

Guohui Lin (Ed.)

LNCS 4598

Computing and Combinatorics

13th Annual International Conference, COCOON 2007
Banff, Canada, July 2007
Proceedings



Springer

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

University of Dortmund, Germany

Madhu Sudan

Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Moshe Y. Vardi

Rice University, Houston, TX, USA

Gerhard Weikum

Max-Planck Institute of Computer Science, Saarbruecken, Germany

Guohui Lin (Ed.)

Computing and Combinatorics

13th Annual International Conference, COCOON 2007
Banff, Canada, July 16-19, 2007
Proceedings



Springer

Volume Editor

Guohui Lin

Algorithmic Research Group and Bioinformatics Research Group

Department of Computing Science

University of Alberta

Edmonton, Alberta T6G 2E8, Canada

E-mail: ghlin@cs.ualberta.ca

Library of Congress Control Number: Applied for

CR Subject Classification (1998): F.2, G.2, I.3.5, C.2.3-4, E.1, E.5, E.4

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

ISSN 0302-9743

ISBN-10 3-540-73544-5 Springer Berlin Heidelberg New York

ISBN-13 978-3-540-73544-1 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2007

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12088638 06/3180 5 4 3 2 1 0

Preface

The Annual International Computing and Combinatorics Conference is an annual forum for exploring research, development, and novel applications of computing and combinatorics. It brings together researchers, professionals and industrial practitioners to interact and exchange knowledge, ideas and progress. The topics cover most aspects of theoretical computer science and combinatorics related to computing. The 13th Annual International Computing and Combinatorics Conference (COCOON 2007) was held in Banff, Alberta during July 16–19, 2007. This was the first time that COCOON was held in Canada.

We received 165 submissions, among which 11 were withdrawn for various reasons. The remaining 154 submissions under full consideration came from 33 countries and regions: Australia, Brazil, Canada, China, the Czech Republic, Denmark, Finland, France, Germany, Greece, Hong Kong, India, Iran, Ireland, Israel, Italy, Japan, the Netherlands, Norway, Pakistan, Poland, Romania, Russia, Slovakia, South Korea, Spain, Sweden, Switzerland, Taiwan, Turkey, the UK, the USA, and the US minor outlying islands.

After a six week period of careful reviewing and discussions, the program committee accepted 51 submissions for oral presentation at the conference. Based on the affiliations, 1.08 of the accepted papers were from Australia, 7.67 from Canada, 3.08 from China, 1 from the Czech Republic, 2 from Denmark, 1 from France, 5.42 from Germany, 0.08 from Greece, 2.18 from Hong Kong, 0.33 from India, 0.17 from Ireland, 1.83 from Israel, 1.5 from Italy, 2.9 from Japan, 0.17 from the Netherlands, 2.67 from Norway, 0.5 from Poland, 1 from Switzerland, 1 from Taiwan, 0.08 from Turkey, 1.33 from the UK, 12.33 from the USA, and 0.33 from the US minor outlying islands. The program of COCOON 2007 also included three keynote talks by Srinivas Aluru, Francis Y. L. Chin, and Ming Li.

Finally, we would like to express our gratitude to the authors of all submissions, the members of the program committee and the external reviewers, the members of the organizing committee, the keynote speakers, our generous sponsors, and the supporting organizations for making COCOON 2007 possible and enjoyable.

July 2007

Guohui Lin

Organization

COCOON 2007 was sponsored by the Department of Computing Science, the University of Alberta, and the Informatics Circle of Research Excellence (iCORE).

Program Committee Chair

Guohui Lin (University of Alberta, Canada)

Program Committee Members

Therese Biedl (University of Waterloo, Canada)
Andreas Brandstädt (Universität Rostock, Germany)
Zhi-Zhong Chen (Tokyo Denki University, Japan)
Ovidiu Daescu (University of Texas at Dallas, USA)
Donglei Du (University of New Brunswick, Canada)
Patricia Evans (University of New Brunswick, Canada)
Yong Gao (University of British Columbia - Okanagan, Canada)
Raffaele Giancarlo (University of Palermo, Italy)
Wen-Lian Hsu (Academia Sinica, Taiwan)
Xiao-Dong Hu (Chinese Academy of Sciences, China)
Ming-Yang Kao (Northwestern University, USA)
Naoki Katoh (Kyoto University, Japan)
Valerie King (University of Victoria, Canada)
Michael A. Langston (University of Tennessee, USA)
Kim Skak Larsen (University of Southern Denmark, Denmark)
Bin Ma (University of Western Ontario, Canada)
Rajeev Motwani (Stanford University, USA)
Mohammad R. Salavatipour (University of Alberta, Canada)
David Sankoff (University of Ottawa, Canada)
Yaoyun Shi (University of Michigan, USA)
Zhiyi Tan (Zhejiang University, China)
Caoan Wang (Memorial University of Newfoundland, Canada)
Lusheng Wang (City University of Hong Kong, China)
Osamu Watanabe (Tokyo Institute of Technology, Japan)
Dong Xu (University of Missouri - Columbia, USA)
Jinhui Xu (State University of New York at Buffalo, USA)
Alexander Zelikovskiy (Georgia State University, USA)
Huaming Zhang (University of Alabama in Huntsville, USA)
Kaizhong Zhang (University of Western Ontario, Canada)
Xizhong Zheng (BTU Cottbus, Germany)
Yunhong Zhou (HP, USA)
Binhai Zhu (Montana State University, USA)

Organizing Committee

Guohui Lin (Chair; University of Alberta, Canada)

Zhipeng Cai (University of Alberta, Canada)

Yi Shi (University of Alberta, Canada)

Meng Song (University of Alberta, Canada)

Jianjun Zhou (University of Alberta, Canada)

External Reviewers

Irina Astrovskaya

Dumitru Brinza

Zhipeng Cai

Shihyen Chen

Qiong Cheng

Arthur Chou

Ye Du

Martin R. Ehmsen

Konrad Engel

Tomas Feder

Fedor Fomin

Lance Fortnow

Loukas Georgiadis

Stephan Gremalschi

MohammadTaghi Hajiaghayi

Qiaoming Han

Jan Johannsen

Valentine Kabanets

George Karakostas

Ker-I Ko

Ekkehard Kohler

Guy Kortsarz

Lap Chi Lau

Hanno Lefmann

Weiming Li

Xueping Li

Xiaowen Liu

Jingping Liu

Shuang Luan

Elvira Mayordomo

Daniele Micciancio

Lopamudra Mukherjee

Shubha Nabar

Javier Pena

Xiaotong Qi

Tim Roughgarden

Adrian Rusu

Amin Saberi

Yi Shi

Amir Shpilka

Vikas Singh

Ram Swaminathan

Dilys Thomas

Iannis Turlakakis

Sergei Vassilvitskii

Jacques Verstraete

Peter Wagner

Duncan Wang

Kelly Westbrooks

Avi Wigderson

Gang Wu

Xiaodong Wu

Lei Xin

Dachuan Xu

Ying Xu

Jin Yan

Yang Yang

Guochuan Zhang

Li Zhang

Jianjun Zhou

Luis Zuluaga

Table of Contents

The Combinatorics of Sequencing the Corn Genome	1
<i>Srinivas Aluru</i>	
Online Frequency Assignment in Wireless Communication Networks	2
<i>Francis Y.L. Chin</i>	
Information Distance from a Question to an Answer	3
<i>Ming Li</i>	
A New Field Splitting Algorithm for Intensity-Modulated Radiation Therapy	4
<i>Danny Z. Chen, Mark A. Healy, Chao Wang, and Xiaodong Wu</i>	
A New Recombination Lower Bound and the Minimum Perfect Phylogenetic Forest Problem	16
<i>Yufeng Wu and Dan Gusfield</i>	
Seed-Based Exclusion Method for Non-coding RNA Gene Search	27
<i>Jean-Eudes Duchesne, Mathieu Giraud, and Nadia El-Mabrouk</i>	
A New Quartet Approach for Reconstructing Phylogenetic Trees: Quartet Joining Method	40
<i>Lei Xin, Bin Ma, and Kaizhong Zhang</i>	
Integer Programming Formulations and Computations Solving Phylogenetic and Population Genetic Problems with Missing or Genotypic Data	51
<i>Dan Gusfield, Yelena Frid, and Dan Brown</i>	
Improved Exact Algorithms for Counting 3- and 4-Colorings	65
<i>Fedor V. Fomin, Serge Gaspers, and Saket Saurabh</i>	
Connected Coloring Completion for General Graphs: Algorithms and Complexity	75
<i>Benny Chor, Michael Fellows, Mark A. Ragan, Igor Razgon, Frances Rosamond, and Sagi Snir</i>	
Quadratic Kernelization for Convex Recoloring of Trees	86
<i>Hans L. Bodlaender, Michael R. Fellows, Michael A. Langston, Mark A. Ragan, Frances A. Rosamond, and Mark Weyer</i>	
On the Number of Cycles in Planar Graphs	97
<i>Kevin Buchin, Christian Knauer, Klaus Kriegel, André Schulz, and Raimund Seidel</i>	
An Improved Exact Algorithm for Cubic Graph TSP	108
<i>Kazuo Iwama and Takuya Nakashima</i>	

Geometric Intersection Graphs: Do Short Cycles Help?	118
<i>Jan Kratochvíl and Martin Pergel</i>	
Dimension, Halfspaces, and the Density of Hard Sets	129
<i>Ryan C. Harkins and John M. Hitchcock</i>	
Isolation Concepts for Enumerating Dense Subgraphs	140
<i>Christian Komusiewicz, Falk Hüffner, Hannes Moser, and Rolf Niedermeier</i>	
Alignments with Non-overlapping Moves, Inversions and Tandem Duplications in $O(n^4)$ Time	151
<i>Christian Ledergerber and Christophe Dessimoz</i>	
Counting Minimum Weighted Dominating Sets	165
<i>Fedor V. Fomin and Alexey A. Stepanov</i>	
Online Interval Scheduling: Randomized and Multiprocessor Cases	176
<i>Stanley P.Y. Fung, Chung Keung Poon, and Feifeng Zheng</i>	
Scheduling Selfish Tasks: About the Performance of Truthful Algorithms	187
<i>George Christodoulou, Laurent Gourvès, and Fanny Pascual</i>	
Volume Computation Using a Direct Monte Carlo Method	198
<i>Sheng Liu, Jian Zhang, and Binhai Zhu</i>	
Improved Throughput Bounds for Interference-Aware Routing in Wireless Networks	210
<i>Chiranjeeb Buragohain, Subhash Suri, Csaba D. Tóth, and Yunhong Zhou</i>	
Generating Minimal k -Vertex Connected Spanning Subgraphs	222
<i>Endre Boros, Konrad Borys, Khaled Elbassioni, Vladimir Gurvich, Kazuhisa Makino, and Gabor Rudolf</i>	
Finding Many Optimal Paths Without Growing Any Optimal Path Trees	232
<i>Danny Z. Chen and Ewa Misiotek</i>	
Enumerating Constrained Non-crossing Geometric Spanning Trees	243
<i>Naoki Katoh and Shin-ichi Tanigawa</i>	
Colored Simultaneous Geometric Embeddings	254
<i>U. Brandes, C. Erten, J. Fowler, F. Frati, M. Geyer, C. Gutwenger, S. Hong, M. Kaufmann, S.G. Kobourov, G. Liotta, P. Mutzel, and A. Symvonis</i>	
Properties of Symmetric Incentive Compatible Auctions	264
<i>Xiaotie Deng, Kazuo Iwama, Qi Qi, Aries Wei Sun, and Toyotaka Tasaka</i>	
Finding Equilibria in Games of No Chance	274
<i>Kristoffer Arnsfelt Hansen, Peter Bro Miltersen, and Troels Bjerre Sørensen</i>	

Efficient Testing of Forecasts	285
<i>Ching-Lueh Chang and Yuh-Dauh Lyuu</i>	
When Does Greedy Learning of Relevant Attributes Succeed?—A Fourier-Based Characterization	296
<i>Jan Arpe and Rüdiger Reischuk</i>	
The Informational Content of Canonical Disjoint NP-Pairs	307
<i>Christian Glaßer, Alan L. Selman, and Liyu Zhang</i>	
On the Representations of NC and Log-Space Real Numbers	318
<i>Fuxiang Yu</i>	
Bounded Computable Enumerability and Hierarchy of Computably Enumerable Reals	327
<i>Xizhong Zheng</i>	
Streaming Algorithms Measured in Terms of the Computed Quantity . . .	338
<i>Shengyu Zhang</i>	
A Randomized Approximation Algorithm for Parameterized 3-D Matching Counting Problem	349
<i>Yunlong Liu, Jianer Chen, and Jianxin Wang</i>	
Optimal Offline Extraction of Irredundant Motif Bases	360
<i>Alberto Apostolico and Claudia Tagliacollo</i>	
Linear Algorithm for Broadcasting in Unicyclic Graphs	372
<i>Hovhannes Harutyunyan and Edward Maraachlian</i>	
An Improved Algorithm for Online Unit Clustering	383
<i>Hamid Zarrabi-Zadeh and Timothy M. Chan</i>	
Linear Time Algorithms for Finding a Dominating Set of Fixed Size in Degenerated Graphs	394
<i>Noga Alon and Shai Gutner</i>	
Single-Edge Monotonic Sequences of Graphs and Linear-Time Algorithms for Minimal Completions and Deletions	406
<i>Pinar Heggernes and Charis Papadopoulos</i>	
On the Hardness of Optimization in Power Law Graphs	417
<i>Alessandro Ferrante, Gopal Pandurangan, and Kihong Park</i>	
Can a Graph Have Distinct Regular Partitions?	428
<i>Noga Alon, Asaf Shapira, and Uri Stav</i>	
Algorithms for Core Stability, Core Largeness, Exactness, and Extendability of Flow Games	439
<i>Qizhi Fang, Rudolf Fleischer, Jian Li, and Xiaoxun Sun</i>	
Computing Symmetric Boolean Functions by Circuits with Few Exact Threshold Gates	448
<i>Kristoffer Arnsfelt Hansen</i>	

On the Complexity of Finding an Unknown Cut Via Vertex Queries	459
<i>Peyman Afshani, Ehsan Chiniforooshan, Reza Dorrigiv, Arash Farzan, Mehdi Mirzazadeh, Narges Simjour, and Hamid Zarrabi-Zadeh</i>	
“Resistant” Polynomials and Stronger Lower Bounds for Depth-Three Arithmetical Formulas	470
<i>Maurice J. Jansen and Kenneth W. Regan</i>	
An Improved Algorithm for Tree Edit Distance Incorporating Structural Linearity	482
<i>Shihyen Chen and Kaizhong Zhang</i>	
Approximation Algorithms for Reconstructing the Duplication History of Tandem Repeats	493
<i>Lusheng Wang, Zhanyong Wang, and Zhizhong Chen</i>	
Priority Algorithms for the Subset-Sum Problem	504
<i>Yuli Ye and Allan Borodin</i>	
Distributed Approximation Algorithms for Weighted Problems in Minor-Closed Families	515
<i>A. Czygrinow and M. Hańćkowiak</i>	
A 1-Local 13/9-Competitive Algorithm for Multicoloring Hexagonal Graphs	526
<i>Francis Y.L. Chin, Yong Zhang, and Hong Zhu</i>	
Improved Algorithms for Weighted and Unweighted Set Splitting Problems	537
<i>Jianer Chen and Songjian Lu</i>	
An $\frac{8}{5}$ -Approximation Algorithm for a Hard Variant of Stable Marriage	548
<i>Robert W. Irving and David F. Manlove</i>	
Approximation Algorithms for the Black and White Traveling Salesman Problem	559
<i>Binay Bhattacharya, Yuzhuang Hu, and Alexander Kononov</i>	
Author Index	569

The Combinatorics of Sequencing the Corn Genome

Srinivas Aluru

Department of Electrical and Computer Engineering, Iowa State University
aluru@iastate.edu

Abstract. The scientific community is engaged in an ongoing, concerted effort to sequence the corn (also known as maize) genome. This genome is approximately 2.5 billion nucleotides long with an estimated 65-80team of university and private laboratory researchers under the auspices of NSF/USDA/DOE is working towards deciphering the majority of the sequence information including all genes, determining their order and orientation, and anchoring them to genetic/physical maps. In this talk, I will present some of the combinatorial problems that arise in this context and outline the role of graph, string and parallel algorithms in solving them.

Online Frequency Assignment in Wireless Communication Networks

Francis Y.L. Chin

Department of Computer Science,
The University of Hong Kong, Hong Kong
`chin@cs.hku.hk`

Abstract. Wireless communication has many applications since its invention more than a century ago. The frequency spectrum used for communication is a scarce resource and the Frequency Assignment Problem (FAP), aiming for better utilization of the frequencies, has been extensively studied in the past 20-30 years. Because of the rapid development of new wireless applications such as digital cellular network, cellular phone, the FAP problem has become more important.

In Frequency Division Multiplexing (FDM) networks, a geographic area is divided into small cellular regions or cells, usually regular hexagons in shape. Each cell contains one base station that communicates with other base stations via a high-speed wired network. Calls between any two clients (even within the same cell) must be established through base stations. When a call arrives, the nearest base station must assign a frequency from the available spectrum to the call without causing any interference with other calls. Interference may occur, which distorts the radio signals, when the same frequency is assigned to two different calls emanating from cells that are geographically close to each other. Thus the FAP problem can be viewed as a problem of multi-coloring a hexagon graph with the minimum number of colors when each vertex of the graph is associated with an integer that represents the number of calls in a cell.

FAP has attracted more attention recently because of the following:

- a) Online analysis techniques: FAP problem is known to be NP-complete and many approximation algorithms have been proposed in the past. As frequency assignments have to be done without knowledge of future call requests and releases, online algorithms have been proposed and competitive analysis has been used to measure their performance.
- b) New technology and application: Wideband Code-Division Multiple-Access (W-CDMA) technology is a new technology used for the implementation of 3G cellular system. Orthogonal Variable Spreading Factor (OVSF) codes are used to satisfy requests with different data rate requirements. FAP with OVSF code trees representing the frequency spectrum becomes an important problem.

Information Distance from a Question to an Answer

Ming Li

School of Computer Science, University of Waterloo, Waterloo,
Ontario N2L 3G1 Canada
mli@uwaterloo.ca

Abstract. We know how to measure distance from Beijing to Toronto. However, do you know how to measure the distance between two information carrying entities? For example: two genomes, two music scores, two programs, two articles, two emails, or from a question to an answer? Furthermore, such a distance measure must be application-independent, must be universal in the sense it is provably better than all other distances, and must be applicable.

From a simple and accepted assumption in thermodynamics, we have developed such a theory. I will present this theory and will present one of the new applications of this theory: a question answering system.

