$ grid can be turned into a query on an $N \times N$ grid, so that the number of I/O operations increases by an additive factor $O(\log_2 \log_B U)$. Hence, it suffices to show that point location queries on $N \times N$ grid can be answered in $O((\log_2 \log_B N)^2)$ time. Our data structure adapts the ideas of [9] for the external memory model. We will also use the data structure for point location on the $(N, 1, d)$ grid from section 2.

We construct a trie $\mathcal{T}_x$ on the set of possible x-coordinates. All internal nodes of $\mathcal{T}_x$ have degree B^c for some constant $c < 1/3$. We associate B consecutive values from the interval $[0, N-1]$ with each leaf of $\mathcal{T}_x$. We say that a rectangle R with x-interval $[a, b]$ x-cuts the node v, such that $rng(v) = [x_1, x_2]$, if $[a, b]$ contains either x_1 or x_2 but $[x_1, x_2] \not\subset [a, b]$. Rectangle R x-cuts v from the left (from the right), if $[a, b]$ contains x_1 but $[x_1, x_2] \not\subset [a, b]$ ($[a, b]$ contains x_2 but $[x_1, x_2] \not\subset [a, b]$).

A d-parent of a node v is a node w, such that w is an ancestor of v, and the path from v to w consists of d edges. In every node v of $\mathcal{T}_x$ there are data structures X_v^l and X_v^r that contain information about all rectangles R that x-cut the node v from the left and from the right respectively. The projections of all rectangles R stored in X_v^l (X_v^r) on the y-axis induce a one-dimensional subdivision Π_v^l (Π_v^r) of size $O(M)$, where M is the number of rectangles that x-cut v from the left (from the right). X_v^l (X_v^r) supports one-dimensional point location queries for Π_v^l (Π_v^r). By Proposition 1, X_v^l and X_v^r use $O(M/B)$ blocks of space and answer queries in $O(\log_2 \log_B N)$ time. In every node v there is also a data structure D_v similar to the data structure of Theorem 1: if the x-interval of some rectangle R is covered by v but is not covered by any child of v, then D_v contains the rectangle $R' = [i, j] \times \pi(R)$, where i and j are chosen so that the x-interval $[a, b]$ of R contains $rng(v_i), rng(v_{i+1}), \ldots, rng(v_j)$, but $rng(v_{i-1}) \not\subset [a, b]$ and $rng(v_{j+1}) \not\subset [a, b]$.

The search procedure consists of two stages. At the end of the first stage we either find the rectangle R that contains the query point p or identify the node v such that p is contained in some rectangle R that is stored in D_v.

The first stage consists of a number (at most $\log_2 \log_B N$) iterations, and during the first stage we examine nodes on the path π_x from the root r_x of $\mathcal{T}_x$ to the leaf l_x that contains the x-coordinate of the query point p. It would cost too much time to examine all nodes on the path π_x, therefore we perform a binary search on the nodes of π_x. During the i-th iteration of the first stage we inspect the part of π_x between nodes u_i and l_i. At the beginning of the first stage, u_1 is initialized to r_x, l_1 is initialized to l_x, and d_1 is set to $\log_B N/2$. At the i-th iteration the d_i-parent m_i of l_i is examined, i.e we examine the node m_i that lies "in the middle" of the path from l_i to u_i. The result of this examination is one of the following: 1.one of the rectangles that x-cuts the examined node contains p and the search is completed. 2. rectangle that contains p x-cuts some ancestor of the examined node m_i (resp. is covered by an ancestor of m_i). In this case, if $d_i > 1$, we set $l_{i+1} = m_i$, $d_{i+1} = d_i/2$, and proceed to the $(i + 1)$-st iteration. If $d_i = 1$, the first stage of the search is completed and the rectangle R that contains p is covered by u_i but is not covered by a child of u_i. Hence, the search continues in the data structure D_{u_i} 3. rectangle that contains p x-cuts some descendant of the examined node m_i (resp. is covered by a descendant of m_i). In this case, we set $u_{i+1} = m_i$, $d_{i+1} = d_i/2$, and proceed to the $(i + 1)$-st iteration. If $d_i = 1$, the first stage of the search is completed and the rectangle R that contains p is covered by m_i but is not covered by a child of m_i. Hence, the search continues in the data structure D_{m_i}. Clearly, the total number of iterations is $O(\log_2 \log_B N)$.

Now we will show how each iteration can be implemented in $O(\log_2 \log_B N)$ I/Os by answering two one-dimensional point location queries. In every visited node v a query y_p is asked to the data structures X_v^l and X_v^r. Suppose that y_p belongs to one-dimensional rectangles $P^l \in \Pi_v^l$ and $P_v^r \in \Pi_v^r$; P_v^l (P_v^r) may be the projection of some two-dimensional rectangle R_v^l (R_v^r) that x-cuts v from the left (from the right). Observe that if y_p belongs to a projection of some rectangle that x-cuts v from the left, then y_p also belongs to a projection of some rectangle that x-cuts v from the right. When the queries to X_v^l and X_v^r are answered, the search proceeds as follows. 1. If p is contained in R_v^l or R_v^r, then the query is answered. 2. If both P_v^l nor P_v^r are projections of rectangles R_v^l and R_v^r that x-cut v, but neither R_v^l nor R_v^r contain p, i.e. if the x-intervals of R_v^r and R_v^l do not contain x_p, then p is belongs to a rectangle that x-cuts some descendant u of v. 3. If neither P_v^l nor P_v^r are projections of rectangles that x-cut v, then p is contained in a rectangle R, such that $rng(v)$ is contained in x-interval of R and

R x-cuts some ancestor w of v. If the rectangle that contains p is not found, we proceed with the next iteration as described in the previous paragraph.

At the end of the first stage, we either find the rectangle R that contains p, or identify the node v, such that R is covered by v but R is not covered by a child of v. In the latter case, we identify the rectangle R' that contains the point $(i, \pi(p))$ in $O(\log_2 \log_B N)$ I/Os using Proposition 1 and Lemma 2. Clearly, a rectangle R' contains a point p' if and only if the corresponding rectangle R contains the point p. Thus the second stage is completed in $O(\log_2 \log_B N)$.time.

Theorem 4. *There exists a $O((N/B) \log_B^2 N)$ space data structure that supports point location queries in a three-dimensional rectangular sub-division in $O(\log_B N)$ time.*

Proof. The proof is quite similar to the proof of Theorem 3. We construct a trie $\mathcal{T}_x$ on the set of possible x-coordinates. Information about all rectangles that x-cut a node v from the left (from the right) is stored in the data structure X_v^l (X_v^r). X_v^l (X_v^r) contains projections of rectangles on the (y, z)-plane; hence, by Theorem 3 two-dimensional point location queries in X_v^l and X_v^r are supported in $O((\log_2 \log_B N)^2)$ time. Data structures D_v are defined in the same way as in Theorem 3. By Lemma 3, D_v support point location queries in $O(\log_B N)$ time.

The first stage of the search for the rectangle that contains the point p is the same as in the proof of Theorem 3. At the end of the first stage we either find the rectangle R containing p or identify the node v, such that R is covered by v but R is not covered by a child of v. Using the data structure D_v, we can find the rectangle R' that corresponds to R in $O(\log_B N)$ time.

4.1 Three-Dimensional Orthogonal Range Reporting

A data structure that three-dimensional dominance range reporting queries with sub-logarithmic number of I/Os can be constructed by combining the data structure of Vengroff and Vitter [23] with the result of Theorem 3.

Theorem 5. *There exists a $O((N/B) \log_2^4 N)$ space data structure that supports three-dimensional orthogonal range reporting queries on a $N \times N \times N$ grid in $O((\log_2 \log_B N)^2 + T/B)$ time, where T is the number of points in the answer.*

A sketch of the proof of Theorem 5 is given in the Appendix. Finally, we can apply the reduction to rank space technique [15] and Proposition 1 and obtain

Corollary 1. *There exists a $O((N/B) \log_2^4 N)$ space data structure that supports three-dimensional orthogonal range reporting queries on a $U \times U \times U$ grid in $O(\log_2 \log_B U + (\log_2 \log_B N)^2 + T/B)$ time, where T is the number of points in the answer.*

5 Conclusion

In this paper we present a data structure that supports d-dimensional point location queries in $O(\log_B^{d-1} N)$ I/Os and uses $O(N/B)$ blocks of space. We also show that there are external memory data structures that answer point location queries in a two-dimensional rectangular subdivision and three-dimensional orthogonal range reporting queries in $o(\log_B n)$ I/Os provided that all point coordinates are bounded by an appropriate parameter U.

Existence of data structures that support rectangular point location queries in d dimensions for $d \geq 3$ in the cache-oblivious model [14] remains an interesting and important open question. The cache-oblivious data structure of Bender, Cole, and Raman [8] supports general planar point location queries in $O(\log_B N)$ I/Os, but even for the case of rectangular subdivisions no d-dimensional data structures for $d \geq 3$ are known. Observe that the approach used to answer point location queries in this paper cannot be extended to the cache-oblivious model: data structures described in this paper are based on B-trees and rely on the fact that the block size B is known.

References

1. Agarwal, P.K., Arge, L., Brodal, G.S., Vitter, J.S.: I/O-Efficient Dynamic Point Location in Monotone Planar Subdivisions. In: Proc. SODA, pp. 11–20 (1999)
2. Agarwal, P.K., Arge, L., Yi, K.: An Optimal Dynamic Interval Stabbing-Max Data Structure? In: Proc. SODA, pp. 803–812 (2005)
3. Aggarwal, A., Vitter, J.S.: The Input/Output Complexity of Sorting and Related Problems. Communications of the ACM 31(9), 1116–1127 (1988)
4. Arge, L.: External Memory Data Structures. In: Meyer auf der Heide, F. (ed.) ESA 2001. LNCS, vol. 2161, pp. 1–29. Springer, Heidelberg (2001)
5. Arge, L., Procopiuc, O., Ramaswamy, S., Suel, T., Vitter, J.S.: Theory and Practice of I/O-Efficient Algorithms for Multidimensional Batched Searching Problems (Extended Abstract). In: Proc. SODA, pp. 685–694 (1998)
6. Arge, L., Vengroff, D.E., Vitter, J.S.: External-Memory Algorithms for Processing Line Segments in Geographic Information Systems. Algorithmica 47(1), 1–25 (2007)
7. Arge, L., Vahrenhold, J.: I/O-Efficient Dynamic Planar Point Location. Computational Geometry 29(2), 147–162 (2004)
8. Bender, M.A., Cole, R., Raman, R.: Exponential Structures for Efficient Cache-Oblivious Algorithms. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 195–207. Springer, Heidelberg (2002)
9. de Berg, M., van Kreveld, M.J., Snoeyink, J.: Two- and Three-Dimensional Point Location in Rectangular Subdivisions. J. Algorithms 18(2), 256–277 (1995)
10. Chazelle, B., Guibas, L.J.: Fractional Cascading: I. A Data Structuring Technique. Algorithmica 1, 133–162 (1986)
11. Crauser, A., Ferragina, P., Mehlhorn, K., Meyer, U., Ramos, E.A.: Randomized External-Memory Algorithms for Line Segment Intersection and Other Geometric Problems. Int. J. Comput. Geometry Appl. 11(3), 305–337 (2001)
12. Edelsbrunner, H., Haring, G., Hilbert, D.: Rectangular point location in d dimensions with applications. Computer J 29, 76–82 (1986)
13. van Emde Boas, P., Kaas, R., Zijlstra, E.: Design and Implementation of an Efficient Priority Queue. Mathematical Systems Theory 10, 99–127 (1977)

14. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-Oblivious Algorithms. In: Proc. FOCS 1999, pp. 285–298 (1999)
15. Gabow, H., Bentley, J.L., Tarjan, R.E.: Scaling and related techniques for geometry problems. In: Proc. STOC 1984, pp. 135–143 (1984)
16. Goodrich, M.T., Tsay, J.-J., Vengroff, D.E., Vitter, J.S.: External-Memory Computational Geometry (Preliminary Version). In: Proc. FOCS 1993, pp. 714–723 (1993)
17. Nekrich, Y.: A Data Structure for Multi-Dimensional Range Reporting. In: Proc. SoCG 2007, pp. 344–353 (2007)
18. Nekrich, Y.: External Memory Range Reporting on a Grid. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, Springer, Heidelberg (2007)
19. Pătraşcu, M., Thorup, M.: Time-space Trade-offs for Predecessor Search. In: Proc. STOC 2006, pp. 232–240 (2006)
20. Smid, M.: Dynamic rectangular point location, with an application to the closest pair problem. Information and Computation 116, 1–9 (1995)
21. Snoeyink, J.: Point Location. In: Handbook of Discrete and Computational Geometry, CRC Press, Inc., Boca Raton (1997)
22. Subramanian, S., Ramaswamy, S.: The P-range Tree: A New Data Structure for Range Searching in Secondary Memory. In: Proc. SODA 1995, pp. 378–387 (1995)
23. Vengroff, D.E., Vitter, J.S.: Efficient 3-D Range Searching in External Memory. In: Proc. STOC 1996, pp. 192–201 (1996)
24. Vitter, J.S.: External Memory Algorithms and Data Structures: Dealing with Massive Data. ACM Computing Surveys 33(2), 209–271 (2001)

Appendix. Proof of Theorem 5

In this Appendix we give a sketch of the data structure for three-dimensional range reporting on a grid of size N.

The result of [23] is based on a very interesting concept of a *t-approximate boundary*: A t-approximate boundary M for the set of points P partitions the set of points P into P^+ and P^-, so that every point of M is dominated by at least t and at most $3t$ points of P. Let I be the minimal layer of maxima of M, i.e. every point of M dominates at least one point of I, and every point of I does not dominate any other point of M. Thus every point p is either dominated by some point of M or dominates a point of I. The problem of finding for an arbitrary point q a point $p \in I$ that is dominated by q or reporting that no such $p \in I$ exists can be reduced to the two-dimensional point location problem in the rectangular subdivision of the (x, y)-plane; see [23] and [17] for details. Recall that using the data structure of Theorem 3 the point location query for a rectangular subdivision of $N \times N$ grid can be answered in $O((\log_2 \log_B N)^2)$ I/Os using a $O((N/B))$ space data structure. Using a variant of the method described in [17], it is possible to report in $O(t/B)$ I/Os all points that dominate an arbitrary point $q \in I$ with help of a $O((N/B))$ space data structure.

The data structure for three-dimensional dominance reporting consists of approximate boundaries $M_1, M_2, \ldots, M_{\log_2 N/2}$, where M_j is a 2^{2j}-approximate boundary. Let I_j be the minimal layer of maxima of M_j. Let $t = \log_2 \log_2 \log_B N + \log_2 B/2$. If q dominates some point c_t that belongs to the minimal set I_t of

M_t, then q is dominated by $O(B(\log_2 \log_B N)^2)$ points. All points that dominate c_t can be examined in $O((\log_2 \log_B N)^2)$ I/Os and all points that dominate q can be found in $O((\log_2 \log_B N)^2)$ time. If q dominates no point of M_t, we test $M_{t+1}, M_{t+2}, \ldots M_{t+j}$ until we find the 2^{2t+2j}-approximate boundary M_{t+j}, such that at least one point c_{t+j} of M_{t+j} is dominated by q. The number of I/O operations necessary to check $M_{t+1}, M_{t+2}, \ldots, M_{t+j}$ is $O(j(\log_2 \log_B N)^2)$, but the number of points that dominate q is $\Omega(2^{2j+2t}) = \Omega(2^{2j} B \log_2 \log_B N)$. Hence, all points that dominate q can be reported in $O(j(\log_2 \log_B N)^2 + T/B) = O(T/B)$ time. Therefore dominance reporting queries can be answered in $O((\log_2 \log_B N)^2 + T/B)$ time.

An $s(N)$ space data structure for dominance reporting queries with $q(N)$ I/Os per query can be transformed into a $O(s(N) \log_2^3 N)$ data structure for three-dimensional orthogonal range reporting queries with $O(q(N))$ I/Os per query; see e.g. [22], [23] or [17] for details.

Finding Heavy Hitters over the Sliding Window of a Weighted Data Stream

Regant Y.S. Hung and H.F. Ting[*]

Department of Computer Science,
The University of Hong Kong, Pokfulam, Hong Kong
{yshung,hfting}@cs.hku.hk

Abstract. We study the problem of identifying items with heavy weights in the sliding window of a weighted data stream. We give a deterministic algorithm that solves the problem within error bound ϵ, uses $O(\frac{R}{\epsilon})$ space and supports $O(\frac{R}{\epsilon})$ query and update times. Here, R is the maximum item weight. We also show that the space can be reduced substantially in practice by showing for any $c > 0$, we can construct an $O(\frac{c \log R}{\epsilon^2})$-space algorithm, which returns correct answers provided that the ratio between the total weights of any two adjacent sliding windows is not greater than c. We also give a randomized algorithm that solves the problem with success probability $1 - \delta$ using $O(\frac{1}{\epsilon^2} \log R \log D \log \frac{\log D}{\delta\epsilon})$ space where D is the number of distinct items in the data stream.

1 Introduction

This paper studies the problem of identifying *heavy hitters* over the sliding window of a weighted data stream. There are four parameters in our study, namely, R the maximum item weight, N the sliding window size, ϵ the error bound and θ the threshold. The input of the problem is a weighted data stream σ, which is a sequence of tuples $(a_1, w_1), (a_2, w_2), \ldots, (a_i, w_i), \ldots$ where a_i is an item name and the integer $w_i \in [1, R]$ is the weight of the tuple. The sliding window covers the N most recently arrived tuples of σ, i.e., $\{(a_{p-N+1}, w_{p-N+1}), (a_{p-N+2}, w_{p-N+2}), \ldots, (a_p, w_p)\}$ where (a_p, w_p) is the most recently arrived tuple. For any item a, we define the *window frequency*, or simply frequency of a to be

$$f_a = \sum \{w_k \mid (a_k, w_k) \text{ is among the } N \text{ most recently arrived items and } a_k = a\},$$

the total weight of the tuples with name a that are covered by the sliding window. We are interested in designing data structures that allow us to answer at any time the following query efficiently:

> Let S be the total weight of all items in the sliding window. Return a set Π of items such that (i) every item in Π has frequency no less than $(\theta - \epsilon)S$ and (ii) all items with frequencies no less than θS must be in Π.

[*] This research was supported in part by Hong Kong RGC Grant HKU-7163/07E.

We call Π a set of heavy hitters. As common to all data stream applications, any solution must be one-pass, use $o(N)$ space, and support $o(N)$ update and query times. Heavy hitters identification problem has been studied extensively for the special case of unweighted data streams, in which all weights are equal to 1 and the frequency of an item is just the number of occurrences of the item in the sliding window. However, we observe that the weighted data stream model is more suitable in practical applications such as the monitoring of network traffic and the usage-based charging scheme in a telecommunication system. Note that it is much more difficult to identify heavy hitters in a weighted data stream. In an unweighted data stream, every item has weight 1 and for an item a to change from a non-heavy hitter to an heavy one, there must be many new arrivals of a. Therefore, a few item arrivals will not change the set of heavy hitters much and a small data structure can keep enough information about the distribution of items in the sliding window for maintaining the heavy-hitters set. However, for a weighted data stream a few arrivals of items may change drastically the set of heavy hitters because these items may have very heavy weight. It is not obvious how a small data structure can still maintain the heavy-hitters set correctly.

Previous work. There are many interesting algorithms for finding heavy hitters for the whole data stream (not over the sliding window). Manku and Motwani [12] gave two algorithms, namely, the sticky counting and lossy counting algorithms for finding heavy hitters in an unweighted data stream. The sticky counting algorithm finds the heavy hitters using $O(\frac{1}{\epsilon}\log\frac{1}{\theta\delta})$ space with success probability $1-\delta$ while lossy counting algorithm finds the heavy hitters using $O(\frac{1}{\epsilon}\log\epsilon N)$ space deterministically. Misra and Gries [13], Demaine *et al.* [4] and Karp *et al.* [8] independently solved the problem using $O(\frac{1}{\epsilon})$-space. Demaine *et al.* [4] also proposed randomized algorithms with substantially smaller space for a restricted model: the items may have different frequency in the stream but every item must be uniformly distributed throughout the whole stream. For weighted data streams, Cormode and Muthukrishnan [2] gave a data structure called CM sketch for estimating the frequency of an item in the whole data stream. Using CM sketch, we can find the set of heavy hitters using $O(\frac{1}{\epsilon}\log D\log\frac{\log D}{\delta\theta})$ space where D is the number of distinct items in the data stream. Ganguly and Majumder [5] proposed a deterministic algorithm for finding heavy hitters using $O(\frac{1}{\epsilon^2}\log D\log\epsilon D)$ space.

There are streaming algorithms for finding quantiles [11,14] and counting bits [3,9,6] over sliding windows. For example, Datar *et al.* [3] studied the following problem: Given a stream of bits, estimate the number of 1-bits in the sliding window with relative error at most ϵ. They solved the problem using $O(\frac{\log\epsilon N}{\epsilon})$ space. Gibbons and Tirthapura [6] proposed another algorithm for this problem that achieves the same space complexity, but with faster update time. Datar *et al.* [3] also considered the problem of finding the sum of the last N items of a stream of integers within the range $[0,R]$; they showed that the problem can be solved in $O(\frac{\log RN}{\epsilon})$ space. Gibbons *et al.* [6] solved the same problem with the same space complexity, but with faster update time. For finding heavy hitters over sliding windows of an unweighted data stream, Golab *et al.* [7] proposed an algorithm that solves the problem with some assumption on the items distribution. Arasu and Manku [1] gave an $O(\frac{1}{\epsilon}\log^2\frac{1}{\epsilon})$-space algorithm for solving the problem without any assumption. Lee and Ting [10] gave an improved algorithm that reduces the space to $O(\frac{1}{\epsilon})$. To the best of our knowledge, there is no algorithm designed for finding heavy hitters in the sliding window of a weighted data stream.

Our Results. We propose a deterministic algorithm for finding the set of heavy hitters in the sliding window of a weighted data stream; our algorithm uses $O(\frac{R}{\epsilon})$ space and supports $O(\frac{R}{\epsilon})$ update and query times. Then, we show that the space complexity can be reduced significantly provided that the total weight of the items in the sliding window will not change drastically. More precisely, we show that for any value $c > 0$, we can construct an $O(\frac{c\log R}{\epsilon^2})$-space algorithm which is correct provided that c is an upper bound on the ratio between the total item weights of any two disjoint adjacent sliding windows. Note that $c \leq R$ because the minimum and maximum total weight of items in the sliding window are respectively N and RN. Furthermore, it can be verified that when $c = o(\frac{\epsilon R}{\log R})$, the new algorithm uses $o(\frac{R}{\epsilon})$ space. We argue that it is reasonable to assume that $c = o(\frac{\epsilon R}{\log R})$. To violate the assumption, most items must have very small weights in one window and have weights near to the maximum in the adjacent window. Such dramatic change in the weights signals abnormal situations and human intervention is usually needed. For instance, in network applications, a sudden increase of network traffic may be due to some attacks or improper use of the networks by some users.

Our $O(\frac{c\log R}{\epsilon^2})$-space algorithm is based on a data structure for solving the Integer-Sum problem, which is defined as follows: given a stream of integers in $[1, R]$, estimate the sum of the last N integers with relative error at most ϵ. We give a data structure solving this problem using $O(\frac{\log R}{\epsilon})$ space (or more precisely, $O((\frac{\log R}{\epsilon})\log RN)$ bits), $O(\frac{\log R}{\epsilon})$ update time and constant query time. We also give a lower bound of $\Omega(\frac{\log^2 R}{\epsilon})$ bits on the space complexity for solving this problem when $R \leq \epsilon N$. Note that the algorithms of Datar *et al.* [3] and Gibbons *et al.* [6] can also solve the problem; the algorithm of Datar *et al.* solves the problem using $O(\frac{1}{\epsilon}\log RN(\log N + \log\log RN))$ bits, $O(\log RN)$ update time and constant query time while the algorithm of Gibbons *et al.* solves the problem using $O(\frac{1}{\epsilon}\log^2 RN)$ bits and constant update and query times. The space complexity of their algorithms depend heavily on N, the sliding window size, which can be in magnitude of hundreds of millions [3]. We aim at finding algorithm whose complexity depends more on R, the maximum weight of an item. Note that our algorithm uses much less space when $\log R = o(\log N)$. For example, when $R = \log N$, our algorithm uses $O(\frac{\log N \log\log N}{\epsilon})$ bits while both algorithms proposed by Datar *et al.* and Gibbons *et al.* use $O(\frac{\log^2 N}{\epsilon})$ bits.

We also study how randomization helps in finding the heavy hitters. We show that by using our data structure for the Integer-Sum problem together with an adaptation of the CM sketch [2] (which was originally designed for estimating the frequency of an item in the whole data stream, not over the sliding window), we can find with success probability $1-\delta$ the set of heavy hitters in the sliding window of a weighted data stream using $O(\frac{1}{\epsilon^2}\log R \log D \log\frac{\log D}{\delta\theta})$ space where D is the number of distinct items appearing in the data stream.

2 A Useful Lemma

The lemma below simplifies our analysis in the whole paper by reducing the problem for finding heavy hitters to the problem for estimating the frequency of every item.

Lemma 1. *Let $G(\epsilon, R)$ be a function of ϵ and R. If an algorithm A that uses $O(G(\epsilon, R))$ space can (1) give an estimate $\hat{S}$ of the total weight S of all the items in any sliding window with guarantee $S \leq \hat{S} \leq (1+\epsilon)S$, and (2) give an estimate $\hat{f}_a$ of the frequency f_a of any item a with guarantee $f_a - \epsilon S \leq \hat{f}_a \leq f_a + \epsilon S$, then A can solve the heavy hitters identification problem over the sliding window of a weighted data stream using $O(G(\frac{\epsilon}{3}, R))$ space.*

Proof. Based on the guarantees of A as stated in the lemma, A can be able to estimate the frequency of any item a with guarantee

$$f_a - \tfrac{1}{3}\epsilon S \leq \hat{f}_a \leq f_a + \tfrac{1}{3}\epsilon S, \tag{1}$$

and estimate the total weight of all items in the sliding window with guarantee

$$S \leq \hat{S} \leq (1+\tfrac{\epsilon}{3})S \tag{2}$$

using $O(G(\epsilon/3, R))$ space. When there is a query, A will return all the items whose estimated frequency no less than $(\theta - \frac{2\epsilon}{3})\hat{S}$. For any item a returned by A, we have $\hat{f}_a \geq (\theta - \frac{2\epsilon}{3})\hat{S}$. Together with the facts that (1) $f_a \geq \hat{f}_a - \frac{1}{3}\epsilon S$ by Inequality 1, and (2) $\hat{S} \geq S$ by Inequality 2, we have $f_a \geq \hat{f}_a - \frac{1}{3}\epsilon S \geq (\theta - \frac{2\epsilon}{3})\hat{S} - \frac{1}{3}\epsilon S \geq (\theta - \frac{2\epsilon}{3})S - \frac{1}{3}\epsilon S = (\theta - \epsilon)S$. Thus, we only return items with frequency no less than $(\theta - \epsilon)S$.

Consider any item e whose frequency is no less than θS. A guarantees that $\hat{f}_e \geq f_e - \frac{1}{3}\epsilon S$ and $S \geq \hat{S}/(1+\frac{\epsilon}{3})$. Together with the fact that $\hat{S}/(1+\frac{\epsilon}{3}) \geq (1-\frac{\epsilon}{3})\hat{S}$, we have $\hat{f}_e \geq f_e - \frac{1}{3}\epsilon S \geq \theta S - \frac{1}{3}\epsilon S = (\theta - \frac{1}{3}\epsilon)S \geq \frac{\theta - \frac{\epsilon}{3}}{1+\frac{\epsilon}{3}}\hat{S} \geq (\theta - \frac{\epsilon}{3})(1 - \frac{\epsilon}{3})\hat{S} \geq (\theta - \frac{2\epsilon}{3})\hat{S}$. Thus, the estimated frequency of e is no less than $(\theta - \frac{2\epsilon}{3})\hat{S}$. Since A will return any item whose estimated frequency is no less than $(\theta - \frac{2\epsilon}{3})\hat{S}$, A will return any item whose frequency is no less than θS. Thus, A solves the problem using $O(G(\epsilon/3, R))$ space. □

By Lemma 1, an algorithm can solve heavy hitters identification problem if it has the two guarantees as stated in the lemma. In the rest of our paper, we will only prove that our algorithms are able to have the two guarantees all the time.

3 A Deterministic Algorithm for Finding Heavy Hitters

The core of our algorithm is a data structure called window counters; we keep at most $\frac{3R}{\epsilon} + 1$ window counters during the execution of our algorithm. Each window counter estimates the frequency of an item in the sliding window based on the sampling technique proposed in [10]. To implement a window counter C, we maintain a deque Q of entries and a variable ℓ. Each entry in the deque Q contains two variables: *pos* and *value*. Let $(pos_1, value_1)$ and $(pos_2, value_2)$ denote the two consecutive entries in Q where $(pos_1, value_1)$ is created before $(pos_2, value_2)$ is created. Then, the entry $(pos_2, value_2)$ samples the total weight of the items in W_{pos_1+1,pos_2}, where $W_{p,q}$ denotes the sliding window covering the items from p-th item to q-th item in the data stream for some $p \leq q$. More precisely, $value_2$ equals the total weight of the items in W_{pos_1+1,pos_2}.

Initially, Q is empty and ℓ equals zero. The value $v(C)$ of C is defined as the sum of the values of all the entries in Q plus ℓ in the window counter. On the other hand, we have to maintain an additional window counter C_{total} that estimates the total weight in the sliding window. Whenever an item (a_i, w_i) arrives in the stream, we do the following steps:

Update every window counter: First, each window counter removes the expired entry, i.e. the entry that samples only the counts outside the sliding window. For the counter C_{a_i} that estimates the count of a_i (if this counter does not exist, we will create one for it), increment its count by w_i. This can be done by adding w_i to the variable ℓ of C_{a_i}. If the sum of w_i and ℓ exceeds $\frac{\epsilon N}{3}$, we will create one entry $(p-1, \ell)$ and insert it at the back of Q, and then set ℓ as w_i. Note that if $w_i > \frac{\epsilon N}{3}$, then $\ell > \frac{\epsilon N}{3}$. For this case, we create one more entry (p, ℓ) and insert it at the back of Q immediately, and then set ℓ to zero. Therefore, this entry samples the frequency of the item in $W_{p,p}$, i.e. only samples the frequency of the item at the single time unit p. The invariant below can be verified easily:

Invariant (*): At any time, an entry in Q either samples the value of at most $\frac{\epsilon N}{3}$, or samples the item at a single time unit.

Batch decrement (if necessary): Note that we may create one window counter when an item arrives. If there exist more than $\frac{3R}{\epsilon}$ window counters (excluding C_{total}), we carry out a batch decrement such that, every counter (except C_{total}) is deducted by the minimum of the values of all the window counters. The window counter with value of zero after batch decrement will be removed.

Update C_{total}: We increment C_{total}, which estimates the total weight of all the items in the sliding window, by w_i and remove the expired entry (if any) in C_{total}.

Recall that the value $v(C)$ of a window counter C is the sum of the values of all the entries in Q plus ℓ. If we keep a window counter C_a for an item a, the estimated frequency of a, $\hat{f}_a$, is defined as the value of C_a, i.e. $v(C_a)$. If no counter is kept for a, $\hat{f}_a = 0$. The estimate $\hat{S}$ of the total weight S in the sliding window is equal to $v(C_{total})$. We prove that our algorithm solves the problem using $O(\frac{R}{\epsilon})$ space as follows.

Lemma 2. *Consider any window $W_{p-N+1,p}$. Our algorithm can (1) estimate the frequency of any item a in the window with guarantee $f_a - \epsilon S \leq \hat{f}_a \leq f_a + \epsilon S$, and (2) estimate the total weight of all the items in the window with guarantee $S \leq \hat{S} \leq (1+\epsilon)S$.*

Proof. Note that the count of an item will decrease only when there is a batch decrement. Let d_a denote the total count deducted for an item a due to batch decrements made during $W_{p-N+1,p}$, more precisely, during the arrival of the items in $W_{p-N+1,p}$. Before proving that $d_a \leq \epsilon N$ for any item a, we first prove that the sum of the values of all the $3R/\epsilon$ window counters is at most $2RN$ when the $(p-N)$-th item arrives. For any sliding window W, the total weight of all the items in W is no larger than RN since each item has weight of at most R and there are exactly N items in W. Note that the entry at the front of Q is the only entry that may sample the count outside the sliding window. Let (p_f, v_f) denote this entry. If (p_f, v_f) samples for one single time unit only, then C_a will not sample any count outside the sliding window since the p_f-th item in the stream must be within the sliding window (otherwise, we will remove this entry). Otherwise, by Invariant (*), v_f must be no larger than $\frac{\epsilon N}{3}$. Since each of the $\frac{3R}{\epsilon}$ window counters samples at most $\frac{\epsilon N}{3}$ counts outside the sliding window, the sum of the values of all the $3R/\epsilon$ window counters is at most $RN + 3R/\epsilon \times \epsilon N/3 = 2RN$ when the $(p-N)$-th item arrives. As we have argued that the total weight of all the items in any sliding window is no larger than RN, the value of all the window counters will increase by at most RN when the items in $W_{p-N+1,p}$ arrive. Note that there will not be too many counts deducted due to batch decrements since every window counter has value no less than zero. Therefore, $(3R/\epsilon + 1) \times d_a \leq 3RN$, that gives $d_a < \epsilon N$.

Moreover, when we estimate the count of an item, we will also include the front entry $e_f = (p_f, v_f)$ in Q that may sample some of the counts that are in $W_{p-N+1,p}$ and some that are not. More precisely, the front entry will sample the counts in $W_{p'_f,p_f}$ where $p'_f \leq p - N + 1 \leq p_f$. If $v_f \leq \epsilon N/3$, then $v(C_a)$ overestimates the count of a by at most $\epsilon N/3$ for this. By Invariant (*), if $v_f > \epsilon N/3$, e_f will sample only a single time unit, i.e. the counts in W_{p_f,p_f}. Note that e_f is at the front of Q and thus, $p_f = p-N+1$ (otherwise, there must exists one more entry (p', v') where $p-N+1 \leq p' < p_f$). For this case, $v(C_a)$ will sample the counts of a exactly in $W_{p-N+1,p}$. Recall that d_a denotes the total amount deducted for C_a due to batch decrement. We have $f_a - d_a \leq v(C_a) \leq f_a + \frac{1}{3}\epsilon N$. Since $0 \leq d_a \leq \epsilon N$ and $\hat{f}_a = v(C_a)$, we have $f_a - \epsilon N \leq \hat{f}_a \leq f_a + \frac{1}{3}\epsilon N \leq f_a + \epsilon N$. Since $N \leq S$, we have $f_a - \epsilon S \leq \hat{f}_a \leq f_a + \epsilon S$.

Similarly, $v(C_{total})$ overestimates the total counts in the sliding window by at most $\epsilon N/3 \leq \epsilon S$ and never underestimates S. Thus, $S \leq \hat{S} \leq (1+\epsilon)S$. □

Theorem 1. *Our algorithm solves the heavy hitters identification problem using* $O(\frac{R}{\epsilon})$ *space and* $O(\frac{R}{\epsilon})$ *time for update and query.*

Proof. We first estimate the space used by our algorithm. Note that we have at most $\frac{3R}{\epsilon} + 1$ window counters any time. This means that we need $\Theta(\frac{3R}{\epsilon})$ space for keeping the auxiliary information of the window counters. With the observation that the values of counters decremented due to batch decrements will not increase the usage of memory of a counter, we assume that there is no batch decrement in our analysis. Since we create one entry or two entries only when the weight of new item plus ℓ exceeds $\epsilon N/3$, the sum of counts of any two consecutive entries must exceed $\epsilon N/3$ if there is no batch decrement. From the proof of Lemma 2, the total value of all the window counters is at most $2RN$ at any moment. Therefore, we have at most $(2RN/\frac{\epsilon N}{3}) \times 2 = 12R/\epsilon$ entries. Each entry stores two integers and thus, the total space for storing all the window counters (excluding C_{total}) is $O(R/\epsilon)$. We have the similar space complexity analysis for C_{total}. The total weight in any sliding window is at most RN and we will keep at most one entry sampling the counts not in the sliding window. Hence we have at most $(RN)/\frac{\epsilon N}{3} \times 2 + 1 = 6R/\epsilon + 1$ entries in C_{total}. It follows that the total space required is $O(\frac{R}{\epsilon})$.

The correctness of the algorithm follows from Lemma 1 and Lemma 2. The bounds on the update and query times follows from the fact that we access the whole data structure in constant number of times for every update or query. □

4 Estimating the Sum in Sliding Window

In this section, we study the Integer-Sum problem: Given a stream of data items which are positive integers in the range $[1, R]$, return an estimate $\hat{S}$ of the sum S of the last N items with the guarantee $(1-\epsilon')S \leq \hat{S} \leq (1+\epsilon')S$. We give a lower bound on the space complexity of the problem and describe an algorithm for solving it. The algorithm given in this section will be useful in the following sections.

Theorem 2. *Any deterministic algorithm with relative error less than ϵ' requires at least $\Omega(\frac{\log^2 R}{\epsilon'})$ bits of memory when $R \leq \epsilon' N$.*

The proof will be given in the full paper. Before describing the algorithm, we give the following definitions.

Definition 1. *We say that an integer i falls in band α, or $i \in band_\alpha$ if $(1+\epsilon')^\alpha N \leq i < (1+\epsilon')^{\alpha+1}N$ for any $\alpha \geq 0$. The integer i falls in band -1 if $0 < i < N$.*

Our algorithm will sample the items in order to save the space. However, instead of having fixed sampling rate, we will sample the item at varying rate: the older items will be sampled less frequently compared with the one that arrived more recently. The details of the algorithm is given as follows.

Initialization: We will maintain a linked list L and a variable γ. Initially, L is empty and $\gamma = 0$. We will use γ to count the sum of integers coming from the data stream.

Update: When the i-th item of some value v arrives for any $i > 0$, we will:
(i) Increment the counter: If the value of γ does not exceed $\epsilon' N$ after adding v to γ, we will increment γ by v. Otherwise, we will first insert a newly created node at the head of L. Let $W_{p,q}$ denote the window containing the set of items from the p-th item to the q-th item in the data stream where $p \leq q$. The new node contains the entry $(i-1, \gamma)$ where γ represents the sum of the integers in $W_{p'+1,i-1}$ and (p', s') is originally at the head of L. (If $(i-1, \gamma)$ is the only entry in L, γ will represent the sum of integers in $W_{p',i-1}$ for some $p' \leq i-N+1$.) Then we set the counter γ to v. If $\gamma > \epsilon' N$ (note that it occurs only when $v > \epsilon' N$),

we will create one more entry (i, γ) and insert it at the head of L immediately, and then set γ to zero. Therefore, this entry samples the weight of the item in $W_{i,i}$, i.e. only samples the frequency of the item for the single time unit, p.
(ii) Delete the expired entry: Delete the node at the tail of L that contains (p, s) if the p-th item is no longer within the sliding window, i.e., $p < i - N + 1$.
(iii) Trim the linked list L**:** Suppose we have m entries. Without loss of generality, we label the entries in order: $e_1 = (p_1, s_1)$, ..., $e_m = (p_m, s_m)$ such that e_1 is at the head of L, e_2 is next to e_1, ..., and e_m is at the tail of L. We trim L by traversing L from the head. Define $t_i = \sum_{j=1}^{i-1} s_j + \gamma$. Suppose that e_i is an entry such that $t_i \in band_0$ and $t_{i-1} \in band_{-1}$. Let $band(t_i)$ denote the band of t_i. Check the succeeding entries to find an integer ι such that $\sum_{j=i}^{\iota} s_j \leq (1+\epsilon')^{band(t_i)}\epsilon' N < \sum_{j=i}^{\iota+1} s_j$. If $\iota \leq i$, we will repeat this procedure again at the entry e_{i+1}. Otherwise, set $s_i = \sum_{j=i}^{\iota} s_j$ and delete the entries $e_{i+1}, e_{i+2}, \ldots, e_\iota$ so that s_i represents the sum of integers in $W_{p_{\iota+1}+1, p_i}$. Repeat this procedure again at the entry $e_{\iota+1}$.