A New Field Splitting Algorithm for Intensity-Modulated Radiation Therapy*

Danny Z. Chen¹, Mark A. Healy¹, Chao Wang^{1,**}, and Xiaodong Wu^{2,***}

¹ Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
{chen,mhealy4,cwang1}@cse.nd.edu

² Department of Electrical and Computer Engineering
Department of Radiation Oncology
University of Iowa
Iowa City, Iowa 52242, USA
xiaodong-wu@uiowa.edu

Abstract. In this paper, we present an almost linear time algorithm for the problem of splitting an intensity map of radiation (represented as an integer matrix) into multiple subfields (submatrices), subject to a given maximum allowable subfield width, to minimize the total delivery error caused by the splitting. This problem arises in intensity-modulated radiation therapy (IMRT) for cancer treatments. This is the first field splitting result on minimizing the total delivery error of the splitting. Our solution models the problem as a shortest path problem on a directed layered graph, which satisfies the staircase Monge property. Consequently, the resulting algorithm runs in almost linear time and generates an optimal quality field splitting.

1 Introduction

In this paper, we study a geometric partition problem, called *field splitting*, which arises in intensity-modulated radiation therapy (IMRT). IMRT is a modern cancer treatment technique that aims to deliver highly conformal prescribed radiation dose distributions, called *intensity maps* (IMs), to target tumors while sparing the surrounding normal tissues and critical structures. The effectiveness of IMRT hinges on its ability to accurately and efficiently deliver the prescribed

* This research was supported in part by the National Science Foundation under Grant CCF-0515203 and NIH NIBIB Grant R01-EB004640-01A2.

** Corresponding author. The research of this author was supported in part by two Fellowships in 2004-2006 from the Center for Applied Mathematics of the University of Notre Dame.

*** The research of this author was supported in part by a faculty start-up fund from the University of Iowa and in part by a seed grant award from the American Cancer Society through an Institutional Research Grant to the Holden Comprehensive Cancer Center, the University of Iowa, Iowa City, Iowa, USA.

IMs. An IM is a dose prescription specified by a set of nonnegative integers on a uniform 2-D grid (see Figure 1(b)) with respect to an orientation in the 3-D space. The value in each grid cell indicates the intensity level of prescribed radiation at the body region corresponding to that IM cell. The delivery of an IM is carried out by a set of cylindrical radiation beams orthogonal to the IM grid.

One of the current most advanced control tools for IM delivery is the *multileaf collimator* (MLC) [15]. An MLC consists of a fixed number of pairs of tungsten alloy leaves of the same rectangular shape and size (see Figure 1(a)). The opposite leaves of each pair are aligned to each other, and can move (say) up or down to form an x -monotone rectilinear polygonal beam-shaping region. The cross-section of a cylindrical radiation beam is shaped by such a region. In delivering a radiation beam for an IM, all IM cells exposed under the beam receive a uniform radiation dose proportional to the exposure time of the beam. The mechanical design of the MLCs restricts what kinds of beam-shaping regions are allowed [15]. A common constraint is called the **maximum field size**: Due to the limitation on the fixed number of MLC leave pairs and the overtravel distance of the leaves, an MLC cannot enclose an IM of a too large size (called the *field size*).

Two key criteria are used to measure the quality of an IMRT treatment. (1) The **treatment time** (efficiency): Minimizing the treatment time is crucial since it not only lowers the treatment cost for each patient but also increases the patient throughput of the hospitals; in addition, it reduces the risk associated with the uncertainties in the treatment. (2) The **delivery error** (accuracy): Due to the special geometric shapes of the MLC leaves [14,15] (i.e., the “tongue-and-groove” interlock feature), an MLC-aperture cannot be delivered perfectly. Instead, there is a *delivery error* between the planned dose and actual delivered dose [14] (called the “*tongue-and-groove*” error in medical literature [15]). Minimizing the delivery error is important because according to a recent study [4], the maximum delivery error can be up to 10%, creating underdose/overdose spots in the target region.

The limited size of the MLC necessitates that a large-size IM field be split into two or more adjacent subfields, each of which can be delivered separately by the MLC subject to the maximum field size constraint [5,9,16]. But, such IM splitting may result in prolonged treatment time and increased delivery error, and thus affect the treatment quality. The **field splitting problem**, roughly speaking, is to split an IM of a large size into multiple subfields whose sizes are all no bigger than a threshold size, such that the treatment quality is optimized.

A few field splitting algorithms are known in the literature [3,10,12,17], which address various versions of the field splitting problem. However, all these field splitting solutions focused on minimizing the *beam-on time* while ignoring the issue of reducing the delivery error. The beam-on time of a treatment is the time while a patient is exposed to actual irradiation [15], which is closely related to the total treatment time. In fact, when splitting an IM into multiple subfields to minimize the total beam-on time, the splitting is along the direction that is perpendicular to the direction of field splitting that aims to minimize the total

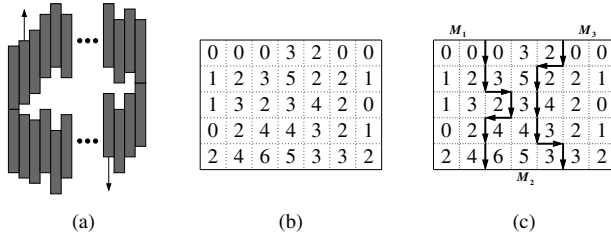


Fig. 1. (a) An MLC. (b) An IM. (c) An example of splitting an IM into three subfields, M_1 , M_2 , and M_3 , using y -monotone paths.

delivery error. For example, in Figure 1(c), the splitting of the IM is along the x -axis (using y -monotone paths) and aims to minimize the total delivery error; a corresponding splitting for minimizing the total beam-on time would be along the y -axis (say, using x -monotone paths). Thus, these two optimization criteria for field splitting, i.e., minimizing the total beam-on time and minimizing the total delivery error, are geometrically complementary to each other, and together provide a somewhat complete solution to the field splitting problem (for example, we may first split an IM along the y -axis, minimizing the total beam-on time, and then split each resulting subfield of the IM along the x -axis, minimizing its delivery error). Although it is useful to consider field splitting to minimize the delivery error, to our best knowledge, no field splitting algorithms are known aiming to minimize the total delivery error.

Several papers (e.g., [2, 11, 14]) have discussed how to minimize the delivery error of IMRT treatments (assuming no field splitting is needed). Chen *et al.* [2] showed that for an IM M of size $m \times n$ (no larger than the maximum allowable field size $l \times w$), the minimum amount of error for delivering M is captured by the following formula (note that M contains only nonnegative integers):

$$Err(M) = \sum_{i=1}^m (M_{i,1} + \sum_{j=1}^{n-1} |M_{i,j} - M_{i,j+1}| + M_{i,n}). \quad (1)$$

They also gave an algorithm for achieving this minimum error [2]. Geometrically, if we view a row of an IM as representing an x -monotone rectilinear curve f , called the *dose profile curve* (see Figure 2(a)), then the (minimum) delivery error associated with this IM row is actually the total sum of the lengths of all vertical edges on f .

In this paper, we consider the following **field splitting using y -monotone paths (FSMP) problem**: Given an IM M of size $m \times n$ and a maximum allowable field size $l \times w$, with $m \leq l$ and $n > w$, split M using y -monotone paths into $d = \lceil \frac{n}{w} \rceil$ (≥ 2) subfields M_1, M_2, \dots, M_d , each with a size no larger than $l \times w$, such that the total delivery error of these d subfields is minimized (e.g., see Figures 2(b)–2(c)). Here, d is the minimum number of subfields required to deliver M subject to the maximum allowable field size $l \times w$.

We present the first efficient algorithm for the above FSMP problem. Our algorithm runs in almost linear time. In our approach, we model the FSMP

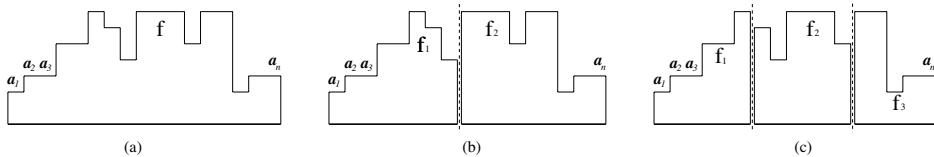


Fig. 2. (a) The dose profile curve f of one row of an IM. The (minimum) delivery error, $Err(f)$, of the row is equal to the sum of the lengths of all vertical edges on the curve f . (b) Splitting the one-row IM in (a) into two subfields. Note that $Err(f_1) + Err(f_2) \geq Err(f)$ always holds. (c) Splitting the one-row IM in (a) into three subfields. $Err(f_1) + Err(f_2) + Err(f_3) \geq Err(f)$ always holds.

problem as a shortest path problem in a directed acyclic graph (DAG) of $O(n)$ vertices and $O(nw)$ edges. Interestingly, the edge weights of this DAG satisfy the staircase Monge property [11, 13] and the edges of the graph can be represented implicitly. Thus, we can solve this shortest path problem by examining only a very small portion of the edges of the graph. One frequent operation in this shortest path algorithm is to compute the weights of the examined edges, which takes $O(mw)$ time if directly using a formula (see Section 3.3.1). We improve the time complexity of this operation to $O(m)$ by using a range-minima data structure [8]. Our final FSMP algorithm runs in $O(mn\alpha(w))$ time, where $\alpha(\cdot)$ is the inverse Ackermann's function.

Our techniques have other applications and extensions. Our FSMP algorithm can be easily adapted to the field splitting scenarios in which we seek to minimize the total horizontal or vertical complexity of the subfields [6] (all we need to do is to redefine the weights of the edges of the DAG). These criteria are believed to be closely related to the beam-on time of the treatment [6].

The rest of the paper is organized as follows. Section 2 gives some observations and notation. Section 3 presents our FSMP algorithm. Section 4 extends our solution to a related intensity map partition problem to minimize the total horizontal or vertical complexity of the resulting subfields. Section 5 shows some implementation and experimental results.

2 Preliminaries

For an IM M of size $m \times n$, we can encode any y -monotone path p on M as $(p(1), p(2), \dots, p(m)) \in [0, n]^m$, where $p(i)$ is the position at which p crosses the i -th row of M (i.e., p crosses the i -th row between the $(p(i))$ -th and $(p(i) + 1)$ -th elements). There are two special y -monotone paths of M : $\mathbf{0} = (0, 0, \dots, 0)$ and $\mathbf{n} = (n, n, \dots, n)$, which are the leftmost and rightmost paths, respectively. We assume that all paths in this paper are y -monotone on M . Define $\min(p) = \min_{i=1}^m \{p(i)\}$ and $\max(p) = \max_{i=1}^m \{p(i)\}$, i.e., $\min(p)$ and $\max(p)$ can be viewed as the smallest and largest column indices on a path p , respectively.

For two paths p' and p'' , we say that $p' \leq p''$ if $p'(k) \leq p''(k)$ holds for every k , i.e., p' lies entirely on or to the left of p'' . If $p' \leq p''$, we denote by $S(p', p'')$

the subfield of M induced by p' and p'' , i.e., $S(p', p'')$ consists of all cells of M lying between p' and p'' . The *width* of a subfield $S = S(p', p'')$ is naturally given by $\text{width}(S) = \max(p'') - \min(p')$.

We denote by $\text{band}(a, b)$ the *band region* $\{(x, y) \mid a \leq x \leq b\}$ in the plane. We say that a subfield S is embedded in a band region B if S lies entirely in B (B is called an embedding band of the subfield S). Clearly, a subfield $S(p', p'')$ is embedded in $\text{band}(a, b)$ if and only if $a \leq \min(p') \leq \max(p'') \leq b$.

Given an IM M of size $m \times n$ and the maximum allowable field size $l \times w$ (with $m \leq l$ and $n > w$), a y -monotone path set $\mathcal{T} = \{p_1, p_2, \dots, p_{d-1}\}$ is said to form a *feasible splitting* of M if (1) $\mathbf{0} \leq p_1 \leq p_2 \leq \dots \leq p_{d-1} \leq \mathbf{n}$, and (2) $\text{width}(S(p_{k-1}, p_k)) \leq w$ for every $k = 1, 2, \dots, d$ (with $p_0 = \mathbf{0}$ and $p_d = \mathbf{n}$).

It is clear that for any feasible field splitting $\{p_1, p_2, \dots, p_{d-1}\}$, the following properties hold:

- (i) For every $k = 1, 2, \dots, d-1$, $\min(p_k) \in [kw - \Delta, kw]$, where $\Delta = w \lceil n/w \rceil - n$ ($0 \leq \Delta \leq w - 1$).
- (ii) $S(\mathbf{0}, p_1)$ (resp., $S(p_{d-1}, \mathbf{n})$) is embedded in $\text{band}(0, w)$ (resp., $\text{band}(n-w, n)$); for every $k = 2, 3, \dots, d-2$, $S(p_k, p_{k+1})$ is embedded in at least one of the following bands: $\text{band}(g, g+w)$, with $g = kw - \Delta, kw - \Delta + 1, \dots, kw$.
- (iii) If $S(p_{k-1}, p_k)$ and $S(p_k, p_{k+1})$ are embedded in bands B' and B'' , respectively, then the path p_k must lie in the common intersection of the two bands, i.e., $p_k \subseteq B' \cap B''$.

3 Our Algorithm for the FSMP Problem

3.1 An Overview of the Algorithm

Our FSMP algorithm starts with the observation that whenever we use a y -monotone path p to split the given IM M , the total delivery error of the resulting subfields will *increase* by a value of $2 \cdot \sum_{i=1}^m \min\{M_{i,p(i)}, M_{i,p(i)+1}\}$, with $M_{i,0} = M_{i,n+1} = 0$ (see Figures 2(b)-2(c)). The reason is that based on the formula $\text{Err}(M)$ for summing up the delivery error of M (i.e., Formula (II) in Section II), when without a splitting between the two adjacent cells $M_{i,p(i)}$ and $M_{i,p(i)+1}$ of M , the delivery error incurred due to these two cells is $|M_{i,p(i)} - M_{i,p(i)+1}|$, but when with a splitting between these two adjacent cells, the delivery error incurred at these two cells is $M_{i,p(i)} + M_{i,p(i)+1}$ (since after the splitting, $M_{i,p(i)}$ becomes the last cell of a row in one subfield and $M_{i,p(i)+1}$ becomes the first cell of a row in another subfield). The net increase in the delivery error from without to with a splitting between $M_{i,p(i)}$ and $M_{i,p(i)+1}$ is $2 \cdot \min\{M_{i,p(i)}, M_{i,p(i)+1}\}$ (see Figures 2(b)-2(c)). Without affecting the correctness of our FSMP algorithm, we will use $\sum_{i=1}^m \min\{M_{i,p(i)}, M_{i,p(i)+1}\}$ as the (increased) cost of a y -monotone path p for splitting M (by dropping the coefficient 2). Note that this cost depends only on p and M , which we call the *cost* of p and denote by $\text{cost}(p)$. Then, the FSMP problem is equivalent to finding a feasible splitting \mathcal{T} of cardinality $d - 1$ that minimizes $\sum_{p \in \mathcal{T}} \text{cost}(p)$.

The above observation helps us to model the FSMP problem as a shortest path problem in a directed graph $G = (V, E)$, which is defined as follows. (1)

G consists of d layers of vertices, with each vertex corresponding to a possible band in M . Precisely, the first layer (i.e., layer 1) contains one vertex $v_{1,0}$, which corresponds to $\text{band}(0, w)$ ($\triangleq B_{1,0}$); the last layer (i.e., layer d) contains one vertex $v_{d,0}$, which corresponds to $\text{band}(n - w, n)$ ($\triangleq B_{d,0}$); the k -th layer ($2 \leq k \leq d - 1$) contains $\Delta + 1$ vertices, $v_{k,0}, v_{k,1}, \dots, v_{k,\Delta}$, with $v_{k,j}$ corresponding to $\text{band}(kw - j, kw - j + w)$ ($\triangleq B_{k,j}$), where $\Delta = w \lceil n/w \rceil - n$ ($= O(w)$). (2) For each vertex v_{k,j_k} in G , there is a directed edge from v_{k,j_k} to every vertex $v_{k+1,j_{k+1}}$ ($0 \leq j_{k+1} \leq \Delta$) in the next layer as long as $B_{k,j_k} \cap B_{k+1,j_{k+1}} \neq \emptyset$. Clearly, for each vertex v_{k,j_k} , the vertices on the $(k + 1)$ -th layer to which an edge from v_{k,j_k} goes form a contiguous subsequence (called an *interval of vertices*), and the first and last vertices of this interval for v_{k,j_k} can be easily determined. (3) For any edge $e = (v_{k,j_k}, v_{k+1,j_{k+1}})$ in G , we define its *induced y -monotone path*, denoted by $p_{k,j_k,j_{k+1}}^*$, as the minimum cost path lying in $B_{k,j_k} \cap B_{k+1,j_{k+1}}$; the weight of the edge $e = (v_{k,j_k}, v_{k+1,j_{k+1}})$ is the cost of its induced y -monotone path $p_{k,j_k,j_{k+1}}^*$.

Our algorithm then computes a shortest path from $v_{1,0}$ to $v_{d,0}$ in G , which has exactly $d - 1$ edges.

Two key questions for our FSMP algorithm remain to be answered.

(1) How does a shortest $v_{1,0}$ -to- $v_{d,0}$ path in G relate to an optimal field splitting of the given IM M ? As we will show in Section 3.2, it turns out that the set of $d - 1$ y -monotone paths induced by the $d - 1$ edges on such a shortest path yields an optimal splitting of M .

(2) How can we efficiently compute a shortest $v_{1,0}$ -to- $v_{d,0}$ path in G ? It is easy to see that G is a DAG with $O(d(\Delta + 1)) = O(n)$ vertices and $O(d(\Delta + 1)^2) = O(n^2/d)$ weighted edges. Thus, in a topological sort fashion, a shortest path in G can be computed in $O(n^2\tau/d)$ time, where τ is the time for computing the induced y -monotone path for each edge of G . In Section 3.3, we will show that by exploiting the staircase Monge property of the graph G [13] and using a range-minima data structure [8], we can dramatically speed up the shortest path computation. Our final FSMP algorithm runs in almost linear time on M .

3.2 Correctness of the Algorithm

In this section, we show the correctness of our FSMP algorithm, i.e., a shortest $v_{1,0}$ -to- $v_{d,0}$ path in the graph G defined in Section 3.1 indeed corresponds to an optimal splitting of the given IM M .

The following lemma states the fact that any $v_{1,0}$ -to- $v_{d,0}$ path in G induces a feasible splitting of M (we leave the proof to the full paper).

Lemma 1. *Let $\pi = v_{1,j_1(=0)} \rightarrow v_{2,j_2} \rightarrow \dots \rightarrow v_{d-1,j_{d-1}} \rightarrow v_{d,j_d(=0)}$ be a $v_{1,0}$ -to- $v_{d,0}$ path in G . Then the set of $d - 1$ y -monotone paths induced by all edges of π , i.e., $\{p_{1,j_1,j_2}^*, p_{2,j_2,j_3}^*, \dots, p_{d-1,j_{d-1},j_d}^*\}$, gives a feasible splitting of the IM M .*

We now show that a shortest $v_{1,0}$ -to- $v_{d,0}$ path in G induces an optimal splitting of M .

Lemma 2. *Let $\pi' = v_{1,j'_1(=0)} \rightarrow v_{2,j'_2} \rightarrow \cdots \rightarrow v_{d-1,j'_{d-1}} \rightarrow v_{d,j'_d(=0)}$ be a shortest $v_{1,0}$ -to- $v_{d,0}$ path in G . Then the set of $d-1$ y -monotone paths induced by all edges of π' , i.e., $\{p_1^*, p_2^*, \dots, p_{d-1}^*\}$, gives an optimal splitting of the IM M .*

Proof. Recall that the FSMP problem seeks a feasible splitting \mathcal{T} of cardinality $d-1$ that minimizes $\sum_{p \in \mathcal{T}} \text{cost}(p)$. To prove the lemma, it suffices to show that for any feasible splitting $\{p_1, p_2, \dots, p_{d-1}\}$ of M , $\sum_{k=1}^{d-1} \text{cost}(p_k) \geq \sum_{k=1}^{d-1} \text{cost}(p_k^*)$ holds.

Since $\{p_1, p_2, \dots, p_{d-1}\}$ is a feasible splitting of M , by Property (ii) in Section 2, there exist $j_1, j_2, \dots, j_{d-1} \in [0, \Delta]$, such that $S(p_k, p_{k+1})$ is embedded in the band B_{k,j_k} . For every $k = 1, 2, \dots, d-1$, by Property (iii) in Section 2, we have $p_k \subseteq B_{k,j_k} \cap B_{k+1,j_{k+1}}$ (for convenience, we define $j_0 = j_d = 0$). Hence there is an edge connecting v_{k,j_k} and $v_{k+1,j_{k+1}}$ in G . It follows that $\pi = v_{1,j_1(=0)} \rightarrow v_{2,j_2} \rightarrow \cdots \rightarrow v_{d-1,j_{d-1}} \rightarrow v_{d,j_d(=0)}$ is a $v_{1,0}$ -to- $v_{d,0}$ path in G .

Recall that p_k^* is a minimum cost y -monotone path that lies in $B_{k,j_k} \cap B_{k+1,j_{k+1}}$. Thus we have $\text{cost}(p_k) \geq \text{cost}(p_k^*)$. It follows $\sum_{k=1}^{d-1} \text{cost}(p_k) \geq \sum_{k=1}^{d-1} \text{cost}(p_k^*)$. Observe that the right-hand side of the above inequality equals the total weight of the $v_{1,0}$ -to- $v_{d,0}$ path π in G , which cannot be smaller than $\sum_{k=1}^{d-1} \text{cost}(p_k^*)$, the total weight of the shortest path π' . It thus follows that $\sum_{k=1}^{d-1} \text{cost}(p_k) \geq \sum_{k=1}^{d-1} \text{cost}(p_k^*)$. \square

3.3 Improving the Time Complexity of the Algorithm

In this section, we will focus on the more detailed aspects of our algorithm. More specifically, we will address how to efficiently compute the weights of the edges of G and a shortest $v_{1,0}$ -to- $v_{d,0}$ path in G .

3.3.1 Computing the Weights of Edges of G

One frequent operation in our FSMP algorithm is to compute the weight of a given edge of G . Recall that for an edge $e = (v_{k,j_k}, v_{k+1,j_{k+1}})$ in G , its weight $w(e)$ is defined as the cost of $p_{k,j_k,j_{k+1}}^*$, which is a minimum cost y -monotone path that lies in $B_{k,j_k} \cap B_{k+1,j_{k+1}}$.

Observe that $B_{k,j_k} \cap B_{k+1,j_{k+1}} = \text{band}(kw - j_k, kw - j_k + w) \cap \text{band}((k+1)w - j_{k+1}, (k+1)w - j_{k+1} + w) = \text{band}((k+1)w - j_{k+1}, (k+1)w - j_k)$. The existence of the edge $e = (v_{k,j_k}, v_{k+1,j_{k+1}})$ in G implies $B_{k,j_k} \cap B_{k+1,j_{k+1}} \neq \emptyset$, which in turn implies $j_k \leq j_{k+1}$.

It is not difficult to show that

$$\begin{aligned} w(e) &= \min\{\text{cost}(p) \mid p \subseteq B_{k,j_k} \cap B_{k+1,j_{k+1}}\} \\ &= \min\left\{\sum_{i=1}^m \min\{M_{i,p(i)}, M_{i,p(i)+1}\} \mid p(i) \in [(k+1)w - j_{k+1}, (k+1)w - j_k], 1 \leq i \leq m\right\} \\ &= \sum_{i=1}^m \min\{M_{i,j} \mid j \in [(k+1)w - j_{k+1}, (k+1)w - j_k + 1]\} \end{aligned} \quad (2)$$

Since $j_k, j_{k+1} \in [0, \Delta]$, it takes $O(\Delta)$ time to compute $\min\{M_{i,j} \mid j \in [(k+1)w - j_{k+1}, (k+1)w - j_k + 1]\}$, and consequently $O(m\Delta) = O(mw)$ time to compute $w(e)$ if we directly use Formula (2) above.

However, since all entries of M are static (i.e., do not change their values during the computation), we can speed up the computation of an edge weight by using a range-minima data structure [8] for each row $R_i(M) = \{M_{i,j}\}_{j=1}^n$ of M , $i = 1, 2, \dots, m$. Note that for an array of values, a range-minima data structure can be built in linear time, which allows each query of finding the minimum value in any interval of the array to be answered in $O(1)$ time [8]. In our algorithm, we build a range-minima data structure for each row $R_i(M)$ of M , in $O(n)$ time per row. This enables us to compute $\min\{M_{i,j} \mid j \in [(k+1)w - j_{k+1}, (k+1)w - j_k + 1]\}$ as a range-minima query, in $O(1)$ time. In this way, we can compute the weight of any edge of G in only $O(m)$ time, after an $O(mn)$ time preprocess (for building the range-minima data structures for the m rows of M).

3.3.2 Computing a Shortest $v_{1,0}$ -to- $v_{d,0}$ Path in G

In this section, we will show that the graph G defined in Section 3.1 satisfies the staircase Monge property [11,13]. This property enables us to compute a shortest $v_{1,0}$ -to- $v_{d,0}$ path in G in almost linear time.

Lemma 3. *Let $v_{k,j}, v_{k,j'}, v_{k+1,r}$, and $v_{k+1,r'}$ be four vertices of G , with $j < j'$, $r < r'$, and $2 \leq k < d - 1$. If $e_1 = (v_{k,j'}, v_{k+1,r})$ is an edge of G , then $e_2 = (v_{k,j}, v_{k+1,r'})$, $e_3 = (v_{k,j}, v_{k+1,r})$, and $e_4 = (v_{k,j'}, v_{k+1,r'})$ are also edges of G . Moreover, $w(e_3) + w(e_4) \leq w(e_1) + w(e_2)$.*

Proof. Since $j', r \in [0, \Delta]$, it is clear that $e_1 = (v_{k,j'}, v_{k+1,r}) \in E(G) \Leftrightarrow B_{k,j'} \cap B_{k+1,r} \neq \emptyset \Leftrightarrow \text{band}(kw - j', kw - j' + w) \cap \text{band}((k+1)w - r, (k+1)w - r + w) \neq \emptyset \Leftrightarrow r \geq j'$. Similarly, $e_2 \in E(G) \Leftrightarrow r' \geq j$, $e_3 \in E(G) \Leftrightarrow r \geq j$, and $e_4 \in E(G) \Leftrightarrow r' \geq j'$. Since $r < r'$ and $j < j'$, we have $e_1 \in E(G) \Rightarrow j < j' \leq r < r' \Rightarrow e_2, e_3, e_4 \in E(G)$.

To show that $w(e_3) + w(e_4) \leq w(e_1) + w(e_2)$, it suffices to show that for any $i \in [1, m]$, $\min_{j \in I_3} \{M_{i,j}\} + \min_{j \in I_4} \{M_{i,j}\} \leq \min_{j \in I_1} \{M_{i,j}\} + \min_{j \in I_2} \{M_{i,j}\}$, where $I_1 = [(k+1)w - r, (k+1)w - j' + 1]$, $I_2 = [(k+1)w - r', (k+1)w - j + 1]$, $I_3 = [(k+1)w - r, (k+1)w - j + 1]$, and $I_4 = [(k+1)w - r', (k+1)w - j' + 1]$. Since $j < j' \leq r < r'$, clearly $I_1 = I_3 \cap I_4$ and $I_2 = I_3 \cup I_4$. It follows that $\min_{j \in I_1} \{M_{i,j}\} \geq \max\{\min_{j \in I_3} \{M_{i,j}\}, \min_{j \in I_4} \{M_{i,j}\}\}$. and $\min_{j \in I_2} \{M_{i,j}\} = \min\{\min_{j \in I_3} \{M_{i,j}\}, \min_{j \in I_4} \{M_{i,j}\}\}$. Hence, we have $\min_{j \in I_3} \{M_{i,j}\} + \min_{j \in I_4} \{M_{i,j}\} = \max\{\min_{j \in I_3} \{M_{i,j}\}, \min_{j \in I_4} \{M_{i,j}\}\} + \min\{\min_{j \in I_3} \{M_{i,j}\}, \min_{j \in I_4} \{M_{i,j}\}\} \leq \min_{j \in I_1} \{M_{i,j}\} + \min_{j \in I_2} \{M_{i,j}\}$. \square

Lemma 3 actually shows that the $(\Delta + 1) \times (\Delta + 1)$ matrix $W^{(k)} = (w_{j,r}^{(k)})$ ($2 \leq k < d - 1$), with $w_{j,r}^{(k)} = w((v_{k,j}, v_{k+1,r}))$ for $0 \leq j, r \leq \Delta$, is a staircase Monge matrix [11,13]. (For convenience, an entry $w_{j,r}^{(k)} = +\infty$ if $(v_{k,j}, v_{k+1,r}) \notin E(G)$.) Recall that an edge $(v_{k,j}, v_{k+1,r}) \in E(G) \Leftrightarrow r \geq j$. Hence, the matrix $W^{(k)}$ can be represented implicitly, such that each of its entry can be obtained, whenever needed, in $O(m)$ time as shown in Section 3.3.1. It is thus easy to see that, by

using the staircase Monge matrix searching algorithms [11,13], given the shortest paths from $v_{1,0}$ to all vertices on the k -th layer, only $O((\Delta + 1)\alpha(\Delta + 1)) = O(w\alpha(w))$ edges need to be examined in order to find the shortest paths from $v_{1,0}$ to all vertices on the $(k + 1)$ -th layer, where $\alpha(\cdot)$ is the inverse Ackermann's function. Hence, only $O(dw\alpha(w)) = O(n\alpha(w))$ edges in total are examined for computing a shortest $v_{1,0}$ -to- $v_{d,0}$ path in G . Thus, we have the following result.

Theorem 1. *Given an IM M of size $m \times n$, and an maximum allowable field width w , the FSMP problem on M can be solved in $O(mn\alpha(w))$ time.*

4 An Extension

In this section, we show that our FSMP technique can be adapted to the field splitting scenarios in which we seek to minimize the total horizontal or vertical complexity of the resulting subfields [6]. It is sufficient for us to focus on solving the horizontal complexity case.

The *horizontal complexity* $HC(M')$ of a subfield M' of size $l \times w$ is defined as follows $HC(M') = \sum_{i=1}^l (M'_{i,1} + \sum_{j=1}^{w-1} \max\{0, M'_{i,j+1} - M'_{i,j}\})$. The horizontal complexity is closely related to the minimum beam-on time for delivering the subfield M' , which can be computed as $\max_{i=1}^l \{M'_{i,1} + \sum_{j=1}^{w-1} \max\{0, M'_{i,j+1} - M'_{i,j}\}\}$ [7]. An intensity map with a smaller horizontal complexity is likely to be delivered more efficiently (i.e., with a smaller treatment time). Thus, it is desirable to split an IM into multiple subfields while minimizing the total horizontal complexity of the resulting subfields. We consider in this section the following **field splitting with minimized total horizontal complexity (FSHC) problem**: Given an IM M of size $m \times n$ and a maximum allowable field size $l \times w$, with $m \leq l$ and $n > w$, split M using y -monotone paths into $d = \lceil \frac{n}{w} \rceil$ (≥ 2) subfields M_1, M_2, \dots, M_d , each with a size no larger than $l \times w$, such that the total horizontal complexity of these d subfields is minimized.

A key observation for solving the FSHC problem is that whenever we use a y -monotone path p to split the given IM M , the total horizontal complexity of the resulting subfields will increase by a value of $\sum_{i=1}^m \min\{M_{i,p(i)}, M_{i,p(i)+1}\}$, where $M_{i,0} = M_{i,n+1} = 0$ and $p(i)$ denotes the position at which p crosses the i -th row of M . The reason is that, similar to that for the delivery error, the *net increase* in the horizontal complexity from without any split between $M_{i,p(i)}$ and $M_{i,p(i)+1}$ to with such a split is $\min\{M_{i,p(i)}, M_{i,p(i)+1}\}$ [17]. We use $cost(p)$ to denote the increased horizontal complexity induced by the y -monotone path p for splitting M , that is, $cost(p) = \sum_{i=1}^m \min\{M_{i,p(i)}, M_{i,p(i)+1}\}$. Then, the FSHC problem is equivalent to finding a feasible splitting \mathcal{T} of cardinality $d - 1$ that minimizes $\sum_{p \in \mathcal{T}} cost(p)$. As in Section 3, we model the FSHC problem as a shortest path problem in a directed acyclic graph G with d layers, which satisfies the staircase Monge property. By using a range-minima data structure [8], the weight of any edge of G can be computed in $O(m)$ time, after an $O(mn)$ time preprocessing. The staircase Monge property enables us to compute the shortest path in G by examining only $O(n\alpha(w))$ edges in total. Hence, we have the following result.

Theorem 2. *Given an IM M of size $m \times n$, and a maximum allowable field width w , the FSHC problem on M can be solved in $O(mn\alpha(w))$ time.*

5 Implementation and Experiments

To study the performance of our new FSMP algorithm with respect to clinical applications, we implemented the algorithm using the C programming language on Linux and Unix systems, and experimented with the resulting software on over 60 sets of clinical intensity maps obtained from the Dept. of Radiation Oncology, Univ. of Maryland Medical School. We conducted a comparison with a common simple splitting scheme that splits intensity maps along vertical lines. The algorithm for this simple splitting scheme is based on computing a d -link shortest path in a weighted directed acyclic graph, which we designate as FSVL (field-splitting using vertical lines). Both implementations are designed to take as input the intensity maps generated by CORVUS, one of the current most popular commercial planning systems.

The widths of the tested intensity maps range from 21 to 27. The maximum intensity level of each field is normalized to 100. The minimum net error increase of each intensity map is calculated for the given maximum subfield width. Table II(a) compares the sums of the total net delivery error increases for splitting each of 63 intensity map fields with the maximum subfield widths w ranging from 7 to 15. For each of the tested fields, the total net delivery error increase using our FSMP algorithm is always less than or equal to the net delivery error increase using FSVL. For all 63 tested fields, the sums of the total net delivery error increases of FSVL and FSMP are 488338 and 351014, respectively. In terms of the net delivery error increase, our FSMP algorithm showed an improvement of about 28.1% over FSVL on the medical data we used. (It should be pointed out that theoretically, it can be shown that the output of FSVL can be arbitrarily worse than FSMP’s in the worst case.)

Table 1. (a) Comparisons of the total sums of net delivery error increase for 63 intensity-modulated fields with the maximum allowable subfield widths w ranging from 7 to 15. (b) Net error increase using the FSMP algorithm with respect to the maximum allowable subfield width w , for three intensity fields from a single clinical case.

w	7	8	9	10	11	12	13	14	15	Total
FSVL	144590	84088	52390	35248	51434	33268	33204	29386	24730	488338
FSMP	142840	65854	34958	14846	41564	18808	18550	9778	3816	351014

(a)

w	7	8	9	10	11	12	13	14	15
Field 1	4000	3120	560	200	680	440	320	200	200
Field 2	4600	2120	920	120	760	600	560	360	0
Field 3	520	160	40	40	1400	200	160	0	0

(b)

On individual intensity-modulated fields, the total net delivery error generally decreases as the maximum allowable subfield width w increases. However, there are some cases in which the net increase in delivery error actually increases with an increased subfield width w . In these relatively rare cases, it may actually be more advantageous to split a field into more subfields rather than less if a smaller total delivery error caused by the field splitting is desired (of course, the increased number of resulting subfields may very well cause a considerable increase in the total delivery time). Table II(b) shows the net delivery error changes for splitting 3 intensity-modulated fields from a single clinical case using our FSMP algorithm with varying maximum allowable subfield widths w .

The FSMP program runs very fast, as predicted by its theoretical linear time bound. It completed the field splitting in less than one second.