Answer query: Give an estimate $\hat{S}$ of the sum S of the integers by returning the total of the s_i in the entries (p_i, s_i) of L plus γ. The following invariant can be verified easily:

Invariant (**): At any time, every entry $e_i = (p_i, s_i)$ either samples the value of at most $(1+\epsilon')^{band(t_i)}\epsilon' N$ if $band(t_i) \geq 0$ and at most $\epsilon' N$ if $band(t_i) < 0$, or samples for a single time unit.

Lemma 3. *The algorithm guarantees that* $S \leq \hat{S} \leq S + \epsilon' \max(S, N)$ *where* S *and* N *denote the total sum of integers in the sliding window and the size of the sliding window, respectively.*

Proof. Suppose we have m entries in L. Let us define e, s and t as the same way in the description of the algorithm. Recall that we will return the sum of the values of all the entries of L plus γ when there is a query. Thus, we will return $\sum_{i=1}^{m} s_i + \gamma = t_m + s_m$.

Since we trim L whenever an item arrives, e_m will not be an expired item (i.e. the p_m-th item should be in sliding window). Thus, s_m is equal to the sum of the weight of the items of which some are in the sliding window while some may not. This implies

$$t_m < S \leq t_m + s_m = \hat{S}. \tag{3}$$

Thus, our algorithm will never underestimate the total weight of the items in the sliding window (i.e. $\hat{S} \geq S$). Note that if e_m only samples for a single time unit, this implies e_m will sample the time unit that is exactly at the boundary of the sliding window. Thus, the sum of the values of all the entries in L plus γ is exactly the sum of the values of all items in the whole sliding window. If e_m does not sample for a single time unit, we consider the following two cases:

$band(t_m) \geq 0$: By Invariant (**), we have $s_m \leq (1+\epsilon')^{band(t_m)}\epsilon' N$. By definition, $t_m \geq (1+\epsilon')^{band(t_m)} N$. So we have $s_m \leq \epsilon' t_m$. Together with the facts that $\hat{S} = t_m + s_m$ and $t_m < S$ (by Inequality 3), we have $\hat{S} = t_m + s_m \leq S + \epsilon' S$.
$band(t_m) = -1$: By Invariant (**), we have $s_m \leq \epsilon' N$. Together with $t_m < S$ and $\hat{S} = t_m + s_m$ (by Inequality 3 again), we have $\hat{S} = t_m + s_m < S + \epsilon' N$. □

Lemma 4. *After the step "Trim the linked list L", there exist at most two entries, e_i and e_{i+1}, such that $t_i = \sum_{\ell=1}^{i-1} s_\ell + \gamma$ and $t_{i+1} = \sum_{\ell=1}^{i} s_\ell + \gamma$ fall in the same band (i.e. $band(t_i) = band(t_{i+1})$) except band -1. There are at most $2/\epsilon' + 1$ entries e_u with t_u fall in band -1.*

Proof. Recall that some entries are deleted during the step "Trim the linked list L" such that $(1+\epsilon')^{band(t_i)}\epsilon' N < s_i + s_{i+1}$. Together with the fact that $t_i \geq (1+\epsilon')^{band(t_i)} N$ by Definition 1, we have $t_{i+2} = \sum_{\ell=1}^{i+1} s_\ell + \gamma = t_i + (s_i + s_{i+1}) > (1+\epsilon')^{band(t_i)} N + (1+\epsilon')^{band(t_i)}\epsilon' N = (1+\epsilon')^{band(t_i)+1} N$. Thus, the value t_{i+2} of e_{i+2} must not fall in $band(t_i)$. Therefore, there exist at most two entries e_i and e_{i+1} such that t_i and t_{i+1} fall in the same band.

Consider any entry $e_u = (p_u, s_u)$ where $t_u = \sum_{\ell=1}^{u-1} s_\ell + \gamma$ is in band -1. Recall that s_u is the sum of the integers in $W_{p_{u+1}+1, p_u}$, and we create e_u since the value of γ will exceed $\epsilon' N$ when we add the value of $(p_u + 1)$-th item to γ. It follows that $s_u + s_{u+1}$ must exceed $\epsilon' N$. Therefore, we can have at most $N/(\epsilon' N) \times 2 + 1 = 2/\epsilon' + 1$ entries since band -1 covers the values from 1 to $N - 1$. □

Lemma 5. *Our algorithm stores at most $O(\frac{\log R}{\epsilon'})$ entries.*

Proof. Suppose we have m entries in L. Recall that the total weight of the sliding window is at most RN. Thus, $t_m \leq S \leq RN$(Inequality 3). Let α be the largest possible band number in $[1, RN]$ such that $(1+\epsilon')^\alpha N \leq RN$. Solving the above inequality and we have $\alpha \leq \frac{\ln R}{\ln(1+\epsilon')}$. By the fact that $\ln(1+\epsilon') > \frac{\epsilon'}{1+\epsilon'}$, $\alpha \leq \frac{1}{\epsilon'}(1+\epsilon')\ln R$. It follows that $band(t_m) \leq \frac{1}{\epsilon'}(1+\epsilon')\ln R$. Since there are at most two entries for every band $b \geq 0$ and we have at most $2/\epsilon' + 1$ entries in band -1 by Lemma 4, our algorithm stores at most $O(\frac{\log R}{\epsilon'})$ entries. □

Theorem 3. *Our algorithm solves the Integer-Sum problem with $O(\frac{\log R}{\epsilon'})$ space. It takes $O(\frac{\log R}{\epsilon'})$ time for update and constant time for answering query.*

Proof. By Lemma 3, we have $S \leq \hat{S} \leq S + \epsilon' \max(S, N)$ where S and N denote the total sum of integers in the sliding window and the size of the sliding window, respectively. Note that $S \geq N$ because there are N items in the sliding window and each of them has value of no less than one. Thus, $S \leq \hat{S} \leq S + \epsilon' S$ and thus, our algorithm solves the Integer-Sum problem.

By Lemma 5 and the fact that each entry in L uses constant space, it follows that our algorithm solves the Integer-Sum problem using $O(\frac{\log R}{\epsilon'})$ space. For update, we need to access the linked list a constant number of times, which takes $O(\frac{\log R}{\epsilon'})$ time. For query, we can keep a counter that maintains the sum of the s_i of all the entries $e_i = (p_i, s_i)$ in L. When some entry expires, the counter will be updated accordingly. When there is a query, we return the value of the counter plus γ. Thus, answering a query takes constant time. □

Therefore, we can estimate the total weight S of all the items in the sliding window using $O(\frac{\log R}{\epsilon'})$ space with the guarantee that $S \leq \hat{S} \leq (1+\epsilon')S$. This is useful for Section 5 since we no longer need to prove that we have this guarantee again when we show how our algorithm solves the heavy hitters identification problem.

5 An Improved Algorithm

We improve our algorithm proposed in Section 3 by keeping $\frac{3c}{\epsilon}$ counters proposed in Section 4 instead of $\frac{3R}{\epsilon}$ window counters (c will be defined soon). This algorithm solves the problem with the assumption that the total weights of all the items in any disjoint adjacent sliding windows will not differ too much. We choose c such that it is an upper bound on the ratio between the total weights of the two adjacent disjoint sliding windows. More precisely, we assume that $S_{p+1,p+N} \geq S_{p-N+1,p}/c$ for any integer $p \geq N$ where $S_{p,q}$ denotes the total weight of all the items $W_{p,q}$. We will show that our algorithm uses $O(\frac{c \log R}{\epsilon^2})$ space for finding heavy hitters.

First of all, we need the following simple modifications to the structure of the counter introduced in the Section 4 before using them directly: (1) We set ϵ' as $\epsilon/4$; (2) Note that our counter originally can handle the increment of the count only, but not the decrement of the count. If we decrement the counter by $v \leq \gamma$, then this can be done by reducing count of the counter γ by v. If we decrement the counter by $v > \gamma$, we will remove the entry (p_m, s_m) at the tail of L and set γ to $\gamma + s_m - v$. If $\gamma + s_m < v$, we will remove more entries until the sum of the values is no less than v.

We will have nearly the same algorithm as in Section 3 except that (i) we keep our counters proposed in Section 4 instead of window counters; and (ii) we carry out batch decrement when there are more than $3c/\epsilon$ counters. By Lemma 1 and the following theorem, our algorithm finds the heavy hitters over sliding windows using $O(\frac{c \log R}{\epsilon^2})$ space.

Theorem 4. *Let $W_{p,q}$ denote the window covering the items from p-th item to q-th item in the data stream for some $p \leq q$. Consider any window $W_{p+1,p+N}$. We can estimate the frequency f for any item in $W_{p+1,p+N}$ with the guarantee $f - \epsilon S_{p+1,p+N} \leq \hat{f} \leq f + \epsilon S_{p+1,p+N}$ using $O(\frac{c \log R}{\epsilon^2})$ space. Our algorithm supports $O(\frac{c \log R}{\epsilon^2})$ update time and $O(\frac{c}{\epsilon})$ query time.*

Proof. Let f_a denote the frequency of a in $W_{p+1,p+N}$ for any item a. Suppose there is a counter that maintains the estimated frequency of an item a. If we are keeping a counter for estimating the frequency of a, then the estimated frequency of a, i.e. $\hat{f}_a$, is defined as the sum of all the entries of L plus γ in that counter. If no counter is kept for a, $\hat{f}_a = 0$. Let $d_{\max}$ denote the largest possible value deducted accumulatively for any counter during $W_{p+1,p+N}$. Recall that $\epsilon' = \epsilon/4$. By Lemma 3, we have

$$f_a - d_{\max} \leq \hat{f}_a \leq f_a + \tfrac{\epsilon}{4} S_{p+1,p+N}. \tag{4}$$

We first estimate $d_{\max}$. Let σ denote the sum of the values of all the $\frac{3c}{\epsilon}$ counters when the p-th item arrives. By Lemma 3, the value of the counter for any item a must be no greater than $f_a + \max(\frac{1}{4}\epsilon N, \frac{1}{4}\epsilon f_a)$ where f_a denotes the frequency of a in $W_{p-N+1,p}$. It can be verified that $\sigma \leq (1+\epsilon/4)S_{p-N+1,p} + \epsilon N/4 \times (3c/\epsilon) \leq (1+\epsilon/4)S_{p-N+1,p} + 3cN/4$. When the items in $W_{p+1,p+N}$ arrive, the sum of the values of all counters will be increased by $S_{p+1,p+N}$. Note that it is impossible to have negative count in our counters. Therefore, $(3c/\epsilon+1)\times d_{\max} \leq \sigma + S_{p+1,p+N} \leq (1+\epsilon/4)S_{p-N+1,p} + 3cN/4 + S_{p+1,p+N}$, and we have $d_{\max} \leq \frac{\epsilon(1+\epsilon/4)}{3c}S_{p-N+1,p} + \epsilon N/4 + \frac{\epsilon}{3c}S_{p+1,p+N} \leq \epsilon S_{p+1,p+N}$ since (1) $S_{p+1,p+N} \geq N$, (2) $S_{p-N+1,p}/c \leq S_{p+1,p+N}$ by assumption, (3) $c \geq 1$, and (4) $\epsilon < 1$. Together with the Inequality 4, we have $f_a - \epsilon S_{p+1,p+N} \leq \hat{f} \leq f_a + \frac{\epsilon S_{p+1,p+N}}{4} \leq f_a + \epsilon S_{p+1,p+N}$ for any item a.

We use at most $O(\frac{c\log R}{\epsilon^2})$ space as each counter uses $O(\frac{\log R}{\epsilon})$ space and we keep at most $3c/\epsilon$ counters. We can see that we will access the whole data structure once for the worst case when an item arrives. When there is a query, we need to check every counter to estimate frequency of the corresponding item. This can be done in constant time for each counter by maintaining a variable storing the value of the counter. □

6 A Randomized Algorithm for Finding Heavy Hitters

We first give a randomized algorithm that modifies CM sketch [2] by using our counter proposed in Section 4. Let e denote the base of the natural logarithm. The algorithm works as follows:

Initialization. We maintain a two-dimensional array of our counters with the width of $w = \lceil \frac{2e}{\epsilon} \rceil$ and depth of $d = \lceil \ln \frac{1}{\delta} \rceil$: $count[1,1], \ldots, count[d,w]$. We set $\epsilon' = \epsilon/2$ when we set up our counters. And each counter has value of zero initially. Consider any window $W_{p-N+1,p}$ for $p \geq N$. Let $f[i,j]$ denote the cumulative counts added to $count[i,j]$ during $W_{p-N+1,p}$. Our counter $count[i,j]$ is used to estimate $f[i,j]$ for any $1 \leq i \leq d$ and $1 \leq j \leq w$. We also maintain d pairwise-independent hash functions $h_1 \cdots h_d : \{1 \cdots D\} \rightarrow \{1 \cdots w\}$ where there are D possible distinct items.

Update. When an item (a_i, w_i) arrives, we increase the value of counters $count[j, h_j(a_i)]$ by w_i, for any $1 \leq j \leq d$, and we will update every counter so that the expired entries will be removed.

Query. When there is a query for the frequency of a_j, we will return $\hat{f}(a_j) = \min_i \hat{f}[i, h_i(a_j)]$ where $\hat{f}[i, h_i(a_j)]$ is the estimation of $f[i, h_i(a_j)]$ by $count$ $[i, h_i(a_j)]$.

Lemma 6. *In our modified CM sketch, the value* $\hat{f} = \min_i count[i, h_i(a)]$ *for any item a has the following guarantees:* $f \leq \hat{f}$*; and* $\hat{f} \leq f + \frac{1}{2}\epsilon S'$ *with probability of* $1-\delta$*, where* S' *is the total weight in the data stream.*

Proof. Similar to the proof of Theorem 1 in [2]. □

Theorem 5. *Our algorithm guarantees that:* $\hat{f}(a) \leq f(a) + \epsilon S$ *with probability of* $1-\delta$*; and* $\hat{f}(a) \geq f(a)$*, where S denotes the total weight of all items in the sliding window. Our algorithm uses* $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\log R)$ *space, updates in* $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\log R)$ *time and estimates the frequency of any item in* $O(\log\frac{1}{\delta})$ *time.*

The proof will be given in full paper. For finding heavy hitters, we keep a counter proposed in Section 4 to estimate the total weight S in sliding window and use the group testing technique as described in [2]: we have to keep $\log D$ different sketches, each for different dyadic range, where D is the number of distinct items in the stream. See [2] for the details. We can prove that

Theorem 6. *We find heavy hitters using* $O(\frac{1}{\epsilon^2}\log R \log D \log \frac{\log D}{\delta\theta})$ *space, update and query times with success probability* $1-\delta$.

References

1. Arasu, A., Manku, G.S.: Approximate counts and quantiles over sliding windows. In: Symposium on Principles of Database Systems (PODS), pp. 286–296 (2004)
2. Cormode, G., Muthukrishnan, S.: An improved data stream summary: The count-min sketch and its applications. Journal of Algorithms 55(1), 58–75 (2005)
3. Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows. In: Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 635–644 (2002)
4. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: European Symposium on Algorithms (ESA), pp. 348–360 (2002)
5. Ganguly, S., Majumder, A.: CR-precis: A deterministic summary structure for update data streams. In: Chen, B., Paterson, M., Zhang, G. (eds.) ESCAPE 2007. LNCS, vol. 4614, pp. 48–59. Springer, Heidelberg (2007)
6. Gibbons, P.B., Tirthapura, S.: Distributed streams algorithms for sliding windows. In: ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 63–72 (2002)
7. Golab, L., DeHaan, D., López-Ortiz, A., Demaine, E.D.: Finding frequent items in sliding windows with multinomially-distributed item frequencies. In: International Conference on Scientific and Statistical Database Management (SSDBM), pp. 425–426 (2004)
8. Karp, R.M., Shenker, S., Papadimitriou, C.H.: A simple algorithm for finding frequent elements in streams and bags. ACM Transactions on Database Systems (TODS) 28(1), 51–55 (2003)
9. Lee, L.K., Ting, H.F.: Maintaining significant stream statistics over sliding windows. In: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms (SODA), pp. 724–732 (2006)
10. Lee, L.K., Ting, H.F.: A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In: Symposium on Principles of Database Systems (PODS), pp. 290–297 (2006)
11. Lin, X., Lu, H., Xu, J., Yu, J.X.: Continuously maintaining quantile summaries of the most recent N elements over a data stream. In: International Conference on Data Engineering (ICDE), pp. 362–374 (2004)
12. Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. In: Very Large Data Bases (VLDB) Conference, pp. 346–357 (2002)
13. Misra, J., Gries, D.: Finding repeated elements. Science of Computer Programming 2, 143–152 (1982)
14. Xu, J., Lin, X., Zhou, X.: Space Efficient Quantile Summary for Constrained Sliding Windows on a Data Stream. In: Li, Q., Wang, G., Feng, L. (eds.) WAIM 2004. LNCS, vol. 3129, pp. 34–44. Springer, Heidelberg (2004)

Fixed-Parameter Algorithms for Cluster Vertex Deletion

Falk Hüffner*, Christian Komusiewicz**, Hannes Moser***, and Rolf Niedermeier

Institut für Informatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, D-07743 Jena, Germany
{hueffner,ckomus,moser,niedermr}@minet.uni-jena.de

Abstract. We initiate the first systematic study of the NP-hard Cluster Vertex Deletion (CVD) problem (unweighted and weighted) in terms of fixed-parameter algorithmics. In the unweighted case, one searches for a minimum number of vertex deletions to transform a graph into a collection of disjoint cliques. The parameter is the number of vertex deletions. We present efficient fixed-parameter algorithms for CVD. Our iterative compression algorithm for CVD seems to be the first nontrivial application of this fairly new technique to a problem that is not a feedback set problem. Moreover, we study the variant of CVD where the number of cliques to be generated is specified. Here, we detect connections to fixed-parameter algorithms for (weighted) Vertex Cover.

1 Introduction

Graph modification problems form a core topic in algorithmic graph theory with many applications. In particular, cluster graph modification problems [21] have recently received considerable interest. Here, the basic problem is, given an undirected graph G, to find a minimum number of editing operations that transform G into a collection of disjoint complete subgraphs, a *cluster graph*. Herein, the three standard editing operations are adding edges, deleting edges, and deleting vertices. For instance, Cluster Editing asks whether a graph can be transformed into a cluster graph by altogether at most k edge additions and edge deletions. Cluster Editing is NP-complete; it recently has shown particularly useful for clustering biological data [6, 19]. Whereas also a factor-2.5 polynomial-time approximation for Cluster Editing is known [3, 23], in practical applications fixed-parameter algorithms (combined with some heuristics) providing optimal solutions seem to dominate [4, 6, 19]. Parameterized

* Supported by the Deutsche Forschungsgemeinschaft, Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4.
** Supported by a PhD fellowship of the Carl-Zeiss-Stiftung.
*** Supported by the Deutsche Forschungsgemeinschaft, project ITKO (iterative compression for solving hard network problems), NI 369/5.

complexity studies for Cluster Editing were initiated by Gramm et al. [11] and have been further pursued in a series of papers [4, 6, 7, 10, 12, 18, 19]. A previously shown bound of $O(1.92^k + n^3)$ for an n-vertex graph [10] can be improved by combining a linear-time problem kernel [7] with the currently best claimed running time of $O(1.83^k + n^3)$ [4] to get an algorithm with running time $O(1.83^k + n + m)$, where m is the number of edges in the graph. Moreover, problem kernels, based on efficient data reduction rules, with only $O(k)$ vertices are known [7, 12], the best upper bound currently being $4k$ [12].

Whereas Cluster Editing has been subject to intensive research, its "sister problem" Cluster Vertex Deletion so far has been widely neglected. Here, we aim at finding a minimum number of vertices such that their deletion transforms a given graph into a cluster graph.[1]

> Weighted Cluster Vertex Deletion
> **Instance:** An undirected graph $G = (V, E)$, a vertex weight function $\omega : V \to [1, \infty)$, and a nonnegative number k.
> **Question:** Is there a subset $X \subseteq V$ with $\sum_{v \in X} \omega(v) \leq k$ such that deleting all vertices in X from G results in a cluster graph (i. e., a graph where every connected component forms a complete graph)?

The unweighted version asks whether there exists a subset $X \subseteq V$ such that $|X| \leq k$ (in other words, all vertices have weight exactly one).

Motivation. As Cluster Editing, Cluster Vertex Deletion may find applications in graph-modeled data clustering: Assume that we have a number of samples, some of which are equivalent (e. g., DNA samples, some of which are from the same species) and a method to test two samples for equivalence. A graph is formed where each vertex corresponds to a sample and an edge between two vertices is added when their samples are tested as equivalent. In the absence of errors, the resulting graph is a cluster graph, where each connected component corresponds to an equivalence class (e. g., a species). However, an unknown subset of samples may be contaminated and can produce unpredictable comparisons to other samples. An optimal solution for unweighted Cluster Vertex Deletion, that is, a minimum-cardinality set of vertices whose deletion produces a cluster graph, then provides the most parsimonious explanation for the data under this model. This clearly extends to the weighted case. Finally, in comparison to Cluster Editing, a small parameter value k (that is, the number of editing operations) appears even more likely for Cluster Vertex Deletion, making a parameterized approach particularly meaningful here.

Known results. By general results for vertex deletion problems for hereditary graph properties, it follows that already unweighted Cluster Vertex Deletion is NP-complete [15]. Only few specific results for (unweighted) Cluster

[1] Parameterized problems (as follows) usually are formulated as decision problems—all our algorithms will also solve the corresponding optimization problem within the same time bounds.

Vertex Deletion are known.[2] These are based on the simple observation that a graph is a cluster graph if and only if it does not contain an induced P_3, a path of three vertices.[3] Gramm et al. [10] used an elaborate case distinction found with computer help to derive a search tree algorithm running in $O(2.26^k m)$ time for an m-edge graph. This can be improved to $O(2.08^k + n^3)$, n denoting the number of vertices, by using a straightforward reduction of unweighted Cluster Vertex Deletion to the 3-Hitting Set problem (transforming each induced P_3 into a three-element set) and employing a sophisticated algorithm for 3-Hitting Set [22]. Moreover, kernelization results for 3-Hitting Set [1] also imply an $O(k^2)$-vertex problem kernel for unweighted Cluster Vertex Deletion, which can be found in $O(n^3)$ time. A weighted Cluster Vertex Deletion instance can be easily transformed into a weighted 3-Hitting Set instance. With this transformation, an $O(k^3)$-vertex problem kernel result for weighted 3-Hitting Set [2] can be adapted to weighted Cluster Vertex Deletion. Moreover, weighted 3-Hitting Set possesses an elaborate search tree algorithm based on case distinction [8], implying an $O(2.25^k + n^3)$ running time for weighted Cluster Vertex Deletion.

New results. One of our main results is an elegant iterative compression algorithm for weighted Cluster Vertex Deletion using matching techniques, running in $O(2^k k^9 + n^3)$ time. Notably, this seems to be the first nontrivial application of the technique of iterative compression (described by Reed et al. [20]; see also [16, Chapter 11]) to a non-feedback set problem. We extend our studies to the (also NP-hard) case where the number of clusters to be generated is given by a second parameter d. Such studies have also been undertaken for Cluster Editing [9, 12, 21], but note that for Cluster Editing clearly $d \leq 2k$. By way of contrast, since vertex deletion is a "stronger" operation than edge deletion, in the case of Cluster Vertex Deletion also $d > 2k$ is possible. Observe that $d = 1$ yields the Clique problem, leading to the NP-hardness of so-called d-Cluster Vertex Deletion also for $d > 1$. Since d-Cluster Vertex Deletion is already NP-hard for $d = 1$, a parameterization only with respect to the parameter d is meaningless. Considering the combined parameter (d, k), however, we can provide further fixed-parameter tractability results. First, we nontrivially extend the kernelization result for weighted Cluster Vertex Deletion to a problem kernel for weighted d-Cluster Vertex Deletion, again achieving an $O(k^3)$-vertex problem kernel. Based on this, we develop three fixed-parameter algorithms for weighted d-Cluster Vertex Deletion with the following running times: $O(3^k + n^3)$, $O(1.40^k k^{3d} + n^3)$, and $O(1.84^{k+d} + n^3)$. Depending on the value of d, each of these algorithms may be preferable in certain constellations.

[2] Jansen et al. [14] studied the closely related problem of finding d pairwise disjoint cliques with maximum overall number of vertices, motivated by applications in scheduling. Note that, other than in Cluster Vertex Deletion, they allowed to have edges between cliques. Jansen et al. gave polynomial-time algorithms for special graph classes, contrasting the NP-complete general case.

[3] In the remainder of this work, when simply writing of containment of a P_3 in a graph we actually always refer to an induced P_3.

```
CompressCVD($G, X$)
1  $X' \leftarrow X$
2  for each $S \subseteq X$:
3      if $G[S]$ is a cluster graph:
4          $G' \leftarrow G \setminus (X \setminus S)$;  $R \leftarrow V(G' \setminus S)$
5          $G' \leftarrow$ReduceRule1($G'$)
6          $G' \leftarrow$ReduceRule2($G'$)
7          $G' \leftarrow$ReduceRule3($G'$)
8          Classify each vertex $u$ in $R$ according to $N(u) \cap S$
9          $H \leftarrow$ auxiliary graph
10         $M \leftarrow$ maximum weight matching in $H$
11         Delete all vertices not in a class corresponding to an edge in $M$
12         $D \leftarrow$ vertices deleted in lines 4–7 and 11
13         if $\omega(D) < \omega(X')$:
14             $X' \leftarrow D$
15 return $X'$
```

Fig. 1. Pseudo-code for CompressCVD, where $\omega(A) := \sum_{v \in A} \omega(v)$ for $A \subseteq V$

In the latter two algorithms, fixed-parameter algorithms for weighted Vertex Cover play a decisive role.

Due to the lack of space, most details are deferred to the full version of this paper.

2 Iterative Compression for Cluster Vertex Deletion

We now describe a novel iterative compression algorithm for weighted Cluster Vertex Deletion. General considerations about iterative compression algorithms can be found in [13] and [16, Chapter 11]. We first describe how to employ a compression routine, and then the compression routine itself. We call a set of vertices whose deletion produces a cluster graph a *CVD set*.

The general idea behind our iterative compression is as follows. We start with $V' = \emptyset$ and $X = \emptyset$; clearly, X is a CVD set for $G[V']$. Iterating over all graph vertices, step by step we add one vertex $v \notin V'$ from V to both V' and X. Then X is still a CVD set for $G[V']$, although possibly not a minimum one. We can, however, obtain a minimum one by applying the compression routine CompressCVD. It takes a graph G and a CVD set X for G, and returns a minimum CVD set for G. Therefore, it is a loop invariant that X is a minimum-size CVD set for $G[V']$. Since eventually $V' = V$, we obtain an optimal solution for G once the algorithm returns X.

In the rest of this section, we describe the compression routine CompressCVD following the pseudo-code in Fig. 1. For this, consider a smaller CVD set X' as a modification of the larger CVD set X. This modification retains some vertices $Y \subsetneq X$, while the other vertices $S := X \setminus Y$ are replaced by new vertices from $V \setminus X$. The idea is to try by brute force all $2^{|X|} - 1$ partitions of X into such sets Y and S (line 2). For each such partition, the vertices from Y are immediately deleted, since we already decided to take them into the CVD set. In the resulting instance $G' = (V', E') := G \setminus Y$, it remains to find an optimal CVD set that is disjoint from S. This is a much easier task than finding a CVD set in general; in fact, it can be done in polynomial time using data reduction and maximum matching.

First, we discard partitions where S does not induce a cluster graph (line 3); these cannot lead to a solution, since we determined that none of the vertices in S would be deleted. Further, $R := V' \setminus S$ also induces a cluster graph, since $R = V \setminus X$ and X is a CVD set. Therefore, the following problem remains:

CVD Compression
Instance: An undirected graph $G = (V, E)$, a vertex weight function $\omega : V \to [1, \infty)$, and a subset $S \subseteq V$ such that $G[S]$ and $G \setminus S$ are cluster graphs.
Task: Find a set $X' \subseteq V \setminus S$ such that $G \setminus X'$ is a cluster graph and $\sum_{v \in X'} \omega(x)$ is minimum.

The instance can now be simplified by a series of data reduction rules. We call a connected component in a cluster graph a *cluster*.

Reduction Rule 1. *Delete all vertices in $R := V \setminus S$ that are adjacent to more than one cluster in $G[S]$.*

Reduction Rule 2. *Delete all vertices in R that are adjacent to some, but not all vertices of a cluster in $G[S]$.*

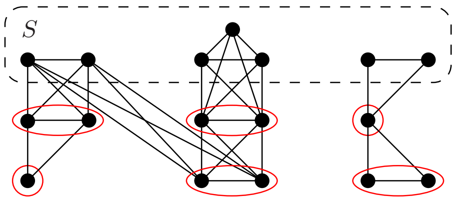
Reduction Rule 3. *Remove connected components that are complete graphs.*

After Rules 1–3 have been applied, the instance is much simplified. In each cluster in $G[R]$, we can divide the vertices into equivalence classes according to their neighborhood in S; each class then contains either vertices adjacent to all vertices of a particular cluster in $G[S]$, or the vertices adjacent to no vertex in S (see Fig. 2a). This classification is useful because of the following lemma.

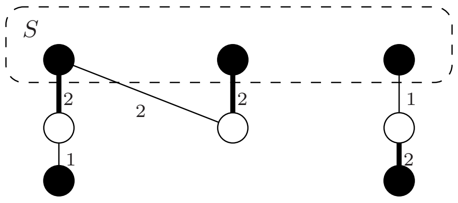
Lemma 1. *In an optimal* CVD compression *solution, for each cluster in $G[R]$, either the vertices of exactly one class are present, or the whole cluster is deleted.*

Because of Lemma 1, the remaining task is an assignment of each cluster in $G[R]$ to one of its classes (corresponding to the preservation of this class, and the deletion of all other classes within the cluster) or to nothing (corresponding to the complete deletion of the cluster). However, we cannot do this independently for each cluster; we must not choose two classes from different clusters in $G[R]$ which are connected to the same cluster in $G[S]$, since that would create a P_3. This can be modelled as a weighted bipartite matching problem in an auxiliary graph H, where each edge corresponds to a possible choice. The graph H is constructed as follows (see Fig. 2b):

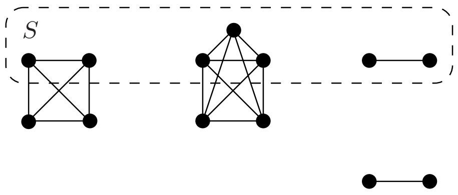
- Add a vertex for every cluster in $G[R]$ (white vertices).
- Add a vertex for every cluster in $G[S]$ (black vertices in S).



(a) Classification



(b) The assignment problem



(c) Final resulting cluster graph

Fig. 2. Assignment problem in the iterative compression, unweighted case

- For a cluster C_S in $G[S]$ and a cluster C_R in $G[R]$, add an edge between the vertex for C_S and the vertex for C_R if there is a class in C_R connected to C_S. This edge corresponds to choosing this class for C_R and is weighted with the total weight of the vertices in this class.
- Add a vertex for each class in a cluster C_R that is not connected to a cluster in $G[S]$ (black vertices outside S), and connect it to the vertex representing C_R. Again, this edge corresponds to choosing this class for C_R and is weighted with the total weight of the vertices in this class.

Since we only added edges between a black and a white vertex, H is bipartite. The task is now to find a *maximum-weight bipartite matching*, that is, a set of edges of maximum weight where no two edges have an endpoint in common. This allows any choice for a cluster, as long as no two clusters share edges to the same cluster in $G[S]$. The following lemma shows that this is a valid approach:

Lemma 2. *A maximum-weight bipartite matching in H provides an optimal* CVD COMPRESSION *solution.*

Fig. 2c shows the resulting cluster graph for our example after deleting the vertex sets corresponding to edges that are not selected by the maximum-weight matching shown in Fig. 2b by bold edges. Note that the size of the solution can be upper-bounded by $k + 1$, since $\forall v \in V : \omega(v) \geq 1$. Altogether, we obtain

Proposition 1. *Weighted* CLUSTER VERTEX DELETION *can be solved in $O(2^k \cdot n^2(m + n \log n))$ time.*

For the unweighted case, we can get better running times, since unweighted matchings can be found faster than weighted ones.

Theorem 1. *Unweighted* Cluster Vertex Deletion *can be solved in* $O(2^k \cdot km\sqrt{n}\log n)$ *time.*

Problem kernelization leads to the following.

Theorem 2. *Unweighted* Cluster Vertex Deletion *can be solved in* $O(2^k \cdot k^6 \log k + n^3)$ *time.*

Curiously, we can use this unweighted algorithm as a subroutine to speed up the weighted case: if we have a solution for an unweighted instance, we can get an optimal weighted solution by executing the compression routine once. This works because the compression does only require that the set X to compress is a CVD set, and does not make any assumptions about its weight.

Theorem 3. *Weighted* Cluster Vertex Deletion *can be solved in* $O(2^k \cdot k^9 + n^3)$ *time.*

In fact, we even have a stronger parameterization in Theorem 3 when compared to Proposition 1: as parameter k, we can use the number of vertices in an optimal unweighted solution, which is less than or equal to the number of vertices in an optimal weighted solution, which in turn is less than or equal to the minimum weight of a weighted solution.

Since the matching subproblem is the bottleneck of the algorithm, it would be nice to replace it with something simpler. However, we can show that the assignment problem in the last step of the compression routine is as hard as the task of finding a maximum weight matching in a bipartite graph, even after applying Reduction Rules 1–3. This indicates that the bottleneck of computing the maximum weight matching might actually be very difficult to overcome with our approach.

3 Cluster Vertex Deletion with a Fixed Number of Clusters

In clustering applications, the number of desired clusters is often known. The deletion of vertices should then produce a *d-cluster graph*, that is, a graph comprising *exactly* d clusters.

Weighted d-Cluster Vertex Deletion
Instance: An undirected graph $G = (V, E)$, a vertex weight function $\omega : V \to [1, \infty)$, and a nonnegative number k.
Question: Is there a subset $X \subseteq V$ with $\sum_{v \in X} \omega(v) \leq k$ such that deleting all vertices in X from G results in a d-cluster graph?

1-Cluster Vertex Deletion is equivalent to Clique, since a 1-cluster graph is a complete graph. Hence, 1-Cluster Vertex Deletion is NP-complete. More generally, we have the following.

Proposition 2. d-Cluster Vertex Deletion *is NP-complete for any constant integer* d.

3.1 An $O(3^k + n^3)$ Time Algorithm

We start with describing a simple search tree algorithm for weighted d-Cluster Vertex Deletion parameterized by the weight k of a solution set. In the search tree, we branch on induced P_3's until the graph is a cluster graph, and then remove surplus clusters in case the graph contains more than d clusters. Before starting the search tree procedure, we perform data reduction. The subsequent problem kernel result makes use of the corresponding result for 3-Hitting Set [2].

Theorem 4. *Weighted d-Cluster Vertex Deletion admits a problem kernel containing $O(k^3)$ vertices, and it can be found in $O(n^3)$ time.*

After kernelization, we perform a search tree procedure. We branch into three cases to destroy a P_3 by vertex deletion, deleting a different vertex in each branch. Since the minimum vertex weight is 1, the parameter is reduced by at least 1 in each search tree branch. Let k' be the sum of the weights of the vertices that may still be removed at a given search tree node. Branching is performed as long as the graph contains a P_3 and $k' \geq 1$. If $k' < 1$, and the graph still contains a P_3, then we have not found a d-CVD set of weight at most k and we cannot remove further vertices. If otherwise the graph contains no P_3, then it is a cluster graph. Let S be the set of vertices that were removed so far. We distinguish four cases.

1. $k' < 0$. The weight of S exceeds k. Therefore, no solution was found.
2. $G \setminus S$ comprises less than d clusters. We can discard S, since S is not a d-CVD set and no superset of S is a d-CVD set.
3. $G \setminus S$ comprises more than d clusters. We compute the sum of the vertex weights for all remaining clusters, and remove a cluster with minimum weight until either $G \setminus S$ is a d-cluster graph (then Case 4 applies) or $k' < 1$ (no solution set was found in this search tree branch).
4. $G \setminus S$ is a d-cluster graph. In this case, S is a d-CVD set of weight at most k.

Clearly, this search tree procedure finds a d-CVD set of minimum weight, since it explores all possibilities to destroy the P_3's of the graph and afterwards optimally removes surplus clusters (in Case 3). Below, we bound the running time of the described algorithm.

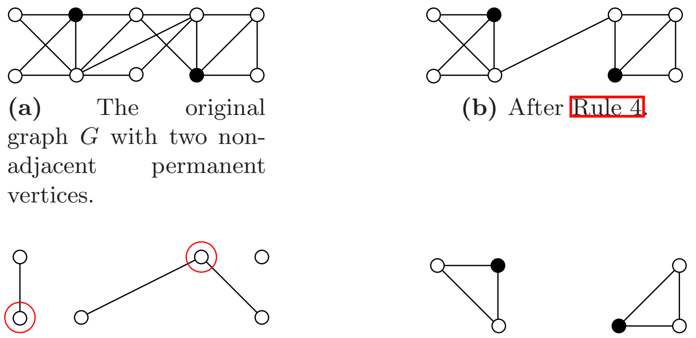
Theorem 5. *Weighted d-Cluster Vertex Deletion can be solved in running time $O(3^k + n^3)$.*

3.2 An $O(1.40^k \cdot k^{3d} + n^3)$ Time Algorithm

Now we present an algorithm that solves weighted d-Cluster Vertex Deletion via the computation of minimum weight vertex covers.[4]

The idea is to try all independent sets of size d and to solve weighted d-Cluster Vertex Deletion for the case that these vertices are not removed

[4] A *vertex cover* of a graph is a set C of graph vertices such that every graph edge has at least one endpoint in C.


(a) The original graph G with two non-adjacent permanent vertices.

(b) After Rule 4.

(c) The graph G' with a vertex cover X (marked with circles).

(d) The 2-cluster graph after the removal of X.

Fig. 3. Example of the algorithm for Weighted 2-Cluster Vertex Deletion when a size-2 independent set of vertices that cannot be deleted is given. Black vertices are permanent.

from the graph. Since in a d-cluster graph any set of d vertices from d different clusters forms an independent set, at least one of the independent sets of size d must be a set of vertices that remain in the graph.

Suppose that such an independent set D of size d is given. We call the vertices in D *permanent*. In the following, we describe how to compute the minimum weight d-CVD set of such a graph; an example is shown in Fig. 3. First, we perform the following reduction rule.

Reduction Rule 4. *Delete all vertices from the graph that are not adjacent to any vertex in D and all vertices that are adjacent to more than one vertex in D.*

The correctness of Rule 4 is obvious; an example of its application is given in Fig. 3b. For each deleted vertex v, we decrease k by $\omega(v)$. Let G be a graph with a size-d independent set of permanent vertices after application of Rule 4. All non-permanent vertices of G are adjacent to exactly one permanent vertex. To produce a cluster graph, we also have to ensure that all neighbors of a permanent vertex are adjacent, and neighbors of different permanent vertices are non-adjacent. These two attributes can be encoded into a graph G' such that a vertex cover of G' is a vertex set whose removal establishes the attributes in G. We construct the graph G' from G as follows: For any pair u, v of non-permanent vertices that is adjacent to the same permanent vertex we do the following: if u and v are adjacent, then remove the edge $\{u, v\}$; otherwise, insert the edge $\{u, v\}$. Furthermore, remove all permanent vertices. After this, we have obtained G' (for an example of this construction see Fig. 3c).

In the following lemma, we show that a vertex cover of G' is a d-CVD set of G; an example of this equivalence is shown in Figures 3c and 3d.

Lemma 3. *Let G be a graph with a size-d independent set of permanent vertices that is reduced with respect to Rule 4 and G' a graph constructed as described above. Then, a vertex set X is a vertex cover of G' iff X is a d-*CVD *set of G.*

We now bound the running time of computing a d-CVD set of a graph, once a size-d independent set that may not be deleted is given. It fundamentally relies on a fixed-parameter algorithm for weighted Vertex Cover [17].

Lemma 4. *Let $G = (V, E)$ be a graph and $D \subseteq V$ an independent set of size d. A minimum weight d-*CVD *set of G of weight at most k that does not delete any vertex $v \in D$ can be computed in $O(1.40^k + n^2)$ time.*

Combining this approach with the kernelization algorithm from Theorem 4, we achieve the following running time.

Theorem 6. *Weighted d-*Cluster Vertex Deletion *can be solved in running time $O(1.40^k \cdot k^{3d} + n^3)$.*

For the unweighted case, we can apply the current fastest algorithm for unweighted Vertex Cover by Chen et al. [5], yielding an improved running time.

Theorem 7. *Unweighted d-*Cluster Vertex Deletion *can be solved in running time $O(1.28^k \cdot k^{3d} + n^3)$.*

3.3 An $O(1.84^{k+d} + n^3)$ Time Algorithm

First, we apply the kernelization algorithm from Theorem 4 that produces a problem kernel consisting of $O(k^3)$ vertices. Next, we perform a search tree algorithm that branches on forbidden subgraphs. For a vertex in a forbidden subgraph, we have two choices: either we have to delete this vertex, or this vertex is one of the remaining vertices in the d-cluster graph. Whenever a vertex v is deleted, the combined parameter $k + d$ decreases by $\omega(v) \geq 1$. Furthermore, explicitly not deleting a vertex means that we assign a cluster to this vertex. Again, we call such a vertex *permanent.* If the permanent vertex does not have any neighbors that are marked as permanent, then we have assigned a new cluster. Hence, $k + d$ also decreases by 1.

Let k' be the sum of the weights of the vertices that may still be removed at a given search tree node and d' the number of clusters that may still be assigned. Before branching, we perform the following data reduction rule.

Reduction Rule 5. *If G contains a P_3 with two permanent vertices u, v and one non-permanent vertex w, then remove w from G and set $k' := k' - \omega(w)$.*

Clearly, if $k' < 1$, then we cannot remove any vertices and either the graph is already a d-cluster graph or this particular branch of the search tree is a dead end. Furthermore, if $d' = 0$, then we have assigned all clusters. This means that there is an independent set of d permanent vertices. By Lemma 4, we can find a d-CVD set of such a graph in $O(1.40^k + k^6) = O(1.40^k)$ time. In the following, we sketch the branching rules in case $k' \geq 1$ and $d' > 0$. After application of Reduction Rule 5, every P_3 contains at most one permanent vertex.

First, we branch on P_3's that consist of vertices that are not adjacent to permanent vertices. If such a P_3 does not exist, then we branch on P_3's that contain a permanent vertex u that is not the middle vertex of the P_3. Next, we branch on isolated clusters that do not contain permanent vertices. Finally, we show that if none of the other cases applies, then we can find a minimum weight d-CVD set of the graph by computing a minimum weight vertex cover.

Theorem 8. *Weighted d-Cluster Vertex Deletion can be solved in running time $O(1.84^{k+d} + n^3)$.*

4 Outlook

Are there any nontrivial polynomial-time approximation algorithms for Cluster Vertex Deletion (weighted and unweighted)? Moreover, the exponential upper bounds for our search-tree based algorithms should be improvable. More importantly, for the unweighted case of Cluster Editing, $O(k)$-vertex problem kernels are known [7, 12], whereas correspondingly for Cluster Vertex Deletion only an $O(k^2)$-vertex kernel is known. Also, improving the $O(k^3)$-vertex problem kernel for the weighted case would be desirable. Finally, all our results are worst-case estimates. Practical tests based on algorithm engineering seem promising.

Acknowledgments. We thank the anonymous *LATIN* referees for pointing out some inconsistencies in the submitted manuscript and for other comments that have improved the presentation of this paper.

References

[1] Abu-Khzam, F.N.: Kernelization algorithms for d-hitting set problems. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 434–445. Springer, Heidelberg (2007)

[2] Abu-Khzam, F.N., Fernau, H.: Kernels: Annotated, proper and induced. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 264–275. Springer, Heidelberg (2006)

[3] Ailon, N., Charikar, M., Newman, A.: Proofs of conjectures in Aggregating inconsistent information: Ranking and clustering. Technical Report TR-719-05, Department of Computer Science, Princeton University (2005)

[4] Böcker, S., Briesemeister, S., Bui, Q.B.A., Truß, A.: PEACE: Parameterized and exact algorithms for cluster editing. Manuscript, Lehrstuhl für Bioinformatik, Friedrich-Schiller-Universität Jena (September 2007)

[5] Chen, J., Kanj, I.A., Xia, G.: Improved parameterized upper bounds for vertex cover. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 238–249. Springer, Heidelberg (2006)

[6] Dehne, F., Langston, M.A., Luo, X., Pitre, S., Shaw, P., Zhang, Y.: The cluster editing problem: Implementations and experiments. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 13–24. Springer, Heidelberg (2006)

[7] Fellows, M.R., Langston, M.A., Rosamond, F.A., Shaw, P.: Efficient parameterized preprocessing for cluster editing. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 312–321. Springer, Heidelberg (2007)
[8] Fernau, H.: Parameterized algorithms for hitting set: The weighted case. In: Calamoneri, T., Finocchi, I., Italiano, G.F. (eds.) CIAC 2006. LNCS, vol. 3998, pp. 332–343. Springer, Heidelberg (2006)
[9] Giotis, I., Guruswami, V.: Correlation clustering with a fixed number of clusters. Theory of Computing 2, 249–266 (2006)
[10] Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Automated generation of search tree algorithms for hard graph modification problems. Algorithmica 39(4), 321–347 (2004)
[11] Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Graph-modeled data clustering: Exact algorithms for clique generation. Theory of Computing Systems 38(4), 373–392 (2005)
[12] Guo, J.: A more effective linear kernelization for cluster editing. In: Chen, B., Paterson, M., Zhang, G. (eds.) ESCAPE 2007. LNCS, vol. 4614, pp. 36–47. Springer, Heidelberg (2007)
[13] Hüffner, F., Niedermeier, R., Wernicke, S.: Techniques for practical fixed-parameter algorithms. The Computer Journal 51(1) (2008)
[14] Jansen, K., Scheffler, P., Woeginger, G.: The disjoint cliques problem. RAIRO Recherche Opérationnelle 31(1), 45–66 (1997)
[15] Lewis, J.M., Yannakakis, M.: The node-deletion problem for hereditary properties is NP-complete. Journal of Computer and System Sciences 20(2), 219–230 (1980)
[16] Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics and Its Applications, vol. 31. Oxford University Press, Oxford (2006)
[17] Niedermeier, R., Rossmanith, P.: On efficient fixed-parameter algorithms for weighted vertex cover. Journal of Algorithms 47(2), 320–331 (2003)
[18] Protti, F., da Silva, M.D., Szwarcfiter, J.L.: Applying modular decomposition to parameterized bicluster editing. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 1–12. Springer, Heidelberg (2006)
[19] Rahmann, S., Wittkop, T., Baumbach, J., Martin, M., Truß, A., Böcker, S.: Exact and heuristic algorithms for weighted cluster editing. In: Proc. 6th CSB. Computational Systems Bioinformatics, vol. 6, pp. 391–401. Imperial College Press (2007)
[20] Reed, B., Smith, K., Vetta, A.: Finding odd cycle transversals. Operations Research Letters 32(4), 299–301 (2004)
[21] Shamir, R., Sharan, R., Tsur, D.: Cluster graph modification problems. Discrete Applied Mathematics 144(1-2), 173–182 (2004)
[22] Wahlström, M.: Algorithms, Measures and Upper Bounds for Satisfiability and Related Problems. PhD thesis, Department of Computer and Information Science, Linköpings universitet (2007)
[23] van Zuylen, A., Williamson, D.P.: Deterministic algorithms for rank aggregation and other ranking and clustering problems. In: Proc. 5th WAOA, LNCS. Springer, Heidelberg (to appear, 2007)

Paths and Trails in Edge-Colored Graphs

A. Abouelaoualim[1], K. Ch. Das[1], L. Faria[2], Y. Manoussakis[1], C. Martinhon[3], and R. Saad[4]

[1] University of Paris-XI, LRI, Bât. 490, 91405 Orsay Cedex, France*
abouela@lri.fr, kinkar@mailcity.com, yannis@lri.fr
[2] Estadual Univ. of Rio de Janeiro, Dept. of Math., São Gonçalo, RJ, Brazil
luerbio@cos.ufrj.br
[3] Fluminense. Fed. Univ., Instit. of Comput., Niterói, RJ, 24210-240, Brazil**
mart@dcc.ic.uff.br
[4] 114/40 rue Charles Albanel, Gatineau (QC) J8Z 1R2, Canada
rachid_saad2003@yahoo.com

Abstract. This paper deals with the existence and search of Properly Edge-Colored paths/trails between two, not necessarily distinct, vertices s and t in an edge-colored graph from an algorithmic perspective. First we show that several versions of the $s-t$ path/trail problem have polynomial solutions including the shortest path/trail case. We give polynomial algorithms for finding a longest Properly Edge-Colored path/trail between s and t for some particular graphs and characterize edge-colored graphs without Properly Edge-Colored closed trails. Next, we prove that deciding whether there exist k pairwise vertex/edge disjoint Properly Edge-Colored $s-t$ paths/trails in a c-edge-colored graph G^c is NP-complete even for $k = 2$ and $c = \Omega(n^2)$, where n denotes the number of vertices in G^c. Moreover, we prove that these problems remain NP-complete for c-colored graphs containing no Properly Edge-Colored cycles and $c = \Omega(n)$. We obtain some approximation results for those maximization problems together with polynomial results for some particulars classes of edge-colored graphs.

Keywords: Edge colored graphs, connectivity, properly edge-colored paths, trails and cycles.

1 Introduction, Notation and Terminology

In the last few years a great number of problems have been dealt with in terms of edge-colored graphs for modeling purposes as well as for theoretical investigation [3,6,7,19]. Previous work on the subject has focused on the determination of particular Properly Edge-Colored subgraphs, such as Hamiltonian or Eulerian configurations, colored in a specified pattern [2,4,5,8,18,21,23], that is, subgraphs such that adjacent edges have different colors. Our first aim in that respect was to extend the graph-theoretic concept of connectivity to colored graphs with a view

* Sponsored by French Ministry of Education.
** Sponsored by Conselho Nacional de Desenvolvimento Científico e Tecnológico-CNPq.

to gaining some insight into our problem from Menger's Theorem in particular. In other words, we intended to define some sort of local alternating connectivity for edge-colored graphs. Local connectivity is a local parameter. For two given vertices x and y, it is the maximum number of (edge-disjoint or vertex-disjoint) paths between them. By contrast, connectivity is a global parameter defined to be the minimum number over all x, y of their local connectivity's. Difficulties arose, however, from local connectivity being not polynomially characterizable in edge-colored graphs, as can easily be seen. Thus, there can be no counterpart to Menger's Theorem as such, and even the notion of a connected component as an equivalence class does not carry over to edge-colored graphs since the concatenation of two properly edge-colored paths is not necessarily properly edge-colored. We settled then for some practical and theoretical results, herein presented, which deal with the existence of vertex-disjoint paths/trails between given vertices in c-edge-colored graphs. Most of those path/trail problems happen to be NP-complete, which thwarts all attempts at systematization.

Formally, let $I_c = \{1, 2, \dots, c\}$ be a set of given colors, $c \geq 2$. Throughout the paper, G^c will denote an edge-colored simple graph such that each edge is in some color $i \in I_c$ and no parallel edges linking the same pair of vertices occur. The vertex and edge-sets of G^c are denoted by $V(G^c)$ and $E(G^c)$, respectively. The *order* of G^c is the number n of its vertices. The *size* of G^c is the number m of its edges. For a given color i, $E^i(G^c)$ denotes the set of edges of G^c colored i. For edge-colored complete graphs, we write K_n^c instead of G^c. If H^c is a subgraph of G^c, then $N^i_{H^c}(x)$ denotes the set of vertices of H^c, linked to x by an edge colored i. The colored $i-$ *degree* of x in H^c, denoted by $d^i_{H^c}(x)$, is $|N^i_{H^c}(x)|$, *i.e.*, the *cardinality* of $N^i_{H^c}(x)$. An edge between two vertices x and y is denoted by xy, its color by $c(xy)$ and its cost (if any) by $cost(xy)$. The cost of a subgraph is the sum of its edge costs. A subgraph of G^c containing at least two edges is said to be *Properly Edge-Colored* if any two adjacent edges in this subgraph differ in color. A *Properly Edge-Colored path* does not allow vertex repetitions and any two successive edges on this path differ in color. A *Properly Edge-Colored trail* does not allow edge repetitions and any two successive edges on this trail differ in color. However, note that the edges on this trail need not form a properly edge-colored subgraph since we can have adjacent and not successive edges with the same color. The *length* of a path/trail is the number of its edges. Given two vertices s and t in G^c, we define a Properly Edge-Colored $s-t$ path/trail (or just, $s-t$ path/trail for short) to be a path/trail with end-vertices s and t. Sometimes s will be called the *source*, and t the *destination* of the path/trail. A Properly Edge-Colored path/trail is said to be *closed* if its endpoints coincide, and its first and last edges differ in color. A closed Properly Edge-Colored path (trail) is usually called a *Properly Edge-Colored cycle* (*closed trail*).

Given a digraph $D(V, A)$, we denote by $\vec{uv}$ an arc of A, where $u, v \in V$. In addition, we define $N_D^+(x) = \{y \in V : \vec{xy} \in A\}$ the *out-neighborhood* of x in D, and by $N_D^-(x) = \{y \in V : \vec{yx} \in A\}$ the *in-neighborhood* of x in D. Finally, we represent by $N_D(x) = N_D^+(x) \cup N_D^-(x)$ the *in-out-neighborhood* of $x \in V(D)$ (or just *neighborhood* for short). Also, given an induced subgraph Q of a non colored graph G, a *contraction* of Q in G consists of replacing Q by a new vertex, say z_Q, so that each vertex x in $G - Q$ is connected to z_Q by an edge, if and only if, there exists an edge xy in G for some vertex y in Q.

This paper is concerned with algorithmic issues regarding various trail/path problems between two given vertices s and t in G^c. First, we consider the $s - t$ path/trail version problem whose objective is to determine the existence or not of an arbitrary Properly Edge-Colored $s - t$ path/trail in G^c. Polynomial algorithms are established for such problems as the *Shortest Properly Edge-Colored path/trail*, the *Shortest Properly Edge-Colored cycles/closed trails* and the *Longest Properly Edge-Colored path/trail* for a particular class of instances. Actually, we show that all these results may be derived from the Szeider's Algorithm for the properly edge-colored $s-t$ paths. We also characterize edge-colored graphs without Properly Edge-Colored closed trails. Next, we deal with the *Maximum Properly Vertex Disjoint Path* and *Maximum Properly Edge Disjoint Trail* problems (respectively, MPVDP and MPEDT for short), whose objective is to find the maximum number of Properly Edge-Colored vertex-disjoint paths (respectively, edge-disjoint trails) between s and t. Although these problems can be solved in polynomial time in general non-colored graphs, most of their instances are proved to be NP-complete in the case of edge-colored graphs. In particular we prove that, given an integer $k \geq 2$, deciding whether there exist k Properly Edge-Colored vertex/edge disjoint $s - t$ paths/trails in G^c is NP-complete even for $k = 2$ and $c = \Omega(n^2)$. Moreover, for an arbitrary k we prove that these problems remain NP-complete for c-colored graphs containing no Properly Edge-Colored cycles/closed trails and $c = \Omega(n)$. We show a greedy procedure for these maximization problems, through the successive construction of Properly Edge-Colored shortest $s - t$ paths/trails. This is a straightforward generalization of the greedy procedure to maximize the number of edge or vertex disjoint paths between k pair of vertices in non-colored graphs (see [17,15] for details). Similarly, we obtain an approximation performance ratio. We finish the paper exhibiting a polynomially solvable class of instances for the related maximization problems.

The following two results will be used in this paper. The first result, initially proved by Grossman and Häggkvist [14] for 2-edge-colored graphs and generalized by Yeo [23], characterizes c-edge-colored graphs without Properly Edge-Colored cycles.

Theorem 1. *(Yeo) Let G^c be a c-edge-colored graph, $c \geq 2$, such that every vertex of G^c is incident with at least two edges colored differently. Then either G^c has a Properly Edge-Colored cycle or for some vertex v, no component of $G^c - v$ is joined to v by at least two edges in different colors.*

In terms of edge-colored graphs, Szeider's main result [20] on graphs with prescribed general transition systems may be formulated as follows:

Theorem 2. *(Szeider) Let s and t be two vertices in a c-edge-colored graph G^c, $c \geq 2$. Then, either we can find a Properly Edge-Colored $s-t$ path or else decide that such a path does not exist in G^c in linear time on the size of the graph.*

Given G^c, the main idea of the proof is based on earlier work by Edmonds (see for instance Lemma 1.1 in [18]) and amounts to reducing the Properly Edge-Colored path problem in G^c to a perfect matching problem in a non-colored graph defined appropriately. The latter graph will be called henceforth *the Edmonds-Szeider graph* and is defined as follows. Given two vertices s and t in G^c, set $W = V(G^c) \setminus \{s,t\}$. Now, for each $x \in W$, we first define a subgraph G_x with vertex- and edge-sets, respectively:

$V(G_x) = \bigcup_{i \in I_c} \{x_i, x'_i | N^i_{G^c}(x) \neq \emptyset\} \cup \{x''_a, x''_b\}$ and
$E(G_x) = \{x''_a x''_b\} \cup \left(\bigcup_{\{i \in I_c | x'_i \in V(G_x)\}} (\{x_i x'_i\} \cup (\bigcup_{j=a,b} \{x'_i x''_j\})) \right)$.

Now, the Edmonds-Szeider non-colored graph $G(V,E)$ is constructed as follows:
$G(V) = \{s,t\} \cup (\bigcup_{x \in W} V(G_x))$, and
$G(E) = \bigcup_{i \in I_c} \left(\{sx_i | sx \in E^i(G^c)\} \cup \{x_i t | xt \in E^i(G^c)\} \cup \{x_i y_i | xy \in E^i(G^c)\} \right) \cup \left(\bigcup_{x \in W} E(G_x) \right)$.

The interesting point in the construction is that, given a particular (trivial) perfect matching M in $G \setminus \{s,t\}$, a Properly Edge-Colored $s-t$ path exists in G^c if and only if there exists an augmenting path P relative to M between s and t in G. Recall that a path P is augmenting with respect to a given matching M if, for any pair of adjacent edges in P, exactly one of them is in M, with the further condition that the first and last edges of P are not in M. Since augmenting paths in G can be found in $O(|E(G)|)$ linear time (see [22], p.122), the same execution time holds for finding Properly Edge-Colored paths in G^c as well, since $O(|E(G)|) = O(|E(G^c)|)$.

2 The $s-t$ Path/Trail Problem

Given two, not necessarily distinct, vertices s and t in G^c, the main question of this section is to give polynomial algorithms for finding (if any) a Properly Edge-Colored $s-t$ path or trail in G^c. The Properly Edge-Colored $s-t$ path problem was first solved by Edmonds for two colors (see Lemma 1.1 in [18]) and then extended by Szeider [20] to include any number of colors. Here we deal with variations of the Properly Edge-Colored trail/path problem, *i.e.*, the problem of finding $s-t$ trails, closed trails, the shortest $s-t$ path/trail and the longest $s-t$ path (trail) in graphs with no Properly Edge-Colored cycles (closed trails).

2.1 Finding a Properly Edge-Colored Trail Between Two Vertices

This section is devoted to the Properly Edge-Colored $s-t$ trail problem. Among other results, we prove that the $s-t$ trail problem reduces to the $s-t$ path problem over a new c-edge-colored graph. As the latter problem has been proved polynomial [20], it follows that our problem is polynomial as well.

Therefore, given G^c and an integer $p \geq 2$, let us consider an edge-colored graph denoted by $p - H^c$ (henceforth called *the trail-path graph*) obtained from G^c as follows. Replace each vertex x of G^c by p new vertices $x_1,\ x_2, \ldots, x_p$.

Furthermore for any edge xy of G^c colored, say j, add two new vertices v_{xy} and u_{xy}, add the edges $x_iv_{xy}, u_{xy}y_i$ for $i = 1, 2, \ldots, p$ all of them colored j, and finally add the edge $v_{xy}u_{xy}$ with color $j' \in I_c$ and $j' \neq j$. For convenience of notation, the edge-colored subgraph of $p - H^c$ induced by the vertices x_i, v_{xy}, u_{xy}, y_i and associated with the edge xy of G^c will be denoted throughout by H^c_{xy}. Moreover for $p = 2$, we write H^c instead of $p - H^c$.

Therefore, we have the following relation between G^c and trail-path graph H^c:

Lemma 1. *(Fundamental Lemma) Consider two vertices s and t in G^c and its associated trail-path graph $p - H^c$ for $p = \lfloor (n-1)/2 \rfloor$. Then, there exists a Properly Edge-Colored $s - t$ trail in G^c, if and only if, there exists a Properly Edge-Colored $s_1 - t_1$ path in $p - H^c$.*

Proof: Assume first that there exists a Properly Edge-Colored trail, say, $T = e_1e_2 \cdots e_k$ between s and t in G^c, where e_i are the edges of T and s is the left endpoint of e_1 while t is the right endpoint of e_k. Note that, no vertices may be visited more than p times in G^c. To see that, consider a Properly Edge-Colored $s - t$ trail T passing by x in G^c with the maximum possible number of cycles through x of length 3.

Thus, to construct a Properly Edge-Colored path P between s_1 and t_1 in $p - H^c$ we consider all edges $e_w = xy$ of T and their associated subgraphs $H^c_{e_w}$ in $p - H^c$. Now, to avoid vertex repetitions in P, we conveniently replace each edge e_w by one of the segments $x_iv_{xy}, v_{xy}u_{xy}, u_{xy}y_j$ (for $i, j \in \{1, \ldots, p\}$) in $H^c_{e_w}$.

Conversely, any Properly Edge-Colored $s_1 - t_1$ path in $p - H^c$ uses precisely one of the sub-paths $x_iv_{xy}, v_{xy}u_{xy}, u_{xy}y_j$ (for some pair $i, j \in \{1, \ldots, p\}$) in each subgraph H^c_{xy} of $p - H^c$. Change one of these sub-paths by xy in G^c. Now it is easy to see that a Properly Edge-Colored $s_1 - t_1$ path in $p - H^c$ will correspond to a Properly Edge-Colored $s - t$ trail T in G^c where no vertices are visited more than p times on T. $\square$

The following result is a straightforward consequence of the fundamental lemma and its proof il left to the reader.

Theorem 3. *Consider two distinct vertices s and t in a c-edge-colored graph G^c. Then we can either find a Properly Edge-Colored $s - t$ trail or else decide correctly that such a trail does not exist in G^c in linear time on the size of G^c.*

We conclude the section with some results on closed trails in edge-colored graphs. In particular, we intend to characterize edge-colored graphs without Properly Edge-Colored closed trails. Recall that the problem of checking whether G^c contains no Properly Edge-Colored cycle was solved by Yeo [23] for an arbitrary number of colors (see Theorem 1 above). The author uses the concept of a cut-vertex separating colors, *i.e.*, a vertex x such that all the edges between each component of $G^c - x$ and x are colored alike. Analogously, let e be a bridge of G^c. We say that e separates colors, if no component of $G^c - e$ is joined to e by at least two edges of different colors. Thus, by introducing the concept of bridges separating colors and using the Fundamental Lemma, we obtain the following:

Theorem 4. *Let G^c be c-edge-colored graph, such that every vertex of G^c is incident with at least two edges colored differently. Then either G^c has a bridge separating colors or G^c has a Properly Edge-Colored closed trail.*

As for the algorithmic aspects of this problem, it suffices to delete all bridges separating colors (if any) and all vertices adjacent to edges of the same color in G^c to test for the existence of a Properly Edge-Colored closed trail in polynomial time. Notice that all such edges and vertices may be deleted without any Properly Edge-Colored closed trail being destroyed. Thus, if the resulting graph is non-empty, it will contain a Properly Edge-Colored closed trail.

2.2 Shortest Properly Edge-Colored Paths/Trails

In this section we consider shortest Properly Edge-Colored $s-t$ paths and trails. By associating appropriate costs with the edges of the Edmonds-Szeider non-colored graph $G(V, E)$, we first show how to find, if any, a shortest Properly Edge-Colored path between (not necessarily distinct) s and t in G^c. As a consequence, this procedure may be used to find a shortest Properly Edge-Colored trail between s and t in G^c. For the shortest Properly Edge-Colored path problem, let us consider the following algorithm:

```
Algorithm 1: SHORTEST PROPERLY EDGE-COLORED PATH
Input: A $c$-edge-colored graph $G^c$, vertices $s, t \in V(G^c)$.
Output: If any, a shortest Properly Edge-Colored $s-t$ path $P$ in $G^c$.
Begin
  1. Define: $W = V(G^c) \setminus \{s, t\}$;
  2. For every $x \in W$, construct $G_x$ as defined in Section 1;
  3. Construct the Edmonds-Szeider graph $G$ associated with $G^c$;
  4. Define: $E' = \cup_{x \in W} E(G_x)$;
  5. For every $pq \in E(G) \setminus E'$ do $cost(pq) \leftarrow 1$;
  6. For every $pq \in E'$ do $cost(pq) \leftarrow 0$;
  7. Find a minimum weighted perfect matching $M$ in $G$ (if any);
  8. Use $M$ to build a path $P$ in $G^c$ and return $P$, or say that $P$ does not exist;
End.
```

Henceforth, we define the weighted graph G above as the *weighted Edmonds-Szeider* graph. Intuitively, the idea in Algorithm 1 is to penalize all edges of G associated with edges in the original graph G^c. To obtain P from M in Step 8, we contract all subgraphs G_x of G to a single vertex x. The remaining edges of M in this resulting non-colored graph, say G', will define a $s-t$ path in G' which is associated to a Properly Edge-Colored $s-t$ path in G^c. Notice that all the vertices not in this $s-t$ path in G^c are isolated, *i.e.*, we cannot have Properly Edge-Colored cycles containing these vertices (otherwise, M would not be a minimum weighted perfect matching in G).

In addition, observe that the overall complexity of Algorithm 1 is dominated by the complexity of a minimum weighted perfect matching (Step 8). Several matching algorithms exist in the literature. See Gerards [13] for a good reference on general matchings. Formally, we have established the following result:

Theorem 5. *Algorithm* 1 *always find, if any, a shortest Properly Edge-Colored $s - t$ path in G^c.*

Now, to solve the shortest trail problem, it suffices to use the above algorithm as follows: Given s and t in G^c, construct the trail-path graph H^c associated with G^c. Notice, in this case, that no vertices are visited more than twice in a shortest Properly Edge-Colored $s - t$. Next, we find a shortest Properly Edge-Colored $s_1 - t_1$ path P in H^c by Algorithm 1. Then, by using path P of H^c, come back and construct a shortest Properly Edge-Colored $s - t$ trail T in G^c. Remember that each subgraph H^c_{xy} of H^c is associated with some edge xy of G^c. Furthermore, observe that a Properly Edge-Colored path between x_i and y_j in H^c_{xy} contains exactly 3 edges. Thus, in order to obtain T in G^c from P in H^c, it suffices to replace each $x_i - x_j$ path of P in H^c_{xy} with the corresponding edge xy in G^c. Therefore, we obtain a shortest $s - t$ trail with $cost(T) = cost(P)/3$. The correctness of this algorithm is guaranteed by the Lemma 1 and Theorem 5.

As a final comment, we can easily adapt these ideas to find a shortest Properly Edge-Colored cycle (closed trail) in G^c, provided that one exists. See [1] for further details.

2.3 The Longest Properly Edge-Colored $s - t$ Path/Trail Problem

The problem of finding the longest Properly Edge-Colored $s-t$ path in arbitrary c-edge-colored graphs is obviously NP-complete since it generalizes the Hamiltonian Path problem in non-directed graphs. Based on the maximum weighted perfect matching problem (see [10,12] for details), we propose a polynomial time procedure for finding the longest Properly Edge-Colored $s - t$ path (trail) in graphs with no Properly Edge-Colored cycles (closed trails).

Theorem 6. *Assume that G^c has no Properly Edge-Colored cycles. Then, we can always find in polynomial time a longest Properly Edge-Colored $s - t$ path or else decide that such a path does not exist in G^c.*

Proof: Construct the weighted Edmonds-Szeder graph G and compute the maximum weighted perfect matching M in G (if any), otherwise, we would not have a Properly Edge-Colored path between s and t. Now, to determine the associated $s - t$ path P in G^c, we construct a new non-colored graph G' by just contracting subgraphs G_x to a single vertex x. It is easy to see that G' will contains a (non-colored) $s - t$ path, cycles and isolated vertices, associated respectively to a Properly Edge-Colored $s - t$ path, Properly Edge-Colored cycles and isolated vertices in G^c . However, by hypothesis G^c does not contains Properly Edge-Colored cycles. Therefore, all edges with unitary costs in M will be associated with edges of P and vice-versa. Then, since M is a maximum weighted perfect matching in G, the associated path P will define a longest Properly Edge-Colored $s - t$ path in G^c. □

Observe in the longest $s - t$ path problem above that every vertex is visited at most once and we do not have Properly Edge-Colored cycles in G^c. However, to find a longest Properly Edge-Colored $s-t$ trail we do not know how many times a given vertex $x \in V(G^c) \setminus \{s, t\}$ will be visited. Nonetheless, we can construct the trail-path graph $p-H^c$ for $p = \lfloor (n-1)/2 \rfloor$ (associated to G^c) and then compute the longest Properly Edge-Colored $s_1 - t_1$ trail in $p - H^c$. Formally, we have:

Theorem 7. *Let G^c be a c-edge-colored graph with no Properly Edge-Colored closed trails and two vertices $s, t \in V(G^c)$. Then, we can always find in polynomial time, a longest Properly Edge-Colored $s - t$ trail in G^c, provided that one exists.*

Notice in the Theorem 7 that, since G^c does not contain Properly Edge-Colored closed trails, the associated trail-path graph $p - H^c$ will not contains Properly Edge-Colored closed cycles and the Theorem 6 may be applied after changing G^c by $p - H^c$. The details are left to the reader.

3 The k-Path/Trail Problem

Let k-PVDP and k-PEDT be the decision versions associated respectively with Maximum Properly Vertex Disjoint Path (MPVDP) and the Maximum Properly Edge Disjoint Trail (MPEDT) problems, *i.e.*, given a c-edge-colored graph G^c, two vertices $s, t \in V(G^c)$ and an integer $k \geq 2$, we want to determine if G^c contains at least k Properly Edge-Colored vertex disjoint paths (respectively, edge disjoint trails) between s and t. We can show that both k-PVDP and k-PEDT are NP-complete even for $k = 2$ and $c = \Omega(n^2)$. In particular, in graphs with no properly colored cycles (respectively, closed trails) and $c = \Omega(n)$ colors, we can prove that k-PVDP (respectively, k-PEDT) is NP-complete for an arbitrary $k \geq 2$. Next, at the end of the section, we establish some approximation results and polynomial algorithms for special cases of both MPVDP and MPEDT problems.

3.1 NP-Complete Results for General Graphs and Graphs with no Properly Edge-Colored Cycles (Closed Trails)

The Theorem 8 stated below guarantees that both 2-PVDP and 2-PEDT are NP-complete for 2-edge-colored graphs. In view of that theorem, let us first describe *two* auxiliary results concerning directed cycles and closed trails in (non-colored) digraphs. Let u and v be two fixed vertices in a digraph D. Deciding if D contains or not a directed cycle containing both u and v (represented here by DC, for short) is known to be NP-complete [9]. We can prove that deciding if D contains or not a directed closed trail containing both u and v is also NP-complete (see [1] for details). We denote this last problem by *Directed Closed Trail* (DCT). Now, using both DC and DCT problems we prove the following result:

Theorem 8. *Both the* 2-PVDP *and* 2-PEDT *problems are NP-Complete for 2-edge-colored graphs.*

Proof: We can easily check in polynomial time that both 2-PVDP and 2-PEDT problems are in NP. To show they are NP-hard, we propose polynomial time reductions from the DC and DCT problems, respectively. Consider two vertices u and v in a digraph D . We show how to construct in polynomial time, a 2-edge-colored graph G^c and a pair of vertices $a, b \in V(G^c)$, such that there is a direct cycle (closed trail) containing u and v in D, if and only if, there are 2 vertex-disjoint $a - b$ paths (2-edge-disjoint $a - b$ trails) in G^c. Let us first define from D another digraph D' by replacing u by two new vertices s_1, s_2 with $N^-_{D'}(s_2) = N^-_D(u)$, $N^+_{D'}(s_1) = N^+_D(u)$. Similarly replace t_1, t_2 and $N^-_{D'}(t_2) = N^-_D(v)$, $N^+_{D'}(t_1) = N^+_D(v)$. Finally, add the arcs (s_2, s_1) and (t_2, t_1) in D'. Now in order to define G^c replace each arc $\vec{xy}$ of D' by a colored segment xzy where z is a new vertex and edges xz, zy are on colors red and blue, respectively. Finally, we define $z = a$ for z between s_1 and s_2, and $z = b$ for z between t_1 and t_2. Observe now that there is a directed cycle (directed closed trail) containing u and v in D, if and only if, there are two vertex-disjoint Properly Edge-Colored $a - b$ paths (Properly Edge-Colored edge-disjoint $a - b$ trails) in G^c. $\square$

Intuitively speaking, note that both 2-PVDP and 2-PEDT problems became easier when 3 colors or more are considered (an extreme case is when all edges of G^c have different colors). As a consequence of that, an interesting question is to study the NP-completeness of these problems for graphs with many colors. Thus, we have established the following result (proved in [1]):

Theorem 9. *Both* 2-PVDP *and* 2-PEDT *problems remain NP-complete even for arbitrary graphs with* $\Omega(n^2)$ *colors.*

In addition, we can prove that k-PVDP (respectively, k-PEDT) for $k \geq 2$, remains NP-complete even for 2-edge-colored graphs with no Properly Edge-Colored cycles (respectively, closed trails). Recall that, as discussed in previous sections, the existence or not of Properly Edge-Colored cycles or closed trails in edge-colored graphs may be checked in polynomial time. The next result was initially based on some ideas similar to those used by Karp [16] for the Discrete Multicommodity Flow problem for non-oriented (and non-colored) graphs (usually known in the literature as the Vertex Disjoint Path problem).

Theorem 10. *Let* G^c *be a* 2*-edge-colored graph without Properly Edge-Colored cycles (respectively, closed trails). Given two vertices* s, t *of* $V(G^c)$ *and an integer* $k \geq 0$*, to decide if there exist* k *Properly Edge-Colored vertex-disjoint* $s - t$ *paths (respectively, edge-disjoint* $s - t$ *trails) in* G^c *is NP-complete.*

Similarly to Theorem 9, we can prove that both k-PVDP and k-PEDT problems remain NP-complete even for c-edge-colored graph without Properly Edge-Colored cycles (respectively, closed trails) and $\Omega(n)$ colors (see [1] for details).

3.2 Some Approximation and Polynomial Results

In this section, we describe greedy procedures for both MPEDT and MPVDP, based in the determination of shortest Properly Edge-Colored $s - t$ trails (respectively $s-t$ paths). Their performance ratio are based on the same arguments used for the Edge/Vertex Disjoint Path problem between k pairs of vertices in non-directed graphs [15,17] and are omitted here. We conclude this section by presenting some polynomial results for some particular instances of both problems.

At each steep of the greedy procedure for the MPEDT problem, we find a shortest properly edge-colored $s-t$ trail T in G^c . We then delete all edges in this trail and repeat the process until no $s-t$ trails are found. We denote this procedure by GREEDY-ED, for short.

Now consider the following definitions: we say that a trail T_1 *hits* a trail T_2, or equivalently, that T_2 is *hitted* by T_1, if and only if T_1 and T_2 share a common edge. If Γ denotes the set of all Properly Edge-Colored $s-t$ trails, we define $I \subseteq \Gamma$ as the subset of trails obtained by the greedy procedure and $J \subseteq \Gamma$ the subset of trails associated to an optimal solution. Then, we have the following:

Theorem 11. *Algorithm* GREEDY-ED *has performance ratio equal to* $O(\frac{1}{\sqrt{m}})$.

In order to give some idea about the determination of the value $\sqrt{m}$ above, suppose that a trail T_1 hits k trails of $J \setminus I_1$ at the first step of the Greedy-ED. Note that, one edge of T_1 can hit at most one other trail of J and therefore T_1 has length at least k. Since T_1 is a shortest $s-t$ trail, all other trails in $J \setminus I_1$ also have at least k edges. Therefore, $k^2 \leq m$, so $k = \sqrt{m}$. This idea may be inductively applied for the remaining steps of the greedy procedure.

Note that we can easily modify the GREEDY-ED to solve the MPVDP. In this case, after the determination of a shortest $s-t$ path P in G^c, it suffices to remove all vertices belonging to $P \setminus \{s,t\}$. We repeat this process until no Properly Edge-Colored $s-t$ paths are found. We call this new procedure GREEDY-VD. Using the same ideas as described in Theorem 11, we can prove that GREEDY-VD has performance ratio equal to $O(1/\sqrt{n})$ for the MPVDP problem.