References

1. Aggarwal, A., Park, J.: Notes on Searching in Multidimensional Monotone Arrays. In: Proc. 29th Annual IEEE Symp. on Foundations of Computer Science, pp. 497–512. IEEE Computer Society Press, Los Alamitos (1988)
2. Chen, D.Z., Hu, X.S., Luan, S., Wang, C., Wu, X.: Mountain Reduction, Block Matching, and Applications in Intensity-Modulated Radiation Therapy. In: Proc. of 21th ACM Symposium on Computational Geometry, pp. 35–44. ACM Press, New York (2005)
3. Chen, D.Z., Wang, C.: Field Splitting Problems in Intensity-Modulated Radiation Therapy. In: Proc. of 17th International Symp. on Algorithms and Computation, pp. 690–700 (2006)
4. Deng, J., Pawlicki, T., Chen, Y., Li, J., Jiang, S.B., Ma, C.-M.: The MLC Tongue-and-Groove Effect on IMRT Dose Distribution. *Physics in Medicine and Biology* 46, 1039–1060 (2001)
5. Dogan, N., Leybovich, L.B., Sethi, A., Emami, B.: Automatic Feathering of Split Fields for Step-and-Shoot Intensity Modulated Radiation Therapy. *Phys. Med. Biol.* 48, 1133–1140 (2003)
6. Dou, X., Wu, X., Bayouth, J.E., Buatti, J.M.: The Matrix Orthogonal Decomposition Problem in Intensity-Modulated Radiation Therapy. In: Proc. 12th Annual International Computing and Combinatorics Conf., pp. 156–165 (2006)
7. Engel, K.: A New Algorithm for Optimal Multileaf Collimator Field Segmentation. *Discrete Applied Mathematics* 152(1-3), 35–51 (2005)
8. Gabow, H.N., Bentley, J., Tarjan, R.E.: Scaling and Related Techniques for Geometric Problems. In: Proc. 16th Annual ACM Symp. Theory of Computing, pp. 135–143. ACM Press, New York (1984)
9. Hong, L., Kaled, A., Chui, C., Losasso, T., Hunt, M., Spirou, S., Yang, J., Amols, H., Ling, C., Fuks, Z., Leibel, S.: IMRT of Large Fields: Whole-Abdomen Irradiation. *Int. J. Radiat. Oncol. Biol. Phys.* 54, 278–289 (2002)
10. Kamath, S., Sahni, S., Li, J., Palta, J., Ranka, S.: A Generalized Field Splitting Algorithm for Optimal IMRT Delivery Efficiency. The 47th Annual Meeting and Technical Exhibition of the American Association of Physicists in Medicine (AAPM), 2005. *Also Med. Phys.* 32(6), 1890 (2005)
11. Kamath, S., Sahni, S., Ranka, S., Li, J., Palta, J.: A Comparison of Step-and-Shoot Leaf Sequencing Algorithms That Eliminate Tongue-and-Groove. *Phys. Med. Biol.* 49, 3137–3143 (2004)

12. Kamath, S., Sahni, S., Ranka, S., Li, J., Palta, J.: Optimal Field Splitting for Large Intensity-Modulated Fields. *Med. Phys.* 31(12), 3314–3323 (2004)
13. Klawe, M.M., Kleitman, D.J: An Almost Linear Time Algorithm for Generalized Matrix Searching. Technical Report RJ 6275, IBM Research Division, Almaden Research Center (August 1988)
14. Luan, S., Wang, C., Chen, D.Z., Hu, X.S., Naqvi, S.A., Wu, X., Yu, C.X.: An Improved MLC Segmentation Algorithm and Software for Step-and-Shoot IMRT Delivery without Tongue-and-Groove Error. *Med. Phys.* 33(5), 1199–1212 (2006)
15. Webb, S.: Intensity-Modulated Radiation Therapy. Institute of Cancer Research and Royal Marsden NHS Trust (2001)
16. Wu, Q., Arnfield, M., Tong, S., Wu, Y., Mohan, R.: Dynamic Splitting of Large Intensity-Modulated Fields. *Phys. Med. Biol.* 45, 1731–1740 (2000)
17. Wu, X.: Efficient Algorithms for Intensity Map Splitting Problems in Radiation Therapy. In: Proc. 11th Annual International Computing and Combinatorics Conf., pp. 504–513 (2005)

A New Recombination Lower Bound and the Minimum Perfect Phylogenetic Forest Problem

Yufeng Wu and Dan Gusfield

Department of Computer Science
University of California, Davis
Davis, CA 95616, U.S.A.
{wuyu,gusfield}@cs.ucdavis.edu

Abstract. Understanding recombination is a central problem in population genetics. In this paper, we address an established problem in Computational Biology: compute lower bounds on the minimum number of historical recombinations for generating a set of sequences [1][3][9][12][15]. In particular, we propose a new recombination lower bound: the forest bound. We show that the forest bound can be formulated as the minimum perfect phylogenetic forest problem, a natural extension to the classic binary perfect phylogeny problem, which may be of interests on its own. We then show that the forest bound is provably higher than the optimal haplotype bound [13], a very good lower bound in practice [15]. We prove that, like several other lower bounds [2], computing the forest bound is NP-hard. Finally, we describe an integer linear programming (ILP) formulation that computes the forest bound precisely for certain range of data. Simulation results show that the forest bound may be useful in computing lower bounds for low quality data.

1 Introduction

Meiotic recombination is an important biological process which has a major effect on shaping the genetic diversity of populations. Recombination takes two equal length sequences and produces a third sequence of the same length consisting of some prefix of one sequence, followed by a suffix of the other sequence. Estimating the frequency or the location of recombination is central to modern-day genetics. Recombination also plays a crucial role in the ongoing efforts of association mapping. Association mapping is widely hoped to help locate genes that influence complex genetic diseases. The increasingly available population genetic variation data provides opportunities for better understanding of recombination.

In this paper, we assume the input data consists of *single nucleotide polymorphisms (SNPs)*. A SNP is a nucleotide site where exactly two (of four) different nucleotides occur in a large percentage of the population. That is, a SNP has binary states (0 or 1). A haplotype is a binary vector, where each bit (called site) of this vector indicates the state of the SNP site for this sequence. Throughout this paper, the input to our computational problems is a set of (aligned) haplotypes (i.e. a binary matrix with n rows and m columns).

An established computational problem on recombination is to determine the *minimum* number of recombinations needed to generate a set of haplotypes from an ancestral sequence, using some specified model of the permitted site *mutations*. A mutation at a SNP site is a change of state from one nucleotide to the other nucleotides at that site. Throughout this paper, we assume that any SNP site can mutate at most once in the entire history of the sequences, which is supported by the standard *infinite sites model* in population genetics.

Given a set of binary sequences M , we let $Rmin(M)$ denote the minimum number of recombinations needed to generate the sequences M from any ancestral sequence, allowing only one mutation per site over the entire history of the sequences. The problem of computing or estimating $Rmin(M)$ has been studied in a number of papers, for example [11,13,9,12,15]. A variation to the problem occurs when a specific ancestral sequence is known in advance. No polynomial-time algorithm for either problem is known, and the second problem is known to be NP-hard [16,3]. Therefore, the problem of computing a good lower bound on the minimum number of recombinations has attracted much attention.

In this paper, we present a new lower bound on $Rmin(M)$, which has a static and intuitive meaning. This lower bound (which we call the forest bound) is closely related to the minimum perfect phylogenetic forest problem, an extension of the classic binary perfect phylogeny problem. We then demonstrate that the forest bound is provably higher than a well-known bound: the optimal haplotype bound [13,15] [4]. We resolve the complexity of computing the forest bound in a negative way with a NP-hardness proof. Finally, we give an integer linear programming formulation whose solution computes the forest bound exactly. We show empirically that this formulation can be solved in practice for data with small number of sites.

2 Background

2.1 Recombination and ARGs

For haplotype data composed of (binary) SNPs, the simplest evolutionary history that derives these haplotypes is the classic binary perfect phylogeny (if we assume the infinite sites model of mutations). The perfect phylogeny problem is to build a (rooted) tree whose leaves are labeled by rows in M and edges labeled by columns in M , and a column can label at most one edge. For the binary perfect phylogeny problem, Gusfield [7] developed a linear time algorithm. However, often the real biological data does not have a perfect phylogeny, which is partly due to recombination. In this case, we need a richer model of evolution: The evolutionary history of a set of haplotypes H , which evolve by site mutations (assuming one mutation per site) and recombination, is displayed on a directed acyclic graph called an “Ancestral Recombination Graph (ARG)” [6] (also called

¹ Throughout this paper, when we say bound A is higher than bound B, we mean that bound A is guaranteed to never be lower than bound B, and that there are examples where it is strictly higher.

phylogenetic networks in some literature). An ARG N , generating n sequences of m sites each, is a directed acyclic graph containing exactly one node (the root) with no incoming edges, and exactly n leaves with one incoming edge each. Every other node has one or two incoming edges. A node with two incoming edges is called a “recombination” node (and the two incoming edges are called recombination edges). Each site (integer) from 1 to m is assigned to exactly one edge in N , and none is assigned to any edge entering a recombination node. The sequences labeling the leaves of N are the extant sequences, i.e., the input sequences. See Gusfield, et al. [8] for a more detailed explanation.

An ARG N is called a minARG if N uses exactly $Rmin(M)$ recombinations. The ARG N may derive sequences that do not appear in M . These sequences are called Steiner sequences. Sequences in M are called input sequences.

2.2 Lower Bounds on $Rmin(M)$

There are a number of papers on *lower bounds* on $Rmin(M)$ [11,13,9,2,15]. In [12,13], Myers and Griffiths introduced the *haplotype lower bound*, which, when combined with additional ideas in [12,13] significantly outperforms the previous lower bounds. The haplotype bound, $h(M)$, is simple and efficiently computable. Consider the set of sequences M arrayed in a matrix. Then $h(M)$ is the number of distinct rows of M , minus the number of distinct columns of M , minus one. It is easy to establish that this is a lower bound on $Rmin(M)$. Simulations show that $h(M)$ by itself is a very poor bound, often a negative number. However, when used with a few more tricks, it leads to impressive lower bounds. One such trick is to compute the haplotype bound on a subset of sites from M that need not be contiguous. For a subset of sites S (not necessarily contiguous), let $M(S)$ be M restricted to the sites in S , and $h(S)$ be the haplotype bound computed on $M(S)$. It is easy to see that $h(S)$ is also a lower bound on the M . The *optimal haplotype bound* is the *highest* $h(S)$ over all choices of S . Since there are an exponential number of subsets, complete enumeration of subsets quickly becomes impractical, and the problem of computing optimal haplotype bound has also been shown to be NP-hard [2]. However, Song et al. [15] showed that integer linear programming (ILP) can be used to efficiently compute the optimal haplotype bound for the range of data of current biological interest. They also showed the optimal haplotype bound is often equal to $Rmin(M)$ in certain biological datasets.

Myers and Griffiths [13] also introduced the so-called “history bound”. The history bound is provably higher than the haplotype bound [1]. However, the history bound is defined only by a computational procedure (described below), and there is no *static* and intuitive meaning provided for this bound in [13], independent of the procedure to compute it. This makes both it difficult to find alternative methods to compute the history bound, or to understand and improve it. To compute the history bound for a set of binary sequences M , we initialize $R = 0$. A site c in M is called non-informative when entries in column c have only a single 0 or a single 1. A *cleanup step* is defined as the removal of any non-informative site in M , or the merging of two duplicate rows in M

into one row. A *row removal step* arbitrarily picks one row in M for removal, provided that no cleanup step is possible. A history is defined by an execution of the following algorithm:

- Repeat a) and b) until M contains only one sequence:
- a) Perform cleanup steps until no more are possible.
 - b) Perform one row removal step; increment R by one.

The history lower bound is equal to the *minimum value* of R over *all* possible histories (i.e. the ways of choosing a row in the row removal step). The correctness of the history bound can be proved by induction [13]. Computing the history bound is NP-hard and a dynamic programming algorithm with $O(2^n m)$ running time is given in Bafna et al. [2] (which improves upon the original implementation by Myers and Griffiths [13]).

3 The Forest Bound and the Minimum Perfect Phylogenetic Forest (MPPF) Problem

3.1 Definition of the Forest Bound

The optimal haplotype bound is currently one of the strongest lower bounds. In the following, we define and describe a lower bound that can be proved to be higher than the optimal haplotype bound.

Given an arbitrary ARG N , suppose we remove *all* recombination edges. N is then decomposed into connected components, each of which is a directed perfect phylogeny (sometimes simply referred to as a directed tree). Some of the tree edges are labeled by site mutations in the original ARG, and thus, we have a forest $\mathcal{F}(N)$ of perfect phylogenies, created by removing all recombination edges. In what follows, we will consider each of these trees after ignoring the edge directions. An important property of these trees in $\mathcal{F}(N)$ is that there is no duplicate mutations at a site in two trees in $\mathcal{F}(N)$. In other words, if a site s labels an edge in tree $T_1 \in \mathcal{F}(N)$, another tree $T_2 \in \mathcal{F}(N)$ can not have a mutation at s . This implies that sequences in T_2 have a uniform value (either all-0 or all-1) at site s . Also note that $\mathcal{F}(N)$ partitions the rows in M , where each partition is a perfect phylogeny and each row in M appears as a label in one of the perfect phylogenies. We call such partitioning of M *perfect* partitioning. It is easy to see that perfect partitioning always exists: a trivial partitioning simply has n partitions, where each partition has a single row. Obviously, there exists a way of partitioning the rows of M such that the *number* of partitions is *minimized*. This motivates the following optimization problem.

The Minimum Perfect Phylogenetic Forest (MPPF) Problem. Given a binary matrix M , find a set of a *minimum* number of perfect phylogenies that derives M s.t. each row is derived by some perfect phylogeny and for any site s , mutations at s occur at most once in at most one tree. We denote the minimum number of perfect phylogenies $F_{min}(M)$.

Note that $F_{min}(M) = 1$ iff M has a perfect phylogeny. That is, there exists a single tree that derives all sequences in M iff M has a perfect phylogeny.

On the other hand, when there is no perfect phylogeny for M , we need more than one tree to derive all the sequences in M . The MPPF problem asks to find the minimum number of trees in the forest. This problem is related to the well-studied maximum parsimony problem (i.e. the Steiner tree problem in phylogeny). In maximum parsimony, we construct a *single* tree (with back or recurrent mutations) which minimizes the number of site mutations. The MPPF problem asks for constructing the minimum number of trees, each of which is a perfect phylogeny, and each site can mutate once in at most one tree.

Now we define the forest bound.

Forest bound. For a binary matrix M , the forest bound is equal to $F_{min}(M) - 1$.

Lemma 1. *The forest bound is a valid lower bound on $Rmin(M)$.*

Proof. Suppose we trim a minARG N by removing all recombination edges in N and we have a forest with $k \geq F_{min}(M)$ trees. Note that N is connected and we need at least one recombination to connect a tree to the rest of trees. So $Rmin(M) \geq k - 1 \geq F_{min}(M) - 1$. The reason that we subtract 1 is because we can start from a tree and this tree does not need a recombination to link it. \square

Now we explain the reason for our interest in the forest bound. Unlike the forest bound, the history bound lacks a static definition. Below we show that the forest bound is higher than the haplotype bound, and hence, the forest bound is the highest lower bound that we know of which has a simple static definition.

Lemma 2. *For a perfect phylogenetic forest \mathcal{F} with n_s Steiner nodes, the number of trees k is equal to $n + n_s - m$.*

Proof. Suppose each tree $T_i \in \mathcal{F}$ contains n_i distinct sequences (nodes) for $i = 1 \dots k$. Here, $\sum_{i=1}^k n_i = n + n_s$, where n_s is the number of Steiner nodes in the ARG N . We know for each tree, there are $n_i - 1$ edges with mutations. Let m_i denote the number of mutations in tree T_i . This means $m_i = n_i - 1$. We require each column to mutate *exactly* once, and it is easy to show that this constraint does not change the forest bound. So we have $m = \sum_{i=1}^k m_i = \sum_{i=1}^k (n_i - 1) = n + n_s - k$ mutations in the forest. So, $k = n + n_s - m$. \square

Lemma 3. *This forest bound is always higher than the haplotype bound, but lower than the history bound.*

Proof. We first show that the forest bound is provably higher than the haplotype lower bound. Suppose a minimum forest has $k = F_{min}(M)$ trees. From Lemma 2, $k = n + n_s - m \geq n - m$. So $k - 1 \geq n - m - 1$, which is the haplotype bound.

Now we show that the history bound is higher than the forest bound. From the algorithm to compute the history bound, it can be seen that the method produces a phylogenetic forest. However, in contrast to the definition of a phylogenetic forest given above, the forest produced by the history bound has additional time-order constraints: the trees in the forest can be time-ordered such that if site s mutates in a tree T_i , the states at s for sequences in earlier trees must be

ancestral states (i.e. not the derived states). But since the forest produced by the history bound is a valid phylogenetic forest, its number of trees in that forest cannot be smaller than the forest bound. \square

We now relate the forest bound to the optimal haplotype bound.

Theorem 1. *The forest bound is higher than the optimal haplotype bound.*

Proof. By Lemma 3 we know that the forest bound applied to any subset of site is higher than the haplotype bound applied to the same subset of sites. In particular, if S^* is the subset of sites of M (called *optimal subset*) that gives the optimal haplotype bound, then the forest bound applied to S^* is higher. Hence to prove the theorem we only need to show that the forest bound applied to all of M cannot increase by restricting to a subset of sites in M .

For a given data M , suppose we have a minimum phylogenetic forest \mathcal{F} for M with $F_{min}(M)$ trees. Now we consider $\mathcal{F}(S)$ when we restrict our attention to S , a subset of sites. To derive $\mathcal{F}(S)$ from \mathcal{F} , we remove all mutation sites in \mathcal{F} that are not in S and cleanup the forest by removing edges with no mutations, and collapsing identical sequences. It is important to note that $\mathcal{F}(S)$ has *at most* $F_{min}(M)$ trees. This is because when we remove sites not in S , we may need to link up two previously disjoint trees (and thus make the number of trees smaller), but we can never *increase* the number of trees. Thus, we know $F_{min}(M(S))$ can not be higher than $F_{min}(M)$. \square

Theorem 1 and Lemma 3 say that the forest bound is higher than the optimal haplotype bound but lower than the history bound. Hence we conclude,

Corollary 1. *The optimal haplotype bound cannot be higher than the history bound 2.*

Experiments show that the forest bound can be strictly higher than the optimal haplotype bound and improve the overall recombination lower bound. For example, consider a simple matrix containing 5 rows and 5 columns: 10001, 00010, 00100, 11011 and 01101. The optimal haplotype bound for this data is 1, while it is not hard to see that a perfect phylogenetic forest contains at least 3 components. Thus, the forest bound for this data is 2.

As mentioned earlier, it is known that the optimal haplotype bound and the history bound are both NP-hard to compute 2. However, if the forest bound could be computed efficiently, we would not need to compute the optimal haplotype bound, but could instead use the forest bound. Unfortunately, the forest bound is also NP-hard to compute.

3.2 The Complexity of the Forest Bound

Theorem 2. *The MPPF problem is NP-hard.*

² Myers 12 asserted (with no proof) that the history bound is higher than the optimal haplotype bound. Here we furnish the proof to this claim.

Proof. The high-level construction of our proof is inspired by Foulds and Graham's NP-completeness proof of Steiner tree in perfect phylogeny problem [4].

As in [4], we reduce from the known NP-complete problem of Exact Cover by 3-sets (X3C) [5]. Recall that the general form of X3C is as the following:

$$\mathcal{S} = \{S_1, S_2, \dots, S_n\}, \text{ where each } |S_i| = 3 \text{ and } S_i \subseteq \{1, 2, \dots, 3m\} = I_{3m}, \text{ for } 1 \leq i \leq n.$$

Does \mathcal{S} contain (non-overlapping) m sets S_{i_1}, \dots, S_{i_m} whose union is I_{3m} ?

High-Level Idea. We construct a binary matrix M for \mathcal{S} (the collection of sets), such that for each set S_i , the set of corresponding sequences in M can be generated on a perfect phylogeny. Thus, if there is a solution for X3C, we have m perfect phylogenies that use up all site mutations, and a collection of isolated sequences (also trivially perfect phylogenies) and the total number of trees is $F_{min}(M)$. If there is no solution for X3C, the number of trees in any perfect phylogenetic forests is more than $F_{min}(M)$. To enforce this property, two sequences corresponding to the same set S_i will have a small Hamming distance. For two sequences corresponding to different sets, their Hamming distance will be large. So, if two far apart sequences are placed into the same tree, there will be too many Steiner sequences needed to connect them, and thus by Lemma [2] and proper manipulation of the construction, we will have more trees than $F_{min}(M)$ in such forest.