We end this section with some polynomial results for some specific families of graphs. To begin with, we introduce the following definition: given an edge-colored graph G^c, we say that a cycle $C_x = xa_1 \cdots a_j x$ with $x \neq a_i$ for $i = 1, \ldots, j$ is an *almost properly colored cycle (closed trail) through x in G^c*, if and only if $c(xa_1) = c(xa_j)$ and both paths (respectively trails) $x-a_1$ and $x-a_j$ are properly colored. If $c(xa_1) \neq c(xa_j)$, then C_x define a *properly edge-colored cycle (closed trail) through x*. In the sequel, we show how to solve the MPVDP (respectively, MPEDT) problem in polynomial time for graphs containing no properly or almost properly colored cycles (respectively, closed trails) through s or t. Notice that to check if an edge-colored graph G^c contains or not a properly or an almost properly cycle (closed trail) through x, it suffices to define an auxiliary graph G^c_x obtained from G^c by replacing x with two new vertices x_a and x_b and setting $N_{G^c_x}(x_a) = N_{G^c}(x)$ and $N_{G^c_x}(x_b) = N_{G^c}(x)$. Now, using Theorem 2 (respectively, Theorem 3) we compute, if any, a Properly Edge-Colored $x_a - x_b$ path (trail) in G^c_x. Clearly if no such $x_a - x_b$ path (trail) exists in G^c_x, then there exists no properly or almost properly edge-colored cycle (closed trail) through x in G^c.

Initially, given an integer $k \geq 1$, we consider the k-PVDP (decision version associated with the MPVDP) restricted to graphs with no (almost) Properly Edge-Colored cycles through s or t.

Theorem 12. *Consider an integer $k \geq 1$ and a c-edge-colored graph G^c with no (almost) Properly Edge-Colored cycles through s or t. Then, the k-*PVDP *problem may be solved in polynomial time.*

Proof: Suppose, *w.l.o.g.*, that we do not have (almost) properly colored cycles through vertex s in G^c. Notice for instance, that (almost) properly colored closed trails through s with vertex repetitions are allowed.

If $k = 1$, the problem is polynomially solved by Edmonds-Szeider's Algorithm. For $k \geq 2$, we construct an auxiliary non-colored graph G' in the following way. As discussed in Section 1, we first define $W = V(G^c) \setminus \{s, t\}$, and non-colored graphs G_x for every $x \in W$ (see the first part in the definition of the Edmonds-Szeider's graph). Now, define $S_k = \{s_1, \ldots, s_k\}$, $T_k = \{t_1, \ldots, t_k\}$ and proceed as follows:
$V(G') = S_k \cup T_k \cup (\bigcup_{x \in W} V(G_x))$, and
$E(G') = \bigcup_{j=1,\ldots,k} \left(\bigcup_{i \in I_c} \left(\{s_j x_i | sx \in E^i(G^c)\} \cup \{x_i t_j | xt \in E^i(G^c)\}\right)\right) \cup \left(\bigcup_{i \in I_c} \{x_i y_i | xy \in E^i(G^c)\}\right) \cup \left(\bigcup_{x \in W} E(G_x)\right)$.

Now, find a perfect matching M (if any) in G' and contract each subgraph G_x into a single vertex x. Let G'' this new graph. Observe that all paths in G'' are defined by edges belonging to $M \cap E(G'')$. In addition, we cannot have a path between s_i and s_j in G'' (otherwise, we would have a (an almost) properly cycle though s in G^c). In this way, all paths in G'' begins at vertex $s_i \in S_k$ and finish at some vertex $t_j \in T_k$. Finally, we construct a non-colored graph G''' by contracting S_k and T_k respectively to vertices s and t. In this way, note that non-colored $s - t$ paths in G''' are associated to Properly Edge-Colored $s - t$ paths in G^c and vice-versa. Therefore, if the construction of a perfect matching M in G' is possible (what is done in polynomial time), we obtain k properly edge-colored $s - t$ paths in G^c. $\square$

Since the perfect matching problem is solved in polynomial time, we can easily construct a polynomial time procedure for the MPVDP in graphs with no (almost) properly colored cycles through s or t. To do that, it suffices to repeat all the steps described in Theorem 12 for $k = 1, \ldots, n-2$ until some non-colored graph G' containing no perfect matchings is found.

Similarly, using the Fundamental Lemma and Theorem 12, we can apply the same ideas to solve the MPEDT in graphs with no (almost) properly colored closed trails through s or t. The details are left to the reader.

4 Conclusions and an Open Problem

In this work, we have considered path and trail problems in edge-colored graphs. We generalized some previous results concerning Properly Edge-Colored paths and cycles in colored graphs, which allowed us to devise efficient algorithms for finding them. On the negative side, we proved that finding k properly vertex/edge disjoint $s - t$ paths/trails is NP-complete even for $k = 2$ and $c = \Omega(n^2)$. In addition, we showed that both problems remain NP-complete for arbitrary $k \geq 2$ in graphs with no Properly Edge-Colored cycles (closed trails) and $c = \Omega(n)$, which led us to investigate approximation. Finally, we showed that both MPVDP (MPEDT) are solved in polynomial time when restricted to graphs with no (almost) Properly Edge-Colored cycles (closed trails) through s or t. However, the following question is left open. Is the problem below NP-complete?

Input: Given a 2-edge-colored graph $G^c(V, E)$ with no Properly Edge-Colored cycles, two vertices $s, t \in V$ and a fixed constant $k \geq 2$.
Question: Does G contains k Properly Edge-Colored vertex/edge disjoint paths between s and t?

As a future direction, another important question is to consider the approximation performance ratio (as well as inapproximability results) for both MPVDP and MPEDT for general colored graphs or for graphs with no Properly Edge-Colored cycles (closed trails).

References

1. Abouelaoualim, A., Das, C.K., Faria, L., Manoussakis, Y., Martinhon, C., Saad, R.: Paths and trail in edge-colored graphs (Extended Version) Technical Report, RT-3/07 (2007), http://www.ic.uff.br/PosGraduacao/lista_relatoriosTecnicos.php?ano=2007
2. Bang-Jensen, J., Gutin, G.: Alternating cycles and paths in edge-coloured multigraphs: a survey. Discrete Mathematics 165/166, 39–60 (1997)
3. Bang-Jensen, J., Gutin, G.: Digraphs: theory, algorithms and applications. Springer, Heidelberg (2002)
4. Benkouar, A., Manoussakis, Y., Paschos, V.T., Saad, R.: On the complexity of some hamiltonian and eurelian problems in edge-colored complete graphs. RAIRO - Operations Research 30, 417–438 (1996)
5. Benkouar, A., Manoussakis, Y., Saad, R.: The number of 2-edge-colored complete graphs with unique hamiltonian properly edge-colored cycle. Disc. Mathematics 263(1-3), 1–10 (2003)
6. Dorniger, D.: On permutations of chromosomes. Contributions of General Algebra 5, 95–103 (1987)
7. Dorniger, D.: Hamiltonian circuits determining the order of chromosomes. Disc. Appl. Math 50, 159–168 (1994)
8. Feng, J., Giesen, H.-E., Guo, Y., Gutin, G., Jensen, T., Rafiey, A.: Characterization of edge-colored complete graphs with properly colored Hamilton paths. Journal of Graph Theory 53(4), 333–346 (2006)
9. Fortune, S., Hopcroft, J., Wyllie, J.: The directed subgraph homeomorphism problem. Theoretical Computer Science 10, 111–121 (1980)
10. Gabow, H.N.: Data structures for weighted matching and nearest common ancestors with linking. In: Proc. SODA 1990, pp. 434–443 (1990)
11. Gabow, H.N., Maheshwari, S.N., Osterweil, L.: On two problems in the generation of program test paths. IEEE Transactions on Software Engineering 2(3) (1976)
12. Gabow, H.N., Tarjan, R.E.: Faster scaling algorithms for general graph matching problems. Journal of ACM 38, 815–853 (1991)
13. Gerards, A.M.H.: Matching, Network Models. In: Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser, G.L. (eds.) Handbooks in Operations Research and Management Science, vol. 7, North Holland, Amsterdam (1995)

14. Grossman, J., Häggkvist, R.: Properly edge-colored cycles in edge-partioned graphs. Journal of Combinatorial Theory, Series B 34, 77–81 (1983)
15. Guruswami, V., Khanna, S., Rajaraman, R., Shepherd, B., Yannakakis, M.: Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. In: Proc. of the 31st Annual ACM Symposium on Theory of Computing, pp. 19–28 (1999)
16. Karp, R.M.: On the Computational Complexity of Combinatorial Problems. Networks 5, 45–68 (1975)
17. Kleinberg, J.M.: Approximation algorithms for disjoint path problems, PhD. Thesis, MIT (1996)
18. Manoussakis, Y.: Properly edge-colored paths in edge-colored complete graphs. Discrete Applied Mathematics 56, 297–309 (1995)
19. Pevzner, P.A.: Computational molecular biology: an algorithmic approach. MIT Press, Cambridge (2000)
20. Szeider, S.: Finding paths in graphs avoiding forbidden transitions. Discrete Applied Mathematics 126(2-3), 239–251 (2003)
21. Saad, R.: Finding a longest properly edge-colered hamiltonian cycle in an edge colored complete graph is not hard. Combinatorics, Probability and Computing 5, 297–306 (1996)
22. Tarjan, R.E.: Data structures and network algorithms, p. 44. SIAM - Philadelphia (1983)
23. Yeo, A.: A note on alternating cycles in edge-coloured Graphs. Journal of Combinatorial Theory, Series B 69, 222–225 (1997)

Efficient Approximation Algorithms for Shortest Cycles in Undirected Graphs

Andrzej Lingas and Eva-Marta Lundell

Department of Computer Science, Lund University, 221 00 Lund, Sweden
{Andrzej.Lingas, Eva-Marta.Lundell}@cs.lth.se

Abstract. We describe a simple combinatorial approximation algorithm for finding a shortest (simple) cycle in an undirected graph. For an undirected graph G of unknown girth k, our algorithm returns with high probability a cycle of length at most $2k$ for even k and $2k+2$ for odd k, in time $\mathcal{O}(n^{\frac{3}{2}}\sqrt{\log n})$. Thus, in general, it yields a $2\frac{2}{3}$ approximation. We study also the problem of finding a simple cycle of minimum total weight in an undirected graph with nonnegative edge weights. We present a simple combinatorial 2-approximation algorithm for a minimum weight (simple) cycle in an undirected graph with nonnegative integer edge weights in the range $\{1, 2, ..., M\}$. This algorithm runs in time $\mathcal{O}(n^2 \log n \log M)$.

1 Introduction

The *girth* of an undirected graph G is defined as the length of a shortest cycle in G, or infinity if G is acyclic. The notion of girth is closely connected with other, important characteristics of a graph, such as chromatic number, connectivity, genus, and many more (cf. [13,14]).

The problems of determining the girth and finding a shortest simple cycle in an undirected graph, or more generally, a minimum weight simple cycle in an undirected graph with non-negative edge weights belong to the basic algorithmic graph problems [2,3,9,12,17,26,27,28,29]. For instance, they occur in the construction of minimum cycle bases of graphs [19,20] and the maximum cycle packing problems [8,21,22].

In a seminal paper from 1978, Itai and Rodeh [17] showed in particular that a shortest cycle in an undirected graph can be found in the same asymptotic time as Boolean matrix multiplication. They also observed that the direct algorithm for a shortest cycle consisting in running breadth first search from each vertex has an $O(nm)$ time behavior, where n and m are the number of vertices and edges in the input graph respectively (see p. 413 and 415 in [17]).

Observe that by the upper bounds on the number of edges of a graph G in terms of the girth of G established by Bollobas in [7] [1], the aforementioned direct algorithm detects a shortest (simple) cycle of G in time $\mathcal{O}(n^{2+\frac{1}{\lfloor (g-1)/2 \rfloor}})$, where

[1] If a graph G on n vertices has girth $\geq 2k+1$ then it has at most $n^{1+\frac{1}{k}}$ edges.

g is the girth of G. Already for graphs of girth greater than four this simple combinatorial method is substantially sub-cubic and for graphs of girth greater than six it is faster than the aforementioned algorithm for a shortest cycle due to Itai and Rodeh based on fast matrix multiplication and running in time $\mathcal{O}(n^{\omega})$, where $\omega < 2.376$ is the exponent of square matrix multiplication.

Itai and Rodeh also modified the direct algorithm to provide a simple quadratic-time heuristic which outputs a cycle whose length is within the minimum plus one in [17].

Surprisingly enough, in spite of large number of deep and interesting papers on related cycle problems in undirected, unweighted graphs [2,3,18,28,14,23,28,29], the aforementioned upper time bounds from 1978 could not be subsumed. Even more surprisingly, no substantially sub-quadratic-time constant-approximation heuristics for a shortest cycle in an undirected graph seems to be known in the literature.

For non-negatively edge weighted graphs on n vertices, the problem of finding a simple cycle of minimum total edge weight is closely related to the all-pairs shortest path problem, where the length of a path means its total edge weight [11] Since the fastest known algorithm for the all-pairs shortest path problem for edge weighted graphs has time complexity $\mathcal{O}(n^3 \log^3 \log n / \log^2 n)$ [9], the aforementioned relationship does not lead to a substantially sub-cubic algorithm for finding a shortest cycle at present.

The situation changes if one can accept a reasonable approximation of a minimum weight cycle. There are several known approximation algorithms for the all-pairs problem [31]. For instance, Kavitha et al. use the following ones to approximate a shortest cycle in order to construct an approximation of minimum cycle basis of a graph with nonnegative edge weights in [19,20]:

- the all-pairs 2-stretch algorithm of Cohen et al. running in time[2] $\tilde{O}(n^{3/2}m^{1/2})$ [10], where m is the number of edges, or
- the all-pairs $(1+\epsilon)$-stretch algorithm of Zwick running in time $\tilde{O}(mn^{\omega}/\epsilon \log(W/\epsilon))$ [30], where W is the largest edge weight, or
- the structure of Thorup et al. for $(2k-1)$-approximate shortest path queries constructible in expected time $\tilde{O}(kmn^{1/k})$ [25].

Unfortunately, each of the approaches requires either substantially super--quadratic time or yields an approximation factor substantially larger than 2. Almost quadratic time is known to be sufficient to achieve solely stretch 3 [10], or more recently, stretch between 3 and 2 [4,5]. For the vast literature on the very related problem of all-pairs shortest paths in graphs with edge weights and their approximation the reader is referred to the excellent survey by Zwick [31], and especially for the more recent developments to the introduction sections of the recent papers by T. Chan (see e.g. [9]).

For the related problems of finding minimum weight (simple) cycles composed of k edges (for a fixed k) in a graph with nonnegative edge weights and those

[2] The notation $\widetilde{O}(f(n))$ stands for $O(f(n) \log^c n)$ for some positive constant c.

of finding minimum weight (simple) cycles in undirected graphs with vertex weights or Euclidean edge weights, which both can be regarded as a subclass of edge weighted undirected graphs, the reader is referred to [9,12,26,27].

Our contributions: Our main result is a simple combinatorial approximation algorithm for finding a shortest (simple) cycle in an undirected graph (Section 2). It combines the idea of "halting BFS" from the simple heuristic of Itai and Rodeh with the idea of small dominating sets for large degree vertices used in the algorithms for shortest paths with additive approximation factors from [1,15]. For an undirected graph G of unknown girth k, our algorithm returns with high probability a cycle of length at most $2k$ for even k and $2k+2$ for odd k, in time $\mathcal{O}(n^{\frac{3}{2}}\sqrt{\log n})$. Thus, in general, it yields a $2\frac{2}{3}$ approximation.

Our approximation algorithm for a shortest cycle leads also to more time-efficient implementations of the best heuristics for maximum packing of edge disjoint cycles as well as vertex disjoint cycles in an undirected graph, due to Krivelevich et al. [21] and Salavatipour et al. [22], respectively.

We also study the problem of finding a shortest (simple) cycle, i.e., a simple cycle of minimum total weight, in an undirected graph with nonnegative integer edge weights (Section 3). We apply a a similar "halting" approach to a specially derived variant of Dijkstra's single-source shortest paths algorithm [11] in order to derive a 2-approximation algorithm for this problem. For an undirected graph with nonnegative integer edge weights in the range $\{1, 2, ..., M\}$, our 2-approximation algorithm runs in time $\mathcal{O}(n^2 \log n \log M)$.

2 The Unweighted Case

For an undirected graph G, we shall denote its girth, i.e., the minimum number of edges in a simple cycle in G, by $girth(G)$.

Our approximation algorithm for finding a short cycle in an unweighted graph G combines the idea of the algorithm of Itai and Rodeh that returns a cycle of length at most $girth(G)+1$ [17] with the ideas behind the algorithms for shortest paths problems with additive approximation factors from [1].

The algorithm of Itai and Rodeh is very simple. It conducts a breadth-first search from each vertex v of G and halts when the first non-tree edge is detected (we shall term such a partial BFS as *halting BFS*). It follows that the running time is at most quadratic. If the first non-tree edge induces a simple cycle of even length passing through v , then this cycle is easily seen to be a shortest even cycle passing through v in G. Therefore, if $girth(G)$ is even, the algorithm of Itai and Rodeh detects a shortest cycle. Also, if the first non-tree edge induces a simple cycle of odd length passing through v then the cycle has to be the shortest one passing through v in G. Thus, the algorithm of Itai and Rodeh may fail to detect a shortest cycle only in the case when $girth(G)$ is odd and it repetitively detects shortest even cycles passing through start vertices belonging to shortest cycles (as seen in Fig. 1).

Our approximation algorithm in part relies on the following lemma.

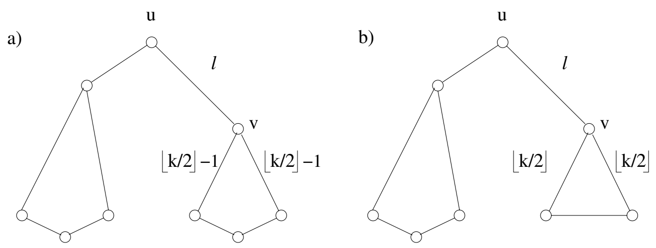


Fig. 1. The detection of a cycle can result in either an even cycle as in (a), or in an odd cycle as in (b)

Lemma 1. *If v is at distance l from u, and there is a simple cycle C of length k going through v then the halting BFS from u will report a simple cycle of length less or equal to $k + 2l$ if k is even, and $k + 2l + 1$ if k is odd.*

Proof. Suppose first that k is even. At most at the $(l + k/2 - 1)$-th level of the BFS tree rooted at u, we are guaranteed to find a cycle. It will be either the cycle C, or another one found while relaxing some vertex w at this level. The distance from u to w is at most $(l + k/2 - 1)$, so the length of the cycle reported can be at most $2(l + k/2 - 1) + 2$, i.e, at most $k + 2l$.

If k is odd, we will find a cycle while relaxing a vertex at the $l + \lfloor k/2 \rfloor$-th level, and the cycle reported will be of length at most $2(l + \lfloor k/2 \rfloor) + 2$, i.e., at most $2l + k + 1$. □

A deterministic version of the following simple combinatorial fact (Remark 2.8 in [1]) lies behind the algorithms for shortest paths with additive approximation factors from [1,15]. For our purposes, we shall use the original randomized version in order to minimize time complexity.

Fact 1. *Let H be the set of all vertices with a degree greater or equal to s, and let $\beta \geq 2$. A set of $\beta s^{-1} n \ln n$ vertices of G chosen uniformly at random is a dominating set for H with probability at least $1 - 1/(n^{\beta-1})$.*

Our algorithm *ApproxUnweightedCycle* is depicted in Fig. 2.

Lemma 2. *For an unweighted undirected graph G of girth k and $\beta \geq 2$, the procedure ApproxUnweightedCycle(G) outputs with probability at least $1 - 1/n^{\beta-1}$ a cycle of length at most $2k$ if k even, and of length at most $2k + 2$ if k is odd.*

Proof. To begin with, recall that the k-th power of a graph $G = (V, E)$ is a graph $G^k = (V, E^k)$ such that $(u,\ v) \in E^k$ if and only if there is a path of length at most k connecting u with v in G.

Let v be a vertex belonging to a cycle of length k in G.

Suppose first that k is even. Then the halting BFS from v either will detect a cycle of length at most k while relaxing a vertex at the $k/2 - 1$ level of the BFS tree rooted at v or it will stop because of visiting n^{ϵ} vertices on levels 1 through $k/2$. In the latter case, the vertex v has at least n^{ϵ} neighbors in the power graph

```
Algorithm ApproxUnweightedCycle(G)
Input: an undirected unweighted graph $G = (V, E)$, and a positive real $\epsilon$.
Output: if $G$ contains a cycle then a simple cycle $C$.

1. Perform the halting BFS from each vertex $v$, running at most $n^\epsilon$
   steps. If a cycle $F$ is detected and $C$ is either unspecified or it is a
   cycle longer than $F$ then set $C$ to $F$.
2. Set $D$ to a set of $\beta n^{1-\epsilon} \ln n$ vertices of $G$ chosen uniformly at random.
3. For each vertex $v$ in $D$, perform the halting BFS. If a cycle $F$ is
   detected and $C$ is either unspecified or it is a cycle longer than $F$
   then set $C$ to $F$.
4. Output $C$.
```

Fig. 2. The procedure *ApproxUnweightedCycle*

$G^{k/2}$. It follows by Fact 1 that v is dominated by a vertex u in D in $G^{k/2}$ with high probability. Hence, by Lemma 1, the halting BFS from u will detect a cycle of length at most $k + 2k/2$, i.e., at most $2k$.

The proof when k is odd is analogous. The halting BFS from v will either detect a cycle of length at most $k+1$ while relaxing a vertex at the $(k-1)/2$ level of the BFS tree rooted at v, or it will stop because of having visiting n^ϵ vertices on levels 1 through $(k-1)/2+1$. In the latter case v is dominated by a vertex u in D in $G^{(k-1)/2+1}$ with high probability. Hence, analogously by Lemma 1, the halting BFS from u will detect a cycle of length at most $k+2((k-1)/2+1)+1$, i.e., at most $2k+2$. □

Lemma 3. *For an unweighted undirected graph G on n vertices, Approx − UnweightedCycle(G) runs in time $O(max\{n^{2-\epsilon}\log n, n^{1+\epsilon}\})$.*

Proof. The first step takes $O(n^{1+\epsilon})$ time whereas the second one can be implemented in time $O(n)$. Finally, since the set D is of size $O(n^{1-\epsilon}\log n)$, the third step requires $O(n^{2-\epsilon}\log n)$ time. □

By Lemmata 2 and 3, and by setting $\epsilon = (1 + \frac{\log\log n}{\log n})/2$, we obtain:

Theorem 1. *For $\beta \geq 2$ and an unweighted undirected graph G on n vertices, the procedure ApproxUnweightedCycle(G) detects with probability at least $1 - 1/n^{\beta-1}$ a cycle of length at most $2girth(G)$ if $girth(G)$ even, and of length at most $2girth(G)+2$ if $girth(G)$ is odd in time $O(n^{\frac{3}{2}}\sqrt{\log n})$.*

Corollary 1. *For an unweighted undirected graph G on n vertices and $\beta \geq 2$, the procedure ApproxUnweightedCycle(G) detects with probability at least $1 - 1/n^{\beta-1}$ a cycle of length at most $2\frac{2}{3}girth(G)$.*

The NP-hard problem of packing a maximum number of edge disjoint cycles in an undirected (or directed) graph is fundamental in algorithmic graph theory

[8,21]. Krivelevich *et al.* provided the best known polynomial-time approximation heuristics for this cycle packing problem both in the undirected and directed case [21]. For an undirected graph on n vertices, their heuristic yields an $O(\sqrt{\log n})$ approximation factor. Their heuristic is a combination of an earlier modified greedy heuristic due to Caprara et al. [8] with the ordinary greedy algorithm repetitively detecting a shortest cycle in the current graph and removing the edges of the cycle. In all the three aforementioned heuristics the most time-consuming step is repetitively detecting a shortest cycle in the current graph. Caprara et al. and Krivelevich *et al.* do not discuss the time complexity of their heuristics, totally concentrating on the derivation of bounds on their approximation factor in [8,21].

Interestingly, their derivation of the $O(\sqrt{\log n})$ upper bound on the approximation factor in [21] can be trivially adapted to the situation when a constant-approximation heuristic for a shortest cycle is used instead of an exact algorithm. Hence, since one can pack at most $m/3$ edge-disjoint cycles in a graph on m edges, we obtain the following theorem by using our approximation algorithm for a shortest cycle in an undirected graph (Corollary 1) instead of an exact algorithm for a shortest cycle.

Theorem 2. *For $\alpha \geq 1$, with probability at least $1 - 1/n^{\alpha}$, one can approximate the problem of maximum packing of edge disjoint cycles in an undirected graph on n vertices and m edges within a factor of $O(\sqrt{\log n})$ in time $O(mn^{\frac{3}{2}}\sqrt{\log n})$.*

Similarly, the problem of packing a maximum number of vertex disjoint cycles in an undirected graph is a fundamental NP-hard problem in algorithmic graph theory [22]. Salavatipour et al. showed that this problem admits an $O(\log n)$ approximation in polynomial time [22]. Their heuristic is also based on repetitively detecting a shortest cycle in the current graph resulting from consecutive contractions (see p. 55 in [22]). Hence, our approximation algorithm for a shortest cycle leads to a more efficient implementation of the best heuristic for maximum packing of vertex disjoint cycles in an undirected graph too.

3 The Weighted Case

In this section, we assume that the input graph $G = (V, E)$ has non-negative real edge weights $weight(e)$, $e \in E$, and the task is to find a shortest cycle in G, i.e., a simple cycle of minimum total weight in G.

Given the close connection between the problem of finding shortest paths and the problem of finding a shortest cycle, it is a natural question to ask whether one could similarly halt Dijkstra's algorithm as one can halt breadth-first search in the unweighted case in order to obtain much faster close approximation of a shortest (simple) cycle.

To elaborate on this question, we have to recall the key concept of edge relaxation from the standard Dijkstra's algorithm [11]. For $v \in V$, let $d(v)$ be the current estimation of the distance from the source vertex in Dijkstra's algorithm and let $\pi(v)$ stand for the predecessor of v on a currently shortest path from the source vertex. Then, for $(u, v) \in E$, the relaxation of (u, v) is defined as follows.

```
**Relax** (u,v)
if $d(v) > d(u) + weight(u, v)$ then
  $d(v) \leftarrow d(u) + weight(u, v)$
  $\pi(v) \leftarrow u$
```

In the standard Dijkstra's algorithm, when a new vertex u is inserted in the set S of vertices for which the shortest distance to the source vertex s is already determined, all edges incident to u are relaxed. Some of them can close cycles (in the shortest-path tree spanning S) whose total weight might be much larger than that of a shortest cycle in G.

To deal with this difficulty, we shall consider a variant of Dijkstra's algorithm where only a lightest among edges incident to u with the other endpoint outside S is relaxed. To extract such an edge, we shall use a priority queue Q_u for edges incident to u. We use a standard heap implementation of our priority queue [11]. Also, when a new vertex u is inserted into S, then a new, currently lightest edge incident to its predecessor $\pi(u)$ in S with the other endpoint outside S is extracted from the priority queue $Q_{\pi(u)}$. We shall term this variant of Dijkstra's algorithm as Priority Dijkstra, see Fig. 3.

```
procedure Priority Dijkstra$(G, s)$
Input: a graph $G = (V, E)$ given by priority queues $Q_u$, $u \in V$, of edges $(u, v)$
incident to $u$ ordered by their weight, and a distinguished start vertex $s \in V$.
Output: for each vertex $v \in V$, its distance $d(v)$ from $s$, and the predecessor $\pi(v)$
on a shortest path from $s$ to $v$.

1. for $v \in V$ do
   (a) $d(v) \leftarrow \infty$;
   (b) $\pi(v) \leftarrow NIL$;
2. $d(s) \leftarrow 0$
3. $S \leftarrow \emptyset$;
4. $Q \leftarrow$ a priority queue of vertices ordered by $d(v)$;
5. while $Q \neq \emptyset$ do
   (a) Extract a vertex $u$ of $G$ of minimum $d(u)$ from $Q$;
   (b) $S \leftarrow S \cup \{u\}$;
   (c) Extract an edge $(u, v)$ satisfying $v \notin S$ of minimum $weight(u, v)$ from $Q_u$;
   (d) if $(u, v)$ is defined then Relax$(u, v)$;
   (e) Extract an edge $(\pi(u), v')$ satisfying $v' \notin S$ of minimum $weight(\pi(u), v')$
       from $Q_{\pi(u)}$;
   (f) if $(\pi(u), v')$ is defined then Relax$(\pi(u), v')$;
```

Fig. 3. The procedure *PriorityDijkstra*

By the definition of Priority Dijkstra, the following holds:

For each $u \in S$, an edge (u, v) incident to u, where $v \notin S$, of minimum weight (if any) is relaxed, i.e., $d(v) \leftarrow \min\{d(v),\ d(u) + weight(u, v)\}$, before the choice of a new vertex for insertion into S.

It follows by induction on the cardinality of S that when the next vertex v is inserted into S, it has the same value $d(v)$, as the one inserted by the standard Dijkstra's algorithm, and hence, we may assume without loss of generality that it is the same vertex as the one inserted by the standard Dijkstra's algorithm. Hence, analogously as for the Dijkstra algorithm, we can obtain the following lemma.

Lemma 4. *If we run Priority Dijkstra procedure on an edge weighted directed (or, undirected) graph $G = (V, E)$ with nonnegative weight function w and source s then for each vertex $u \in V$ after its insertion in S, $d(u)$ is equal to the distance from s to u, and $d(v) \leq d(u)$ for all v inserted into S prior to u.*

Of course, the preprocessing in the form of the priority queue already requires super-quadratic time, however as we show later it can be reused for the runs for all starting vertices.

As for halting *Priority Dijkstra*(G, s) when an edge closing a cycle in the shortest-path tree of the current S is encountered, we still face the problem that the weight of such an edge can be much larger than that of a shortest cycle in G passing through s.

To tackle with the latter problem, we introduce a t-bounded variant of Priority Dijkstra, Bounded Priority Dijkstra (BPD for short), where t is a positive real. $BPD(G, s, t)$ simply disregards steps 5(c,d) if the currently lightest edge in Q_u has weight exceeding $t - d(u)$. Similarly, it disregards steps 5(e,f) if the currently lightest edge in $Q_{\pi(u)}$ has weight exceeding $t - d(\pi(u))$.

It is easy to observe that $BPD(G, s, t)$ computes correct distances for all vertices u of G whose distance from s is at most t.

Now, let us modify BPD further so it halts whenever it detects an edge (u, v) satisfying $d(u) + weight(u, v) \leq t$ and $v \in S$ in step 5(c), or an edge $(\pi(u), v')$ satisfying $d(\pi(u)) + weight(\pi(u), v') \leq t$ and $v' \in S$ in step 5(e), and reports the simple cycle closed respectively by (u, v) or $(\pi(u), v')$ in the shortest-path tree spanning the current S. Let HBPD denote the so modified procedure.

Lemma 5. *$HBPD(G, s, t)$ runs in time $\mathcal{O}(n \log n)$.*

Proof. Each iteration of the block under the while instruction which does not detect a cycle takes $\mathcal{O}(\log n)$ time (including updating Q after the relaxation of $(u,\ v)$ or $(\pi(u),\ v')$). Reporting the detected cycle takes $\mathcal{O}(n)$ time by using the pointer function π. □

Lemma 6. *If $HBPD(G, s, t)$ detects a cycle then the weight of the cycle is at most $2t$.*

Proof. Let (u, v) be an edge that closes the detected cycle. Then, the weight of the cycle is at most $d(u) + weight(u, v) + d(v)$. By the t-boundedness, we have $d(u) + weight(u, v) \leq t$. Since v already belongs to S, we have also $d(v) \leq t$. □

Lemma 7. *If the input graph G has a cycle of weight not exceeding t then there is a vertex s in G such that $HBPD(G, s, t)$ detects a cycle.*

Proof. Let C be a cycle in G of weight not greater than t. By Theorem 4 in [16], there is a vertex s and an edge (u, v) in C such that C consists of shortest, i.e., minimum weight, paths from s to u and v, respectively, and the edge (u, v). Run $HBPD(G, s, t)$. We may assume without loss of generality that $HBPD(G, s, t)$ never detects a cycle, thus, all vertices of G are in the final S. We may also assume without loss of generality that v is inserted before u into S. The queue Q_u cannot be discarded before the edge (u, v) is considered and relaxed since $d(u) + weight(u, v) \leq t$. We obtain a contradiction, since when (u, v) is relaxed the cycle C should be detected. □

By running HBPD with all possible start vertices, we obtain the following theorem by Lemmata 6, 7.

Theorem 3. *If G is known to contain a cycle of weight at most t, then one can detect a cycle of weight at most $2t$ in G in time $\mathcal{O}(n^2 \log n)$.*

Proof. By combining Lemmata 6, 7, we conclude that $HBPD(G, s, t)$ detects a cycle of weight at most $2t$ if s belongs to a cycle of weight at most t.

To recover the priority queues Q_n, we can store the elements extracted from them on corresponding separate lists and then use the lists to reinsert the elements back into the queues. This modification does not alter the asymptotic time complexity of $HBPD$. □

By combining Theorem 3 with a binary search, we obtain our second main result.

Theorem 4. *Let G be an undirected graph with nonnegative integer edge weights in the range $\{1, 2, ..., M\}$. A 2-approximation to a minimum weight (simple) cycle in G can be determined in time $\mathcal{O}(n^2 \log n \log M)$.*

4 Final Remarks

It is an interesting open problem whether or not there is a substantially subquadratic-time c-approximation algorithm for a shortest cycle in an unweighted undirected graph for $c \leq 2$.

It is also an interesting open problem whether or not there is a roughly quadratic-time c-approximation algorithm for a minimum weight cycle in an undirected graph with nonnegative integer edge weights of polynomial size for $c < 2$.

References

1. Aingworth, D., Chekuri, C., Indyk, P., Motwani, R.: Fast Estimation of Diameter and Shortest Paths (Without Matrix Multiplication). SIAM J. Comput. 28(4), 1167–1181 (1999)
2. Alon, N., Yuster, R., Zwick, U.: Color-coding. Journal of the ACM 42(4), 844–856 (1995)

3. Alon, N., Yuster, R., Zwick, U.: Finding and counting given length cycles. Algorithmica 17(3), 209–223 (1997)
4. Baswana, S., Goyal, V., Sen, S.: All-Pairs Nearly 2-Approximate Shortest-Paths in $\mathcal{O}(n^2\text{polylog } n)$ Time. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 666–679. Springer, Heidelberg (2005)
5. Baswana, S., Kavitha, T.: Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In: Proc. FOCS 2006, pp. 591–602. IEEE, Los Alamitos (2006)
6. Berman, P., Kasiviswanathan, S.P.: Faster approximation of distances in graphs. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 541–552. Springer, Heidelberg (2007)
7. Bollobás, B.: Extremal Graph Theory. Academic Press, New York (1978)
8. Caprara, A., Panconesi, A., Rizzi, R.: Packing cycles in undirected graphs. J. Algorithms 48, 239–256 (2003)
9. Chan, T.: More algorithms for all-pairs shortest paths in weighted graphs. In: Proc. STOC 2007, pp. 590–598 (2007)
10. Cohen, E., Zwick, U.: All-pairs small-stretch paths. Journal of Algorithms 38(2), 335–353 (2001)
11. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. The MIT Press, Cambridge (1990)
12. Czumaj, A., Lingas, A.: Finding a heaviest triangle is not harder than matrix multiplication. In: Proc. SODA 2007, pp. 986–994 (2007)
13. Diestel, R.: Graph Theory. Springer, Heidelberg (2000)
14. Djidjev, H.: Computing the Girth of a Planar Graph. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 821–831. Springer, Heidelberg (2000)
15. Dor, D., Halperin, S., Zwick, U.: All Pairs Almost Shortest Paths. SIAM J. Computing 29, 1740–1759 (2000)
16. Horton, J.D.: A polynomial-time algorithm to find a shortest cycle basis of a graph. SIAM Journal on Computing 16(2), 358–366 (1987)
17. Itai, A., Rodeh, M.: Finding a minimum circuit in a graph. SIAM Journal on Computing 7(4), 413–423 (1978)
18. Karger, D.R., Koller, D., Phillips, S.J.: Finding the Hidden Path: Time Bounds for All-Pairs Shortest Paths. In: Proc. FOCS 1991, pp. 560–568 (1991)
19. Kavitha, T., Mehlhorn, K., Michail, D., Paluch, K.: A faster algorithm for Minimum Cycle Basis of graphs. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 846–857. Springer, Heidelberg (2004)
20. Kavitha, T., Mehlhorn, K., Michail, D.: New Approximation Algorithms for Minimum Cycle Bases of Graphs. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 512–523. Springer, Heidelberg (2007)
21. Krivelevich, M., Nutov, Z., Yuster, R.: Approximation Algorithms for Cycle Packing Problems. In: Proc. SODA 2005, pp. 556–561 (2005)
22. Salavatipour, M.R., Verstraete, J.: Disjoint Cycles: Integrality Gap, Hardness, and Approximation. In: Jünger, M., Kaibel, V. (eds.) IPCO 2005. LNCS, vol. 3509, pp. 51–65. Springer, Heidelberg (2005)
23. Schank, T., Wagner, D.: Finding, Counting and Listing all Triangles in Large Graphs, An Experimental Study. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 606–609. Springer, Heidelberg (2005)
24. Takaoka, T.: A Faster Algorithm for the All-Pairs Shortest Path Problem and Its Application. In: Chwa, K.-Y., Munro, J.I.J. (eds.) COCOON 2004. LNCS, vol. 3106, pp. 278–289. Springer, Heidelberg (2004)

25. Thorup, M., Zwick, U.: Approximate distance oracles. In: Proc. STOC 2001, pp. 183–192 (2001)
26. Vassilevska, V., Williams, R., Yuster, R.: Finding the smallest H-subgraph in real weighted graphs and related problems. In: Proc. ICALP 2006, pp. 262–273 (2006)
27. Vassilevska, V., Williams, R.: Finding a maximum weight triangle in $n^{3-\delta}$ time, with applications. In: Proc. STOC 2006, pp. 225–231 (2006)
28. Yuster, R., Zwick, U.: Finding Even Cycles Even Faster. In: Shamir, E., Abiteboul, S. (eds.) ICALP 1994. LNCS, vol. 820, pp. 532–543. Springer, Heidelberg (1994)
29. Yuster, R., Zwick, U.: Detecting short directed cycles using rectangular matrix multiplication and dynamic programming. In: Proc. SODA 2004, pp. 254–260 (2004)
30. Zwick, U.: All pairs shortest paths using bridging rectangular matrix multiplication. Journal of the ACM 49(3), 289–317 (2002)
31. Zwick, U.: Exact and Approximate Distances in Graphs - A survey. In: Meyer auf der Heide, F. (ed.) ESA 2001. LNCS, vol. 2161, pp. 33–48. Springer, Heidelberg (2001)

Domination in Geometric Intersection Graphs

Thomas Erlebach[1] and Erik Jan van Leeuwen[2,*]

[1] Department of Computer Science, University of Leicester,
University Road, Leicester LE1 7RH, UK
T.Erlebach@mcs.le.ac.uk
[2] CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
E.J.van.Leeuwen@cwi.nl

Abstract. For intersection graphs of disks and other fat objects, polynomial-time approximation schemes are known for the independent set and vertex cover problems, but the existing techniques were not able to deal with the dominating set problem except in the special case of unit-size objects. We present approximation algorithms and inapproximability results that shed new light on the approximability of the dominating set problem in geometric intersection graphs. On the one hand, we show that for intersection graphs of arbitrary fat objects, the dominating set problem is as hard to approximate as for general graphs. For intersection graphs of arbitrary rectangles, we prove APX-hardness. On the other hand, we present a new general technique for deriving approximation algorithms for various geometric intersection graphs, yielding constant-factor approximation algorithms for r-regular polygons, where r is an arbitrary constant, for pairwise homothetic triangles, and for rectangles with bounded aspect ratio. For arbitrary fat objects with bounded ply, we get a $(3 + \epsilon)$-approximation algorithm.

1 Introduction

We study the approximability of the minimum dominating set problem in geometric intersection graphs. Given an undirected graph $G = (V, E)$, a set $D \subseteq V$ is a *dominating set* if every $v \in V$ is in D or has a neighbor in D. The aim of *Minimum Dominating Set* (MDS) is to compute for a given graph a dominating set of minimum cardinality. Although for general graphs the approximability of MDS has been settled [15,8], the problem is open for numerous graph classes, such as geometric intersection graphs.

Geometric intersection graphs are graphs in which the vertices represent geometric objects and two vertices are adjacent if the corresponding objects intersect. Studying approximation algorithms for fundamental graph optimization problems on such graphs has led to several new techniques, in particular the geometric *shifting technique* [17], which can be used to obtain polynomial-time approximation schemes (PTASs) for a number of problems, such as Maximum Independent Set and Minimum Vertex Cover in unit disk graphs [18] and in general disk graphs [13,6,26]. These algorithms extend to any constant number of dimensions and arbitrary fat objects (including e.g. squares or other regular polygons in the two-dimensional case).

Interestingly, as pointed out in [13], these techniques do not seem sufficient for handling MDS in intersection graphs of objects of different sizes. To the best of our

* Partially supported by the Dutch BSIK/BRICKS project.

knowledge, there are no results for intersection graphs of disks, squares, etc. beyond the $(1 + \ln n)$-approximation ratio that can be achieved by the greedy algorithm. In particular, we know of no constant-factor approximation algorithm or approximation hardness results. In this paper, we address this open problem by studying the minimum dominating set problem for intersection graphs of different types of fat objects and providing new insights into its approximability.

In Sect. 3 we present a new general approach to deriving approximation algorithms for MDS on geometric intersection graphs. We apply it to obtain the first constant-factor approximation algorithms for MDS on intersection graphs of r-regular polygons, of pairwise homothetic triangles, and of rectangles of bounded aspect-ratio.

We also obtain a constant-factor approximation algorithm for MDS on disk graphs of constant ply (see Sect. 4). A surprising corollary of this is a constant integrality gap for MDS on planar graphs. For disk graphs of bounded ply, this result can be improved to a $(3 + \epsilon)$-approximation algorithm by using a new variant of the shifting technique. This algorithm extends to intersection graphs of arbitrary fat objects of bounded ply.

The type of fat objects considered impacts the approximability of MDS: We prove that for n arbitrary fat objects, approximation ratio $(1 - \epsilon) \ln n$ is not achievable for any $\epsilon > 0$, unless $NP \subset DTIME(n^{O(\log \log n)})$. We also solve an open problem of Chlebík and Chlebíková [9], who asked whether their APX-hardness results for intersection graphs of d-dimensional axis-parallel boxes extend to the case $d = 2$. We affirm this by showing that MDS is APX-hard for rectangle intersection graphs.

1.1 Known Results

MDS in general graphs is essentially equivalent to Minimum Set Cover. For n-vertex graphs, approximation ratio $1+\ln n$ is achievable by a greedy algorithm, and one cannot get ratio $(1 - \epsilon) \ln n$ for any $\epsilon > 0$, unless $NP \subset DTIME(n^{O(\log \log n)})$ [15,8].

Even though geometric intersection graphs have properties exploitable to approximate several problems [13,6,26], only few approximation algorithm are known for MDS in such graphs. For unit disk graphs, Marathe et al. [22] gave a constant-factor approximation algorithm, before a PTAS was presented by Hunt et al. [18] and Nieberg et al. [25]. MDS in unit disk graphs seems harder in the weighted than in the unweighted case, but has a constant-factor approximation algorithm by Ambühl et al. [2].

On the negative side, MDS cannot have an FPTAS (unless P=NP), as it is NP-hard for geometric intersection graphs (even for simple classes such as unit disk graphs [10]). Chlebík and Chlebíková [9] have shown that for any $d \geq 3$, Minimum Dominating Set and several other problems are APX-hard on intersection graphs of d-dimensional axis-parallel boxes. It follows from Marx [23] that Minimum Dominating Set cannot have an EPTAS (Efficient PTAS) for unit square/disk graphs (unless FPT=W[1]).

Some of our algorithms use ϵ-nets, which were used to approximate geometric optimization problems before, e.g. geometric hitting set [5,14], geometric set cover [11].

2 Preliminaries

A *ρ-approximation algorithm* for a minimization problem is an algorithm that runs in polynomial time and always produces a solution whose value is at most $\rho \cdot OPT$, where OPT is the optimal objective value. The value ρ is also referred to as the approximation ratio. An algorithm that achieves approximation ratio $1 + \epsilon$, for arbitrary $\epsilon > 0$, and whose running-time is polynomial in the size of the input for any fixed ϵ, is called a *polynomial-time approximation scheme* or *PTAS*. If its running-time is polynomial also in $\frac{1}{\epsilon}$, it is called a *fully polynomial-time approximation scheme* or *FPTAS*. A *c-asymptotic fully polynomial-time approximation algorithm* or *c-FPTAA*$^{\omega}$ is an algorithm giving for any $\epsilon > 0$ a feasible solution in time polynomial in $\frac{1}{\epsilon}$ and the size of the input, such that the objective value of the solution is at most $(c + \epsilon)OPT$ if the size of the input is at least c_ϵ, where c_ϵ is a constant depending only on ϵ. If $c = 1$, it is called an *asymptotic fully polynomial-time approximation scheme* or *FPTAS*$^{\omega}$.

2.1 ϵ-Nets

Our main algorithmic results rely on the availability of small ϵ-nets. Given a universe $\mathbb{U}$ and a family $\mathcal{S}$ of n subsets of $\mathbb{U}$ (called *objects*), we say $\mathcal{R} \subseteq \mathcal{S}$ is an *ϵ-net* for $\mathcal{S}$ if any element $u \in \mathbb{U}$ covered by more than $\epsilon\,|\mathcal{S}|$ sets in $\mathcal{S}$ is also covered by $\mathcal{R}$ (i.e., covered by $\bigcup \mathcal{R}$). The size of the net is equal to the cardinality of $\mathcal{R}$. Suppose that for objects of a certain type (e.g. disks in the plane), we have a *decomposition bound function* $f(n)$ bounding the number of simple regions in a canonical decomposition of the complement of the union of n such objects. Then Clarkson and Varadarajan have proved the following result.

Theorem 1 ([11]). *For any $0 < \epsilon \leq 1$, there is an ϵ-net for $\mathcal{S}$ of size $O(f(1/\epsilon) + 1/\epsilon)$.*

Such a net can be found by a randomized algorithm with polynomial expected running time. For details and a formal definition of f, we refer to [11]. By derandomizing the algorithm using the method of conditional expectations, we obtain the following result.

Theorem 2. *For any $0 < \epsilon \leq 1$, we can find an ϵ-net for $\mathcal{S}$ of size $O(f(1/\epsilon) + 1/\epsilon)$ in time polynomial in $|\mathcal{S}|$, $1/\epsilon$, and $f(1/\epsilon)$.*

Pseudo-disks (subsets of the plane bounded by simple closed Jordan curves where each pair of curves intersects at most twice) have a linear decomposition bound function [19,11], giving a linear sized net.

3 Domination in Geometric Intersection Graphs

We introduce the novel notion of $\preceq$-dominating sets, which we use with ϵ-nets to approximate geometric dominating set. Let $\preceq$ be a binary reflexive relation on the vertices of a graph $G = (V, E)$. An example, say for geometric intersection graphs, is that $u \preceq v$ if the object representing u is at most as large as the object representing v. We call $v \in V$ *$\preceq$-larger* than $u \in V$ if $u \preceq v$. Denote by $N_{\preceq}(u) = \{v \in V \mid (u, v) \in E, u \preceq v\}$ the set of $\preceq$-larger neighbors of u and u's closed $\preceq$-larger neighborhood by $N_{\preceq}[u] = N_{\preceq}(u) \cup \{u\}$.

Call a set $D \subseteq V$ a *$\preceq$-dominating set* if for any $u \in V$, $u \in D$ or there is a $\preceq$-larger neighbor of u in D, i.e. $D \cap N_{\preceq}[u] \neq \emptyset$. In light of the following theorem, we will be interested in binary reflexive relations $\preceq$ where the *$\preceq$-factor* (the size of a minimum $\preceq$-dominating set divided by the size of a minimum dominating set) is at most a constant.

Theorem 3 (Main Theorem). *Let G be the intersection graph of a set $\mathcal{S} = \{s_u \subseteq \mathbb{R}^d \mid u \in V(G)\}$ of closed topological balls with decomposition bound function f. Let $\preceq$ be a binary reflexive relation on the vertices of G with $\preceq$-factor c_1 such that for any vertex u there exist c_2 points in s_u jointly hitting all objects s_v with $v \in N_{\preceq}(u)$. If the size of a minimum dominating set of G is k, then we can find in polynomial time a dominating set of size $O(f(2c_1c_2k) + c_1c_2k)$.*

Proof. Solve the LP-relaxation of the $\preceq$-dominating set problem. The integer LP is

$$\begin{aligned} z_I^* = \min\ & \textstyle\sum_{u \in V} x_u \\ \text{s.t.}\ & \textstyle\sum_{v \in N_{\preceq}[u]} x_v \geq 1 \quad (\forall u \in V) \\ & x_u \in \{0,1\} \quad (\forall u \in V). \end{aligned}$$

Observe that $z_I^* \leq c_1 k$. In the relaxation, the last constraint is replaced by $x_u \geq 0$ ($\forall u \in V$). Let x^* be a vector attaining the optimum fractional value z^*. Since for any vertex u all objects s_v with $v \in N_{\preceq}(u)$ can be hit by c_2 points in s_u, each s_u contains a point p such that $\sum_{v : p \in s_v} x_v^* \geq 1/c_2$.

Now construct a set $\mathcal{S}'$ from $\mathcal{S}$ by taking $\lceil x_u^* \cdot |\mathcal{S}|/z^* \rceil$ copies of each object s_u. Following the previous observation, this means that for any object $s \in \mathcal{S}$ there is a point p in s such that at least $|\mathcal{S}|/(c_2 z^*)$ objects of $\mathcal{S}'$ contain p. Furthermore,

$$|\mathcal{S}'| = \textstyle\sum_{u \in V} \lceil x_u^* \cdot |\mathcal{S}|/z^* \rceil < \sum_{u \in V} (1 + x_u^* \cdot |\mathcal{S}|/z^*) = |\mathcal{S}| + \frac{|\mathcal{S}|}{z^*} \sum_{u \in V} x_u^* = 2|\mathcal{S}|.$$

Applying Theorem 2, we find a set $\mathcal{R}' \subseteq \mathcal{S}'$ of size $O(f(2c_2z^*) + 2c_2z^*)$ such that any point covered by more than $|\mathcal{S}'|/(2c_2z^*)$ objects of $\mathcal{S}'$, and thus also any point covered by at least $|\mathcal{S}|/(c_2z^*)$ objects of $\mathcal{S}'$, is covered by $\mathcal{R}'$. Then $\mathcal{R}'$ intersects each object of $\mathcal{S}$ and thus $\mathcal{R}'$ is a dominating set of G. It has size

$$O(f(2c_2z^*) + 2c_2z^*) \leq O(f(2c_2z_I^*) + c_2z_I^*) \leq O(f(2c_1c_2k) + c_1c_2k).$$

Following Theorem 2, $\mathcal{R}'$ can be found in polynomial time. □

Hence the integrality gap[1] of the LP relaxation of $\preceq$-MDS is $O(c_1 f(2c_2z^*)/z^* + c_1c_2)$.

In the remainder, we do not distinguish between a vertex $v \in V$ and the geometric object s_v it represents, i.e., v can refer both to the vertex and to the geometric object.

Before we can apply Theorem 3, we need more concrete relations $\preceq$. Consider the intersection graph of a set $\mathcal{S}$ of closed topological balls in $\mathbb{R}^d$. Define a relation $\preceq_{\mathsf{Leb}}$ such that $u \preceq_{\mathsf{Leb}} v$ if and only if the Lebesgue measure of u is at most the Lebesgue measure of v. Clearly, $\preceq_{\mathsf{Leb}}$ is a (total) preorder (i.e. $\preceq_{\mathsf{Leb}}$ is reflexive and transitive). The following two easy lemmas are sufficient to show that the $\preceq_{\mathsf{Leb}}$-factor is a constant for many intersection graph classes. We use $N(u)$ to denote the set $\{v \mid (u,v) \in E\}$.

Lemma 1. *Let $\preceq$ be a binary reflexive relation on the vertices of G such that for any vertex u a minimum $\preceq$-dominating set for $U_u = \{v \mid v \not\preceq u, v \in N(u)\}$ has size at most c. Then the $\preceq$-factor is at most $c+1$.*

The observation here is that if D is a dominating set of G and D_u is a minimum $\preceq$-dominating set for U_u, then $D \cup \bigcup_{u \in D} D_u$ is a $\preceq$-dominating set. In fact, a bound on the size of a minimum $\preceq$-dominating set for U_u is only needed for vertices u appearing in a particular minimum dominating set.

Lemma 2. *Let $\preceq$ be a total preorder on the vertices of G s.t. for any vertex u the size of any independent set of $N_{\preceq}(u)$ is bounded by c. Then the $\preceq$-factor is at most $c + 1$.*

These lemmas also hold for the *fractional $\preceq$-factor* (the ratio of the value of the optimum fractional $\preceq$-dominating set and fractional dominating set). By Thm. 3, the integrality gap[1] of MDS is $O(f(2c_2c_3z^*)/z^* + c_2c_3)$ if the fractional $\preceq$-factor is at most c_3.

3.1 Regular Polygons

We apply Theorem 3 to give constant-factor approximation algorithms for Minimum Dominating Set on intersection graphs of regular polygons. We assume the polygons are pairwise *homothetic*: one polygon can be obtained from another by scaling and translating (i.e. rotations are not allowed). Applying results of Kim, Kostochka, and Nakprasit [20] (bounding sizes of independent sets in neighborhoods of larger objects) and Lemma 2, we can show that the $\preceq_{\mathsf{Leb}}$-factor is at most 5 for intersection graphs of homothetic parallelograms and at most 6 for intersection graphs of homothetic copies of any other planar convex object (including disks and regular polygons).

For even regular polygons (i.e. $2r$-regular polygons), $2r$ points suffice to hit all $\preceq_{\mathsf{Leb}}$-larger neighbors of a vertex (take the corners[2] of the polygon). As pairwise homothetic regular polygons are pseudo-disks, we can apply Theorem 3 with a linear decomposition bound to yield the following.

Theorem 4. *Let $r > 0$ be an integer. There is a polynomial-time $O(r)$-approximation algorithm for Minimum Dominating Set on intersection graphs of pairwise homothetic $2r$-regular polygons.*

Corollary 1. *Minimum Dominating Set on square intersection graphs is in APX.*

Although Theorem 4 also works for intersection graphs of 2-regular polygons (i.e. interval graphs), a linear-time exact algorithm exists in this case [7].

The $\preceq_{\mathsf{Leb}}$ relation does not seem sufficient to give a constant-factor approximation algorithm for odd regular polygons, as it is not possible to hit all $\preceq_{\mathsf{Leb}}$-larger neighbors of a vertex by a constant number of points *inside* the object, even though a constant number of points outside the object would suffice. The algorithm of Theorem 3 does not seem to extend to this case. However, we can introduce a more restrictive relation $\preceq$ such that $u \preceq v$ if in addition to $u \preceq_{\mathsf{Leb}} v$, v also covers a constant fraction of the boundary of u or covers a corner of u. For this relation, a constant number of points inside an object suffices to hit all $\preceq$-larger neighbors. We can also show it has a constant $\preceq$-factor for odd regular polygons. Detailed analysis reveals the restriction that $u \preceq_{\mathsf{Leb}} v$ is not necessary. So for two vertices $u, v \in V$, define $u \preceq_{1/3} v$ if and only if v contains a corner of u or v covers at least one third of a side of u. We first consider triangles.

Theorem 5. *There is a polynomial-time $O(1)$-approximation algorithm for Minimum Dominating Set on intersection graphs of pairwise homothetic equilateral triangles.*

[1] The *integrality gap* is the ratio of the optimum integral and optimum fractional value of an LP.
[2] To disambiguate between vertices of a graph and vertices of a polygon, vertices of a polygon will be referred to as *corners*.

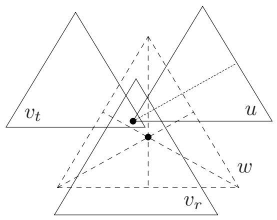


Fig. 1. Triangles u, v_t, v_r, and w of the proof of Theorem 5. The two dots represent p and the barycenter of w.

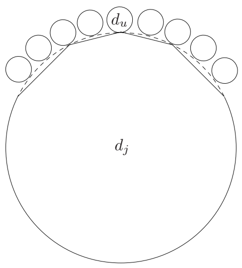


Fig. 2. A cut-off disk d_j and the disks d_u for elements $u \in \mathbb{U}$ of Theorem 13

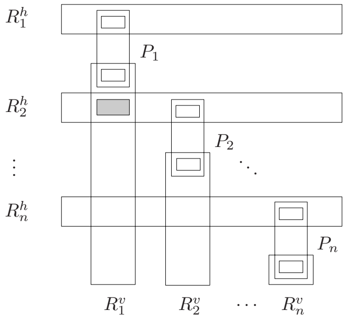


Fig. 3. The intersection graph of Theorem 15. If $(v_1, v_2) \in E$, then the shaded rectangle $S_{1,2}$ is in G'.

Proof. All $\preceq_{1/3}$-larger neighbors of a vertex can be hit by 9 points inside the triangle (its three corners and two points equidistantly on each side). To apply the $\preceq_{1/3}$ relation with Theorem 3, we show that the $\preceq_{1/3}$-factor is at most 7. Let $\mathcal{S}$ be a set of equilateral triangles with base parallel to the x-axis. Consider a triangle u that is not fully contained in any other triangle of $\mathcal{S}$ and let $U = \{v \mid v \not\preceq_{1/3} u, v \in N(u)\}$. For any $v \in U$, u does not contain a corner of v. Hence all $v \in U$ must contain a corner of u. Look at one particular corner of u, say the left corner, and let $U_l \subseteq U$ be the set of triangles intersecting it. Now let v_t be a vertex in U_l such that the top corner of v_t has the largest distance to the altitude[3] of the left corner of u. Similarly, let $v_r \in U_l$ be a vertex such that the right corner of v_r has the largest distance to this altitude (see Fig. 1). We claim v_t, v_r, and u form a $\preceq_{1/3}$-dominating set for U_l.

Let w be a vertex in U_l. We may assume w has no corner in v_t, v_r, or u. Then w contains a corner of v_t, v_r, and u. Furthermore, by the choice of v_t and v_r, w cannot fully contain either v_t or v_r, as the top (right) corner of w would be further from the altitude than the top (right) corner of v_t (v_r). Triangles v_t, v_r, and u share a common point p inside w (the leftmost corner of u). There must be a side of w such that p is at least as far from this side as the barycenter[4] of w. Suppose w.l.o.g. that v_r protrudes this side. Then the corner of v_r in w is at least as far from this side as p, and thus at least as far from the side as the barycenter of w. An easy calculation shows that v_r covers at least one third of the side of w.

Similarly, two triangles can be chosen for the other two corners of u. This gives a $\preceq_{1/3}$-dominating set for U of size at most 7. There is a minimum dominating set D such that no triangle in D is strictly contained inside another triangle of $\mathcal{S}$. As we can find a $\preceq_{1/3}$-dominating set for U of size at most 7 for any u not fully contained in some other triangle of $\mathcal{S}$, it follows similar to Lemma 1 that the $\preceq_{1/3}$-factor is at most 7. □

[3] An *altitude* of a triangle τ is the line through a corner of τ, perpendicular to the side opposite the corner.

[4] The *barycenter* or *centroid* of a triangle is the intersection point of the three straight lines going through a corner of the triangle and the midpoint of the opposite side.

For odd regular polygons with a larger number of sides, a similar proof as in Theorem 5 bounds the $\preceq_{1/3}$-factor. However, we can do better. Define a relation $\preceq_{1/2}$ such that $u \preceq_{1/2} v$ if and only if v contains a corner of u or v covers at least half of a side of u. Using the $\preceq_{1/2}$ relation yields the following result.

Theorem 6. *For any $r \in \mathbb{Z}_{>1}$, there is a polynomial-time $O(r^2)$-approximation algorithm for MDS on intersection graphs of pairwise homothetic $(2r+1)$-regular polygons.*

Our results imply $O(1)$-approximation algorithms for Minimum Connected/Total Dominating Set on intersection graphs of r-regular polygons, for constant r. Also the results imply a constant bound on the integrality gap of the LP relaxation for these problems, as the bounds on the integral $\preceq_{\mathsf{Leb}}$-, $\preceq_{1/3}$-, and $\preceq_{1/2}$-factors extend easily to their fractional variants by the fractional versions of Lemma 1 and 2.

3.2 More General Objects

The proof of Theorem 5 also goes through for pairwise homothetic triangles in general. Alternatively, one can use an affine transformation to map pairwise homothetic triangles into an equivalent set of pairwise homothetic equilateral triangles (i.e. two mapped triangles intersect if and only if they do so in the original set) and then apply Theorem 5.

Theorem 7. *There is a polynomial-time $O(1)$-approximation algorithm for Minimum Dominating Set on intersection graphs of pairwise homothetic triangles.*

We also consider intersection graphs of axis-parallel rectangles whose aspect-ratio (the ratio of the length of the longer side over that of the shorter side) is bounded by an integer constant c. It is easy to see that any rectangle with aspect-ratio at most c can be represented as the union of at most c squares. Hence the union of n axis-parallel rectangles of aspect-ratio at most c is also the union of cn axis-parallel squares. This implies that the decomposition bound function is $O(cn)$ (as squares are pseudo-disks). Furthermore, a $\preceq_{\mathsf{Leb}}$-larger rectangle intersecting a rectangle u must contain a corner of u or a $\frac{1}{c}$-fraction of a side of u. Hence $O(c)$ points in u suffice to hit all $\preceq_{\mathsf{Leb}}$-larger rectangles intersecting u and the $\preceq_{\mathsf{Leb}}$-factor is $O(c)$. Now apply Theorem 3.

Theorem 8. *For any $c > 1$, there is a polynomial-time $O(c^3)$-approximation algorithm for MDS on intersection graphs of axis-parallel rectangles with aspect-ratio at most c.*

These methods do not seem to extend to intersection graphs in higher dimensions.

4 Disk Graphs of Bounded Ply

We do not know how to use the above approach to obtain a constant-factor approximation algorithm for Minimum Dominating Set in general disk graphs, even though the $\preceq_{\mathsf{Leb}}$-factor is 6. However, this low $\preceq_{\mathsf{Leb}}$-factor can be used to give an approximation algorithm if the ply of the set of disks is bounded. The *ply* of a set of objects is the maximum over all points p of the number of objects strictly containing p [24].

Using Theorem 3, we can give an $O(\gamma)$-approximation algorithm for MDS on disk intersection graphs of ply γ. Different techniques however can improve the hidden constant of this result and make it explicit.

Theorem 9. *The integrality gap of the LP relaxation of Minimum Dominating Set on disk intersection graphs of ply γ is at most $54 \cdot \gamma$. If the ply is 1, the gap is at most 42. Hence the gap of the LP relaxation of MDS on planar graphs is at most 42.*

Proof (Sketch). We transform the minimum $\preceq_{\mathsf{Leb}}$-dominating set problem on the input graph to a Minimum Set Cover (MSC) instance in which the element frequency is at most $\max_{u \in V} |N_{\preceq_{\mathsf{Leb}}}[u]|$. Following a result of Hochbaum [16], the integrality gap of the MSC instance is at most the element frequency. But then the integrality gap of the original problem is at most the fractional $\preceq_{\mathsf{Leb}}$-factor times $\max_{u \in V} |N_{\preceq_{\mathsf{Leb}}}[u]|$. Using an area bound, one can show that the closed $\preceq_{\mathsf{Leb}}$-larger neighborhood of a disk in a set of disks of ply γ has size at most $9 \cdot \gamma$ [24]. If the ply is 1, the size is at most 7. The theorem then follows from the fact that the fractional $\preceq_{\mathsf{Leb}}$-factor for disk graphs is 6.

The bound for planar graphs follows immediately from the above and the fact that planar graphs are disk graphs of ply 1 [21,24]. □

A PTAS for MDS on planar graphs is known [3], but we are not aware of any previous results on the integrality gap of the LP relaxation for this class of graphs.

By using Bar-Yehuda and Even's approximation algorithm for MSC instances of bounded element frequency [4], we can give a linear-time $(54 \cdot \gamma)$-approximation algorithm for Minimum Dominating Set on disk intersection graphs of ply γ.

We can improve on the $O(\gamma)$ ratio given above by using the shifting technique. One way is to approximate Minimum $\preceq_{\mathsf{Leb}}$-Dominating Set.

Theorem 10. *Minimum $\preceq_{\mathsf{Leb}}$-Dominating Set on disk graphs of bounded ply, i.e. of ply $\gamma = \gamma(n) = o(\log n)$, has an FPTAS$^{\omega}$. Hence Minimum Dominating Set on disk graphs of bounded ply has a 6-FPTAA$^{\omega}$.*

The proof of Theorem 10 is omitted. Instead we use similar ideas to give a simpler algorithm for Minimum Dominating Set with better approximation ratio. The algorithm uses a new variant of the classic geometric shifting technique [17,26]. Assume the disks in a set $\mathcal{D}$ are scaled such that the smallest disk has radius $\frac{1}{2}$. Partition the disks into levels. A disk with radius r has level j ($j \in \mathbb{Z}_{\geq 0}$) if $2^{j-1} \leq r < 2^j$. The level of the largest disk is denoted by l. Define $\mathcal{D}_{=j}$ as the set of disks in $\mathcal{D}$ having level j. Similarly, $\mathcal{D}_{\geq j}$ denotes the set of disks having level at least j, and so on.

For each level j, define a grid by lines $y = hk2^j$ and $x = vk2^j$ ($h, v \in \mathbb{Z}$) for some $k \geq 9$ (an odd multiple of 3), whose value we determine later. The grid partitions the plane into squares of size $k2^j \times k2^j$, called *j-squares*. A j-square is contained in precisely one $(j+1)$-square and each $(j+1)$-square contains exactly four j-squares. Let $\mathcal{D}^S$ denote the set of disks intersecting a j-square S and $\mathcal{D}^{\mathrm{b}(S)}$ the set of disks intersecting the boundary of S. Similarly, $\mathcal{D}^{\mathrm{i}(S)} = \mathcal{D}^S - \mathcal{D}^{\mathrm{b}(S)}$ is the set of disks fully inside S. Combinations such as $\mathcal{D}^{\mathrm{b}(S)}_{=j}$ should be self-explanatory. The level of a square S is denoted $j(S)$. Let $\mathcal{D}^b = \bigcup_S \mathcal{D}^{\mathrm{b}(S)}_{=j(S)}$ be the set of disks intersecting the boundary of a j-square at their level.

Theorem 11. *Let $\mathcal{D}$ be a set of n disks of ply γ, $k \geq 9$ an odd multiple of 3, and OPT a minimum dominating set. Then in time $O(k^2 n^2\, 3^{32k\gamma/\pi} 2^{16k\gamma/\pi} 4^{16(k+1)\gamma/\pi})$, we can find a set $DS \subseteq \mathcal{D}$ dominating $\mathcal{D} - \mathcal{D}^b = \bigcup_S \mathcal{D}^{\mathrm{i}(S)}_{=j(S)}$ such that $|DS| \leq \sum_S |OPT^S_{=j(S)}|$, where the union and the sum is over all squares S.*

The proof of this theorem is quite involved and is omitted due to space limitations.

The shifting technique is applied in the following novel way. For an integer a ($0 \leq a \leq k-1$), a line of level j is *active* if it has the form $y = (hk + a2^{l-j})2^j$ or $x = (vk + a2^{l-j})2^j$ ($h, v \in \mathbb{Z}$). The active lines partition the plane into j-squares as before, but are shifted w.r.t. a. However, we can still use the algorithm of Theorem 11.

Let DS_a denote the set returned by the algorithm for the j-squares induced by a and let $\mathcal{D}_a^b$ be the set D^b for these j-squares ($0 \leq a \leq k-1$). We join three such sets to ensure we dominate the entire graph. So let $DS_i^3 = DS_i \cup DS_{i+k/3} \cup DS_{i+2k/3}$ for each $i = 0, \ldots, k/3 - 1$. This is properly defined, as k is a multiple of 3. Denote the smallest DS_i^3 by $DS_{\min}^3$.

Theorem 12. *There is a* 3*-FPTAA*$^\omega$ *for Minimum Dominating Set on disk graphs of bounded ply, i.e. of ply* $\gamma = \gamma(n) = o(\log n)$*. If* $\gamma = O(1)$*, there is a* $(3+\epsilon)$*-approximation algorithm for any fixed* $\epsilon > 0$.

Proof. We first show DS_i^3 is a dominating set of $\mathcal{D}$, for any $i \in \{0, \ldots, k/3 - 1\}$. A level j disk is in $\mathcal{D}_a^b$ if and only if it intersects an active line of level j for a. We know ([26], Lemma 9) that any disk intersects an active horizontal line for at most two (consecutive) values of a and an active vertical line for at most two (consecutive) values of a. As $k \geq 9$ is an odd multiple of 3, $k/3 > 1$, and thus $i, i+k/3, i+2k/3$ are non-consecutive integers (modulo k). Hence any disk is in at most two of the sets $\mathcal{D}_i^b, \mathcal{D}_{i+k/3}^b, \mathcal{D}_{i+2k/3}^b$. Theorem 11 shows that DS_a is a dominating set for $\mathcal{D} - \mathcal{D}_a^b$. Given the previous argument, $(\mathcal{D} - \mathcal{D}_i^b) \cup (\mathcal{D} - \mathcal{D}_{i+k/3}^b) \cup (\mathcal{D} - \mathcal{D}_{i+2k/3}^b) = \mathcal{D}$. Then DS_i^3 is a dominating set of $\mathcal{D}$.

We now prove $\left|DS_{\min}^3\right| \leq (3 + \frac{36}{k})\,|OPT|$. A level j disk is in $\mathcal{D}_a^b$ for at most 4 values of a ([26], Lemma 9). Therefore $\sum_{a=0}^{k-1} \left|\mathcal{D}_a^b\right| \leq 4\,|\mathcal{D}|$ and $\sum_{a=0}^{k-1} \left|OPT \cap \mathcal{D}_a^b\right| \leq 4\,|OPT|$.

Also, for fixed a, any level j disk intersects at most 4 j-squares. Hence $|DS_a| \leq \sum_S \left|OPT_{=j(S)}^S\right| \leq |OPT| + 3\left|OPT \cap \mathcal{D}_a^b\right|$ and thus

$$\tfrac{1}{3}k\left|DS_{\min}^3\right| \leq \sum_{i=0}^{k/3-1} \left|DS_i^3\right| \leq \sum_{a=0}^{k-1} \left(|OPT| + 3\left|OPT \cap \mathcal{D}_a^b\right|\right) \leq (k+12)\,|OPT|\,.$$

Then $DS_{\min}^3 \leq (3 + \frac{36}{k})\,|OPT|$. Choose k as the smallest odd multiple of 3 greater than 9 and $\frac{36}{\epsilon}$ and apply Theorem 11 to get the $(3+\epsilon)$-approximation for constant ply and fixed ϵ. The 3-FPTAA$^\omega$ is obtained along similar lines using methods of [26]. □

Analogously, we can obtain 3-FPTAA$^\omega$'s for arbitrary fat objects of bounded ply. The algorithms of this section extend to d-dimensional fat objects for any constant d. We do not know if the shifting technique can be used to give a constant approximation (or even a PTAS) for MDS on disk graphs of arbitrary ply, because (1) there is no upper bound on the number of 'large' disks intersecting a j-square in the dominating set, and (2) we cannot track which j-square is 'responsible' for dominating a disk intersecting more than one j-square on its level. We avoided (1) by assuming bounded ply and (2) by considering $\preceq_{\mathsf{Leb}}$-dominating sets (Thm. 10), or by disregarding the domination of disks intersecting a boundary on their level and combining three result sets (Thm. 12).

5 Hardness Results

The approximation schemes [6,13,26] for Maximum Independent Set and Minimum Vertex Cover on disk intersection graphs extend easily to fat object intersection graphs. It is unlikely that an approximation algorithm for MDS would extend this way, as on intersection graphs of fat objects that are almost disks, MDS becomes hard to approximate. A convex subset s of $\mathbb{R}^2$ is *α-fat* for some $\alpha \geq 1$ if the ratio between the radii of the smallest disk circumscribing s and the largest disk inscribed in s is at most α [12].

Theorem 13. *For any $\alpha > 1$ and any $\epsilon > 0$, MDS on α-fat object intersection graphs is not approximable within $(1-\epsilon)\ln n$, unless $NP \subset DTIME(n^{O(\log\log n)})$.*

Proof. Reduce from Minimum Set Cover (MSC). For instance x of MSC with universe $\mathbb{U}$ and collection $\mathcal{F} = \{S_1, \ldots, S_m\}$ of subsets of $\mathbb{U}$, construct instance y of MDS on α-fat object intersection graphs as in Fig. 2. Each $u \in \mathbb{U}$ corresponds to a 'small' disk d_u. Each S_j corresponds to a disk d_j with the top replaced by a polyhedral structure such that d_j intersects d_u if and only if $u \in S_j$. Packing the d_u close together makes the fatness of the construction arbitrarily close to 1. As any object dominated by a d_u is also dominated by a d_j for which $u \in S_j$, we have $|OPT_x| = |OPT_y|$. Constructing y takes time polynomial in $|\mathbb{U}|$ and m. The theorem follows from Feige [15,8]. □

An object has *constant description complexity* if it is a semialgebraic set defined by a constant number of polynomial (in)equalities of constant maximum degree [12]. The objects modeling the S_j are the intersection of a disk with a polyhedron (each d_j can be described by one quadratic inequality and $|S_j| + 1$ linear inequalities) and might not have constant description complexity. So for constant description complexity objects, better approximation ratios than $\ln n$ could be attained. However, we can prove APX-hardness by reducing from Minimum k-Set Cover, the variant of MSC where $|S_j| \leq k$ for any $S_j \in \mathcal{F}$. This problem is APX-hard for $k = 3$ (follows e.g. from [1]). Using the same gadget as before, the objects of Theorem 13 have constant description complexity.

Theorem 14. *For any $\alpha > 1$, MDS on α-fat, constant description complexity object intersection graphs is APX-hard. Hence it has no PTAS (unless P=NP).*

These results say something about intersection graphs of fat objects in general, and of fat almost disks in particular. But we can easily prove similar results for almost squares, almost bounded aspect ratio rectangles, almost triangles, etc. Basically, if we slightly relax the shape constraints for a given object, Minimum Dominating Set on the intersection graphs of such relaxed objects is hard to approximate.

The above reductions can also be used to prove the hardness of other problems, including Minimum Connected Dominating Set and Minimum Total Dominating Set. Furthermore, by replacing each disk d_u in the reductions by a point, we obtain theorems equivalent to Theorem 13 and 14 for the $\ln n$-hardness of Geometric Set Cover on general α-fat objects and α-fat almost disks, almost squares, etc. and APX-hardness if these objects have constant description complexity.

Finally, we solve an open problem of Chlebík and Chlebíková [9] by proving that Minimum Dominating Set is APX-hard for intersection graphs of 2-dimensional boxes. The reduction can be extended to Minimum Connected/Total Dominating Set, to ellipse intersection graphs, and to Geometric Set Cover on rectangles and ellipses.

Theorem 15. *MDS on rectangle intersection graphs (MDSr) is APX-hard.*

Proof. We give an L-reduction from the APX-hard [1] problem Minimum Vertex Cover on graphs $G = (\{v_1, \ldots, v_n\}, E)$ of degree three (MVC3) to MDS in a rectangle intersection graph G'. Rectangles R_i^h and R_i^v represent vertex v_i, and are connected by three *plates*, the largest of which is the *big plate* P_i (see Fig. 3). Edge $(v_i, v_j) \in E$ for $i < j$ corresponds to rectangle $S_{i,j}$ in the intersection of rectangles R_i^v and R_j^h.

Let C be a minimum vertex cover of G and let $k = |C|$. Construct a set D from C by adding R_i^h and R_i^v to D for each $v_i \in C$ and adding P_i for each $v_i \notin C$. By the construction of D, all R_i^h, R_i^v, and all plates are dominated. As C is a vertex cover, D dominates each $S_{i,j}$. Since the graph has degree three, $|C| \geq n/4$, and thus $|D| \leq 2|C| + (n - |C|) = n + k \leq 5k$.

Let D be a dominating set of G'. We can assume that D contains only rectangles of type R_i^h, R_i^v and P_i. Construct a set C from D by adding v_i to D if R_i^h or R_i^v is in D. Because D dominates all $S_{i,j}$, C is a vertex cover. Let $R^2[D]$ be the set of rectangles for v_i for which both R_i^h and R_i^v occur in D, $R^1[D]$ the set for v_i for which only one of R_i^h and R_i^v occurs in D, and $P[D]$ the set of big plates in D. To dominate all small plates, $|P[D]| + \left|R^2[D]\right|/2 \geq n$. Then $|D| \geq |P[D]| + \left|R^1[D]\right| + \left|R^2[D]\right| \geq n + \left|R^1[D]\right| + \left|R^2[D]\right|/2 \geq n + k$. Hence $OPT_{MDSr}(G') = n + k$. Suppose $|D| = OPT_{MDSr}(G') + c$, for a certain $c \geq 0$. Then $|D| = n + k + c$ and thus $\left|R^1[D]\right| + \left|R^2[D]\right|/2 + n \leq n + k + c$, implying $|C| - OPT_{MVC3}(G) \leq c$. $\square$

6 Conclusion

The immediate open question is whether Minimum Dominating Set admits a constant-factor approximation algorithm or even a PTAS for disk graphs of arbitrary ply. The hardness results of Sect. 5 show that for objects whose boundaries can intersect an arbitrary number of times, MDS is very hard to approximate. On the contrary, if object boundaries intersect at most twice (i.e. the objects are pseudo-disks), the decomposition bound is linear and at least for cases such as r-regular polygons with constant r or rectangles with bounded aspect-ratio, we get constant-factor approximation algorithms. An intriguing question is whether MDS on disk graphs is harder to approximate than for other intersection graph classes such as intersection graphs of squares, or whether the algorithmic ideas can be extended to disks or maybe even to arbitrary pseudo-disks (the decomposition bound starts to fail ‘naturally’ beyond pseudo-disks).