WLOG we assume there is no duplicate sets in \mathcal{S} . Given an instance of X3C, we construct a binary matrix M as follows. We let $K = m + 1$. Note that $2K - 3 > m$, when $m > 1$. For each S_i , we construct a set of $3K$ sequences of length $3mK$. We have K sequences corresponding to each of the three elements in S_i . Each of these sequences is composed of $3m$ blocks of K sites. Each block is arranged sequentially in the increasing order for each integer in S_i . The sequences are constructed as follows. Suppose we are constructing the j th sequence ($j \in \{1 \dots K\}$) for an element $p \in S_i$. For block (of number q) $B_{i,p,j,q}$ in this sequence, if the corresponding integer $q \notin S_i$, then block $B_{i,p,j,q}$ contains all 1. If $q \in S_i$ and $q \neq p$, then we set $B_{i,p,j,q}$ to be all 0. If $q \in S_i$ and $q = p$, we set the j th bit in $B_{i,p,j,q}$ to 1 and 0 for all other bits. Note that for a given row associated to a set S_i , all bits corresponding to elements not in this set are 1. Also note that for the K sequences corresponding to an integer $q \in S_i$, the K blocks $B_{i,p,j,q}$ form a diagonal matrix with all 1 on the main diagonal. One example is shown in Table [3] for the simple case when $m = 2$.

The following facts (proof omitted) about M are important.

- P1. There are *no* two identical sequences in M .
- P2. The $3K$ sequences corresponding to a single set S_i have a star-shaped perfect phylogeny, with the center sequence as the only Steiner sequence.
- P3. For two sequences s_1, s_2 coming from the same set S_i , the Hamming distance between s_1, s_2 is 2. For two sequences s_1, s_2 coming from different sets S_i, S_j , the Hamming distance is at least $2K - 2$.

Now we claim that X3C problem has a solution (i.e. union of S_{i_1}, \dots, S_{i_m} is equal to I_{3m}) if and only if there is a phylogenetic forest for M with exactly $3nK - 3mK + m$ perfect phylogenies.

Table 1. Example of the constructed matrix when $m = 2$ (i.e. there are 6 elements in I_{3m}), and thus $K = 3$. The table lists the constructed rows for a set $\{1, 2, 4\}$.

	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	6	6	6
r_1	1	0	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	1
r_2	0	1	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	1
r_3	0	0	1	0	0	0	1	1	1	0	0	0	1	1	1	1	1	1
r_4	0	0	0	1	0	0	1	1	1	0	0	0	1	1	1	1	1	1
r_5	0	0	0	0	1	0	1	1	1	0	0	0	1	1	1	1	1	1
r_6	0	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	1	1
r_7	0	0	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1
r_8	0	0	0	0	0	0	1	1	1	0	1	0	1	1	1	1	1	1
r_9	0	0	0	0	0	0	1	1	1	0	0	1	1	1	1	1	1	1

We first show that given a solution (i.e. S_{i_1}, \dots, S_{i_m}) of X3C, we can build a forest with $3nK - 3mK + m$ trees. From property P2, we construct m perfect phylogenies, each from $3K$ sequences corresponding to each of S_{i_j} . Then, we treat the remaining $3nK - 3mK$ as isolated sequences (trivially perfect phylogenies). So the total number of perfect phylogeny is $3nK - 3mK + m$.

Now we show the other direction: if there is a phylogenetic forest for M that has $3nK - 3mK + m$ perfect phylogenies, then there is a solution for problem X3C. We first argue that no two sequences from different sets S_i, S_j can appear together in a same perfect phylogeny. For contradiction, suppose s_1, s_2 coming from different sets are together in one perfect phylogeny. Consider the path from s_1 to s_2 in the perfect phylogeny. From Property P3, the Hamming distance between s_1 and s_2 is at least $2K - 2$. This means there are at least $2K - 3$ intermediate nodes on that path whose states changes from s_1 to s_2 on these $2K - 2$ sites. It is also easy to see that none of these $2K - 3$ nodes can be part of M , since each sequence in M has either exactly a single 1 or all 1s within a block and intermediate nodes between s_1, s_2 must contain from 2 to $K - 1$ 1s for the block corresponding to the element not shared by S_i, S_j . That is, we know that the phylogenetic forest contains at least $2K - 3$ Steiner nodes. But from Lemma 2, we will have at least $3nk - 3mK + (2K - 3)$ perfect phylogenies, which is larger than $3nK - 3mK + m$ since $2K - 3 > m$. That is a contradiction, and thus each phylogeny can only have sequences derived from the same set S_i .

Now it is easy to see that within the forest there can be at most m non-degenerated trees. This is because each non-degenerated tree contains at least 1 Steiner node (see Property P3). Also note that we can not have fewer than m non-degenerated trees. To prove this, suppose for contradiction, that we have at most $m - 1$ non-degenerate perfect phylogenies. Since each such tree comes from a single set S_i , if there are at most $m - 1$ trees, there are at most $3(m - 1)K$ nodes in the trees. Then there are at least $3nK - 3(m - 1)K = 3nK - 3mK + 3K$ degenerated trees. Since $K = m + 1$, we know we will have more than $3nK - 3mK + m$ (isolated) trees. That is a again a contradiction.

Therefore, we know we will have exactly m non-degenerated trees. Now we need to show that these m non-degenerated trees correspond to a solution for X3C. Note that each of such trees does correspond to a set in \mathcal{S} . What we

need to show is that every element in I_{3m} is covered, and no element is covered more than once. Suppose a tree has a block whose corresponding integer is not covered by other picked sets, then we can easily enlarge the tree by adding the sequences of that block. Now we want to argue that no two sets picked by this phylogenetic forest can overlap. For contradiction, suppose there is an overlap between S_i, S_j when we select all $3K$ sequences corresponding to S_i, S_j . Then for the corresponding two trees, there must be mutations in *both* trees for the overlapped sites. This contradicts the assumption that the set of trees are perfect phylogenies with no duplicate mutations (sites). Therefore, the given phylogenetic forest leads to a valid X3C solution. \square

Corollary 2. *Computing the forest bound is NP-hard.*

A Variant of the MPPF Problem. The MPPF problem requires that if a mutation occurs at a site in one tree, then this site does not mutate in any other tree. Now, suppose we allow a site to mutate in more than one of the perfect phylogenies (but still mutate at most once in any single perfect phylogeny). This problem is NP-complete even when we just want to partition the matrix into two perfect phylogenies. We omit the proof due to the space limit.

4 Practical Computation of the Forest Bound

In this section, we focus on using integer programming to compute the *exact* forest bound for data within certain range.

4.1 Computing the Forest Bound Precisely Using Integer Linear Programming

Consider an input matrix M with n rows and m sites. Our goal is to compute the minimum forest that derives the input sequences. There are 2^m possible sequences (which form a hypercube) that could be part of the minimum forest. Of course, the n input sequences must appear in this forest. From Lemma 2, in order to compute the forest bound we need to *minimize* the number of Steiner nodes. Thus, we create a variable v_i for each sequence s_i in the set of 2^m possible sequences at Steiner nodes, where $v_i = 1$ means sequence s_i appears in the forest. Next, we create a variable $e_{i,j}$ for two sequences s_i, s_j that differ at exactly one column. We create constraints to ensure $e_{i,j} = 1$ *implies* $v_i = 1$ and $v_j = 1$. We define a set E_c as the set of $e_{i,j}$ where s_i, s_j differ exactly at the single site c . The infinite sites mutation model requires that exactly one $e \in E_c$ has value 1.

Optimization goal Minimize $(\sum_{i=1}^{2^m} v_i) - m - 1$
Subject to

$$v_i = 1, \text{ for each row } s_i \in M.$$

$$e_{i,j} \leq v_i, \text{ and } e_{i,j} \leq v_j, \text{ for each edge } (s_i, s_j).$$

$$\sum_{(s_i, s_j) \in E_c} e_{i,j} = 1, \text{ for each site } c$$

Binary Variables

v_i for each sequence s_i with m binary characters

$e_{i,j}$ for each pair of sequences s_i, s_j such that $d(s_i, s_j) = 1$.

The formulation can also be extended easily to handle the situation where there are missing values in the input data, which is important for handling real biological data. To handle missing data in the ILP formulation, for a sequence s_i with missing values, we change the constraint $v_i = 1$ to $\sum_j v_j \geq 1$, for each sequence s_j that matches the values of s_i at all non-missing positions. Our experience shows that the formulation can be solved reasonably fast for data with up to 8 sites (by a powerful ILP package CPLEX).

4.2 Simulations of Data with Missing Data

Now we describe computations of the forest bound and how they compare to the haplotype bound on simulated data. In this simulation study, to make comparison easier, we do not use the composite method [13], which often gives higher lower bound. We show here that the forest bound can be effectively computed for certain range of data with missing values.

We generated 100 datasets with Hudson’s program MS [10] for each parameter setting. We fix the number of sites in these data to 7 or 8. We want to compare the forest bound with the optimal haplotype bound when the data contains various level of missing data. Currently, the only known method computing optimal haplotype bound with missing data can only work with very small data. So instead, we compare with a weaker haplotype bound method (implemented in program HapBound [15]) that can handle missing data but not always give the optimal bound [14] when there is missing data. Missing values are added to the datasets by setting an entry to be missing with a fixed probability P_{mv} .

Table 2. Comparing the forest bound with haplotype bound. We report the percentage of datasets where the forest bound is strictly higher than the haplotype bound.

% Missing value	0%	10%	20%	30%
20 rows, 7 sites	0%	0%	0%	3%
20 rows, 8 sites	0%	1%	0%	0%
30 rows, 7 sites	0%	1%	0%	8%
30 rows, 8 sites	0%	0%	0%	7%

Table 2 shows that the forest bound can outperform HapBound in some cases. On the other hand, the haplotype bound method used by the program HapBound appears to be quite good for the range of data we tested. Our tests show that when the missing value level is low or moderate, the program HapBound performs quite well for the range of data we generated. However, on random data, the forest bound was seen to often give higher bounds than the optimal haplotype bound (results not shown). In fact, for 50 simulated random datasets with

15 rows and 7 sites (and with no missing entries), 10% of the data had a strictly higher forest bound compared to the optimal haplotype bound (and 20% of the data had a strictly higher history bound than the optimal haplotype bound).

Acknowledgments. The research reported here is supported by grants CCF-0515278 and IIS-0513910 from National Science Foundation.

References

1. Bafna, V., Bansal, V.: The number of recombination events in a sample history: conflict graph and lower bounds. *IEEE/ACM Trans. on Computational Biology and Bioinformatics* 1, 78–90 (2004)
2. Bafna, V., Bansal, V.: Inference about Recombination from Haplotype Data: Lower Bounds and Recombination Hotspots. *J. of Comp. Bio.* 13, 501–521 (2006)
3. Bordewich, M., Semple, C.: On the computational complexity of the rooted subtree prune and regraft distance. *Annals of Combinatorics* 8, 409–423 (2004)
4. Foulds, L.R., Graham, R.L.: The Steiner Tree in Phylogeny is NP-complete, *Advances in Applied Math*, v. 3 (1982)
5. Garey, M., Johnson, D.: *Computers and intractability*, Freeman (1979)
6. Griffiths, R.C., Marjoram, P.: Ancestral inference from samples of DNA sequences with recombination. *J. of Comp. Bio.* 3, 479–502 (1996)
7. Gusfield, D.: Efficient algorithms for inferring evolutionary history. *Networks* 21, 19–28 (1991)
8. Gusfield, D., Eddhu, S., Langley, C.: Optimal, efficient reconstruction of phylogenetic networks with constrained recombination. *J. Bioinformatics and Computational Biology* 2, 173–213 (2004)
9. Gusfield, D., Hickerson, D., Eddhu, S.: An Efficiently-Computed Lower Bound on the Number of Recombinations in Phylogenetic Networks: Theory and Empirical Study. *Discrete Applied Math* 155, 806–830 (2007)
10. Hudson, R.: Generating Samples under the Wright-Fisher neutral model of genetic variation. *Bioinformatics* 18(2), 337–338 (2002)
11. Hudson, R., Kaplan, N.: Statistical properties of the number of recombination events in the history of a sample of DNA sequences. *Genetics* 111, 147–164 (1985)
12. Myers, S.: The detection of recombination events using DNA sequence data, PhD dissertation. Dept. of Statistics, University of Oxford, Oxford, England (2003)
13. Myers, S.R., Griffiths, R.C.: Bounds on the minimum number of recombination events in a sample history. *Genetics* 163, 375–394 (2003)
14. Song, Y.S., Ding, Z., Gusfield, D., Langley, C., Wu, Y.: Algorithms to distinguish the role of gene-conversion from single-crossover recombination in the derivations of SNP sequences in populations. In: Apostolico, A., Guerra, C., Istrail, S., Pevzner, P., Waterman, M. (eds.) *RECOMB 2006*. LNCS (LNBI), vol. 3909, Springer, Heidelberg (2006)
15. Song, Y.S., Wu, Y., Gusfield, D.: Efficient computation of close lower and upper bounds on the minimum number of needed recombinations in the evolution of biological sequences. *Bioinformatics* 421, i413–i422 (2005) *Proceedings of ISMB 2005*
16. Wang, L., Zhang, K., Zhang, L.: Perfect Phylogenetic Networks with Recombination. *J. of Comp. Bio.* 8, 69–78 (2001)

Seed-Based Exclusion Method for Non-coding RNA Gene Search

Jean-Eudes Duchesne¹, Mathieu Giraud², and Nadia El-Mabrouk¹

¹ DIRO – Université de Montréal – H3C 3J7 – Canada
{duchesnj,mabrouk}@iro.umontreal.ca

² Bioinfo/Sequoia – LIFL/CNRS, Université de Lille 1 – France
giraud@lifl.fr

Abstract. Given an RNA family characterized by conserved sequences and folding constraints, the problem is to search for all the instances of the RNA family in a genomic database. As seed-based heuristics have been proved very efficient to accelerate the classical homology based search methods such as BLAST, we use a similar idea for RNA structures. We present an exclusion method for RNA search allowing for possible nucleotide insertion, deletion and substitution. It is based on a partition of the RNA stem-loops into consecutive seeds and a preprocessing of the target database. This algorithm can be used to improve time efficiency of current methods, and is guaranteed to find all occurrences that contain at least one exact seed.

1 Introduction

The last 20 years have seen an explosion in the quantity of data available for genomic analysis. Much work has been devoted to speeding up data mining of proteins or gene coding DNA, but these sequences account for only a fraction of the genome. In addition, many non-coding RNA genes (ncRNAs) are known to play key roles in cellular expression, yet few efforts have been made to facilitate their search in large scale databases. Classical homology based search methods like Blast [1] often fail when searching for non-coding genes since the input is stripped from structural information down to its bare sequence. Searching algorithms that permits inputs with structural information should yield better results.

Historically, the first computer scientists to interest themselves with ncRNAs have created tailor made algorithms for specific RNA families such as tRNAs [6,4,9]. Other more general search tools were created to give control of the biological context to the user [3,12,7]. Still these tools lacked the capacity to efficiently parse large genomic databases. Klein and Eddy provided a database specialized search tool [8] for ncRNA including structural information, but is self admittedly slow for large scale databases. More recently, Zhang and Bafna presented a method to efficiently filter databases with a set of strings matching a profile to specific parameters [2]. Their experimentation gave rise to specialized filters for specific RNA families. As such this strategy would require prior

knowledge on the RNA families of interest when generating database, this can become restrictive in some experimental contexts which would benefit from an all-purpose filtering method for ncRNA. Although this can be offset by combining filters with different parameters in an attempt to maximize efficiency and accuracy.

In addition to the capacity of parsing large genomic databases, as sequence and structure constraints are established from a restricted set of an RNA family representatives, any search method should account for a certain flexibility and deviation from the original consensus, allowing for possible mismatches and insertion/deletion (indel) of nucleotides. In particular RNAMotif [12] (one of the most popular and time efficient tool for RNA search), does not explicitly allow for base pair indels. In [5], we have considered a more general representation of folding constraints and developed an approximate matching algorithm allowing for for both mismatches and indels of base pairs. The major drawback of the method was its time inefficiency.

In this paper, we develop a seed-based exclusion method allowing for mismatches and indels, able to speed up existing RNA search methods. Similar heuristics have been proved very efficient to accelerate the classical homology based methods. In particular, PatternHunter [11,10] based on multiple spaced seeds has become one of the most popular method for sequence search at a genomic scale. Recently, Zhang et al. [16] proposed a formalization of the filtering problem and a demonstration that the combination of several filters can improve the search of ncRNAs. Here, we develop a new seed-based heuristic for RNA search, using seeds with distance and folding constraints. It is based on a partition of the RNA stem-loops into consecutive seeds and a preprocessing of the target database storing the occurrences of all possible seeds in a hash table. The search phase then reports, in constant time, the position lists of all seeds of the query stem-loops, and uses extension rules to account for possible errors. The heuristic is guaranteed to find all occurrences containing at least one exact seed.

The rest of the paper is organized as follows. Section 2 presents the basic concepts and definitions, and introduces the general idea of the Sagot-Viari algorithm [15] that will be used in our algorithm's search phase. Section 3 describes our new exclusion method. In Section 4, we study the choice of seeds and anchor elements. Finally, we present our experimental results in Section 5, and show how our method can be used in conjunction with RNAMotif to improve its running time.

2 Preliminary Definitions

2.1 RNA Structures

An RNA *primary structure* is a strand of consecutive nucleotides linked by phosphodiester bonds: Adenine (A), Cytosine (C), Guanine (G) and Thymine (T). When transcribed from DNA to RNA, thymine is substituted into uracil (U). As

such, U and T are considered synonymous for most purposes. We denote Σ_{DNA} the alphabet of nucleotides $\{A, C, G, T\}$.

Considering that an individual nucleotide’s main biological property is to form a structural bond with other nucleotides, primary structure alone ill-defines ncRNAs. An RNA *secondary structure* is represented by a series of base pairings, the most frequent ones being the canonical Watson-Crick A-T and C-G. The secondary structure is organized in a set of nested stems and loops, where a stem is a sequence of paired and unpaired nucleotides, and a loop is a sequence of unpaired nucleotides. A stem followed by a loop is called a *stem-loop* (Figure 1(a)).

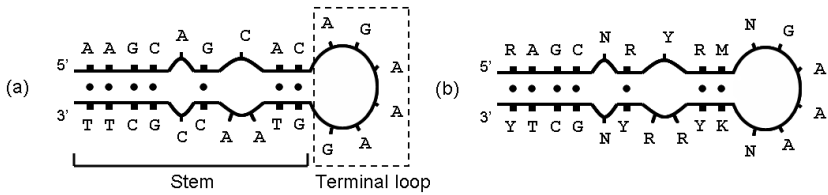


Fig. 1. (a) A stem-loop with canonical base pairings represented as dots; (b) A stem-loop descriptor. (a) is an occurrence of (b).

It is well documented that the functional properties of an RNA molecule is dependent on its final structure obtained by additional foldings over its secondary structure. Our work relies on the hypothesis that there is enough signal in the primary and secondary structure to find the overall molecule with this simplified view.

2.2 Descriptors

Descriptors are user-defined sets of conserved elements of a specific molecule’s primary and secondary structure. They are often obtained from multiple alignments of different instances of the same molecule’s sequence from various species, but how a good descriptor is obtained is beyond the scope of this current work.