References

1. Alimonti, P., Kann, V.: Some APX-completeness results for cubic graphs. Theoret. Comput. Sci. 237(1-2), 123–134 (2000)
2. Ambühl, C., Erlebach, T., Mihalák, M., Nunkesser, M.: Constant-Factor Approximation for Minimum-Weight (Connected) Dominating Sets in Unit Disk Graphs. In: Díaz, J., Jansen, K., Rolim, J.D., Zwick, U. (eds.) Proc. APPROX-RANDOM 2006. LNCS, vol. 4110, pp. 3–14. Springer-Verlag, Berlin/Heidelberg (2006)
3. Baker, B.S.: Approximation Algorithms for NP-Complete Problems on Planar Graphs. J. ACM 41(1), 153–180 (1994)

4. Bar-Yehuda, R., Even, S.: A Linear-Time Approximation Algorithm for the Weighted Vertex Cover Problem. J. Algorithms 2(2), 198–203 (1981)
5. Brönnimann, H., Goodrich, M.T.: Almost Optimal Set Covers in Finite VC-Dimension. Discrete Comput. Geometry 14(4), 463–479 (1995)
6. Chan, T.M.: Polynomial-time Approximation Schemes for Packing and Piercing Fat Objects. J. Algorithms 46(2), 178–189 (2003)
7. Chang, M.-S.: Efficient Algorithms for the Domination Problems on Interval and Circular-Arc Graphs. SIAM J. Comput. 27(6), 1671–1694 (1998)
8. Chlebík, M., Chlebíková, J.: Approximation Hardness of Dominating Set Problems. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 192–203. Springer-Verlag, Berlin/Heidelberg (2004)
9. Chlebík, M., Chlebíková, J.: The Complexity of Combinatorial Optimization Problems on d-Dimensional Boxes. SIAM J. Discrete Math. 21(1), 158–169 (2007)
10. Clark, B.N., Colbourn, C.J., Johnson, D.S.: Unit Disk Graphs. Discrete Math. 86(1–3), 165–177 (1990)
11. Clarkson, K.L., Varadarajan, K.R.: Improved Approximation Algorithms for Geometric Set Cover. Discrete Comput. Geometry 37(1), 43–58 (2007)
12. Efrat, A., Sharir, M.: The Complexity of the Union of Fat Objects in the Plane. Discrete Comput. Geometry 23(2), 171–189 (2000)
13. Erlebach, T., Jansen, K., Seidel, E.: Polynomial-time Approximation Schemes for Geometric Intersection Graphs. SIAM J. Comput. 34(6), 1302–1323 (2005)
14. Even, G., Rawitz, D., Sharar, S.: Hitting Sets when the VC-Dimension is Small. Inform. Process. Lett. 95(2), 358–362 (2005)
15. Feige, U.: A Threshold of $\ln n$ for Approximating Set Cover. J. ACM 45(4), 634–652 (1998)
16. Hochbaum, D.S.: Approximation Algorithms for the Set Covering and Vertex Cover Problems. SIAM J. Comput. 11(3), 555–556 (1982)
17. Hochbaum, D.S., Maass, W.: Approximation Schemes for Covering and Packing Problems in Image Processing and VLSI. J. ACM 32(1), 130–136 (1985)
18. Hunt III, D.B., Marathe, M.V., Radhakrishnan, V., Ravi, S.S., Rosenkrantz, D.J., Stearns, R.E.: NC-Approximation Schemes for NP- and PSPACE-Hard Problems for Geometric Graphs. J. Algorithms 26(2), 238–274 (1998)
19. Kedem, K., Livne, R., Pach, J., Sharir, M.: On the Union of Jordan Regions and Collision-Free Translational Motion Amidst Polygonal Obstacles. Discrete Comput. Geometry 1, 59–70 (1986)
20. Kim, S.-J., Kostochka, A., Nakprasit, K.: On the Chromatic Number of Intersection Graphs of Convex Sets in the Plane. Electr. J. Combinatorics 11, #R52 (2004)
21. Koebe, P.: Kontaktprobleme der konformen Abbildung. Ber. Ver. Sächs. Ak. Wiss. Leipzig, Math.-Phys. Kl. 88, 141–164 (1936)
22. Marathe, M.V., Breu, H., Hunt III, H.B., Ravi, S.S., Rosenkrantz, D.J.: Simple Heuristics for Unit Disk Graphs. Networks 25, 59–68 (1995)
23. Marx, D.: Parameterized Complexity of Independence and Domination on Geometric Graphs. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 154–165. Springer-Verlag, Berlin/Heidelberg (2006)
24. Miller, G.L., Teng, S.-H., Thurston, W., Vavasis, S.A.: Separators for Sphere-Packings and Nearest Neighbor Graphs. J. ACM 44(1), 1–29 (1997)
25. Nieberg, T., Hurink, J.L.: A PTAS for the Minimum Dominating Set Problem in Unit Disk Graphs. In: Erlebach, T., Persinao, G. (eds.) WAOA 2005. LNCS, vol. 3879, pp. 296–306. Springer-Verlag, Berlin/Heidelberg (2006)
26. van Leeuwen, E.J.: Better Approximation Schemes for Disk Graphs. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 316–327. Springer-Verlag, Berlin/Heidelberg (2006)

An Efficient Quantum Algorithm for the Hidden Subgroup Problem in Nil-2 Groups*

Gábor Ivanyos[1], Luc Sanselme[2], and Miklos Santha[3]

[1] SZTAKI, Hungarian Academy of Sciences, H-1111 Budapest, Hungary
[2] LRI, UMR 8623, Université Paris-Sud, Orsay, France, F-91405
[3] CNRS–LRI, Université Paris–Sud, 91405 Orsay, France and Centre for Quantum Technologies, National University of Singapore

Abstract. In this paper we extend the algorithm for extraspecial groups in [12], and show that the hidden subgroup problem in nil-2 groups, that is in groups of nilpotency class at most 2, can be solved efficiently by a quantum procedure. The algorithm presented here has several additional features. It contains a powerful classical reduction for the hidden subgroup problem in nilpotent groups of constant nilpotency class to the specific case where the group is a p-group of exponent p and the subgroup is either trivial or cyclic. This reduction might also be useful for dealing with groups of higher nilpotency class. The quantum part of the algorithm uses well chosen group actions based on some automorphisms of nil-2 groups. The right choice of the actions requires the solution of a system of quadratic and linear equations. The existence of a solution is guaranteed by the Chevalley-Warning theorem, and we prove that it can also be found efficiently.

1 Introduction

Efficient solutions to some cases of the hidden subgroup problem (HSP), a paradigmatic group theoretical problem, constitute probably the most notable success of quantum computing. The problem consists in finding a subgroup H in a finite group G hidden by some function which is constant on each coset of H and is distinct in different cosets. The hiding function can be accessed by an oracle, and in the overall complexity of an algorithm, a query counts as a single computational step. To be efficient, an algorithm has to be polylogarithmic in the order of G. While classically not even query efficient algorithms are known for the HSP, it can be solved efficiently in abelian groups by a quantum algorithm. A detailed description of the so called standard algorithm can be found for example in [19]. The main quantum tool of this algorithm is Fourier sampling, based on the efficiently implementable Fourier transform in abelian groups. Factorization and discrete logarithm [23] are special cases of this solution.

* Research supported by the European Commission IST Integrated Project Qubit Applications (QAP) 015848, the OTKA grants T42559 and T46234, the NWO visitor's grant Algebraic Aspects of Quantum Computing, and the ANR Blanc AlgoQP grant of the French Research Ministry.

After the settling of the abelian case, substantial research was devoted to the HSP in some finite non-abelian groups. Beside being the natural generalization of the abelian case, the interest of this problem is enhanced by the fact, that important algorithmic problems, such as graph isomorphism, can be cast in this framework. The standard algorithm has been extended to some non-abelian groups by Rötteler and Beth [21], Hallgren, Russell and Ta-Shma [8], Grigni, Schulman, Vazirani and Vazirani [6] and Moore, Rockmore, Russell and Schulman [17]. For the Heisenberg group, Bacon, Childs and van Dam [1] used the pretty good measurement to reduce the HSP to some matrix sum problem that they could solve classically. Ivanyos, Magniez and Santha [11] and Friedl, Ivanyos, Magniez, Santha and Sen [5] have efficiently reduced the HSP in some non-abelian groups to HSP instances in abelian groups using classical and quantum group theoretical tools, but not the non-abelian Fourier transform. This latter approach was used recently by Ivanyos, Sanselme and Santha [12] for extraspecial groups.

The so far unknown complexity of two special cases of the HSP would be of particular interest. The first one is the hidden subgroup problem in the symmetric group because it contains as special instance the graph isomorphims problem. Recently Moore, Russell and Sniady [18] have shown that no algorithm based one a particular approach can solve the graph isomorhism problem efficiently. The other one is the hidden subgroup problem in the dihedral group because of its relation to certain lattice problems investigated by Regev [20].

In this work we extend the class of groups where the HSP is efficiently solvable by a quantum algorithm to nilpotent groups of nilpotency class at most 2 (shortly nil-2 groups). These are groups whose lower (and upper) central series are of length at most 2. Equivalently, a group is nil-2 group if the derived group is a subgroup of the center. Nilpotent groups form a rich subclass of solvable groups, they contain for example all (finite) p-groups. Extraspecial groups are, in particular, in nil-2 groups. Our main result is:

Theorem 1. *Let G be a nil-2 group.Let us given an oracle f which hides the subgroup H of G. Then there is an efficient quantum procedure which finds H.*

The overall structure of the algorithm presented here is closely related to the algorithm in [12] for extraspecial groups, but has also several additional features. The quantum part of the algorithm is restricted to specific nil-2 groups, which are also p-groups and are of exponent p. It consists essentially in the creation of a quantum hiding procedure (a natural quantum generalization of a hiding function) for the subgroup HG' of G. The procedure uses certain automorphisms of the groups to define some appropriate group actions, and is analogous to what have been done in [12] for extraspecial p-groups of exponent p.

While dealing with extraspecial p-groups of exponent p basically solves the HSP for all extraspecial groups (the case of remaining groups, of exponent p^2, easily reduces to groups of exponent p), this is far from being true for nil-2 groups. Indeed, one of the main new features of the current algorithm is a classical reduction of the HSP in nil-2 groups to the HSP in nil-2 p-groups of exponent p, where moreover the hidden subgroup is either trivial or of cardinality p. In fact, our result is much more general: we prove an analogous reduction in nil-k

groups for any constant k. We believe that this general reduction might be useful for designing efficient quantum algorithms for the HSP in groups of higher nilpotency class.

Our second main novel feature concerns the quantum hiding procedure. While in extraspecial groups it was reduced to the efficient solvability of a single quadratic and a single linear equation modulo p, here we look for a nontrivial solution of a homogeneous system of d quadratic and d linear equations, where d can be any integer. The reason for this is that while in extraspecial groups the derived subgroup is one dimensional, in nil-2 groups we have no a priori bound on its dimension. If the number of variables is superior to the global degree of the system then the solvability itself is an immediate consequence of the Chevalley-Warning theorem [3,24]. In fact, we are in presence of a typical example of Papdimitriou's complexity class of total functions [16]: the number of solutions is divisible by p and therefore there is always a nontrivial one. Our result is that if the number of variables is sufficiently large, more precisely is of $O(d^3)$, then we can also find a nontrivial solution in polynomial time.

The structure of the paper is the following. In Section 2 we shortly describe the extension of the standard algorithm for quantum hiding procedures, and then we discuss some basic properties of nilpotent groups, in particular nil-2 p-groups of exponent p. Section 3 contains the description of the classical reduction of the HSP in groups of constant nilpotency class to instances where the group is also p-group of exponent p, and the subgroup is either trivial or cyclic of order p (Theorem 2). Section 4 gives the description of the quantum algorithm in nil-2 p-groups of exponent p: Theorem 3 briefly describes the reduction to the design of an efficient hiding procedure for HG', and Theorem 4 proves the existence of such a procedure. Finally Section 5 gives the proof of Theorem 5, the efficient solvability of the system of quadratic and linear equations. The proof of Theorem 1 follows from Corollary 1 and Theorems 3 and 4.

Even if the hidden subgroup problem is hard for the symmetric group and also for general solvable groups, it may happen that there is an efficient solution in nilpotent groups. The works [1,12] and this paper can be considered as the first steps in investigating the complexity of the HSP in that group family.

2 Preliminaries

2.1 Extension of the Standard Algorithm for the Abelian HSP

We will use standard notions of quantum computing for which one can consult for example [15]. For a set X, let $|X\rangle = \frac{1}{\sqrt{|X|}} \sum_{x \in X} |x\rangle$. We denote by $\mathsf{supp}(|\Psi\rangle)$ the support of $|\Psi\rangle$, that is the set of basis elements with non-zero amplitude.

The standard algorithm for the abelian HSP repeats polynomially many times the Fourier sampling involving the same hiding function, to obtain in each iteration a random element from the subgroup orthogonal to the hidden subgroup. In fact, for the repeated Fourier samplings, the existence of a common hiding function can be relaxed in several ways. Firstly, in different iterations different hiding functions can be used, and secondly, classical hiding functions can be replaced by quantum hiding functions. This was formalized in [12], and we recall here the precise definition. A set of vectors $\{|\Psi_g\rangle : g \in G\}$ from some Hilbert space $\mathcal{H}$ is a *hiding set* for the subgroup H of G if:

- $|\Psi_g\rangle$ is a unit vector for every $g \in G$,
- if g and g' are in the same left coset of H then $|\Psi_g\rangle = |\Psi_{g'}\rangle$,
- if g and g' are in different left cosets of H then $|\Psi_g\rangle$ and $|\Psi_{g'}\rangle$ are orthogonal.

A quantum procedure is *hiding* the subgroup H of G if for every $g_1, \ldots, g_N \in G$, on input $|g_1\rangle \ldots |g_N\rangle|0\rangle$ it outputs $|g_1\rangle \ldots |g_N\rangle|\Psi^1_{g_1}\rangle \ldots |\Psi^N_{g_N}\rangle$, where $\{|\Psi^i_g\rangle : g \in G\}$ is a hiding set for H for all $1 \le i \le N$.

The following fact whose proof is immediate from Lemma 1 in [11] recasts the existence of the standard algorithm for the abelian HSP in the context of hiding sets.

Fact 1. *Let G be a finite abelian group. If there exists an efficient quantum procedure which hides the subgroup H of G then there is an efficient quantum algorithm for finding H.*

2.2 Nilpotent Groups

Let G be a finite group. For two elements g_1 and g_2 of G, we usually denote their product by g_1g_2. If we conceive group multiplication from the right as a group action of G on itself, we will use the notation $g_1 \cdot g_2$ for g_1g_2. We write $H \le G$ when H is a subgroup of G, and $H < G$ when it is a proper subgroup. Normal subgroups and proper normal subgroups will be denoted respectively by $H \trianglelefteq G$ and $H \triangleleft G$. For a subset X of G, let $\langle X\rangle$ be the subgroup generated by X. The *normalizer* of X in G is $N_G(X) = \{g \in G \ : \ gX = Xg\}$. For an integer n, we denote by $\mathbb{Z}_n$ the group of integers modulo n, and for a prime number p, we denote by $\mathbb{Z}_p^*$ the multiplicative group of integers relatively prime with p.

The commutator $[x, y]$ of elements x and y is $x^{-1}y^{-1}xy$. For two subgroups X and Y of G, let $[X, Y]$ be $\langle\{[x, y] \ : \ x \in X, y \in Y\}\rangle$. The derived subgroup G' of G is defined as $[G, G]$, and its center $Z(G)$ as $\{z \in G \ : \ gz = zg \text{ for all } g \in G\}$. The *lower central series* of G is the series of subgroups $G = A_1 \trianglerighteq A_2 \trianglerighteq A_3 \ldots$, where $A_{i+1} = [A_i, G]$ for every $i > 1$. The *upper central series* of G is the series of subgroups $\{1\} = Z_0 \trianglelefteq Z_1 \trianglelefteq Z_2 \ldots$, where $Z_{i+1} = \{x \in G \ : \ [x, g] \in Z_i \text{ for all } g \in G\}$ for every $i > 0$. Clearly $A_2 = G'$ and $Z_1 = Z(G)$. The group G is *nilpotent* if there is a natural number n such that $A_{n+1} = \{1\}$. If n is the smallest integer such that $A_{n+1} = \{1\}$ then G is *nilpotent of class n*. It is a well known fact that G is nilpotent of class n if and only if $Z_n = G$ in the upper central series. Nilpotent groups of class 1 are simply the nontrivial abelian groups. A nilpotent group of class at most n is called a *nil-n* group.

A detailed treatment of nilpotent groups can be found for example in Hall [7]. Let us just recall here that nilpotent groups are solvable, and that every p-group is nilpotent, where a p-group is a finite group whose order is a power of some prime number p.

2.3 Nil-2 p-Groups of Exponent p

It is clear from the definition of nilpotent groups that G is a nil-2 group exactly when $G' \le Z(G)$. It is easy to see that this property implies that the commutator is a bilinear function in the following sense: for every g_1, g_2, g_3, g_4 in G, we have $[g_1g_2, g_3g_4] = [g_1, g_2][g_1, g_3][g_2, g_3][g_2, g_4]$.

The quantum part of our algorithm will deal only with special nilpotent groups of class 2, which are also p-groups of exponent p. The structure of these special groups is well known, and is expressed in the following simple fact.

Fact 2. *Let G be a p-group of exponent p and of nilpotency class 2. Then there exist positive integers m and d, group elements $x_1, \ldots, x_m \in G$ and $z_1, \ldots, z_d \in G'$ such that:*
(1) $G/G' \cong \mathbb{Z}_p^m$ *and* $G' \cong \mathbb{Z}_p^d$,
(2) $\forall g \in G,\ \exists!(e_1, \ldots, e_m, f_1, \ldots f_d) \in \mathbb{Z}_p^{m+d}$ *such that* $g = x_1^{e_1} \ldots x_m^{e_m} z_1^{f_1} \ldots z_d^{f_d}$,
(3) $G = \langle x_1, \ldots, x_m \rangle$ *and* $G' = \langle z_1, \ldots, z_d \rangle$.

We will say that a nil-2 p-group G of exponent p has parameters (m, d) if $G/G' \cong \mathbb{Z}_p^m$ and $G' \cong \mathbb{Z}_p^d$. In those groups we will indentify G' and $\mathbb{Z}_p^d$. Thus, for two elements z and z' of G', the product zz' is just $z \oplus z'$ where $\oplus$ denotes the coordinate-wise addition modulo p. If G is a such a group then $|G| = p^{m+d}$. The elements of G can be encoded by binary strings of length $O((m+d)\log p)$, and an efficient algorithm on input G has to be polynomial in m, d and $\log p$.

For $j = 1, \ldots, p-1$, we consider on generators the maps x_i to x_i^j. It turns out that these maps extend to automorphisms ϕ_j of G. We also define the map ϕ_0 by letting $\phi_0(g) = 1$, for every $g \in G$.

Proposition 1. *Let G be a p-group of exponent p and of nilpotency class 2. Then the mappings ϕ_j have the following properties:*

$$(1)\ \forall j \in \mathbb{Z}_p, \forall z \in G', \quad \phi_j(z) = z^{j^2},$$
$$(2)\ \forall g \in G, \exists z_g \in G', \forall j \in \mathbb{Z}_p, \quad \phi_j(g) = g^j z_g^{j-j^2}.$$

3 Groups of Constant Nilpotency Class: Classical Reductions

In order to present the reduction methods in a sufficiently general way, in this section we assume that our groups are presented in terms of so-called *refined polycyclic presentations* (RPP) [9]. Such a presentation of a finite solvable group G is based on a sequence $G = G_1 \rhd \ldots \rhd G_{s+1} = \{1\}$, where for each $1 \leq i \leq s$ the subgroup G_{i+1} is a normal subgroup of G_i and the factor group G_i/G_{i+1} is cyclic of prime order r_i. For each $i \leq s$ we choose $g_i \in G_i \setminus G_{i+1}$. Then $g_i^{r_i} \in G_{i+1}$. Every element g of G can be uniquely represented as a product of the form $g_1^{e_1} \cdots g_s^{e_s}$, called the normal word for g, where $0 \leq e_i < r_i$.

In the abstract presentation the generators are $g_1, \ldots, g_s$, and for each index $1 \leq i \leq s$, the following relations are included:

- $g_i^{r_i} = u_i$, where $u_i = g_{i+1}^{a_{i,i+1}} \cdots g_s^{a_{i,s}}$ is the normal word (n.w.) for $g^{r_i} \in G_{i+1}$,
- $g_i^{-1} g_j g_i = w_{ij}$ for $j > i$, where $w_{ij} = g_{i+1}^{b_{i,j,i+1}} \cdots g_s^{b_{i,j,s}}$ is the n.w. for $g_i^{-1} g_j g_i \in G_{i+1}$.

Using a quantum implementation [11] of an algorithm of Beals and Babai [2], RPP for a solvable black box group can be computed in polynomial time. We assume that elements of G are encoded by normal words and there is a polynomial time algorithm in $\log|G|$, the so called *collection procedure*, which computes normal words representing products. This is the case for nilpotent groups of constant class [10]. If there is an efficient collection procedure then RPP for subgroups and factor groups can be obtained in polynomial time [9]. Also, the major notable subgroups including Sylow subgroups, the center and the commutator can be computed efficiently. Furthermore, in p-groups with RPP, normalizers of subgroups can be computed in polynomial time using the technique of [4], combined with the subspace stabilizer algorithm of [14].

Our first theorem is a classical reduction for the HSP in groups of constant nilpotency class. The proof is given by the subsequent three lemmas.

Theorem 2. *Let $\mathcal{C}$ be a class of groups of constant nilpotency class that is closed under taking subgroups and factor groups. Then the hidden subgroup problem in members of $\mathcal{C}$ can be reduced to the case where the group is a p-group of exponent p, and the the subgroup is either trivial or of cardinality p.*

Corollary 1. *The hidden subgroup problem in nil-2 groups can be reduced to the case where the group is a p-group of exponent p, and the the subgroup is either trivial or of cardinality p.*

Lemma 1. *Let $\mathcal{C}$ be a class of groups of constant nilpotency class that is closed under taking subgroups and factor groups. Then the HSP in $\mathcal{C}$ can be reduced to the HSP of p-groups belonging to $\mathcal{C}$.*

Lemma 2. *Let $\mathcal{C}$ be a class of p-groups of constant nilpotency class that is closed under taking subgroups and factor groups. Then the hidden subgroup problem in members of $\mathcal{C}$ can be reduced to the case where the subgroup is either trivial or of cardinality p.*

Proof. Assume that we have a procedure $\mathcal{P}$ which finds hidden subgroups in $\mathcal{C}$ under the promise that the hidden subgroup is trivial or is of order p. Let G be a group in $\mathcal{C}$ and let f be a function on G hiding the subgroup H of G. We describe an iterative procedure which uses $\mathcal{P}$ as a subroutine and finds H in G. The basic idea is to compute a refined polycyclic sequence $G = G_1 \rhd \ldots \rhd G_s \rhd 1$ for G and to proceed calling $\mathcal{P}$ on the subgroups in the sequence starting with G_s. When $\mathcal{P}$ finds for the first time a nontrivial subgroup generated by h, then we would like to restart the process in $G/\langle h\rangle$, and at the end, collect all the generators. Since $\langle h\rangle$ is not necessarily a normal subgroup of G we will actually restart the process instead in $N_G(\langle h\rangle)$. More formally, if f hides H in G, and $\widetilde{H}$ be some subgroup of H. Then f hides $N_G(\widetilde{H}) \cap H$ in $N_G(\widetilde{H})$, and therefore $(N_G(\widetilde{H}) \cap H)/\widetilde{H}$ in $N_G(\widetilde{H})/\widetilde{H}$. We consider the following algorithm:

Algorithm 1.

```
success:= TRUE, $\widetilde{H} = \{1\}$.
WHILE success=TRUE DO
  IF $G \neq \widetilde{H}$ THEN compute $N_G(\widetilde{H})/\widetilde{H} = G_1 \rhd \ldots \rhd G_s \rhd 1$ a RPP, $i := s$,
                             WHILE $i > 0$ DO call $\mathcal{P}$ on $G_i$,
                                              IF $\mathcal{P} \to \langle h\rangle$ THEN $\widetilde{H} := \langle \widetilde{H} \cup \{h\}\rangle, i := 0$
                                              ELSE $i := i - 1$
                                                     IF $i = 0$ THEN success := FALSE
  ELSE success:=FALSE
```

Algorithm 1 stops when the subgroup $\widetilde{H}$ is such that $(N_G(\widetilde{H}) \cap H)/\widetilde{H} = \{1\}$, that is when $N_G(\widetilde{H}) \cap H = \widetilde{H}$. We claim that this implies $\widetilde{H} = H$. Indeed, suppose that $\widetilde{H}$ is a proper subgroup of H. Since in nilpotent groups a proper subgroup is also a proper subgroup of its normalizer, $\widetilde{H}$ is also a proper subgroup of $N_H(\widetilde{H}) = N_G(\widetilde{H}) \cap H$.

Finally observe that the whole process makes $O(\log_p^2 |G|)$ calls to $\mathcal{P}$. □

Lemma 3. *Let $\mathcal{C}$ be a class of p-groups of constant nilpotency class that is closed under taking subgroups and factor groups. Then the instances of the hidden subgroup problem in members of $\mathcal{C}$, when the subgroup is either trivial or of cardinality p, can be reduced to groups in $\mathcal{C}$ of exponent p.*

Proof. If p is not larger than the class of G, the algorithm of [5] is applicable. Otherwise the elements of order p or 1 form a subgroup G^*, see Chapter 12 of [7]. The hidden subgroup H is also a subgroup of G^* since $|H| \leq p$. The function hiding H in G also hides it in G^*, therefore the reduction will consist in determining G^*.

We design an algorithm that finds G^* by induction on the length of RPP. If $|G| = p$ then $G^* = G$. Otherwise, let $G = G_1 \rhd G_2 \rhd \ldots \rhd G_s \rhd \{1\}$ be a RPP with $s \geq 2$. It is easy to construct a presentation where G_s is a subgroup of the center of G, which we suppose from now on. For the ease of notation we set $M = G_2$ and $N = G_s$.

We first describe the inductive step in a simplified case, with the additional hypothesis $(G/N)^* = G/N$. Observe that the hypothesis is equivalent to saying that the map $\phi : x \mapsto x^p$ sends every element of G into N. From this it is also clear that the hypothesis carries over to M, that is $(M/N)^* = M/N$. We further claim that either $G^* = G$ or G^* is a subgroup of G of index p. In fact this follows Theorem 12.4.4 of [7] which states that the map ϕ is constant on cosets of G^* and distinct on different cosets. From a polycyclic presentation of G it can be read off whether or not $G = G^*$. If $G^* = G$ we are done. Otherwise we compute inductively M^*. If $M^* = M$ then $G^* = M$. If M^* is a proper subgroup of M then M^* has index p^2 in G. Pick an arbitrary $u \in M \setminus M^*$ and $y \in G \setminus M$. By the assumptions, $u^p = g_s^{j_u}$ for some integer $0 < j_u < p$, and $y^p = g_s^{j_y}$ for some integer $0 \leq j_y < p$. Recall that in the polycyclic presentation model, computing normal words for u^p and y^p – using fast exponentiation – amounts to computing j_u and j_y. Set $x = u^{j_y j_u^{-1}}$. For this x we have $x^p = y^p$, and therefore $xy^{-1} \in G^*$. Since $xy^{-1} \in G^* \setminus M^*$, we have $G^* = \langle M^*, xy^{-1} \rangle$.

In the general case first $(G/N)^*$ is computed inductively. If $(G/N)^* = G/N$ then one proceeds as in the simplified case. Otherwise we set $K = (G/N)^* N$. We claim that $G^* = K^*$. For this we will show that. $G^* \subseteq K$. To see this, let x be an element of G^*. Then $x = yz$ where $y \in G/N$ and $z \in N$. We show that y is in $(G/N)^*$ which implies that $x \in K$. Indeed, $y^p = y^p z^p = (yz)^p = 1$, where the first equality follows from $|N| = p$, the second from $N \leq Z(G)$ and the third from $x \in G^*$. Finally observe that $(K/N)^* = K/N$ since $K/N = (G/N)^*$. Therefore one can determine K^* inductively as in the simplified case.

Let $c(s)$ denote the number of recursive calls when the length of a presentation is s. In the simplified case the number of calls is $s - 1$. Therefore in the general case we have $c(s) = c(s-1) + s - 2$, whose solution is $c(s) = O(s^2)$. □

4 The Quantum Algorithm

The quantum part of our algorithm, up to technicalities, follows the same lines as the algorithm given in [12] for extraspecial groups. Some proofs in this section are emitted, they are analogous to the ones given there.

Theorem 3. *Let G be a nil-2 p-group of exponent p, and let us given an oracle f which hides a subgroup H of G whose cardinality is either 1 or p. If we have an efficient quantum procedure (using f) which hides HG' in G then H can be found efficiently.*

Proof. First observe that finding H is efficiently reducible to finding HG'. Indeed, HG' is an abelian subgroup of G since H is abelian. The restriction of the hiding function f to HG' of G hides H. Therefore the standard algorithm for solving the HSP in abelian groups applied to HG' with oracle f yields H.

Let us now suppose that G has parameters (m, d). We will show that finding HG' can be efficiently reduced to the hidden subgroup problem in an abelian group. Let us denote for every element $g = x_1^{e_1} \ldots x_m^{e_m} z_1^{f_1} \ldots z_d^{f_d}$ of G, by $\overline{g}$ the element $x_1^{e_1} \ldots x_m^{e_m}$. We define the group $\overline{G}$ whose base set is $\{\overline{g} : g \in G\}$. Observe that this set of elements does not form a subgroup in G. To make $\overline{G}$ a group, its law is defined by $\overline{g_1} * \overline{g_2} = \overline{g_1 g_2}$ for all $\overline{g_1}$ and $\overline{g_2}$ in $\overline{G}$. It is easy to check that $*$ is well defined, and is indeed a group multiplication. In fact, the group $\overline{G}$ is isomorphic to G/G' and therefore is isomorphic to $\mathbb{Z}_p^m$. For our purposes a nice way to think about $\overline{G}$ as a representation of G/G' with unique encoding. Observe also that $HG' \cap \overline{G}$ is a subgroup of $(\overline{G}, *)$ because HG'/G' is a subgroup of G/G'. Since $HG' = (HG' \cap \overline{G})G'$, finding HG' is efficiently reducible to finding $HG' \cap \overline{G}$ in $\overline{G}$.

To finish the proof, let us remark that the procedure which hides HG' in G hides also $HG' \cap \overline{G}$ in $\overline{G}$. Since $\overline{G}$ is abelian, Fact 1 implies that we can find efficiently $HG' \cap \overline{G}$. $\square$

Theorem 4. *Let G be a nil-2 p-group of exponent p, and let us given an oracle f which hides a subgroup H of G. Then there is an efficient quantum procedure which hides HG' in G.*

Proof. The basic idea of the quantum procedure is the following. Suppose that we could create, for some $a \in G$, the coset state $|aHG'\rangle$. Then the group action $g \rightarrow |aHG' \cdot g\rangle$ is a hiding procedure. Unfortunately, $|aHG'\rangle$ can only be created efficiently when p and d are constant. In general, we can create efficiently $|aHG'_u\rangle$ for random $a \in G$ and $u \in G'$, where by definition $|G'_u\rangle = \frac{1}{\sqrt{|G'|}} \sum_{z \in \mathbb{Z}_p^d} \omega^{-<u,z>} |z\rangle$. Then $|aHG'_u \cdot h\rangle = |aHG'_u\rangle$ for every $h \in H$, and $|G'_u \cdot z\rangle = \omega^{<u,z>} |G'_u\rangle$. To cancel the disturbing phase we will use more sophisticated group action via the group automorphisms ϕ_j on several copies of the states $|aHG'_u\rangle$.

Lemma 4. *There is an efficient quantum procedure which creates $\frac{1}{\sqrt{p^d}} \sum_{u \in \mathbb{Z}_p^d} |u\rangle |aHG'_u\rangle$ where a is a random element from G.*

Now we claim that due to Proposition 1, the states $|aHG'_u\rangle$ are eigenvectors of the group action of multiplication from the right by $\phi_j(g)$, whenever g is from HG'.

Lemma 5. *We have*
(1) $\forall z \in \mathbb{Z}_p^d, \forall a \in G, \forall u \in \mathbb{Z}_p^d, \forall j \in \mathbb{Z}_p, \quad |aHG'_u \cdot \phi_j(z)\rangle = \omega^{<u,z>j^2} |aHG'_u\rangle$,
(2) $\forall h \in H, \forall a \in G, \forall u \in \mathbb{Z}_p^d, \forall j \in \mathbb{Z}_p, \quad |aHG'_u \cdot \phi_j(h)\rangle = \omega^{<u,z_h>(j-j^2)} |aHG'_u\rangle$.

The principal idea now is to take several copies of the states $|a_i HG'_{u_i}\rangle$ and choose the j_i so that the product of the corresponding eigenvalues becomes the unity. Therefore the combined actions $\phi_{j_i}(g)$, when g is from HG', will not modify the combined state. It turns out that we can achieve this with a sufficiently big enough number of copies. Let $n = n(d)$ some function of d to be determined later.

For $\overline{a} = (a_1, \ldots, a_n) \in G^n$, $\overline{u} = (u_1, \ldots, u_n) \in (\mathbb{Z}_p^d)^n$, $\overline{j} = (j_1, \ldots, j_n) \in (\mathbb{Z}_p)^n \setminus \{0^n\}$ and $g \in G$, we define the quantum state $|\Psi_g^{\overline{a},\overline{u},\overline{j}}\rangle$ in $\mathbb{C}^{G^n}$ by $|\Psi_g^{\overline{a},\overline{u},\overline{j}}\rangle = \bigotimes_{i=1}^n |a_i HG'_{u_i} \cdot \phi_{j_i}(g)\rangle$.

Our purpose is to find an efficient procedure to generate triples $(\overline{a}, \overline{u}, \overline{j})$ such that for every g in HG', $|\Psi_g^{\overline{a},\overline{u},\overline{j}}\rangle = \bigotimes_{i=1}^n |a_i HG'_{u_i}\rangle$. We call such triples *appropriate*. The reason to look for appropriate triples is that they lead to hiding sets for HG' in G as stated in the next lemma.

Lemma 6. *If $(\overline{a}, \overline{u}, \overline{j})$ is an appropriate triple then $\{|\Psi_g^{\overline{a},\overline{u},\overline{j}}\rangle : g \in G\}$ is hiding for HG' in G.*

Let us now address the question of existence of appropriate triples and efficient ways to generate them. Let $(\overline{a},\overline{u},\overline{j})$ be an arbitrary element of $G^n \times (\mathbb{Z}_p^d)^n \times (\mathbb{Z}_p)^n \setminus \{0^n\}$, and let g be an element of HG'. Then $g = hz$ for some $h \in H$ and $z \in \mathbb{Z}_p^d$, and $\phi_{j_i}(g) = \phi_{j_i}(h)\phi_{j_i}(z)$ for $i = 1,\ldots,n$. By Lemma 5, we have $|a_iHG'_{u_i} \cdot \phi_{j_i}(z)\rangle = \omega^{<u_i,z>j_i^2}|a_iHG'_{u_i}\rangle$, and $|a_iHG'_{u_i} \cdot \phi_{j_i}(h)\rangle = \omega^{<u_i,z_h>(j_i-j_i^2)}|a_iHG'_{u_i}\rangle$, and therefore $|\Psi_g^{\overline{a},\overline{u},\overline{j}}\rangle = \omega^{\sum_{i=1}^n <u_i,z_h>(j_i-j_i^2)+<u_i,z>j_i^2} \bigotimes_{i=1}^n |a_iHG'_{u_i}\rangle$.

For a given $\overline{u}$, we consider the following system of quadratic equations, written in vectorial form:

$$\begin{cases} \sum_{i=1}^n u_i(j_i - j_i^2) & = & 0^d \\ \sum_{i=1}^n u_i j_i^2 & = & 0^d. \end{cases}$$

It should be clear that when this system has a nontrivial solution $\overline{j}$ (that is $\overline{j} \neq 0^d$) then $(\overline{a},\overline{u},\overline{j})$ is an appropriate triple, for every $\overline{a}$. In fact, the Chevalley-Warning theorem [3,24] implies that the following equivalent system of vectorial equations has a nontrivial solution for every $\overline{u}$, whenever $n > 3d$.

$$\begin{cases} \sum_{i=1}^n u_i j_i^2 & = & 0^d \\ \sum_{i=1}^n u_i j_i & = & 0^d. \end{cases} \tag{1}$$

Moreover, if we take a substantially larger number of variables, we can find a solution in polynomial time.

Theorem 5. *If $n = (d+1)^2(d+2)/2$ then we can find a nontrivial solution for the system (1) in polynomial time.*

The proof of Theorem 5 will be given in the next section. To finish the proof of Theorem 4 we describe the efficient hiding procedure. On input $|g\rangle$, it computes, for some $\overline{a} \in G^n$, the superposition $\frac{1}{p^d} \bigotimes_{i=1}^n \sum_{u_i \in \mathbb{Z}_p} |u_i\rangle|a_iHG'_{u_i}\rangle$, which by Lemma 4 can be done efficiently, and then it measures the registers for the u_i. Then, by Theorem 5 it finds efficiently a nontrivial solution $\overline{j}$ for system (1). Such a triple $(\overline{a},\overline{u},\overline{j})$ is appropriate, and therefore by Lemma 6 $\{|\Psi_g^{\overline{a},\overline{u},\overline{j}}\rangle : g \in G\}$ is hiding for HG' in G. Using the additional input $|g\rangle$, the procedure finally computes $|\Psi_g^{\overline{a},\overline{u},\overline{j}}\rangle$. □

5 Solving the System of Equations

This section is fully dedicated to the proof of Theorem 5. If $p = 2$ then the d quadratic and the d linear equations coincide, and the (linear) system can easily be solved in polynomial time. Therefore, from now on, we suppose that $p > 2$. Let us detail system (1), where we set $u_i = (u_{1,i}, u_{2,i}, \ldots, u_{d,i})$. We have the following system of d homogenous quadratic and d homogenous linear one equations with n variables:

$$\begin{cases} \forall \ell \in [|1,d|], & \sum_{i=1}^n u_{\ell,i} j_i^2 & = & 0 \\ \forall \ell \in [|1,d|], & \sum_{i=1}^n u_{\ell,i} j_i & = & 0. \end{cases} \tag{2}$$

We start by considering only the quadratic part of the (2), that is for some integer n':

$$\begin{cases} \forall \ell \in [|1,d|], & \sum_{i=1}^{n'} u_{\ell,i} j_i^2 & = & 0. \end{cases} \tag{3}$$

Claim. If $n' = (d+1)(d+2)/2$ then we can find a nontrivial solution for (3) in polynomial time.

Proof. For the ease of notation we are going to represent this system by the $d \times n'$ matrix $M = (u_{\ell,i})_{1 \le \ell \le d, 1 \le i \le n'}$.

We will present a recursive algorithm whose complexity will be polynomial in d and in $\log p$. When $d = 1$, the unique quadratic equation is of the form $u_{1,1}j_1^2 + u_{1,2}j_2^2 + u_{1,3}j_3^2 = 0$. According to a special case of the main result in the thesis of van de Woestijne (Theorem A3 of [25]), a nontrivial solution for this can be found in polynomial time in $\log p$.

Let us suppose now that we have d equations in $n' = (d+1)(d+2)/2$ variables. We can make elementary operations on M (adding two lines and multiplying a line with a nonzero constant) without changing the solutions of the system. Our purpose is to reduce it with such operations to $d-1$ equations in at least $d(d+1)/2$ variables. If the system is of rank less than d, then we can erase an equation and get an equivalent system with only $d-1$ equations in the same number of variables. Otherwise, we perform Gaussian elimination resulting in the matrix

$$M_1 = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & u^{(1)}_{1,d+1} & \cdots & u^{(1)}_{1,n'} \\ 0 & 1 & 0 & \dots & 0 & u^{(1)}_{2,d+1} & \cdots & u^{(1)}_{2,n'} \\ \vdots & & \ddots & & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & 1 & 0 & u^{(1)}_{d-1,d+1} & \cdots & u^{(1)}_{d-1,n'} \\ 0 & \dots & 0 & 0 & 1 & u^{(1)}_{d,d+1} & \cdots & u^{(1)}_{d,n'} \end{pmatrix}.$$

Since checking quadratic residuosity is simple, and for odd p, half of the elements of $\mathbb{Z}_p^*$ are squares, we can easily compute a quadratic non-residue λ in probabilistic polynomial time. Then every quadratic non-residue is the product of a square and λ. We will look at column $d+1$ of M_1. If the column is everywhere 0 then $j_{d+1} = 1$ and $j_i = 0$ for $i \neq d+1$ is a nontrivial solution of the whole system. Otherwise, without loss of generality, we can suppose that for some $(k_1, k_2) \neq (0,0)$ the first k_1 elements are squares, the following k_2 elements are the product of λ and a square, and the last $d - k_1 - k_2$ elements are zero. Thus there exist $v_1, \dots, v_{k_1+k_2}$ different from 0, such that $u^{(1)}_{i,d+1} = v_i^2$ for $1 \le i \le k_1$, and $u^{(1)}_{i,d+1} = \lambda v_i^2$ for $k_1 + 1 \le i \le k_1 + k_2$. Once we have a quadratic non-residue, the square roots $v_1, \dots, v_{k_1+k_2}$ can be found in deterministic polynomial time in $\log p$ by the Shanks–Tonelli algorithm [22]. We set the variables $j_{k_1+k_2+1}, \dots, j_d$ to 0, and eliminate columns $k_1 + k_2 + 1, \dots, d$ from M_1. Then for $i = 1, \dots, k_1 + k_2$, we divide the line i by v_i^2. Introducing the new variables $j'_i = j_i v_i^{-1}$ for $1 \le i \le k_1 + k_2$, the matrix of the system in the $n' - d + k_1 + k_2$ variables $j'_1, \dots, j'_{k_1+k_2}, j_{d+1}, \dots j_{n'}$ is

$$M_2 = \begin{pmatrix} 1 & 0 & \dots & & 0 & 1 & u^{(2)}_{1,d+2} & \cdots & u^{(2)}_{1,n'} \\ 0 & \ddots & & & & \vdots & \vdots & & \vdots \\ & 1 & \ddots & & \vdots & 1 & u^{(2)}_{k_1,d+2} & \cdots & u^{(2)}_{k_1,n'} \\ \vdots & & \ddots & 1 & & \lambda & u^{(2)}_{k_1+1,d+2} & \cdots & u^{(2)}_{k_1+1,n'} \\ & & & \ddots & 0 & \vdots & \vdots & & \vdots \\ 0 & \dots & & 0 & 1 & \lambda & u^{(2)}_{k_1+k_2,d+2} & \cdots & u^{(2)}_{k_1+k_2,n'} \\ 0 & & \dots & & & 0 & u^{(2)}_{k_1+k_2+1,d+2} & \cdots & u^{(2)}_{k_1+k_2+1,n'} \\ \vdots & & & & \vdots & \vdots & \vdots & & \vdots \\ 0 & & \dots & & & 0 & u^{(2)}_{d,d+2} & \cdots & u^{(2)}_{d,n'} \end{pmatrix}.$$

In M_2 we subtract the first line from lines $2,\dots,k$ and line k_1+1 from lines $k_1+2,\dots,k_1+k_2$. Then we set the variables $j'_2,\dots,j'_{k_1}$ to j'_1, and variables $j'_{k_1+2},\dots,j'_{k_1+k_2}$ to j'_{k_1+1}. The corresponding changes in the matrix are eliminating columns $2,\dots k_1$ and $k_1+2,\dots k_1+k_2$ and putting in columns 1 and k_1+1 everywhere 0 but respectively in line 1 and line k_1+1. Finally, by exchanging line 2 and line k_1+1, we get the matrix

$$M_3 = \begin{pmatrix} 1 & 0 & 1 & u^{(3)}_{1,d+2} & \cdots & u^{(3)}_{1,n'} \\ 0 & 1 & \lambda & u^{(3)}_{2,d+2} & \cdots & u^{(3)}_{2,n'} \\ 0 & 0 & 0 & u^{(3)}_{3,d+2} & \cdots & u^{(3)}_{3,n'} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & u^{(3)}_{d,d+2} & \cdots & u^{(3)}_{d,n'} \end{pmatrix} \text{ in variables } j'_1, j'_{k_1+1}, j_{d+1}, \dots, j_{n'}.$$

To finish the reduction, we will distinguish two cases, depending on the congruency class of p modulo 4. When $p \equiv 1$, the element -1 is a square, and in polynomial time in $\log p$ we can find s such that $s^2 = -1$. We set $j_1 = s j_{d+1}$, eliminate column 1 from matrix M_3, put 0 in line 1 column $d+1$, and exchange line 1 and line 2. When $p \equiv 3$ modulo 4, the element -1 is not a square, and therefore we can choose $\lambda = -1$. We set $j_2 = j_{d+1}$, eliminate column 2, and put 0 in line 2 column $d+1$. In both cases we end up with a matrix of the form

$$M_4 = \begin{pmatrix} 1 & \alpha & u^{(3)}_{1,d+2} & \cdots & u^{(3)}_{1,n'} \\ 0 & 0 & u^{(3)}_{2,d+2} & \cdots & u^{(3)}_{2,n'} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & u^{(3)}_{d,d+2} & \cdots & u^{(3)}_{d,n'} \end{pmatrix}$$

in the variables $j', j_{d+1}, \dots, j_{n'}$ where $\alpha = \lambda$ and $j' = j'_{k_1+1}$ when $p \equiv 1$, and $\alpha = 1$ and $j' = j'_1$ otherwise. Without the first line it represents a system of $d-1$ equations in $n' - (d+1) = d(d+1)/2$ variables for which we can find a nontrivial solution by induction. Let $j_{d+2}, \dots, j_{n'}$ such a solution, and set $b = \sum_{k=d+2}^{n'} u^{(3)}_{1,k} j_k$. To give values to the remaining two variables we have to solve the equation $j'^2 + \alpha j^2_{d+1} + b = 0$. It is easy to see that the equation is always solvable, and then by Theorem A3 of [25] a solution can be found deterministically in polynomial time.

Gaussian elimination on M can be done in time $O(d^4)$. Finding a nontrivial solution for a quadratic homogeneous equation in 3 variables takes time $q_1(\log p)$, solving a quadratic equation in two variables takes time $q_2(\log p)$, and finding a square roots modulo p takes time $q_3(\log p)$ where q_1, q_2 and q_3 are polynomials. Therefore the complexity of solving system (1) is $O(d^5 + d^2 q_3(\log p) + d q_2(\log p) + q_1(\log p))$. □

We now turn to the system (2). Let $n' = n/(d+1)$, and for $0 \le k \le d$, consider the the system of d quadratic equations in n' variables:

$$\Big\{ \forall \ell \in [|1,d|], \quad \sum_{i=kn'+1}^{(k+1)n'} u_{\ell,i} j_i^2 \;=\; 0.$$

By Claim 5, each of these systems has a nontrivial solution that we can find in polynomial time. For each k, let $(j_{kn'+1}, \dots, j_{(k+1)n'})$ such a solution of the kth quadratic system. Then the set

$$\{(\lambda_0 j_1, \dots, \lambda_0 j_{n'}, \dots, \lambda_d j_{dn'+1}, \dots, \lambda_d j_{(d+1)n'}) \;:\; (\lambda_0, \dots, \lambda_d) \in \mathbb{Z}_p^{d+1}\}$$

is a $d+1$ dimensional subspace of of $\mathbb{Z}_p^n$ whose elements are solutions of the d quadratic equations in (2). Since in (2) there are d linear equations, we can find a a nontrivial

$(\lambda_0, \ldots, \lambda_d) \in \mathbb{Z}_p^{d+1}$ such that $(\lambda_0 j_1, \ldots, \lambda_0 j_{n'}, \ldots, \lambda_d j_{dn'+1}, \ldots, \lambda_d j_{(d+1)n'})$ is a (non-trivial) solution of the linear part of (2), and therefore of the whole system. □

Observe that the only probabilistic part of the algorithm is the generation of a quadratic non-residue modulo p

References

1. Bacon, D., Childs, A., van Dam, W.: From optimal measurement to efficient quantum algorithms for the hidden subgroup problem over semidirect product groups. In: Proc. 46th IEEE FOCS, pp. 469–478 (2005)
2. Beals, R., Babai, L.: Las Vegas algorithms for matrix groups. In: Proc. 34th IEEE FOCS, pp. 427–436 (1993)
3. Chevalley, C.: Démonstration d'une hypothèse de M. Artin. Abhand. Math. Sem. Univ. Hamburg 11, 73–75 (1936)
4. Eick, B.: Orbit-stabilizer problems and computing normalizers for polycyclic groups. J. Symbolic Comput. 34, 1–19 (2002)
5. Friedl, K., Ivanyos, G., Magniez, F., Santha, M., Sen, P.: Hidden translation and orbit coset in quantum computing. In: Proc. 35th ACM STOC, pp. 1–9 (2003)
6. Grigni, M., Schulman, L., Vazirani, M., Vazirani, U.: Quantum mechanical algorithms for the nonabelian Hidden Subgroup Problem. In: Proc. 33rd ACM STOC, pp. 68–74 (2001)
7. Hall, M.: Theory of groups. AMS Chelsea Publishing (1999)
8. Hallgren, S., Russell, A., Ta-Shma, A.: Normal subgroup reconstruction and quantum computation using group representations. SIAM J. Comp. 32(4), 916–934 (2003)
9. Holt, D.F., Eick, B., O'Brien, E.: Handbook of computational group theory. Chapman & Hall/CRC Press (2005)
10. Hoefling. Efficient multiplication algorithms for finite polycyclic groups (preprint, 2004)
11. Ivanyos, G., Magniez, F., Santha, M.: Efficient quantum algorithms for some instances of the non-Abelian hidden subgroup problem. Int. J. of Foundations of Computer Science 14(5), 723–739 (2003)
12. Ivanyos, G., Sanselme, L., Santha, M.: An efficient quantum algorithm for the hidden subgroup problem in extraspecial groups. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 586–597. Springer, Heidelberg (2007)
13. Kitaev, A.: Quantum measurements and the Abelian Stabilizer Problem. Technical report, Quantum Physics e-Print archive (1995), http://xxx.lanl.gov/abs/quant-ph/9511026
14. Luks, E.M.: Computing in solvable matrix groups. In: Proc. 33rd IEEE FOCS, pp. 111–120 (1992)
15. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge (2000)
16. Meggido, N., Papadimitriou, C.: On total functions, existence theorems, and computational complexity. Theor. Comp. Sci. 81, 317–324 (1991)
17. Moore, C., Rockmore, D., Russell, A., Schulman, L.: The power of basis selection in Fourier sampling: Hidden subgroup problems in affine groups. In: Proc. 15th ACM-SIAM SODA, pp. 1106–1115 (2004)
18. Moore, C., Russell, A., Sniady, P.: On the impossibility of a quantum sieve algorithm for graph isomorphism. In: Proc. 39th ACM STOC, pp. 536–545 (2007)

19. Mosca, M.: Quantum Computer Algorithms. PhD Thesis, University of Oxford (1999)
20. Regev, O.: Quantum Computation and Lattice Problems. SIAM J. Comp. 33(3), 738–760 (2004)
21. Rötteler, M., Beth, T.: Polynomial-time solution to the Hidden Subgroup Problem for a class of non-abelian groups. Technical report, Quantum Physics e-Print archive (1998), http://xxx.lanl.gov/abs/quant-ph/9812070
22. Shanks, D.: Five number-theoretic algorithms. In: Proc. 2nd Manitoba Conference on Numerical Mathematics, pp. 51–70 (1972)
23. Shor, P.: Algorithms for quantum computation: Discrete logarithm and factoring. SIAM J. Comp. 26(5), 1484–1509 (1997)
24. Warning, E.: Bemerkung zur vorstehenden Arbeit von Herr Chevalley. Abhand. Math. Sem. Univ. Hamburg 11, 76–83 (1936)
25. van de Woestijne, C.: Deterministic equation solving over finite fields. PhD thesis, Universiteit Leiden (2006)

Quantum Property Testing of Group Solvability

Yoshifumi Inui[1,2] and François Le Gall[2]

[1] Department of Computer Science, The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan
[2] ERATO-SORST Quantum Computation and Information Project, JST
Hongo White Building, 5-28-3 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan
legall@qci.jst.jp

Abstract. Testing efficiently whether a finite set Γ with a binary operation $\cdot$ over it, given as an oracle, is a group is a well-known open problem in the field of property testing. Recently, Friedl, Ivanyos and Santha have made a significant step in the direction of solving this problem by showing that it it possible to test efficiently whether the input $(\Gamma, \cdot)$ is an *Abelian* group or is far, with respect to some distance, from any Abelian group. In this paper, we make a step further and construct an efficient quantum algorithm that tests whether $(\Gamma, \cdot)$ is a *solvable* group, or is far from any solvable group. More precisely, the number of queries used by our algorithm is polylogarithmic in the size of the set Γ.

1 Introduction

In property testing, the problem considered is to decide whether an object given as an oracle has some expected property or is far from any object having that property. This is a very active research area and many properties including algebraic function properties, graph properties, computational geometry properties and regular languages were proved to be testable. We refer to, for example, [13,17] for surveys on classical property testing. Quantum testers have also been studied [6,9,14], and they are known to be strictly more powerful than classical testers in some cases [6,14].

In this paper, we focus on testing group-theoretical properties. A famous example is testing whether a function $f : G \to H$, where H and G are groups, is a homomorphism. It is well known that such a test can be done efficiently [4,5,18]. Another kind of of problems deals with the case where the input is a finite set Γ and an oracle of a binary operation $\cdot : \Gamma \times \Gamma \to \Gamma$ over it. A classical algorithm testing associativity of the oracle $\cdot$ using $O(|\Gamma|^2)$ queries to the oracle has been constructed by Rajagopalan and Schulman [16], and Ergün et al. [7] have proposed an algorithm, using a number of queries polynomial in $|\Gamma|$, testing if f corresponds to the multiplication of a group. But notice that, since each element in Γ needs $\Theta(\log |\Gamma|)$ bits to be encoded, the query complexities of these algorithms can be considered as exponential in the input length when not Γ, but only $|\Gamma|$ is given (e.g., Γ is supposed to be the set of binary strings of length $\lceil \log_2 |\Gamma| \rceil$). Designing an algorithm deciding whether $(\Gamma, \cdot)$ is a group that uses a number of queries to $\cdot$ polynomial in $\log |\Gamma|$ is indeed a well-known open problem. Recently, Friedl et al. [8] have made a significant step in the direction of solving this problem by constructing a classical algorithm with query and time complexities polynomial in $\log |\Gamma|$ that tests whether $(\Gamma, \cdot)$ is an Abelian group or is far from any Abelian group.

In this work, we make a step further and construct an efficient quantum algorithm that tests whether $(\Gamma, \cdot)$ is a solvable group or the distance between $(\Gamma, \cdot)$ and any solvable group is at least $\epsilon|\Gamma|^2$. More precisely, our algorithm uses a number of queries polynomial in $\log |\Gamma|$ and ϵ^{-1}, and its time complexity is polynomial $exp((\log \log |\Gamma|)^2)$ and ϵ^{-1}, i.e. subexponential in $\log |\Gamma|$. Notice that the class of solvable groups is far much larger than the class of Abelian groups and includes a vast class of non-Abelian groups. To deal with those groups, we introduce new ideas relying on the ability of quantum computation to solve fundamental group-theoretical problems, such as finding orders of elements or working with superpositions of all the elements of a subgroup.

Besides the theoretical interest of this result, our algorithm can be used when studying group-theoretical problems where the input is a black-box solvable group (i.e., given as a set a generators and an oracle performing group operations). In these problems, the input is usually promised to be a solvable group. By applying our algorithm, in the quantum setting, we can relax this promise, obtaining the new promise that the input corresponds to a solvable group or is far from any solvable group. We thus obtain robust versions of the quantum algorithms already known for solvable black-box groups [11,12,20]. We also hope that this will be useful to design new quantum property testers or group-theoretical quantum algorithms. In particular, our tester may be useful when considering quantum versions of classical algorithms solving problems over black-box solvable groups [1,2,3] as well.

Finally, we believe that our quantum algorithm may also be a first step in the direction of designing efficient classical testers for solvable groups. Indeed, the efficient classical tester for Abelian groups proposed by Friedl et al. [8] was inspired by a quantum algorithm solving the same problem. In this case, they were able to "dequantumize" the algorithm. A similar approach may be possible for our algorithm too.

2 Definitions

2.1 Distances Between Sets

Let Γ be a set and $\cdot : \Gamma \times \Gamma \to X$ a binary operation over it, where X is some set. We say that such couple $(\Gamma, \cdot)$ is a pseudo-magma. If $X \subseteq \Gamma$, we say that $(\Gamma, \cdot)$ is a magma. When there is no ambiguity we will denote a pseudo-magma or a magma $(\Gamma, \cdot)$ simply by Γ. We now define a distance between two pseudo-magmas. In this paper we adopt the so-called edit distance. This is the same distance as the one used by Friedl et al. [8].

Define a table of size k as a $k \times k$ matrix of dimensions $k \times k$. We consider three operations to transform a table to another. An exchange operation replaces elements in a table by arbitrary elements and its cost is the number of replaced elements. An insert operation at index i inserts a row and a column of index i. Its cost is $2k+1$ if the original table is of size k. A delete operation at index i deletes both the row of index i and the column of index i, giving a table of size $(k-1) \times (k-1)$. Its cost is $(2k-1)$.

Let $(\Gamma, \cdot)$ be a pseudo-magma, with $\cdot : \Gamma \times \Gamma \to X$. A multiplication table for Γ is a table of size $|\Gamma|$ with entries in X for which both rows and columns are in one-to-one correspondence with elements in Γ, i.e. there exists a bijection $\sigma : \{1, \cdots, |\Gamma|\} \to \Gamma$ such that the element in the i-th row and the j-th column is equivalent to $\sigma(i) \cdot \sigma(j)$. The distance between two pseudo-magmas is defined as follows.

Definition 1. *The edit distance between two tables T and T' is the minimum cost needed to transform T to T'. The edit distance between two pseudo-magmas Γ and Γ', denoted $d(\Gamma, \Gamma')$, is the minimum edit distance between T and T' where T (resp. T') runs over all tables corresponding to a multiplication table of Γ (resp. Γ'). For $\delta \geq 0$, we say that a pseudo-magma Γ is δ-close to another pseudo-magma Γ' if $d(\Gamma, \Gamma') \leq \delta$. Otherwise we say that Γ and Γ' are δ-far.*

Notice that if the sizes of Γ and Γ' are the same, then the edit distance becomes the minimal Hamming distance of the corresponding tables.

2.2 Property Testing of Group Solvability

In this paper we assume that the reader is familiar with the standard notions of group theory. We refer to any standard textbook for details. For completeness, we only recall the definition of solvable groups.

Definition 2. *A group G is solvable if there exists a collection of normal subgroups $G_0, \ldots, G_k$ such that $\{e\} = G_0 \subseteq \cdots \subseteq G_k = G$ and G_i/G_{i-1} is Abelian for $0 < i \leq k$.*

We now give our definition of a quantum property tester of group solvability. We define such a tester as a quantum algorithm $\mathscr{A}$ receiving as input a magma $(\Gamma, \cdot)$. More precisely, the actual input of the algorithm is the value $|\Gamma|$, and two oracles are available: an oracle that generates random elements in Γ (the details of the implementation of this oracle are not essential because this oracle will only be used in a classical subprocedure), and a quantum oracle that performs the binary operation $\cdot$. Since the elements of Γ can be encoded by binary strings of length $k = \lceil \log_2 |\Gamma| \rceil$, we identify the elements with their encoding and suppose that this quantum oracle performs the map $|g\rangle|h\rangle|c\rangle \mapsto |g\rangle|h\rangle|c \oplus g \cdot h\rangle$, where g and h are elements in Γ and c is a string in $\{0,1\}^k$. We denote by $\mathscr{A}(\Gamma)$ the behavior of the algorithm $\mathscr{A}$ on an input $(\Gamma, \cdot)$ given in this way. A more formal definition of a quantum property tester can be given but the following definition will be sufficient for our purpose.

Definition 3. *Let d be the distance defined in Subsection 2.1. A quantum ϵ-tester of group solvability is a quantum algorithm $\mathscr{A}$ such that, for any magma $(\Gamma, \cdot)$, the following holds:*

$$\begin{cases} \mathbf{Pr}[\mathscr{A}(\Gamma)\,accepts] > 2/3 & if\, d(\Gamma, \mathscr{S}) = 0 \\ \mathbf{Pr}[\mathscr{A}(\Gamma)\,rejects] > 2/3 & if\, d(\Gamma, \mathscr{S}) > \epsilon|\Gamma|^2. \end{cases}$$

Here we use $d(\Gamma, \mathscr{S})$ to represent $\inf_{G \in \mathscr{S}} d(\Gamma, G)$, where $\mathscr{S}$ denotes the set of finite solvable groups.

Notice that, a priori, requiring that the oracle is quantum may seem to give a problem different than in the classical setting, where the oracle is classical. But this is not really the case: if a classical procedure that computes the product $g \cdot h$ from g and h is available, such a quantum oracle can be effectively constructed for standard techniques of quantum computation [15].

The main result of this paper in the following theorem.

Theorem 1. *There exists a quantum ϵ-tester of group solvability that uses a number of queries polynomial in $\log|\Gamma|$ and ϵ^{-1}. The running time of this algorithm is polynomial in exp$((\log\log|\Gamma|)^2)$ and ϵ^{-1}.*

2.3 Quantum Algorithms for Solvable Groups

As stated in the following theorem, efficient quantum algorithms for studying the structure of solvable groups have been constructed by Watrous [20]. Our algorithm deeply relies on these algorithms.

Theorem 2. *([20]) Let G be a solvable group given as a black-box group. Then there exists a quantum algorithm running in time polynomial in $\log|G|$ that outputs, with probability at least 3/4, t elements $h_1, \ldots, h_t$ of G and t integers $m_1, \ldots, m_t$ such that, if we denote $H_i = \langle h_1, \ldots, h_i \rangle$ for $1 \le i \le t$, the following holds.*

(a) $\{e\} = H_0 \lhd H_1 \lhd \cdots \lhd H_{t-1} \lhd H_t = G$;
(b) H_i/H_{i-1} *is cyclic, for* $1 \le i \le t$, *with* $|H_i|/|H_{i-1}| = m_i$.

Moreover, given any $0 \le i \le t$, and any element h in H_i, there exists a quantum algorithm running in time polynomial in $\log|G|$ that outputs, with probability at least 3/4, the (unique) factorization of g over H_i, i.e. integers $a_1, \ldots, a_i$ with each $a_k \in \mathbb{Z}_{m_k}$, such that $h = h_i^{a_i} h_{i-1}^{a_{i-1}} \cdots h_1^{a_1}$.

In the algorithm of Theorem 2, the group is supposed to be input as a black-box group. A black-box group is a representation of a group G where elements are represented by strings (of the same length, say $k = \lceil \log G \rceil$). The input is a set of strings representing a set of generators of the group and an oracle performing the group product is available. The oracle necessary for Watrous's algorithm [20] is the map : $|g\rangle|h\rangle|c\rangle \mapsto |g\rangle|h\rangle|c \oplus g \cdot h\rangle$, for any elements $g, h \in G$ and any string c in $\{0,1\}^k$. Notice that this is the same oracle as the one given to a quantum tester of group solvability as defined in Subsection 2.2.

3 Our Quantum Algorithm

In this section we describe our quantum algorithm. We first give an overview of the algorithm in Subsection 3.1. Then, in Subsection 3.2, we explain the details. Finally, we analyse its correctness and complexity in Subsection 3.3.

3.1 Outline of Our Algorithm

Our algorithm consists in four parts.

Decomposition of Γ
We first construct, using Theorem 2, $t = O(\log^2(|\Gamma|))$ elements $h_1, \ldots, h_t$ of Γ that satisfy, if Γ is a solvable group, the relations $\{e\} = H_0 \lhd H_1 = \langle h_1 \rangle \lhd \cdots \lhd H_i = \langle h_1, \cdots, h_i \rangle \lhd \cdots \lhd H_t = \langle h_1, \cdots, h_t \rangle = \Gamma$, where each H_i is a subgroup of Γ, normal in H_{i+1}, such that H_i/H_{i-1} is cyclic. If Γ is a solvable group, this decomposition gives a so-called power-conjugate presentation of Γ. If Γ is not a solvable group, these elements $h_1, \ldots, h_t$ still define some sets $H_0, \ldots, H_t$, although in general these sets satisfy no group-theoretical property.

Test of embedding
Then, we take sufficiently many elements of Γ and check that they are all in H_t. Success of this test implies that $|\Gamma \backslash H_t|$ is small enough. Of course, if Γ is a solvable group, then $\Gamma = H_t$ with high probability and this test always succeed. Assume that Γ is far from any solvable group $\tilde{H}_t$. If the test succeed, since the inequality $d(\Gamma, \tilde{H}_t) \leq d(\Gamma, H_t) + d(H_t, \tilde{H}_t)$ holds for any solvable group $\tilde{H}_t$, this will imply that H_t is far from any solvable group $\tilde{H}_t$ too (because the value of $d(\Gamma, H_t)$ is basically a function of $|\Gamma \backslash H_t|$, and thus small).

Construction of the group G_t
We construct, using the information about the structure of Γ obtained at the first part of the algorithm, t solvable groups $G_1, \ldots, G_t$ and a function $\psi : G_t \rightarrow H_t$ in a way such that, if Γ is a solvable group, then ψ is a group isomorphism from G_t to H_t.

Test of homomorphism
Finally, the algorithm will test whether ψ is "almost" an homomorphism. We will show that this test is robust: is ψ is close to an homomorphism, then H_t is close to the solvable group G_t. If H_t is far from any solvable group, then this cannot hold and the homomorphism test must fail with high probability.

Again, the similar idea of constructing a group G, a function $\psi : G \rightarrow \Gamma$ and use homomorphism tests was at the heart of the property tester for Abelian groups proposed by Friedl et al. [8] and inspired this work (notice that the Friedl et al. first constructed a quantum property tester for Abelian groups, and then were able to remove the quantum part in their algorithm). However there are new difficulties that arise when considering property testers for solvable groups. The first one is that analyzing the decomposition the H_i's is more difficult and the power of quantum computation seems necessary to perform this task efficiently.

The second complication is that, now, the group G_i's we are considering are solvable, i.e. in general not commutative. In this case, we have to be very careful in the definition of G_i's and additional tests have to be done to ensure that the G_i's we define are really groups.

3.2 Algorithm

Our algorithm appears in Figure 1 and each of the four parts are explained in details in Subsections 3.2.1 to 3.2.4. If all the tests performed succeed, we decide that Γ is a solvable group. Otherwise we decide that Γ is $(\epsilon|\Gamma|^2)$-far from any solvable group.

PART I: Decomposition of Γ

1. Take $O(\log|\Gamma|)$ elements of Γ.
2. Use the first algorithm of Theorem 2 on them and obtain the set $\{h_1, \ldots, h_t\}$ and integers $m_1, ..., m_t$.
3. Compute the decompositions of all $h_i^{m_i}$ and $h_i^{-1} \cdot (h_k \cdot h_i)$ over H_{i-1}, for $i \in \{1, \ldots, t\}$ and $k \in \{1, \ldots, i-1\}$, and check the obtained decompositions.

PART II: Test of embedding

4. Check that $|\Gamma| = m_1 \times \cdots \times m_t$ and $|\Gamma \backslash H_t|/|\Gamma| < \epsilon/4$.

PART III: Construction of the group G_t

5. For j from 2 to t check that conditions (a), (b) and (c) of Proposition 1 hold.

PART IV: Test of homomorphism

6. Check that $\mathbf{Pr}_{x,y\in G_t}[\psi(x \circ y) = \psi(x) \cdot \psi(y)] > 1 - \eta$ with $\eta = \epsilon/422$.

Fig. 1. Quantum ϵ-tester of group solvability

3.2.1 Decomposition of Γ

The first step in our algorithm finds a power-conjugate representation of Γ when Γ is a solvable group. We will prove that when Γ is far from any solvable group, then the output of this step cannot be a power-conjugate representation of a group close to Γ and that this can be detected by our algorithm at part II, III or IV.

First, we pick $s = \Theta(\log|\Gamma|)$ random elements $\alpha_1, \cdots, \alpha_s$ from the ground set Γ. For simplicity, suppose first that Γ is a solvable group. Denote $\Gamma' = \langle\alpha_1, \cdots, \alpha_s\rangle$. Then, with high probability, $\Gamma = \Gamma'$. Here we rely on the standard fact in computational group theory that $\Theta(\log|\Gamma|)$ random elements of Γ constitute, with high probability, a generating set of Γ. We now run the first algorithm of Theorem 2 with input Γ' presented as a black-box group as follows: $\alpha_1, \cdots, \alpha_s$ is the set of generators and the operation $\cdot$ is the oracle performing group multiplication. The output of the algorithm is then, with high probability, a set of t elements $h_1, \ldots, h_t$ of Γ and t integers $m_1, \ldots, m_t$ such that, if we denote $H_i = \langle h_1, \ldots, h_i\rangle$ for $1 \le i \le t$, the following holds:

(a) $\{e\} = H_0 \lhd H_1 \lhd \cdots \lhd H_{t-1} \lhd H_t = \Gamma'$; and
(b) H_i/H_{i-1} is cyclic for $1 \leq i \leq t$ and satisfies $|H_i|/|H_{i-1}| = m_i$.

Moreover, we further analyse the structure of Γ' and use the second algorithm of Theorem 2 to decompose the elements $h_i^{m_i}$ and $h_i^{m_i-1} \cdot (h_k \cdot h_i)$ over H_{i-1}, for each $i \in \{2, \ldots, t\}$ and each $k \in \{1, \ldots, i-1\}$. Notice that, indeed, each $h_i^{m_i}$ and $h_i^{m_i-1} \cdot (h_k \cdot h_i) = h_i^{-1} \cdot h_k \cdot h_i$ are in H_{i-1} when Γ is a solvable group. We denote the decompositions obtained by

$$h_i^{m_i} = h_{i-1}^{r_{i-1}^{(i)}} \cdot \left(\cdots \left(h_3^{r_3^{(i)}} \cdot \left(h_2^{r_2^{(i)}} \cdot h_1^{r_1^{(i)}} \right) \right) \right) \text{ for } 2 \leq i \leq t, \tag{1}$$

$$h_i^{m_i-1} \cdot (h_k \cdot h_i) = h_{i-1}^{s_{k,i-1}^{(i)}} \cdot \left(\cdots \left(h_3^{s_{k,3}^{(i)}} \cdot \left(h_2^{s_{k,2}^{(i)}} \cdot h_1^{s_{k,1}^{(i)}} \right) \right) \right) \text{ for } 1 \leq k < i \leq t, \tag{2}$$

where each $r_\ell^{(i)}$ and each $s_{k,\ell}^{(i)}$ are in $\mathbb{Z}_{m_\ell}$.

In general, we do not know whether Γ is a solvable group or not but we do exactly the same as above: we first run the first algorithm of Theorem 2 on the set $\{\alpha_1, \cdots, \alpha_s\}$ with the oracle $\cdot$. If this algorithm errs or outputs something different from a set of elements $h_1, \ldots, h_t$ and a set of integers $m_1, \ldots, m_t$, we conclude that Γ is not a solvable group (this decision is correct with high probability because, if Γ is a group, then the algorithm of Theorem 2 succeeds with high probability). Now suppose that we have obtained elements $h_1, \ldots, h_t$ and a set of integers $m_1, \ldots, m_t$. We define the following sets by recurrence: $H_1 = \{h_1^a | a \in \mathbb{Z}_{m_1}\}$, and, for $2 \leq j \leq t$, $H_j = \{h_j^a \cdot h | a \in \mathbb{Z}_{m_j}, h \in H_{j-1}\}$. Here, and in many other places in this paper, we use the notation h^r, for $h \in \Gamma$ and $r \geq 1$, to denote the product $h \cdot (\cdots (h \cdot (h \cdot h)))$, since $\cdot$ is not in general associative. Moreover we use the convention $h^0 = h_1^{m_1}$ for any $h \in \Gamma$.

Notice that, in general, the H_i's have no group-theoretical structure at all. Then we run the second algorithm of Theorem 2 to decompose the elements $h_i^{m_i}$ and $h_i^{m_i-1} \cdot (h_k \cdot h_i)$ over H_{i-1}, for each $2 \in \{1, \ldots, t\}$ and each $k \in \{1, \ldots, i-1\}$. If the algorithm errs or outputs something irrelevant, we conclude that Γ is not a solvable group. Suppose that the algorithm succeeds and outputs decompositions. We use the notations of equations (1) and (2) to denote the decompositions obtained. We check whether these decompositions are correct, i.e. we compute the right sides of equations (1) and (2) and check that they match the left sides. If they are correct, we move to the next step (Subsection 3.2.2). Else, we conclude that Γ is not a solvable group.

3.2.2 Test of Embedding

In the second part of our algorithm, we first check that $|\Gamma| = m_1 \times \cdots \times m_t$. Then, we want to check whether $|\Gamma \backslash H_t|$ is small enough. Otherwise we conclude that Γ is not a solvable group. Indeed, if Γ is a group, then with high probability (on the choice of $\alpha_1, \ldots, \alpha_s$ and on the randomness of the algorithm of Theorem 2) $\Gamma = \Gamma' = H_t$.

More precisely we check whether $|\Gamma \backslash H_t|/|\Gamma| < \epsilon/4$ holds. In order to perform this test, we simply take c_1 elements of Γ and check whether they are all in

H_t (by using the second algorithm of Theorem 2 and checking the obtained decompositions). It is easy to show that, when taking $c_1 = \Theta(\epsilon^{-1})$, we can detect whether $|\Gamma \backslash H_t|/|\Gamma| > \epsilon/4$ with constant probability.

3.2.3 Construction of the Group G_t

We now construct the abstract group G_t defined by the power-conjugate presentation found in Part I of our algorithm (relations (1) and (2)) when such a group exists, i.e. when the presentation is consistent with the definition of a group.

We first define by recurrence the family of magmas $\{G_j\}_{1\leq j\leq t}$, where each G_j is equal (as a set) to $\mathbb{Z}_{m_j} \times \cdots \times \mathbb{Z}_{m_1}$. G_1 is defined as the cyclic group $(\mathbb{Z}_{m_1}, +)$, where $+$ is the addition modulo m_1. For any $i \in \{2, \ldots, t\}$, denote by u_i the element $(r^{(i)}_{i-1}, \ldots, r^{(i)}_1)$ of G_{i-1} and, for any $i \in \{2, \ldots, t\}$ and $k \in \{1, \ldots, i-1\}$, denote by $v_{i,k}$ the element $(s^{(i)}_{k,i-1}, \ldots, s^{(i)}_{k,1})$ of G_{i-1}.

Definition 4. *Define $G_1 = (\mathbb{Z}_{m_1}, +)$ and, for $2 \leq j \leq t$, let G_j be the magma $(\mathbb{Z}_{m_j} \times G_{j-1}, \circ_j)$ with*

$$(a, x) \circ_j (b, y) = \begin{cases} \left(a + b, \phi^{(b)}_j(x) \circ_{j-1} y\right) & \text{if } \ a + b < m_j \\ \left(a + b - m_j, u_j \circ_{j-1} \phi^{(b)}_j(x) \circ_{j-1} y\right) & \text{if } \ a + b \geq m_j \end{cases}$$

where $\phi_j : G_{j-1} \to G_{j-1}$ maps any element $(a_{j-1}, \cdots, a_1)$ of G_{j-1} to the element $\phi_j((a_{j-1}, \cdots, a_1)) = v^{a_{j-1}}_{j,j-1} \circ_{j-1} \left(\cdots \circ_{j-1} \left(v^{a_2}_{j,2} \circ_{j-1} v^{a_1}_{j,1}\right)\right)$.

We will usually denote $\circ_j$ or $\circ_{j-1}$ simply by $\circ$ when there is no ambiguity.

In order to illustrate this definition, let us consider the case where all the H_j's are solvable groups. In this case, each $H_j = \{h^{a_j}_j \cdot \cdots \cdot h^{a_1}_1 \mid a_j \in \mathbb{Z}_{m_j}\}$ is in bijection with $\mathbb{Z}_{m_j} \times \cdots \times \mathbb{Z}_{m_1}$. Fix a j and consider H_j. Each element $h^{a_j}_j \cdots h^{a_1}_1$ is associated with the element $(a_j, \ldots, a_1)$ of G_j. Now the element $\phi_j(a_{j-1}, \cdots, a_1)$ corresponds to the element

$$h^{-1}_j \cdot (h^{a_{j-1}}_{j-1} \cdots h^{a_1}_1) \cdot h_j = \left(h^{s^{(j)}_{j-1,j-1}}_{j-1} \cdots h^{s^{(j)}_{j-1,1}}_1\right)^{a_{j-1}} \cdots \left(h^{s^{(j)}_{1,j-1}}_{j-1} \cdots h^{s^{(j)}_{1,1}}_1\right)^{a_1}.$$

In other words, the map ϕ_j in G_{j-1} corresponds to the inner automorphism $h \mapsto h^{-1}_j h h_j$ of H_j. For any two elements g and g' in H_{j-1}, since $h^a_j \cdot g \cdot h^b_j \cdot g' = h^{a+b}_j \cdot (h^{-b}_j \cdot g \cdot h^b_j) \cdot g'$ we see that the the G_j's are defined to be isomorphic to the H_j's in the case where the H_j's are solvable groups.