In [5], we have introduced a rigorous and very flexible representation of folding constraints in term of “secondary expressions”. In this paper, we focus on a more restrictive descriptor form, though allowing to represent most of the RNA families found in the literature. The considered constraints are:

1. Positions characterized by a possible subset of nucleotides and represented by a degenerate alphabet, the IUPAC code, over all possible substitutions (Table 1). For example, N allows for any nucleotide at the observed position.
2. Correlated constraints due to canonical base pairings. For example, in Figure 1(b), the left-most pairing (R, Y) means that the upper nucleotide can be either A or G, but if it is A (respec. G) then the opposite nucleotide should be T (respec. C).
3. Bounded range of possible lengths for unpaired parts of the structure.

Table 1. The standard IUPAC code defines symbols for sets of nucleotides

A : A	K : G T	B : C G T
C : C	M : A C	D : A G T
G : G	R : A G (purine)	V : A C G
T : T	S : C G	H : A C T
	W : A T	
	Y : C T (pyrimidine)	N : A C G T

2.3 The Sagot-Viari Notations

The Sagot-Viari algorithm [15] is designed to search for all stem-loops in a genomic sequence, allowing for possible mispairings. More precisely, given four parameters s , e , d_{min} and d_{max} , the algorithm finds all possible stem-loops in the genome G characterized by a maximum stem length s , a loop of size d with $d_{min} \leq d \leq d_{max}$, and a maximum number of e mispairings and nucleotide insertion and deletion (indels).

The interesting design feature of their method was to keep separate the two complementary parts of the stems until the final reconstruction step. Another way to look at their method is to consider that they filter a complete genome for sequences that can potentially form a stem-loop structure but differ the actual verification until the sequences have been extended to the full length of the pattern.

We first introduce some basic notations. Given a sequence $u = u_1u_2 \dots u_n$, we denote by $u_{i,j}$ the subsequence $u_{i,j} = u_iu_{i+1} \dots u_j$. The sequence \bar{u} is the complementary inverse of u . For example, if $u = \text{AATGC}$, then $\bar{u} = \text{GCATT}$. Given a sequence u of size k on Σ_{DNA} , we denote by $Oc(u)$ the list of positions of all occurrences of u in the genomic sequence G , eventually within a threshold of error e . The occurrences list of a stem-loop described by u is:

$$S_{(u,\bar{u})} = \{(p, q) \mid p \in Oc(u), q \in Oc(\bar{u}), \text{good}(p, q)\}$$

The predicate $\text{good}(p, q)$ checks the distance ($d_{min} \leq q - p \leq d_{max}$) and the error constraints.

The algorithm proceeds by successive *extensions* and *filtering* steps, starting from sets $Oc(\alpha)$ for each $\alpha \in \Sigma_{\text{DNA}}$. Each set $Oc(u_{i,j})$ could be constructed by extending $Oc(u_{i+1,j})$ and $Oc(u_{i,j-1})$. However, a majority of the positions in $Oc(u)$ can be eliminated before the final filtering. In fact, the algorithm never computes any Oc list beyond the initial step. It considers only *possible occurrences* (of the stem-loop) position lists:

$$POc(u_{i,j}) = \{p \in Oc(u_{i,j}) \mid \exists q \in Oc(\overline{u_{i,j}}), \text{good}(p, q)\}$$

The lists $POc(u_{i+1,j})$ and $POc(u_{i,j-1})$ are extended and merged into one list $POc'(u_{i,j})$. Filtering that list for the distance and error constraints give rise to

the list $POc(u_{i,j})$. At the end, the solution set $S_{(u,\bar{u})}$ is obtained by a (quadratic) filtering between $POc(u)$ and $POc(\bar{u})$.

The POc and POc' lists are represented through stacks, and all the extension and filtering operations are done in linear time relative to the size of the stacks.

3 An Exclusion Method for RNA Search

Given an RNA descriptor D and a genomic sequence (or database) G , the goal is to find the position list S_D of the occurrences of D in G , possibly with an error threshold e . We propose to search the descriptor D starting with a set of n anchor sequences extracted from the descriptor’s stem-loops. A heuristic based on an exclusion method is developed for an efficient search of anchor sequences: each anchor is partitioned into consecutive (and overlapping) seeds of a given size, and a preliminary step consists in building a seed database over the genomic sequence G . In section 4, we discuss the choice of appropriate “constraining” anchors allowing a good speed-up with a convenient sensibility.

A high level sketch of the exclusion method is given below and is schematized in Figure 2. Details are in the following subsections.

1. **Preprocessing phase:** Build a seed database over the genomic sequence G .
2. **Partition phase:** Choose a set of anchor sequences from D (with their relative distance constraints) and a set of seed-shapes, and partition the anchor sequences into consecutive seeds.
3. **Anchor search phase:** Query the database for the seeds, giving lists of occurrences Oc . Then extend occurrences and filter them while checking length, error and folding constraints.
4. **Check phase:** Check whether each RNA candidate verifies the descriptor constraints that were not used as anchors in the search phase.

3.1 The Preprocessing Phase

The genomic database G is first processed to output all elementary motifs of a given size. The preprocessing phase is designed to allow for a constant time access to the position list of all occurrences of elementary motifs, represented by seeds. Rigorous definitions follow.

A *seed of size k* or *k -seed* is a sequence of size k on the alphabet Σ_{DNA} . To allow the possibility of spaced seeds, we define two types of characters: $\#$ and $-$, where $-$ denotes the don’t care character. A *k -seed-shape* is a sequence of k elements from the alphabet $\{\#, -\}$.

Given a set of seed-shapes, the preprocessing phase builds a hash table containing an entry for each set of sequences with the same $\#$ positions. For example, for seed-shape $\#\#-\#$, $AGAC$ et $AGTC$ are stored at the same position. We discuss the choice of appropriate seed-shapes and lengths in Section 4.

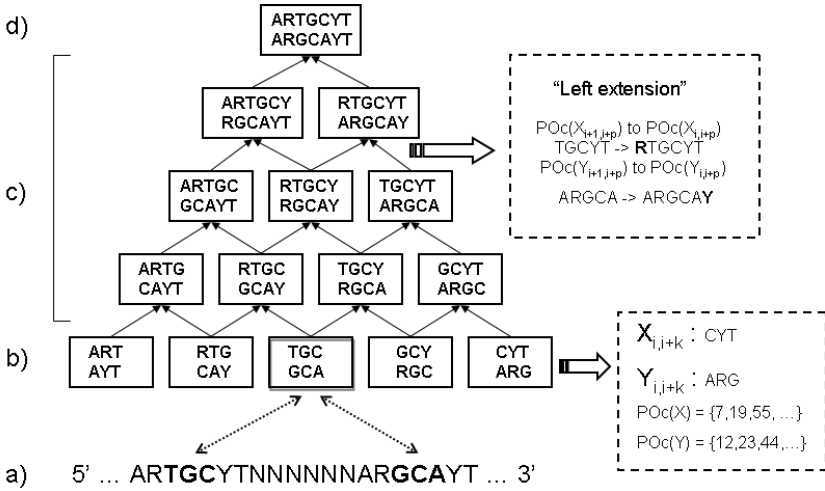


Fig. 2. (a) A specific descriptor sequence with the set of anchors $\mathcal{A} = \{\text{ARTGCGYT}, \text{ARGCAYT}\}$ of common length $m = 7$. The distance constraints are $d_{min}^{1,2} = d_{max}^{1,2} = 6$. Anchors are partitioned into consecutive 3-patterns. The elements in bold represent a single pair of seeds (seed shape ###); (b) The initial step of the search phase is to query the database for all positions of the selected seeds. In the figure, boxes are labeled by their implicit sequences. Their actual data is the lists of positions of these sequences, as illustrated by the rightmost box; (c) Next, the algorithm iterates over a series of extensions and merges to filter the seeds that cannot possibly extend into the desired motif. Each level represents a single iteration. A single box receives incoming extensions from two sources, hence the need for merging sets of positions into a single set. One of these extension is shown in greater detail; (d) After the final iteration, the algorithm returns a list of candidate positions for the full anchor sequences. Each position needs to be validated to confirm the presence or absence of the desired motif at the given position in the genome.

3.2 The Partition Phase

The RNA descriptor is first parsed to extract a given number of *anchors* that are ordered in a priority search list (see section 4). More precisely an anchor \mathcal{A} is a set of sequences $\{\mathcal{A}^1, \dots, \mathcal{A}^l\}$ on the IUPAC alphabet, with a set of distance constraints $\{(d_{min}^{i,j}, d_{max}^{i,j})\}$. Anchor sequences can be related with complementary relations, but that is not mandatory.

A sequence of size k over the IUPAC alphabet is called a k -*pattern*. For a given length k , each anchor sequence \mathcal{A}^i is partitioned into its consecutive k -patterns $\mathcal{A}_{1,k}^i, \mathcal{A}_{2,k+1}^i, \dots, \mathcal{A}_{m-k+1,m}^i$. For a given k -seed-shape sh , we then report the set of seeds corresponding to each k -pattern. A formal definition follows.

Definition 1. Let $u = u_1 \dots u_k$ be a k -pattern and $sh = sh_1 \dots sh_k$ be a k -seed-shape. We say that a seed $s = s_1 \dots s_k$ is a representative of u with respect to sh iff, for any i such that $sh_i = \#$, $s_i \in u_i$.

Given a k -seed-shape sh and a k -pattern u , we denote by $L(u)$ the list of seed representative of u with respect to sh . For example, if $u = ARYC$ and $sh = \#\#\#$ we have $L(u) = \{AAAC, AACC, AAGC, AATC, AGAC, AGCC, AGGC, AGTC\}$. We also denote by $L(\mathcal{A}^i)$ the list of representative of all k -patterns of \mathcal{A}^i . The partition phase reports the lists $L(u)$ of each k -pattern of each anchor sequence \mathcal{A}^i .

A final definition is required for the following section. Given a genomic database G and a k -pattern u , the list of all occurrence positions of $L(u)$ in G is denoted by $Oc(u)$. For example, if $G = TAGACTAAAC$ and u is the k -pattern introduced above, then $Oc(u) = \{2, 7\}$.

3.3 The Anchor Search Phase

For clarity of presentation, we describe the search phase for an anchor with two anchor sequences of the same length, and a unique seed-shape of size k . Generalization to anchors of different lengths only requires a final step to extend the longest anchor sequence. Generalization to anchors with more than two sequences requires to consider one POc and POc' list per sequence. Anchors with a single sequence are usually inefficient to consider during the search phase of an RNA descriptor. Generalization to multiple seed-shapes is straightforward.

Let $\mathcal{A} = \{X, Y\}$ be the considered anchor, with the distance constraint (d_{min}, d_{max}) , and m be the common length of X and Y . Let k be the size of the considered seed-shape, and the consecutive k -patterns of each anchor sequence be $X_{1,k} \cdots X_{m-k+1,m}$ (respec. $Y_{1,k} \cdots Y_{m-k+1,m}$).

The initialization step consists in computing $m - k + 1$ pairs of lists $(Oc(X_{i,i+k-1}), Oc(Y_{i,i+k-1}))$ with respect to the genomic sequence G . Following the partition phase, each seed is an entry in the hash table and accessed in constant time. Following the Sagot-Viari methodology (Section 2.3), the two lists are then traversed and filtered with respect to the distance constraints (d_{min}, d_{max}) . The list's elements are of the form (pos, num_errors) , where pos represents a position in the genome and num_errors is the minimum number of errors between the G subsequence at position p and the considered k -pattern with respect to the seed-shape (errors are computed on the # positions of the seed-shape).

The following $m - k$ steps extend the consecutive k -seed surviving lists to $k + 1$ -seeds, then $k + 2$ -seeds, until the m -seeds surviving lists representing the complete anchor. As allowed seed lengths vary from k to m , we will number the following steps from k to m .

Step p, for $k \leq p < m$:

For each i , $1 \leq i < m - p + 1$ do:

1. **Extend left** $POc(X_{i+1,i+p})$ to $POc(X_{i,i+p})$ and respectively $POc(Y_{i+1,i+p})$ to $POc(Y_{i,i+p})$ iff $1 \leq i$. To do so we use the Sagot-Viari rules of model construction (extension by the character X_i or Y_i) with the exception that elements of both $POc(X_{i,i+p})$ and $POc(Y_{i,i+p})$ need to satisfy one condition out of the match, mismatch, insertion and deletion. This is because we allow

for errors in both X and Y with respect to the descriptor while the original Sagot-Viari algorithm did not have that restriction.

2. **Extend right** $POc(X_{i,i+p-1})$ to $POc(X_{i,i+p})$ and respectively $POc(Y_{i,i+p-1})$ to $POc(Y_{i,i+p})$ iff $i+p \leq m$. This extension mirrors the previous step but uses equivalent symmetric extensions to add characters X_{i+p} and Y_{i+p} respectively.
3. **Merge** the two resulting lists into a new pair of lists $POc'(X_{i,i+p})$ and $POc'(Y_{i,i+p})$. If the resulting lists contain consecutive elements representing the same position but with different numbers of errors, we keep a single copy with the minimum number of errors.
4. **Filter** $POc'(X_{i,i+p})$ and $POc'(Y_{i,i+p})$ with respect to the distance, folding and error bound constraints. The resulting lists are $POc(X_{i,i+p})$ and $POc(Y_{i,i+p})$.

In contrast with the original Sagot-Viari algorithm, errors should be allowed for both anchor sequences. The filtering step should then account, not only for the distance constraint, but also for the combined error constraints. Moreover if X and Y are two strands of a given stem, then folding constraints must be checked.

At the end of the search phase, the two remaining lists $POc(X) = POc(X_{1,m})$ and $POc(Y) = POc(Y_{1,m})$ contain all possible occurrences of both anchor sequences. The last step is then to return all occurrence pairs $S_{X,Y}$ respecting the distance, error and folding constraints.

3.4 The Check Phase

The rest of the descriptor D should finally be validated against the positions of the anchor \mathcal{A} . For this purpose any existing RNA search method can be used, such as BioSmatch [5] or RNAMotif if indels are not allowed.

4 Choosing the Anchor Sequences and Seed Shapes

The first idea is to choose the most constraining anchor sequences (those that are likely to give rise to the minimum number of occurrences in the database), that is those with the lowest p -value. Statistical work on structured motifs of form $Xx(\ell, \ell+\delta)Y$, where X and Y are correlated by secondary structure constraints, have been done in [14]. The difficulty arise from the overlapping structure of the patterns. The p -value can be computed by brute enumeration or by sampling.

However, the most constraining anchors are not necessarily the easiest to parse. Indeed, degenerated symbols (representing sets with more than one nucleotide) can give rise to a large list of seed representative in the partition phase (see section 3.2). More precisely, anchor sequences of the same size and with the same occurrence probability may give rise to different lists of seeds representatives depending on the distribution of their degenerated positions. For example, in Table 2, though both anchor sequences ARTGCT and ACTNCAT have the same occurrence probability of $1/4^6$ under the Bernoulli model, the second

Table 2. Size of the lists involved in each stage of the extension phases, for an exact search with the seed-shape ### on a 10M test database (from E. coli. and B. subtilis)

Anchor \mathcal{A}	Number of seeds in $L(\mathcal{A})$	seed occurrences in the database	seeds remaining after extensions			
			step 3	step 4	step 5	step 6
ACTGCAT	5	856408	17846	871	42	4
ARTGCTT	8	1427873	35303	2380	206	10
ACTNCAT	14	2130755	60553	3599	167	6

sequence gives rise to a larger list of seed representative leading to a much larger list of occurrences in the database. As the first extension phase of our algorithm is the most time-consuming phase, a good estimation of the total time comes from the number of seed representative.

This is further illustrated in Figure 3 where all possible anchors represented by pairs of 5-patterns from the 5S RNA helix III (see Figure 3) are searched in a database of bacterial genomes. Not surprisingly, the anchors with the fewest seeds are significantly faster. Therefore, among the most constraining anchor sequences (those with the lowest p -value), we choose those that give rise to the shortest list of seed representative $L(\mathcal{A})$.

Finally, Table 3 shows the sensibility and the speed obtained with different seed-shapes. It appears that longer shapes lead to a smaller execution time, but at the cost of a lower sensibility: some sequences are missed. Here the best compromise is to use the spaced seed-shape #-#-#: the parsing time is more than 40% smaller than the time needed by RNAMotif and the sensibility remains at 99%.

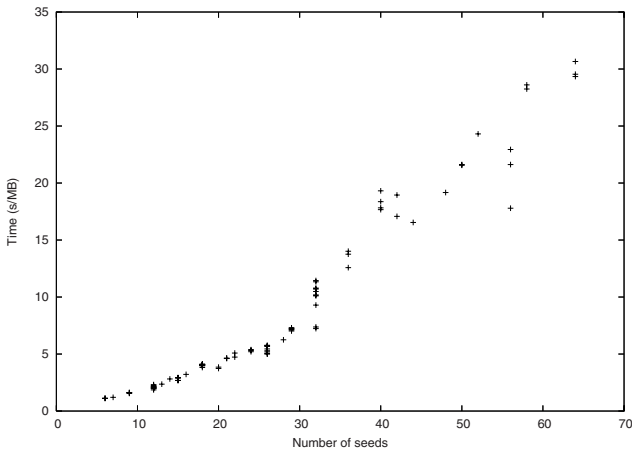
**Fig. 3.** Relation between the speed of the exclusion method and the number of seed representative of the anchor. We tested each possible anchor represented by pairs of 5-patterns from the 5S RNA helix III, with the seed-shape ####. The horizontal axis gives the number of seeds corresponding to each anchor pair, and the vertical axis the time taken for the search on a 130 MB bacterial database.

Table 3. Speed and sensitivity of the Exclusion method. The descriptor is an helix with 6-base stems and a loop $x(10, 50)$, searched with 1 error. It occurs 2110 times in on a 10M test database (sequences from *E. coli.* and *B. subtilis*). Sensibilities of our method are lowered by 1% due to an additional heuristic in stack transversal during the search phase. The time ratios are against the time for RNAMotif.

	RNAMotif	Exclusion			
	v. 3.0.4	###	##-#	####	##-##
sensibility	100%	97%	99%	68%	67%
preprocessing time (ms)	–	684	927	1068	1262
parsing time (ms)	2709	4144 (153%)	1512 (56 %)	352 (13%)	135 (5%)
total time (ms)	2709	4828 (178%)	2439 (90 %)	1420 (51%)	1407 (45%)

5 Testing on RNA Stem-Loops

Here, we tested our new method for both quality and speed, by comparing with RNAMotif. Indeed though RNAMotif has the limitation of ignoring possible nucleotide insertions and deletions, it is an exact method thus giving a good benchmark to test our heuristic’s sensitivity. Moreover it is the fastest RNA search method developed so far.