If the H_j's are not groups, then the G_j's constructed in Definition 4 are not necessarily groups. But we now show that when some additional conditions are satisfied, the G_j's become groups. In the next proposition, we denote by $x_{j,k}$, for $1 \leq k \leq j \leq t$, the element of G_j with one 1 at the index k (from the right) and zeros at all the other indexes.

Proposition 1. *Let $1 < j < t$. Suppose that G_{j-1} is a solvable group. Assume that the following three conditions holds.*

(a) $\phi_j(x_{j-1,k}) \circ \phi_j(x_{j-1,i}) = \phi_j(x_{j-1,i}) \circ \phi_j(v_{k,i})$ *for all* $1 \leq k < i \leq j-1$*; and*
(b) $\phi_j(u_j) = u_j$*; and*
(c) $\phi_j^{(m_j)}(x_{j-1,i}) = u_j^{-1} \circ x_{j-1,i} \circ u_j$ *for all* $1 \leq i \leq j-1$.

Then G_j *is a solvable group.*

Proof. If ϕ_j is an automorphism of G_{j-1}, then conditions (b) and (c) imply that G_j is a cyclic extension of G_{j-1} and thus a solvable group (see [19, Section 9.8]). It is easy to check that condition (a) implies that ϕ_j is a homomorphism. Since $\phi_j^{(m_j)}$ is an automorphism from condition (c), ϕ_j is thus an automorphism too. □

Notice that the three conditions of Proposition 1 obviously hold when $(\Gamma, \cdot)$ is a group: conditions (b) and (c) come from the fact that u_j in G_{j-1} corresponds to the element $h_j^{m_j}$ and condition (a) follows from equations (2) and from the fact that ϕ_j is a homomorphism when $(\Gamma, \cdot)$ is a group.

For each $j \in \{2, \ldots, t\}$, testing that conditions (a) and (b) hold can be done using a number of multiplications in the group G_{j-1} polynomial in $\log |\Gamma|$. The best known classical algorithm for computing products in a solvable group given as a power-conjugate presentation is an algorithm by Höfling [10] with time complexity $O(exp((\log\log |G_{j-1}|)^2)) = O(exp((\log\log |\Gamma|)^2))$. Notice that if condition (a) holds then ϕ_j is a homomorphism. Then each term $\phi_j^{(m_j)}(x_{j-1,i})$ in condition (c) can be computed using a polynomial number of group products by computing, step by step for increasing ℓ from 0 to $\lfloor \log m_j \rfloor$, the values $\phi_j^{(2^\ell)}(x_{j-1,k})$ for all $1 \leq k \leq j-1$. The total time complexity of checking that all the G_i's are solvable groups is thus $O(exp((\log\log |\Gamma|)^2))$. No query to the oracle $\cdot$ is needed.

3.2.4 Test of Homomorphism

We now suppose that the G_i's have passed all the tests of Proposition 1 and thus G_t is a solvable group. Let ψ be the function from G_t to H defined as

$$\psi(a_t, a_{t-1}, \cdots, a_1) = h_t^{a_t} \cdot (h_{t-1}^{a_{t-1}} \cdot (\cdots (h_2^{a_2} \cdot h_1^{a_1})).$$

We will test whether ψ is a homomorphism from G_t to H_t. If $(\Gamma, \cdot)$ is a solvable group, then ψ is an homomorphism by construction. We now show that this test is robust.

Proposition 2. *Suppose that* $\eta < 1/120$*. Assume that* $|H_t| > 3|G_t|/4$*. Suppose that*

$$\mathbf{Pr}_{x,y \in G_t}[\psi(x \circ y) = \psi(x) \cdot \psi(y)] > 1 - \eta. \tag{3}$$

Then there exists a solvable group $\tilde{H}_t$ *that is* $(211\eta|\Gamma|^2)$*-close to* H_t*.*

Proof. From condition (3), Theorem 2 of [8] implies that there exists a group $(\tilde{H}_t, *)$ with $|\tilde{H}_t| \leq |G_t|$, and a homomorphism $\tilde{\psi} : G_t \to \tilde{H}_t$ such that:

(a) $|\tilde{H}_t \backslash H_t| \leq 30\eta|\tilde{H}_t|$;

(b) $\mathbf{Pr}_{h,h'\in\tilde{H}_t}[h * h' \neq h \cdot h'] \leq 91\eta$; and
(c) $\mathbf{Pr}_{x\in G_t}[\tilde{\psi}(x) \neq \psi(x)] \leq 30\eta$.

Notice that, strictly speaking, Theorem 2 of [8] can be used only if H_t is a magma, i.e. closed under $\cdot$. This is not the case here because H_t may not be a magma, but only a pseudo-magma. However, careful inspection of the proof of Theorem 2 of [8] shows that the same result holds for pseudo-magmas too. The distance between $\tilde{H}_t$ and H_t is determined by the number of elements being a member of either set and the number of pairs of two elements for which the result of the multiplication differ. In particular, this distance has for upper bound the cost of the following transform: starting from the table of $\tilde{H}_j$, we first delete rows and columns corresponding to elements in $\tilde{H}_t \backslash H_t$, insert rows and columns corresponding to elements in $H_t \backslash \tilde{H}_t$, and then exchange multiplication entries which differ between two tables. It follows from (a) and (b) that the number of elements in $\tilde{H}_t \backslash H_t$ is less than $30\eta|\tilde{H}_t|$ and the number of pairs $(h, h') \in \tilde{H}_t \times \tilde{H}_t$ such that $h * h' \neq h \cdot h'$ is less than $91\eta|\tilde{H}_t|^2$. It remains to show that $H_t \backslash \tilde{H}_t$ is small enough too and that $\tilde{H}_t$ is a solvable group.

Suppose towards a contradiction that $|\tilde{\psi}(G_t)| < |G_t|$. Then $|\tilde{\psi}(G_t)| \leq |G_t|/2$. From condition (c), we obtain $|\psi(G_t)| \leq |G_t|/2 + 30\eta|G_t| \leq 3|G_t|/4$. This gives a contradiction. Thus $|\tilde{\psi}(G_t)| = |\tilde{H}_t| = |G_t|$ and $\tilde{\psi}$ is an isomorphism from G_t to $\tilde{H}_t$. Since G_t is a solvable group, $\tilde{H}_t$ is solvable too. Since $|H_t| \leq |G_t|$, it also follows that $|H_t| \leq |\tilde{H}_t|$ and thus $|H_t \backslash \tilde{H}_t| \leq |\tilde{H}_t \backslash H_t| \leq 30\eta|\tilde{H}_t|$.

Deleting $|\tilde{H}_t \backslash H_t|$ rows and column from the table of $\tilde{H}_t$ costs

$$2|\tilde{H}_t||\tilde{H}_t \backslash H_t| - |\tilde{H}_t \backslash H_t|^2 \leq 60\eta|\tilde{H}_t|^2.$$

Then inserting $|H_t \backslash \tilde{H}_t|$ rows and columns similarly costs at most $60\eta|\tilde{H}_t|^2$ too. Thus the distance between H_t and the solvable group $\tilde{H}_t$ and is at most $[(60 + 60 + 91)\eta|\tilde{H}_t|^2] \leq 211\eta|\Gamma|^2$. □

More precisely, we perform the following test. We want to test which of $\mathbf{Pr}_{x,y\in G}[\psi(x \circ y) = \psi(x) \cdot \psi(y)] = 1$ and $\mathbf{Pr}_{x,y\in G_t}[\psi(x \circ y) = \psi(x) \cdot \psi(y)] \leq 1 - \eta$ with $\eta = \epsilon/422$ holds. We take c_2 pairs (x, y) of elements of G_t and test whether they all satisfy $\psi(x \circ y) = \psi(x) \cdot \psi(y)$. It is easy to show that, when taking $c_2 = \Theta(\eta^{-1}) = \Theta(\epsilon^{-1})$, we can decide which case holds with constant probability.

3.3 Correctness and Complexity

We now evaluate the performance of our algorithm. This gives the result of Theorem 1.

First, suppose that the magma $(\Gamma, \cdot)$ is a solvable group. With high probability the set of elements taken at step 1 is a generating set of Γ and the first algorithm of Theorem 2 succeeds on this set. In this case, each of the tests realized at steps 3 and 4 succeeds with high probability (since the success probability of the second algorithm of Theorem 2 can be amplified), and then all the tests at steps 5 and 6 succeed with probability 1. Thus the global error probability is constant.

Now, we would like to show that any magma Γ that is $(\epsilon|\Gamma|^2)$-far from any solvable group is rejected with high probability. Take such a magma Γ. Then H_t is $(\frac{\epsilon}{2}|\Gamma|^2)$-far from any solvable group $\tilde{H}_t$ or $|\Gamma\backslash H_t|/|\Gamma| > \frac{\epsilon}{4}$. This assertion holds because for any solvable group $\tilde{H}_t$, the inequalities $\epsilon|\Gamma|^2 < d(\Gamma, \tilde{H}_t) \leq d(\Gamma, H_t) + d(H_t, \tilde{H}_t)$ hold and $d(\Gamma, H_t) = 2|\Gamma\backslash H_t||\Gamma| - |\Gamma\backslash H_t|^2 \leq 2|\Gamma\backslash H_t||\Gamma|$ since $H_t \subseteq \Gamma$ and the operation is the same. If the latter holds, it should be rejected with high probability at test 4. Now suppose that the former holds and that all the steps 1–5 succeed. Then with high probability $|H_t| \geq 3|\Gamma|/4 = 3|G_t|/4$. From Proposition 2 this implies that $\mathbf{Pr}_{x,y\in G}[\psi(x\circ y) = \psi(x)\cdot\psi(y)] \leq 1 - \epsilon/422$. This is detected with high probability at step 6.

The algorithm queries the oracle Γ a number of times polynomial in $\log|\Gamma|$ at each steps 1, 2 and 3, and a number of times polynomial in $\log|\Gamma|$ and ϵ^{-1} at steps 4 and 6. Additional computational work is needed at steps 5 and 6 to compute a polynomial number of products in the groups G_i's. Since each product can be done (without queries) using $O(exp((\log\log|G_i|)^2)) = O(exp((\log\log|\Gamma|)^2))$ time using the algorithm by Höfling [10], the total time complexity of the algorithm is polynomial in $exp((\log\log|\Gamma|)^2)$ and ϵ^{-1}.

Acknowledgments

The authors thank an anonymous reviewer for helpful comments and suggestions.

References

1. Arvind, V., Vinodchandran, N.V.: Solvable black-box group problems are low for PP. Theoretical Computer Science 180(1-2), 17–45 (1997)
2. Babai, L., Cooperman, G., Finkelstein, L., Luks, E., Seress, Á.: Fast Monte Carlo algorithms for permutation groups. Journal of Computer and System Sciences 50(2), 296–307 (1995)
3. Babai, L., Szemerédi, E.: On the complexity of matrix group problems. In: Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science, pp. 229–240 (1984)
4. Ben-Or, M., Coppersmith, D., Luby, M., Rubinfeld, R.: Non-Abelian homomorphism testing, and distributions close to their self-convolutions. In: Proceedings of the 8th International Workshop on Randomization and Computation, pp. 273–285 (2004)
5. Blum, M., Luby, M., Rubinfeld, R.: Self-testing/correcting with applications to numerical problems. In: Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, pp. 73–83 (1990)
6. Buhrman, H., Fortnow, L., Newman, I., Röhrig, H.: Quantum property testing. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 873–882 (2001)
7. Ergün, F., Kannan, S., Kumar, R., Rubinfeld, R., Viswanathan, M.: Spot-checkers. Journal of Computer and System Sciences 60(3), 717–751 (2000)
8. Friedl, K., Ivanyos, G., Santha, M.: Efficient testing of groups. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, pp. 157–166 (2005)

9. Friedl, K., Magniez, F., Santha, M., Sen, P.: Quantum testers for hidden group properties. In: Rovan, B., Vojtáš, P. (eds.) MFCS 2003. LNCS, vol. 2747, pp. 419–428. Springer, Heidelberg (2003)
10. Höfling, B.: Efficient multiplication algorithms for finite polycyclic groups (preprint, 2004), http://www-public.tu-bs.de:8080/~bhoeflin/
11. Inui, Y., Le Gall, F.: Efficient algorithms for the hidden subgroup problem over a class of semi-direct product groups. Quantum Information and Computation 7(5&6), 559–570 (2007)
12. Ivanyos, G., Magniez, F., Santha, M.: Efficient quantum algorithms for some instances of the non-Abelian hidden subgroup problem. International Journal of Foundations of Computer Science 14(5), 723–740 (2003)
13. Kiwi, M., Magniez, F., Santha, M.: Exact and approximate testing/correcting of algebraic functions: a survey. In: Theoretical Aspects of Computer Science 2000. LNCS, vol. 2292, pp. 30–83. Springer, Heidelberg (2002)
14. Magniez, F., Nayak, A.: Quantum complexity of testing group commutativity. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1312–1324. Springer, Heidelberg (2005)
15. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge (2000)
16. Rajagopalan, S., Schulman, L.: Verification of identities. In: Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, pp. 612–616 (1996)
17. Ron, D.: Property testing. In: Handbook of Randomized Computing, pp. 597–649. Kluwer Academic Publishers, Dordrecht (2001)
18. Shpilka, A., Wigderson, A.: Derandomizing homomorphism testing in general groups. In: Proceedings of the 36th Annual ACM Symposium on Theory of Computing, pp. 427–435 (2004)
19. Sims, C.: Computation with Finitely Presented Groups. Cambridge University Press, Cambridge (1994)
20. Watrous, J.: Quantum algorithms for solvable groups. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, pp. 60–67 (2001)

Solving NP-Complete Problems with Quantum Search

Martin Fürer[⋆]

Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802, USA
furer@cse.psu.edu
http://cse.psu.edu/~furer

Abstract. In his seminal paper, Grover points out the prospect of faster solutions for an NP-complete problem like SAT. If there are n variables, then an obvious classical deterministic algorithm checks out all 2^n truth assignments in about 2^n steps, while his quantum search algorithm can find a satisfying truth assignment in about $2^{n/2}$ steps.

For several NP-complete problems, many sophisticated classical algorithms have been designed. They are still exponential, but much faster than the brute force algorithms. The question arises whether their running time can still be decreased from $T(n)$ to $\tilde{O}(\sqrt{T(n)})$ by using a quantum computer. Isolated positive examples are known, and some speed-up has been obtained for wider classes. Here, we present a simple method to obtain the full $T(n)$ to $\tilde{O}(\sqrt{T(n)})$ speed-up for most of the many non-trivial exponential time algorithms for NP-hard problems. The method works whenever the widely used technique of recursive decomposition is employed.

This included all currently known algorithms for which such a speed-up has not yet been known.

1 Introduction

Grover's [1,2] quantum algorithm for searching a database to "find a needle in a haystack" has an obvious application for solving many hard combinatorial problems. In the example of SAT (Satisfiability of boolean formulas), the quantum computer goes into a uniformly weighted superposition of 2^n states representing all possible truth assignments to the n variables, each represented by a binary string of length n.

For a given formula and a given truth assignment the trivial satisfiability test runs in polynomial time. The quantum algorithm uses the same kind of test, but runs it simultaneously for all the 2^n truth assignments. Every one of the 2^n superpositioned computations stores the result in the same boolean variable. Then all auxiliary results are erased by an operation sometimes referred to as

⋆ Research supported in part by NSF Grant CCR-0209099 and CCF-0728921.

"uncomputing." If there are successful truth assignments, then their values are Grover's needles in the haystack. With constant probability, one of them is found in time $\tilde{O}(2^{n/2})$[1].

This example seems to suggest that Grover's method can only be applied to a class of very simply structured brute force search algorithms. It is certainly unrealistic to believe that a method could be designed that could speed up every thinkable satisfiability test by the use of quantum computers.

Nevertheless, Angelsmark, Dahllöf, and Jonsson [3] have shown that for some nontrivial nicely structured classical algorithms, a significant speed-up by using quantum computers is indeed possible. They present a quantum algorithm for constraint satisfaction problems and a special version for 3-coloring running in time $\tilde{O}(1.2185^n)$. This is in drastic contrast to our approach. Our goal is to provide a quadratic speed-up for every known deterministic test. This is possible, because all such non-trivial tests are based on recursive decomposition. We can take any state of the art classical algorithm based on recursive decomposition, like Eppstein's [4] 3-coloring algorithm with a provable running time of $\tilde{O}(1.3289^n)$. We immediately obtain a quantum algorithm running in time $\tilde{O}(1.3289^{n/2}) = \tilde{O}(1.1527^n)$. Our method applies as well to all the known deterministic algorithms for other NP-complete decision problems.

Our method does not provide a mechanical transformation from classical decision algorithms to quantum algorithms. To build the quantum algorithm, we have to take the time analysis of the classical algorithm too, and build it into the quantum algorithm.

A different approach has been taken by Cerf et al. [5]. For well structured classical solutions, they propose to speed up the search by doing many nested quantum searches, applied to subtrees of a given search tree. Our method is much simpler, more widely applicable, and provides a higher speed-up.

Cerf et al. [5] cut a search tree at a small number of levels. Useless subtrees rooted at a cut level can be omitted. The others have still to be fully searched down to the next level. They obtain a quadratic speed-up compared to a classical algorithm that also searches these complete subtrees. But naturally, a classical algorithm has no need to fully search these sutrees and could often stop earlier. Therefore, instead of speeding up from $T(n)$ to $\tilde{O}(\sqrt{T(n)})$, their quadratic speed-up is from the running time of an artificially slow classical algorithm. As an example, the authors point out that while the currently best classical algorithm for 3-SAT runs in time $\tilde{O}(2^{0.446n})$, their quantum algorithm runs in time $\tilde{O}(2^{0.34n})$. As this is a significant improvement, they consider it a success for their algorithm, even though the exponent is far from being divided by 2.

A more widely applicable method has been proposed by Brassard, Høyer, Mosca and Tapp [6]. They show how any randomized classical algorithm with one-sided error can be transformed into a faster quantum algorithm. The idea is to run a superposition of the computations with all possible strings produced by coin tossing. The needles in the haystack are then the accepting computation paths.

[1] Here, $\tilde{O}$ refers to upper bounds up to polynomial factors.

In their expository articles, Ambainis [7] as well as Dantsin et al. [8] point out in particular that Schöning's classical algorithm [9], as well as its later improvements can easily be modified for a quantum computation with a quadratic speed up. The reason is that these algorithms do many runs of the same procedure from different starting points. Where these algorithms use sequential repetitions, a quantum algorithm just uses quantum amplitude amplification instead, in effect doing many searches from different starting points in parallel.

If a classical randomized algorithm with one-sided error succeeds with probability c^{-n} in polynomial time $T(n)$, then the directly corresponding quantum algorithm running these superpositions also has a success probability of order c^{-n}.

In the classical setting the algorithm can be repeated c^n times to obtain constant success probability and running time $O(T(n)c^n)$. The quantum algorithm instead can use amplitude amplification [6], which is a generalization of Grover's searching algorithm. Using amplitude amplification, a constant success probability can already be obtained by a quantum computation with running time $O(T(n)c^{\frac{n}{2}})$. This is a quadratic improvement, because $T(n)$ is a polynomial.

The resulting running time is much better than the time obtained by a direct application of Grover's method to a trivial algorithm. The classical randomized algorithm running in time $T(n)$ does not have to be the brute force search algorithm that tosses a coin for every boolean variable and checks whether it defines a satisfying assignment. Instead, the classical algorithm can already be a sophisticated search algorithm. Hence, this is a significant generalization of Grover's algorithm.

In the current paper we do something similar, corresponding to (sophisticated) deterministic exponential time algorithms. We focus on the wide middle ground between simplistic brute force approaches or other well structured algorithms on one hand, and unstructured or arbitrarily structured methods on the other hand. There might be some mere heuristics, but in all algorithms published with a complexity analysis, there is just enough regularity to allow the full quantum search speed-up. We are considering the broad class of algorithms using recursive decomposition based on self reducibility known as the Davis-Putnam [10,11] procedure. Thus our method applies to all the classical and recent exponential time algorithms for NP-hard problems based on recursive decomposition, including dynamic programming and search tree pruning techniques. For a recent survey see Woeginger [12].

For our method to work, we have to assume that the recursive decomposition does not happen completely irregularly. It can be quite complicated, but it has to be so predictable as to allow the proof of a rigorous complexity bound for the classical algorithm. Thus all the many algorithms based on recursive decomposition qualify, if they have been published together with a provable time analysis (upper bound). The presentation of such an efficient exponential time algorithm always comes in two quite distinct parts, the definition of the algorithm and the analysis of its running time. Here, we use both, a classical algorithm and also its time analysis as building blocks for a quantum algorithm. Thus to be precise, our algorithm does not produce a quadratic speed-up form an unknown running time of an arbitrary classical algorithm, but from the currently provable upper bound on the classical running time of such an algorithm.

Our quantum algorithm directly applies to all deterministic exponential time algorithms for NP-hard search problems, that have been published with rigorous time bounds. Our method is not directly applicable to the few randomized algorithms in this area, like the one based on the approach pioneered by Schöning [9]. His method is based on local search starting from random points.

As mentioned before, a quadratic speed-up is possible for Schöning's algorithm due to its regularity. Our algorithm can easily handle the derandomized version of his algorithm [13]. The important point is that this derandomized version is based on a computation tree with strict bounds on the sizes of its subtrees rooted at any internal node.

Alternatively, we can easily extend our method using the ideas of Brassard et al. [6]. Every coin toss is expanded to a branching where both paths are followed. It does not matter that the resulting deterministic algorithm would be too slow. When many paths are accepting, then the amplitude amplification is less costly.

In this way we could also handle algorithms that might be composed of randomized parts and parts based on recursive decomposition. For example, one could imagine an algorithm for SAT based on a random selection of sub-clauses followed by a sophisticated deterministic satisfiability test of the resulting formula by recursive decomposition.

2 The Classical Analysis of Algorithms Based on Recursive Decomposition

Typical recursive decomposition algorithms for SAT branch on a variable x occurring in the boolean formula with n variables. Two branches are created by setting x true and false respectively, and simplifying the resulting formulas. A bound on the branching depth of n and a bound on the number of leaves of 2^n follows immediately. Many more efficient exponential time algorithms have a computation tree with only $\tilde{O}(c^n)$ leaves for some $c < 2$.

Typical efficient exponential algorithms for this and other problems can be quite complicated and involve many cases. This implies a little more programming, but is not an obstacle for obtaining a fast quantum algorithm based on our general design.

Just to illustrate the principle, let us consider a simple but still much improved algorithm for 3SAT. Instead of branching on an arbitrary variable, it branches on a variable occurring in a shortest clause. Naturally, clauses of size 1 just simplify without any branching. If we are always lucky and find a clause with 2 variables, then on one branch the occurring literal is set to false and the other literal is forced to be true for free. This would result in a computation tree with at most $L(n)$ leaves, where $L(n)$ satisfies the following recurrence.

$$L(n) \leq L(n-1) + L(n-2)$$

The resulting running time is also $L(n)$ up to a polynomial factor. Clearly in the worst case we would not always be so lucky. Sometimes, we have to branch on variables from 3 literal clauses. Still on one branch we have then produced a 2 literal clause, resulting in

$$L(n) \leq L(n-1) + L(n-2) + L(n-3)$$

Using the terminology of Kullmann [14,15], we say that in the first case, the corresponding node ν in the tree has 2 branches labeled 1 and 2. The *branching tuple* for ν is $(1,2)$. In the second case, the corresponding node ν' in the tree has 3 branches and a branching tuple of $(1,2,3)$, representing the decreases of the number of variables in each branch.

When a node has a branching tuple $(t_1, \ldots, t_d)$, then its *branching number* is the unique positive real solution of the equation

$$\sum_{j=1}^{d} x^{-t_j} = 1$$

It is important to notice that the branching number is not at all an upper bound on the degree of the computation tree, even though for a trivial brute-force algorithm, the branching number is just 2. In general the branching number is a tool to measure the progress in one recursive step even when the tree is very unbalanced and not nicely structured.

If x is the largest branching number in a tree, then x^n is an upper bound on the number of leaves (and thus an upper bound on the corresponding running time up to a polynomial factor).

For various NP-complete problems, many nontrivial (but still exponential time) algorithms have been designed based on this principle. Sometimes, such algorithms need an elaborate case analysis. Nevertheless, they are immediately amenable to our solution.

3 The Quantum Algorithm Based on Recursive Decomposition

We consider a problem in NP for which we know a classical exponential time algorithm based on recursive decomposition. This means that the problem is self reducible, and a computation consists of repeated reductions of a current instance to a finite number of smaller instances. The size of an instance is measured by a real parameter value p. Often the parameter is actually restricted to be a non-negative integer, like $p = n$, the number of variables in our 3SAT example.

We represent the algorithm by a computation tree. The children of a node can be viewed as the recursive calls made form that node. Hence, a sequential algorithm traverses the whole computation tree. On a parallel machine with an unbounded number of processors, one could handle all nodes of the same depth simultaneously, resulting in a parallel running time given by the height of the computation tree.

This is not how our quantum algorithm works. Its running time is the square root of the upper bound on the number of leaves (up to a polynomial factor), as long as the height is polynomial. For the sequential algorithm, this upper bound on the number of leaves (based on using the worst case branching number in each vertex) is the proved upper bound on the running time (up to a polynomial factor). The corresponding quantum algorithm runs a superposition of the computations corresponding to all paths from the root to any leaf.

The root of the computation tree represents the input instance. Every node in the tree represents an instance and its children represent the smaller instances to which the parent instance is reduced. Every node has a branching number, based on the decreases of the parameter value from the parent to its children, as defined in the previous section. We know an upper bound on all branching numbers in the tree. Typically, we have a small fixed number of possible types of nodes (2 in our simple 3SAT example) and thus a small number of different branching numbers.

If x is an upper bound on the branching numbers of a computation tree, then it is easy to see that every subtree of a node with size parameter p has at most x^p leaves. Imposing an arbitrary fixed order on the children of every node, it is very easy to compute an estimate of the number of leaves to the left of the subtree of a currently visited node, inductively assuming that such an estimate is available for the parent node, and that the type of branching in the parent node is known.

As x is just a real algebraic number, there might well be some rounding problems involved, because we want our estimate not just to be an approximate value, but an exact upper bound. On the other hand, the bound does not have to be tight. By just doubling the estimate and rounding, we are safe. Every partial result $2x^p$ is rounded to any one of the two adjacent integers. (If we cannot easily figure out what the 2 integers are, then we are actually very close to one of them, and we obviously choose that one.)

What we have done in effect is enumerating the leaves from left to right, but with using at most half the numbers in the appropriate range. The gaps between any two numbers assigned to adjacent leaves fluctuate due to the rounding errors, with additional gaps introduced by overestimation of some branching numbers by the maximal branching number. But in any case, the leaves are enumerated from left to right, and no two leaves are assigned the same number. The differences between adjacent numbers assigned to leaves are at least 1 (and in the average at least 2).

We assume now that our problem instance has just one solution. Then the quantum algorithm proceeds as follows.

Algorithm $\mathcal{A}$:

```
Let $x$ be the maximal branching number and $p$ the size parameter value for the input.
Let $k = \lceil \log_2(2x^p) \rceil = \lceil p \log_2 x + 1 \rceil$.
Start with the input and the string $0^k$
Apply a Fourier transform to obtain a superposition of all non-negative integers less
```

than 2^k, represented by all 0-1-strings s of length k, each with amplitude $2^{-k/2}$.
In parallel for each s in the superposition execute the one branch of the exponential size classical computation tree that leads to the leaf numbered s (if it exists). (This takes polynomial time.)
Produce a result bit b of 1, if the searched branch of the computation has found a solution. The result bit is 0, if no solution has been found. The latter includes the case where s does not occur as the name of any leaf (because there is a gap at this point).
Undo the computation, erasing auxiliary partial results, just keeping the result bit b (together with s and the input).
Run Grover's algorithm to search for $b = 1$.

This algorithm is of the 1-sided Monte Carlo type. If solutions exist, then one of them is found with high probability, but no proof of nonexistence of solutions is accomplished.

As the running time is dominated by Grover's search algorithm, we obtain the following result.

Theorem 1. *Let $\mathcal{P}$ be a problem in NP that has a classical solution by recursive decomposition. Assume the maximal branching number is x with respect to a parameter with initial value p. Further assume, the input is an instance with a unique solution. Then with constant probability, the quantum Algorithm $\mathcal{A}$ finds the solution in time $\tilde{O}(\sqrt{x^p})$.*

If the problem instance possibly has multiple solutions, then we employ the generalization of Grover's algorithm to multiobject search by Boyer, Brassard, Høyer and Tapp [16]. The algorithm $\mathcal{A}'$ obtained by this replacement gives us the following result.

Theorem 2. *Let $\mathcal{P}$ be a problem in NP that has a classical solution by recursive decomposition. Assume the maximal branching number is x with respect to a parameter with initial value p. Then with constant probability, the quantum Algorithm $\mathcal{A}'$ finds the solution in time $\tilde{O}(\sqrt{x^p})$.*

One could also use any one of a newer class of search algorithms by Grover [17,18]. These algorithms have been extended to multiobject search by Chen and Sun [19].

Repeating $\mathcal{A}'$ polynomially often immediately produces the following result.

Corollary 1. *Let $\mathcal{P}$ be a problem in NP that has a classical solution by recursive decomposition. Assume the maximal branching number is x with respect to a parameter with initial value p. Then there is a polynomial time algorithm solving $\mathcal{P}$ with exponentially small error probability by making a polynomial number of calls to an $\tilde{O}(\sqrt{x^p})$ time quantum algorithm.*

Note that under the conditions of the Corollary, the corresponding classical algorithm takes time $\tilde{O}(x^p)$.

4 Applications

There are plenty of applications. After a slow start, many sophisticated and efficient exponential time deterministic algorithms have appeared over the last dacade, e.g., [20,14,21,4,22,23,24]. All of these algorithms for NP-hard problems fit into the scheme of recursive decomposition. This includes algorithms for Satisfiability, Maximum Independent Set, Graph Coloring, and other Constraint Satisfaction Problems. The method works fine for optimization problems too. One just applies binary search for the optimal value. Thus, for example, to find a maximum independent set, one would run the quantum algorithm with a logarithmic number of choices for the size of such a set.

References

1. Grover, L.K.: Quantum mechanics helps in searching for a needle in a haystack. Physical Review Letters 79(2), 325–328 (1997)
2. Grover, L.K.: A framework for fast quantum mechanical algorithms. In: Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC 1998), pp. 53–62. ACM Press, New York (1998)
3. Angelsmark, O., Dahllöf, V., Jonsson, P.: Finite domain constraint satisfaction using quantum computation. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 93–103. Springer, Heidelberg (2002)
4. Eppstein, D.: Improved algorithms for 3-Coloring, 3-Edge-Coloring, and constraint satisfaction. In: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2001), pp. 329–337. ACM Press, New York (2001)
5. Cerf, N., Grover, L., Williams, C.: Nested quantum search and structured problems. Phys. Rev. A 61(3) (2000) 14 032303.
6. Brassard, G., Høyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. In: Lomonaco Jr., S.J., Brandt, H.E. (eds.) Quantum Computation and Information, AMS Contemporary Mathematics, vol. 305, pp. 53–74 (2002), http://arxiv.org/abs/quant-ph/0005055
7. Ambainis, A.: Quantum search algorithms. ACM SIGACT News 35(2), 22–35 (2004)
8. Dantsin, E., Kreinovich, V., Wolpert, A.: On quantum versions of record-breaking algorithms for sat. ACM SIGACT News 36(4), 103–108 (2005)
9. Schöning, U.: A probabilistic algorithm for k-SAT and constraint satisfaction problems. In: 40th Annual Symposium on Foundations of Computer Science (FOCS 1999), Washington - Brussels - Tokyo, pp. 410–414. IEEE, Los Alamitos (1999)
10. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of Association Computer Machinery 7, 201–215 (1960)
11. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM 5(7), 394–397 (1962)
12. Woeginger, G.: Exact algorithms for NP-hard problems: A survey. In: Jünger, M., Reinelt, G., Rinaldi, G. (eds.) Combinatorial Optimization - Eureka, You Shrink! LNCS, vol. 2570, pp. 185–207. Springer, Heidelberg (2003)
13. Dantsin, E., Goerdt, A., Hirsch, E.A., Kannan, R., Kleinberg, J., Papadimitriou, C., Raghavan, P., Schöning, U.: A deterministic $(2 - 2/(k + 1))^n$ algorithm for k-SAT based on local search. Theoretical Computer Science 289(1), 69–83 (2002)

14. Kullmann, O.: New methods for 3-SAT decision and worst-case analysis. Theoretical Computer Science 223, 1–72 (1999)
15. Dahllöf, V., Jonsson, P., Wahlström, M.: Counting models for 2SAT and 3SAT formulae. Theoretical Computer Science 332(1-3), 265–291 (2005)
16. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. Fortsch. Phys. 46, 493–506 (1998)
17. Grover, L.K.: Quantum computers can search rapidly by using almost any transformation. Physical Review Letters 80, 4329–4332 (1998)
18. Grover, L.K.: Rapid sampling through quantum computing. In: Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC 2000), pp. 618–626 (2000)
19. Chen, G., Sun, S.: Generalization of Grover's algorithm to multiobject search in quantum computing, Part II: general unitary transformations. In: Brylinski, R., Chen, G. (eds.) Mathematics of Quantum Computation. Computational Mathematics, pp. 161–168. Chapman & Hall/CRC, Boca Raton, London, New York, Washington, D.C (2002)
20. Tarjan, R.E., Trojanowski, A.E.: Finding a maximum independent set. SIAM J. Comput. 6(3), 537–546 (1977)
21. Beigel, R.: Finding maximum independent sets in sparse and general graphs. In: SODA 1999. Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, pp. 856–857 (1999)
22. Dantsin, E., Hirsch, E.A., Ivanov, S., Vsemirnov, M.: Algorithms for SAT and upper bounds on their complexity. Technical Report TR01-012, Electronic Colloquium on Computational Complexity (ECCC) (2001)
23. Fürer, M.: A faster algorithm for finding maximum independent sets in sparse graphs. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 491–501. Springer, Heidelberg (2006)
24. Fürer, M., Kasiviswanathan, S.P.: Exact Max 2-SAT: Easier and faster. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 272–283. Springer, Heidelberg (2007)

Author Index

Abouelaoualim, A. 723
Afonin, Sergey 121
Akhavi, Ali 293
Aloupis, Greg 146
Angelopoulos, Spyros 399
Angelov, Stanislav 48
Awerbuch, Baruch 580
Azimian, Kooshiar 676

Bansal, Mukul S. 72
Bansal, Nikhil 240
Bárány, Imre 654
Bereg, Sergey 630
Bern, Marshall 617
Bienkowski, Marcin 252
Blondel, Vincent D. 84
Bonsma, Paul 531
Bose, Prosenjit 170
Brandstädt, Andreas 479
Bui-Xuan, B.-M. 492
Bunde, David P. 240

Cardinal, Jean 146
Carmi, Paz 170
Chan, Ho-Leung 240
Cheng, Christine T. 568
Chung, Christine 228
Collette, Sébastien 146
Cornuéjols, Gérard 317
Couture, Mathieu 170
Czyzowicz, J. 158

Das, K.Ch. 723
Daudé, Hervé 12
Dean, Brian C. 449
Dellamonica, Domingos 664
Deshpande, Amol 436
Dobrev, S. 158
Dong, Jianrong 72
Dorrigiv, Reza 399
Dragan, Feodor F. 555
Dudek, Andrzej 473

Epstein, Leah 264
Erlebach, Thomas 747

Faria, L. 723
Fernández-Baca, David 72
Fernandes, Cristina G. 329
Ferreira, Carlos E. 329
Fevens, T. 158
Fomin, Fedor V. 194
Fürer, Martin 784

González, Rodrigo 374
González-Aguilar, H. 158
Griffis, Adam 449
Gupta, Anupam 36
Gupta, Arvind 182

Habib, M. 492
Hayes, Barry 617
Hazan, Elad 306
Heggernes, Pinar 216
Hell, Pavol 182, 520
Hoefer, Martin 339
Hong, Seok-Hee 461
Hüffner, Falk 711
Hundt, Christian 479
Hung, Regant Y.S. 699
Hwang, H.-K. 1

Ibarra, Oscar H. 94
Inui, Yoshifumi 772
Ivanyos, Gábor 759

Jiang, Minghui 630
Jungers, Raphaël M. 84

Karakostas, George 350
Karhumäki, Juhani 94
Karimi, Mehdi 182
Khandekar, Rohit 580
Khuller, Samir 436
Komusiewicz, Christian 711
Kortsarz, Guy 423
Kranakis, Evangelos 158, 605
Kratochvíl, Jan 194
Kratsch, Dieter 216
Krizanc, Danny 605
Kunal, Keshav 48

Langerman, Stefan 146
Le Gall, François 772
Li, Zhentao 206
Lingas, Andrzej 736
Lokshtanov, Daniel 194
Lopez, Mario A. 642
López-Ortiz, Alejandro 399
Lundell, Eva-Marta 736

Mądry, Aleksander 252
Maheshwari, Anil 170
Malekian, Azarakhsh 436
Maletti, Andreas 106
Mancini, Federico 194
Manoussakis, Y. 723
Margot, François 317
Markou, Euripides 350
Martínez, Conrado 504
Martinhon, C. 723
Mayster, Yan 642
McCutchen, Richard Matthew 593
McGregor, Andrew 48
Meister, Daniel 216
Mirrokni, Vahab S. 423
Morin, Pat 170, 605
Moser, Hannes 711
Moura, Lucia 504

Nagamochi, Hiroshi 461
Navarro, Gonzalo 362, 374
Nekrich, Yakov 687
Nicodème, P. 1
Niedermeier, Rolf 711
Nutov, Zeev 411, 423

Okhotin, Alexander 94
Oliveira, Arlindo L. 362
Opatrny, J. 158

Panario, Daniel 504
Panigrahy, Rina 387
Park, G. 1
Pór, Attila 654
Protasov, Vladimir Yu. 84
Pruhs, Kirk 228, 240

Rafiey, Arash 182
Rapaport, I. 24
Raspaud, André 520
Ravelomanana, Vlady 12
Reed, Bruce 206
Rödl, Vojtěch 473
Russo, Luís M.S. 362

Saad, R. 723
Sanselme, Luc 759
Santha, Miklos 759
Scheder, Dominik 60
Shparlinski, Igor E. 276, 284
Silveira, Rodrigo I. 133
Smid, Michiel 170
Smorodinsky, Shakhar 146
Stacho, Juraj 520, 544
Stehlé, Damien 293
Stevens, Brett 504
Suchan, K. 24
Szegedy, Mario 676
Szpankowski, W. 1

Talwar, Kunal 36
Telle, Jan Arne 194
Ting, H.F. 699
Tjandraatmadja, Christian 329
Todinca, I. 24
Toossi, Mohammed 436
Tsanko, Elena 423

Urrutia, J. 158
Uthaisombut, Patchrawat 228

Valtr, Pavel 654
van Kreveld, Marc 133
van Leeuwen, Erik Jan 747
van Stee, Rob 264
Verstraete, J. 24
von zur Gathen, Joachim 276

Wakabayashi, Yoshiko 329
Wang, Wencheng 630
Whitley, Adam 449

Xiang, Yang 555

Yan, Chenyu 555
Yang, Boting 630

Zhu, Binhai 630
Zickfeld, Florian 531