We considered three RNA families: 5S rRNAs and RNase P RNAs as in [5] as well as group II introns. In each case, the most conserved region was considered, namely helix III for 5S rRNAs, P4 region for RNase P RNAs and domain V for group II introns. The tests are performed exclusively on stem-loop signatures because of technical limitations in our current implementation. This will be extended to full structures of ncRNAs in the near future. We used a database containing 25 randomly selected microbial genomes from GenBank representing a total of more than 75 million base pairs. All tests were performed on an intel Pentium 4 PC with a 2800MHz processor, 2 GB of memory and running Fedora Core 2. The stem-loop signatures were chosen to represent various testing conditions and parameters (stem and loop size).

We considered the seed-shape ####, and two anchor sequences of size 5. Since computational time rises exponentially with the number of seed representative generated by anchor pairs, a cutoff value was selected to avoid anchors likely

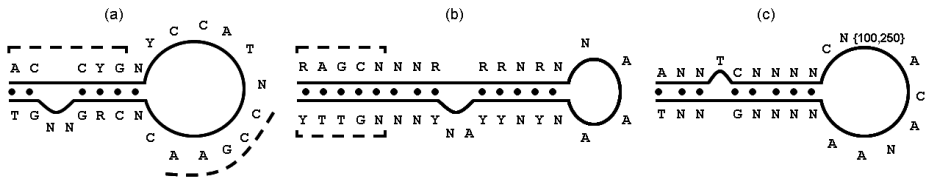


Fig. 4. The stem-loop signature used for (a) 5S rRNAs helix III, (b) group II intron domain V and (c) RNase P RNA P4 region. The dotted lines represent the specific anchor sequences selected for searching.

to generate large initial sets of occurrences from the database. It was set to 16 seed representative, based on experimental results (Figure 3). This cutoff could be raised or lowered on execution to influence speed (lower cutoff) or sensitivity (higher cutoff), but 16 has been a good compromise thus far. Both the 5s rRNA and the Intron group II consensus had several anchor pairs of size 5 falling under that cutoff. The chosen anchors are illustrated in Figure 4. The anchor sequences used for the Intron group II consensus are related by folding constraints, where as those used for the 5s rRNA are only related by distance constraints. Unfortunately, no suitable pair of anchor sequences was found for the RNase P. This is more likely a limitation of the current implementation rather than the method since the whole consensus structure could have yielded for adequate anchors which were not present in the P4 region. However, this illustrates that certain ncRNA might not have sufficient conserved regions to select adequate anchor sequences.

We tested the ability of our exclusion method to speed up RNAMotif, in other words, the check phase was completed by using RNAMotif. Running times (Table 4, third column) are clearly improved for both 5S rRNAs and group II intron domain V. This clearly shows that the exclusion method can shave off significant amount of computational time for ncRNA searching methods. Finally, we used our method not as a filtering strategy but rather as a stand alone algorithm. In other words the exclusion method was used over the full ncRNA stem-loop signatures (Table 4, last column). As expected from the many degenerated positions in the structure consensus, the execution times are fairly slow for this setup. Here we can clearly see the relationship between conservation and execution times with the most conserved consensus structure (5S rRNAs) being significantly faster to search than the other candidates.

Table 4. Computation times obtained by running our exclusion algorithm and RNAMotif v3.0.0 on the stem-loop signature considered for each structured motif family on a database of bacterial genomes. For each method we show the times obtained when the full helix is searched and when only the most conserved subsets are searched. No suitable anchor subset was available for the RNase P RNA P4 region.

	RNAMotif	RNAMotif with Exclusion method	Exclusion on full helix
5S RNA, helix III	2.6 s/Mb	1.1 s/Mb	12.7 s/Mb
Intron group II, domain V	3.3 s/Mb	2.7 s/Mb	31.0 s/Mb
RNase P RNA P4 region	3.1 s/Mb	–	59.1 s/Mb

The database contained 71 annotated sequences of the 5S rRNA and 17 sequences of the group II intron. Of these 89 annotated ncRNA genes, only 2 weren't found by RNAMotif, both of the 5S rRNA variety. The exact same results were found by the exclusion method in combination with RNAMotif. In the case of the RNase P RNA, although we couldn't find a suitable pair of anchor sequences, using the exclusion method as a stand alone algorithm did provide

the same predictions as RNAMotif, where 36 of 39 annotated sequences were found. In other words, no loss in sensitivity was observed over the tested data when compared to an exhaustive method like RNAMotif.

6 Conclusion

We have developed an exclusion method allowing for nucleotide mismatches and indels, that can be used in combination with other existing RNA search methods to speed up the search. We have shown that given sub-motifs with small degeneracy values, a hashing method built on the preprocessing of the target database can significantly improve search times. The idea is to select in the descriptor anchors which yield the least computation. That being the case, it's not given that any descriptor contains enough consecutive conservations to permit sublinear filtering. By using distance constraints we can significantly reduce the number of needed consecutive conserved positions by introducing gaps between pairs of anchors.

Furthermore, we have shown that restricting these features to the helical structures alone is not an efficient method to filter a database. This result concurs with previous literature on the subject of finding signals in secondary structure alone [13]. Generalizing the problem to seeds with distance constraints without considering the secondary structures yields the best results as it takes into account signal in both secondary and primary sequence.

This filtering method is still in its early stage as we can explore many other features. It is evident from our current results that there is a bias for selecting small elements and using the largest possible seed to gain the greatest speed increase. In [16], Zhang et al. present a more robust way to select target anchors from a pattern by creative use of the pigeonhole principle. In this paper we address the same sensitivity issue through the use of "don't care" characters which gives added flexibility in choosing the anchor sequences for seeding the search. We have not yet determined if both approaches are compatible and can be defined into a single model. In any case, we plan to incorporate Zhang's filter definition into our future work to facilitate the comparison and/or addition of our parameters. Furthermore, we plan to generalize the method to an arbitrary number of anchors separated by constraint distances. This could be a viable avenue to limit the number of initial candidates to process and further lower computational times.

References

1. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. *Journal of Molecular Biology* 215, 403–410 (1990)
2. Bafna, V., Zhang, S.: FastR: Fast database search tool for non-coding RNA. In: *Proceedings of IEEE Computational Systems Bioinformatics (CSB) Conference*, pp. 52-61 (2004)

3. Eddy, S.R.: RNABOB: a program to search for RNA secondary structure motifs in sequence databases (1992) <http://bioweb.pasteur.fr/docs/man/man/rnabob.1.html#toc1>
4. El-Mabrouk, N., Lisacek, F.: Very fast identification of RNA motifs in genomic DNA. Application to tRNA search in the yeast genome. *Journal of Molecular Biology* 264, 46–55 (1996)
5. El-Mabrouk, N., Raffinot, M., Duchesne, J.E., Lajoie, M., Luc, N.: Approximate matching of structured motifs in DNA sequences. *J. Bioinformatics and Computational Biology* 3(2), 317–342 (2005)
6. Fichant, G.A., Burks, C.: Identifying potential tRNA genes in genomic DNA sequences. *Journal of Molecular Biology* 220, 659–671 (1991)
7. Gautheret, D., Major, F., Cedergren, R.: Pattern searching/alignment with RNA primary and secondary structures. *Comput. Appl. Biosci.* 6(4), 325–331 (1990)
8. Klein, R., Eddy, S.: RSEARCH: Finding homologs of single structured RNA sequences (2003)
9. Laslett, D., Canback, B.: ARAGORN, a program to detect tRNA genes and tmRNA genes in nucleotide sequences. *Nucleic Acids Research* 32, 11–16 (2004)
10. Li, M., Ma, B., Kisman, D., Tromp, J.: PatternHunter II: Highly Sensitive and Fast Homology Search. *Journal of Bioinformatics and Computational Biology* 2(3), 417–439 (2004) Early version in GIW 2003
11. Ma, B., Tromp, J., Li, M.: PatternHunter: faster and more sensitive homology search. *Bioinformatics* 18(3), 440–445 (2002)
12. Macke, T., Ecker, D., Gutell, R., Gautheret, D., Case, D.A., Sampath, R.: RNAmotif – a new RNA secondary structure definition and discovery algorithm. *Nucleic Acids Research* 29, 4724–4735 (2001)
13. Rivas, E., Eddy, S.R.: Secondary Structure Alone is Generally Not Statistically Significant for the Detection of Noncoding RNAs. *Bioinformatics* 16(7), 583–605 (2000)
14. Robin, S., Daudin, J.-J., Richard, H., Sagot, M.-F., Schbath, S.: Occurrence probability of structured motifs in random sequences. *J. Comp. Biol.* 9, 761–773 (2002)
15. Sagot, M.F., Viari, A.: Flexible identification of structural objects in nucleic acid sequences: palindromes, mirror repeats, pseudoknots and triple helices. In: Hein, J., Apostolico, A. (eds.) *Combinatorial Pattern Matching*. LNCS, vol. 1264, pp. 224–246. Springer, Heidelberg (1997)
16. Zhang, S., Borovok, I., Aharonovitz, Y., Sharan, R., Bafna, V.: A sequence-based filtering method for ncRNA identification and its application to searching for riboswitch elements. *Bioinformatics* 22(14), e557–e565 (2006)

A New Quartet Approach for Reconstructing Phylogenetic Trees: Quartet Joining Method

Lei Xin, Bin Ma, and Kaizhong Zhang

Computer Science Department, University of Western Ontario,
London N6A 5B7, Canada

lxin3@uwo.ca, bma@csd.uwo.ca, kzhang@csd.uwo.ca

Abstract. In this paper we introduce a new quartet-based method for phylogenetic inference. This method concentrates on reconstructing reliable phylogenetic trees while tolerating as many quartet errors as possible. This is achieved by carefully selecting two possible neighbor leaves to merge and assigning weights intelligently to the quartets that contain newly merged leaves. Theoretically we prove that this method will always reconstruct the correct tree when a completely consistent quartet set is given. Intensive computer simulations show that our approach outperforms widely used quartet-based program TREE-PUZZLE in most of cases. Under the circumstance of low quartet accuracy, our method still can outperform distance-based method such as Neighbor-joining. Experiments on the real data set also shows the potential of this method. We also propose a simple technique to improve the quality of quartet set. Using this technique we can improve the results of our method.

1 Introduction

With the accumulation of phylogenetic data in recent years, the computational biology community has shown great interest in the reconstruction of large evolutionary trees from smaller sub-trees [5, 9, 10, 11, 12]. The quartet-based method may be the simplest and most natural approach for this kind of problem. This approach usually takes two major steps to complete the reconstruction. First, a set of four-leaf subtrees i.e. quartets are built for every possible four sequences in a DNA or protein sequence set using other phylogeny methods such as maximum likelihood (ML) [2] or neighbor-joining (NJ) [8]. Then a combinatorial technique is applied to reconstruct the entire evolutionary tree according to the topology relations between the quartets built in the first step. It is well known that with the currently existing methods it is hard to build accurate quartet set [1]. So the efficiency of a quartet-based method really depends on how successful it is in tolerating quartet errors. The advantage of this approach is that it can be quite flexible. It can use the distance matrix if in the first step NJ is used. It can also be totally independent of the distance matrix if ML or some other methods are used.

In the last ten years, many efforts have been made to develop efficient quartet-based algorithms. A prominent approach is known as quartet puzzling (QP) [9].

The corresponding program package is called TREE-PUZZLE which is widely used in practice. Recently even a parallelized version was developed for this package [11]. The main dispute around quartet puzzling is that it is outperformed by faster neighbor-joining method in computer simulations. One attempt to improve quartet puzzling is weighted optimization (WO). Although WO is better than QP, it is still outperformed by neighbor-joining in most of cases. The authors stated in their paper [5]: “Despite the fact that there were about 90% of the correct quartets in Q_{max} , we observed that the correct tree was not well inferred in comparison with other inference methods (including ML and NJ)”. A recent paper takes another approach to solve the problem of quartet errors [12]. This approach builds a series of evolutionary trees rather than a single final tree. Their results show that with multiple trees this approach can ensure high accuracy in computer simulations. However, we noticed if only one final tree was allowed, their method was still outperformed by neighbor-joining except on one type of tree model.

In this article, we concentrate on the second step of reconstructing entire phylogenetic tree from the quartet set. We introduce a new approach based on quartet: quartet joining (QJ). This method achieves error toleration by carefully selecting two possible neighbor leaves to merge and assigning weights intelligently to the quartets that contain newly merged leaves. Theoretically we can prove that this method will always reconstruct correct tree when a completely consistent quartet set is given. Intensive computer simulations also show that our approach outperforms TREE-PUZZLE package in most of cases. Under the circumstance of low quartet accuracy, our method can outperform distance-based method such as Neighbor-joining. The potential of this algorithm is also demonstrated by the experiments on the real data set. This algorithm executes in $O(n^4)$ time which is much faster than QP’s $O(Mn^4)$ where M is usually greater than 1000. n is the number of sequences in the data set.

This article is organized as follows: in Section 2, we introduce some necessary notations and definitions. In Section 3, we describe quartet joining method and prove its important property. In Section 4, computer simulation results and experiment results on real data set are given. In Section 5, a simple technique is introduced to improve the results from section 4. Conclusion and future work are discussed in Section 6.

2 Notations and Definitions

In the field of molecular phylogeny, DNA or protein sequences are represented by leaf nodes on the evolutionary tree. A quartet is a set of four leaf nodes which is associated with seven possible pathway structures: three of them are fully resolved unrooted trees, three are partially resolved trees which we can not distinguish between two of three fully resolved tree, the last one is a fully unresolved tree. A figure of three fully resolved trees and one fully unresolved tree of leaf nodes $\{a, b, c, d\}$ are shown in figure 1. We will use $\{ab|cd\}$, $\{ad|cb\}$, $\{ac|bd\}$, $\{abcd\}$ to represent these four different structures. Clearly a tree with n leaf

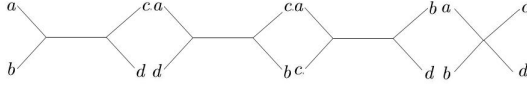


Fig. 1. Four different quartet structures

nodes will have C_n^4 quartets. This quartet set is denoted by Q_T . We say a quartet is *consistent* with an evolutionary tree T if it belongs to Q_T . A quartet set Q is *completely consistent* with evolutionary tree T if $Q = Q_T$.

Two leaf nodes are called *neighbors* if they are connected to the same internal node of the evolutionary tree. This is an important notation and will be mentioned frequently in this paper. When two leaf nodes are merged together to form a new node of the tree, this new node is called *supernode*. Any of four nodes of a quartet could be supernode. If all four nodes of a quartet are single leaf nodes, this quartet is called *single-node quartet* otherwise it is called *supernode quartet*. In this paper we will use capital letters to represent supernodes and small letters to represent single leaf nodes.

3 Quartet-Joining Algorithm

The quartet-joining algorithm follows the bottom-up scheme to build an unrooted evolutionary tree. It starts from n leaf nodes. At every step it merges two possible neighbors according to the quartet set and form one new leaf node until three nodes are left. At last, it joins these three nodes together to get a fully resolved evolutionary tree. The input and output of quartet-joining algorithm are listed below:

Input: A set of quartets.

Output: A fully resolved tree with sequences on leaf nodes.

The key to this kind of algorithms is how to decide which two nodes are most likely to be neighbors on the true evolutionary tree. We notice that for a given tree of n leaf nodes, if any two leaf nodes i, j are neighbors on the tree then the number of the quartets of the form $\{ij|kl\}$ is C_{n-2}^2 . If any two leaf nodes are not neighbors then the number of the quartets of the form $\{ij|kl\}$ must be less than C_{n-2}^2 . In other words, the quartets number of a pair of leaf nodes which are neighbors is maximal among any pair of leaf nodes. This fact has been used by several other methods to deduce neighbors. In this article, rather than apply the fact directly, we design a special mechanism to fully mine the information contained in the quartet set. Here we define *support* to be the weight that supports two leaf nodes to be neighbors. We use *confidence* to represent the weight of quartets including single-node quartets and supernode quartets. Now we need to choose support and confidence wisely such that the algorithm would not have bias on any type of trees. An intuition is that the confidence of supernode quartet can be defined as the percentage of consistent single node quartets it contains. The support will then be the sum of confidence of all the quartets that support

two nodes to be pairs. However experiments show this definition has bias on certain type of trees. So here we use a different approach. In the quartet-joining algorithm, the confidence of the supernode quartet $\{XY|UV\}$ is defined as:

$$C(X, Y; U, V) = \sum_{x \in X, y \in Y, u \in U, v \in V} w(x, y; u, v) \quad (1)$$

where X, Y, U, V are supernodes and we use $|X|$ to denote the number of single nodes which are contained in the supernode X . $w(x, y; u, v)$ is the weight of input quartets.

For a supernode quartet $\{XY|UV\}$, a single-node quartet induced from $\{XY|UV\}$ is the quartet whose four single nodes are taken from supernodes X, Y, U, V separately. So the number of single-node quartets that can be induced from $\{XY|UV\}$ is $|X||Y||U||V|$. When every supernode only contains one single node, $C(X, Y; U, V)$ degenerates to $w(x, y; u, v)$. The support of a supernode pair (X, Y) is defined as:

$$supp(X, Y) = \frac{\overline{supp}(X, Y)}{T(X, Y)} \quad (2)$$

where

$$\overline{supp}(X, Y) = \sum_{U \neq V, U, V \neq X, Y} C(X, Y; U, V) \quad (3)$$

$$T(X, Y) = |X||Y| \sum_{U \neq V, U, V \neq X, Y} |U||V| \quad (4)$$

From the formula above, we can see the support of a pair (X, Y) is basically the averaged weights of the quartets that support (X, Y) to be neighbors. The advantage of this definition is that it balances the weights contributed by supernode quartets and single node quartets. Thus there will be no information loss during the process of merging leaf nodes. All the information from quartet set is utilized. Therefor QJ method can reconstruct the evolutionary tree more accurately than other quartet-based methods. Later on we will also prove that this definition of support will ensure reconstructing the true evolutionary tree when a completely consistent quartet set is given.

If at every step we calculate $C(X, Y; U, V)$ and $supp(X, Y)$ directly, it results in an algorithm of $O(n^5)$ time complexity. So we will use the equations below to update the values of $C(X, Y; U, V)$ and $supp(X, Y)$ instead of recalculating all the values of the two matrices at every step. This will give us an algorithm of $O(n^4)$ time complexity.

Let $C_n(X, Y; U, V)$ and $supp_n(X, Y)$ denote the matrices at step n . A_1, A_2 are the supernodes to be merged at step $n - 1$ and A_1, A_2 will be replaced by a new node A . Then we have the updating formulae:

$$C_n(A, B; C, D) = C_{n-1}(A_1, B; C, D) + C_{n-1}(A_2, B; C, D) \quad (5)$$

For the number of single nodes contained in supernode A , we have:

$$|A| = |A_1| + |A_2| \quad (6)$$

Let S_n denote the remaining nodes set at step n . For the support of the pairs that contain the new node, we have:

$$\overline{supp}_n(A, B) = \sum_{C \neq D, C, D \in S_n} C_n(A, B; C, D) \quad (7)$$

For other nodes, we have:

$$\overline{supp}_n(C, D) = \overline{supp}_{n-1}(C, D) - \sum_{V \in S_{n-1}} C_{n-1}(C, D; A_1, V) \quad (8)$$

$$- \sum_{V \in S_{n-1}} C_{n-1}(C, D; A_2, V) + C_{n-1}(C, D; A_1, A_2) \quad (9)$$

$$+ \sum_{V \in S_n} C_n(C, D; A, V) \quad (10)$$

$$= \overline{supp}_{n-1}(C, D) - C_{n-1}(C, D; A_1, A_2) \quad (11)$$

The formula above comes from fact that all the confidences don't change except confidences concerning A_1, A_2 and A . We will delete nodes A_1 or A_2 and insert node A . So for the new support matrix, we just subtract the influence of A_1 and A_2 then add back the influence of A . We also add one $C_{n-1}(C, D; A_1, A_2)$ just because we subtract it twice before. Then we use equation (5), the final updating formula can be simplified as (11). Also we need to update $T(X, Y)$. For the new nodes we just recalculate using equation (4). For other nodes:

$$T_n(C, D) = T_{n-1}(C, D) - |A_1||A_2||C||D| \quad (12)$$

The quartet-joining algorithm are formally described below:

1. Use $w(x, y, u, v)$ to initialize confidence matrix $C(X, Y; U, V)$.
2. Compute support matrix $supp(X, Y)$ from confidence matrix using equation (2) (3) (4).
3. Find the pair (i, j) which has the maximal value in the support matrix
4. Connect nodes i, j to an internal node then replace them with a supernode k : delete i, j from the set of organisms and insert k .
5. Update confidence matrix, support matrix and $T(X, Y)$ with the updating equations above.
6. Repeat 3 to 5 until only three nodes remain.
7. Connect the three remaining nodes to an internal node. Output the final tree.

We have the following important property for quartet-joining algorithm:

Theorem 1. *If the input quartet set is completely consistent with evolutionary tree T , the quartet-joining algorithm will reconstruct the exact evolutionary tree T .*

To prove this theorem we need to prove a lemma first. Let $T_0 = T$. T_n is a subtree of T with some nodes merged and removed at step n . A figure of T_n is shown in figure 2. If two nodes A, B of T_{n-1} are merged at step $n - 1$, T_n will be a subtree of T_{n-1} with A, B removed. Here we point out that although A, B are supernodes in T , they are treated as single nodes in T_{n-1} .

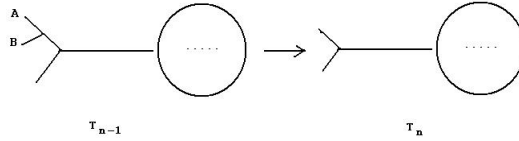


Fig. 2. Definition of T_n tree

Lemma 1. *If the input quartet set is completely consistent with evolutionary tree T , in the quartet-joining algorithm, $C_n(A, B; C, D)$ equals $|A||B||C||D|$ if and only if quartet topology $\{AB|CD\}$ is consistent with tree T_n . Otherwise $C_n(A, B; C, D) = 0$.*

We use induction to prove this lemma.

Proof. At step 0, every supernode only contains 1 single node. Because the input quartet set is completely consistent with T , it is clear that $C_0(A, B; C, D) = 1 = |A||B||C||D|$ when $\{AB|CD\}$ is consistent with tree T_0 . $C_0(A, B; C, D) = 0$ when $\{AB|CD\}$ is not consistent with tree T_0 .

Assume lemma holds at step $n - 1$. Assume A_1, A_2 are the neighbors of T_{n-1} to be merged at step $n - 1$. Because A_1, A_2 are neighbors, both $\{A_1B|CD\}$ and $\{A_2B|CD\}$ will be consistent with T_{n-1} or both will be inconsistent with T_{n-1} at the same time. We assume lemma holds at step $n - 1$, so $C_{n-1}(A_1, B; C, D) = C_{n-1}(A_2, B; C, D) = |A||B||C||D|$ or $C_{n-1}(A_1, B; C, D) = C_{n-1}(A_2, B; C, D) = 0$. When $\{A_1B|CD\}$ and $\{A_2B|CD\}$ are consistent with T_{n-1} , from the definition of T_n , we can see at this time $\{AB|CD\}$ is consistent with T_n . From equation (5), we have $C_n(A, B; C, D) = |A_1||B||C||D| + |A_2||B||C||D|$. Take a look of equation (6), we will know $C_n(A, B; C, D) = |A||B||C||D|$. When $\{A_1B|CD\}$ and $\{A_2B|CD\}$ are inconsistent with T_{n-1} , $\{AB|CD\}$ is inconsistent with T_n and $C_{n-1}(A_1, B; C, D) = C_{n-1}(A_2, B; C, D) = 0$. Surely at this time $C_n(A, B; C, D) = 0$. For other nodes, confidence concerning them don't change, lemma also holds. \square

Now we prove Theorem 1.

Proof. From Lemma 1 and definition of $T_n(C, D)$ in equation (12), we know $supp_n(A, B) \leq 1$ and $supp_n(A, B) = 1$ if and only if (A, B) are neighbors in tree T_n . This means that the maximal value in the support matrix will give us a pair of neighbors. Notice the fact that for a bottom up tree-building scheme, if at every step, a pair of neighbors on the evolutionary tree are merged, this will finally give us the exact evolutionary tree T . Thus Theorem 1 holds. \square

Quartet-joining algorithm will take $n - 3$ steps to build a fully resolved tree, n is the number of sequences to be studied. Initialization needs $O(n^4)$ time. At every step updates of confidence and support matrix need $O(n^3)$. So the time complexity for the algorithm is $O(n^4)$. Because we need to keep the confidence matrix, it is a four-dimension matrix, the space complexity is also $O(n^4)$.

4 Experimental Results

4.1 Computer Simulation Results

In the computer simulation experiment, we use the benchmarks developed by Ranwez and Gascuel to test and compare different phylogeny methods [5]. Six model trees, each consisting of eight leaf nodes, are used to generate test sets under various situations. Among them, AA, BB, AB are molecular clock-like trees while the other three CC, DD, CD, have different substitution rate among lineages. Three evolutionary rates are considered: their maximal pair-wise divergences (MD) are 0.1, 0.5, 0.8 substitution per site ranging from very low to fast. For each group of parameters, sequences of length 300, 600 and 900 sites are generated. A total data set of 86,400 DNA sequences are generated using Seq-Gen [4]. The evolutionary model used here is F81.

In this paper, we concentrate on reconstructing evolutionary tree from a set of quartets. So we just use a very simple method to assign weights to quartets. Let D_{ij} to be the distance between generated sequences i and j . The weights of input quartets are computed as:

$$w(x, y; u, v) = \begin{cases} 1 & D_{xy} + D_{uv} < D_{xu} + D_{yv}, D_{xy} + D_{uv} < D_{xv} + D_{yu}; \\ \frac{1}{2} & D_{xy} + D_{uv} = D_{xu} + D_{yv} < D_{xv} + D_{yu} \\ & \text{or } D_{xy} + D_{uv} = D_{xv} + D_{yu} < D_{xu} + D_{yv}; \\ \frac{1}{3} & D_{xy} + D_{uv} = D_{xv} + D_{yu} = D_{xu} + D_{yv}; \\ 0 & \text{else.} \end{cases} \quad (13)$$

Three phylogeny methods are tested on this data set: QJ, NJ and QP. Here we emphasize again that QJ is not a distance based method. It can reconstruct the phylogenetic tree without the distance matrix as long as the quartet set is provided. QP has the same property. On the other hand, NJ is a pure distance based method. Its reconstruction must be based on the distance matrix. In this experiment, for QP method, the tree appears most frequently in the building process is selected as the output.

The final results are shown in Fig 3. The Y-axis in Fig 3 is the average correctly reconstructed tree percentage for six model trees. The X-axis is the expected number of substitution in DNA sequences i.e. MD times sequence length. The corresponding data table is shown in table 1. The figures in the table are the correct tree percentage for each type of trees. The figures in the parenthesis are the correct quartet percentage i.e. the percentage of input quartet set that actually belongs to the quartet set of model trees.

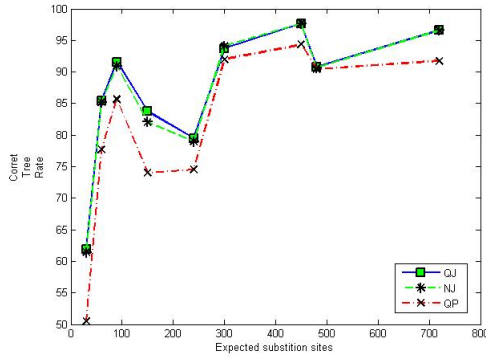


Fig. 3. Average correctly reconstructed tree percentage

From Fig 3, we can clearly see that in the average sense, the performance of QJ and NJ are quite close: their performance lines are almost overlapping except a small portion where QJ outperforms NJ slightly. But when compared to QP, the performance advantage of QJ is quite obvious: QJ outperforms QP in all the combination of evolutionary rate and sequence length. If we check the data table, we can find some other patterns. First is that within the evolutionary rates considered, faster evolutionary rates correspond to more accurate quartet set. When the correct quartet rate is relatively low, QJ performs slightly better than NJ. For example, when sequence length=300, MD=0.1, for the model tree AA, the correct quartet rate is 88%, QJ outperforms NJ about 4%. However, if the correct quartet rate is high, both QJ and NJ can do very well. Another thing should be noticed is that compared to other five tree models, QP method performs better on CD type of tree. When the evolutionary rates are high, it even can outperform QJ and NJ slightly.

4.2 Real Data Set Experiment

The real data set comes from GenBank. They are mitochondrial DNA sequences of 36 different mammals. The evolutionary tree of these 36 mammals are reconstructed by biologists using maximum likelihood method. [6].

Each of these 36 DNA sequences has about 16k sites. To estimate the pairwise evolutionary distances between these sequences, we will use the metric of strings introduced by Bin Ma and Ming Li [3]. We apply QJ, NJ and QP methods to this data set and compare the results with the tree reconstructed by maximum likelihood method. The RF distances [7] between the trees reconstructed by above three methods and the tree rebuilt by maximum likelihood method is shown in Table 2. We can see from the table that on the real data set, the results of quartet-joining method is closer to the result of maximum likelihood method than the other two methods.

Table 1. Data for each type of trees

	Algorithms	WITH CLOCK			NO CLOCK			
		AA	BB	AB	CC	DD	CD	
sequence length 300	MD≈ 0.1	QJ	32.0	73.5	76.5	56.5	71.5	61.5
		NJ	28.0(88)	77.0(96)	76.5(96)	51.0 (92)	72.5 (96)	63.5 (95)
		QP	8.0	65.5	67.0	34.0	75.0	53.5
	MD≈ 0.5	QJ	59.5	97.5	96.0	73.0	93.0	83.5
		NJ	55.5 (91)	98.0 (98)	96.5 (98)	65.5 (94)	92.5 (98)	84.5(95)
		QP	24.5	90.5	86.5	67.5	89.5	85.5
	MD≈ 0.8	QJ	55.5	94.5	95.5	60.5	97.0	74.0
		NJ	51.0(91)	95.0 (97)	96.0 (98)	54.5 (92)	96.0 (98)	81.5 (97)
		QP	23.5	92.0	87.0	69.0	93.5	82.0
sequence length 600	MD≈ 0.1	QJ	63.5	97.0	96.5	78.0	92.5	85.0
		NJ	62.0 (94)	97.5 (99)	96.0(99)	77.5 (96)	92.0 (99)	86.0(98)
		QP	42.0	92.5	92.0	62.5	92.5	84.5
	MD≈ 0.5	QJ	80.5	100.0	100.0	89.0	100.0	93.0
		NJ	82.5 (96)	99.5 (99)	100.0 (99)	85.5 (96)	100.0 (99)	97.0(98)
		QP	66.5	99.0	99.5	91.5	98.0	97.5
	MD≈ 0.8	QJ	78.5	99.0	100.0	78.0	100.0	89.5
		NJ	75.5 (95)	99.5 (99)	100.0 (99)	76.0 (93)	100.0 (99)	93.0(97)
		QP	53.0	98.5	98.0	95.5	99.5	98.0
sequence length 900	MD≈ 0.1	QJ	78.0	99.5	99.0	79.5	97.5	96.0
		NJ	77.0 (95)	98.5 (99)	99.0 (99)	75.5 (95)	97.5 (99)	98.0(99)
		QP	64.0	97.5	98.5	65.5	95.5	93.0
	MD≈ 0.5	QJ	92.5	100.0	99.5	94.5	100.0	99.5
		NJ	92.5 (98)	100.0 (100)	99.5 (99)	94.0 (98)	100.0 (99)	100.0(99)
		QP	81.5	99.5	99.5	86.0	99.5	100.0
	MD≈ 0.8	QJ	93.5	100.0	100.0	93.5	100.0	92.5
		NJ	93.0 (97)	100.0 (100)	100.0 (100)	89.5 (97)	100.0 (99)	97.0(99)
		QP	75.0	100.0	100.0	76.0	100.0	99.5

Table 2. Real Data Set Results

Algorithms	RF distances
QJ	12
NJ	16
QP	20

5 Improvement of Experimental Results

No matter how successfully a quartet-based method can tolerate quartet errors, the accuracy of input quartet set will still influence final results. Here, we propose a simple technique than can improve our experimental results when the evolutionary rate is low. Notice that for a quartet 1,2,3,4, if we have

$$D_{13} + D_{24} < D_{14} + D_{23}, D_{13} + D_{24} < D_{12} + D_{34} \tag{14}$$

but

$$D_{12} + D_{34} \neq D_{14} + D_{23} \tag{15}$$

Then we will have unresolved case as shown in Fig 4

It could either take the form of {13|24} or {12|34}. The internal branch length a, b can be directly computed from pairwise distance. The intuition is if a is longer, then the quartet will have more chance to take the form of {13|24}. If

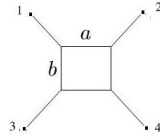


Fig. 4. Unresolved quartet

Table 3. Improved Results

	Algorithms	WITH CLOCK			NO CLOCK		
		AA	BB	AB	CC	DD	CD
sequence length 300	MD \approx 0.1 QJ Imp	36.5	75.5	76.5	66	78.5	67.5
	QJ	32.0	73.5	76.5	56.5	71.5	61.5
	NJ	28.0	77.0	76.5	51.0	72.5	63.5
	QP	8.0	65.5	67.0	34.0	75.0	53.5

b is longer, the quartet will have more chance to take the other form. So here instead of assigning a binary value to quartet weights, we will assign probability values estimated from a and b . But we find this technique works only when the evolutionary rate is low. The improved results are shown in Table 3. From the first row, we can clearly see the improvement of QJ method. By using this technique, QJ totally outperform NJ when MD equals 0.1.

6 Conclusion

In this paper, we propose a new quartet approach for reconstructing phylogenetic trees: quartet-joining method. QJ method will execute in $O(n^4)$ time. Theoretically it will build a fully resolved tree that perfectly represents the input quartet set when the quartet set is completely consistent with the evolutionary tree. In computer simulations, it totally outperforms popular QP method in the average sense. When compared to NJ, their performances are quite close except QJ performs a little bit better. However when a special technique is used, QJ can gain quite performance advantage over NJ when the evolutionary rate is low. We also test QJ method on the real data set and the results show its output is closer to maximum likelihood method than QP and NJ. Currently QJ gets its input quartet set from a very simple method. We expect when more complex methods like maximum likelihood are used to generate quartet weights, QJ can gain some performance advantage over NJ under most conditions.

Acknowledgement

This research is supported by Natural Sciences and Engineering Research Council of Canada.

References

1. Adachi, J., Hasegawa, M.: Instability of quartet analyses of molecular sequence data by the maximum likelihood method: the cetacean/artiodactyla relationships *Cladistics*, Vol. 5, pp.164-166 (1999)
2. Felsenstein, J.: Evolutionary trees from DNA sequences: a maximum likelihood approach. *J. Mol. Evol.* 17, 368–376 (1981)
3. Li, M., Chen, X., Li, X., Ma, B., Paul, M.B.: The Similarity Metric. *IEEE Transactions On Information Theory*, vol. 50(12) (2004)
4. Rambaut, A., Grassly, N.C.: Seq-Gen: An application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. *Comput. Appl. Biosci.* (1996)
5. Ranwez, V., Gascuel, O.: Quartet-Based Phylogenetic Inference:Improvement and Limits. *Mol. Biol. Evol.* 18(6), 1103–1116 (2001)
6. Reyes, A., Gissi, C., Pesole, G., Catzeflis, F.M., Saccone, C.: Where Do Rodents Fit? Evidence from the Complete Mitochondrial Genome of *Sciurus vulgaris*. *Molecular Biology and Evolution* 17, 979–983 (2000)
7. Robinson, D.F., Foulds, L.R.: Comparison of phylogenetic trees. *Math. Biosci.* 53, 131–147 (1981)
8. Saitou, N., Nei, M.: The Neighbor-joining Method: A New Method for Reconstructing Phylogenetic Trees. *Mol. Bio. Evol.* 4(4), 406–425 (1987)
9. Strimmer, K., Goldman, N., Von Haeseler, A.: Quartet puzzling:a quartet maximum-likelihood method for reconstructing tree topologies. *Mol. biol. E* 13, 964–969 (1996)
10. Strimmer, K., Goldman, N., Von Haeseler, A.: Bayesian probabilities and quartet puzzling. *Mol. biol. E* 14, 210–211 (1997)
11. Schmidt, H.A., Strimmer, K., Vingron, M.: TREE-PUZZLE: maximum likelihood phylogenetic analysis using quartets and parallel computing. *Bioinformatics* 18(3), 502–504 (2002)
12. Zhou, B.B., Tarawneh, M., Wang, C.: A Novel Quartet-based Method for Phylogenetic Inference. In: *Proceeding of 5th IEEE Symposium on Bioinformatics and Bioengineering 2005*, IEEE Computer Society Press, Los Alamitos (2005)

Integer Programming Formulations and Computations Solving Phylogenetic and Population Genetic Problems with Missing or Genotypic Data

Dan Gusfield¹, Yelena Frid¹, and Dan Brown²

¹ Department of Computer Science, University of California, Davis
gusfield@cs.ucdavis.edu

² David R. Cheriton School of Computer Science, University of Waterloo, Canada
browndg@cs.uwaterloo.ca

Abstract. Several central and well-known combinatorial problems in phylogenetics and population genetics have efficient, elegant solutions when the input is complete or consists of haplotype data, but lack efficient solutions when input is either incomplete, consists of genotype data, or is for problems generalized from decision questions to optimization questions. Unfortunately, in biological applications, these harder problems arise very often. Previous research has shown that integer-linear programming can sometimes be used to solve hard problems in practice on a range of data that is realistic for current biological applications. Here, we describe a set of related integer linear programming (ILP) formulations for several additional problems, most of which are known to be NP-hard. These ILP formulations address either the issue of missing data, or solve *Haplotype Inference Problems* with objective functions that model more complex biological phenomena than previous formulations. These ILP formulations solve efficiently on data whose composition reflects a range of data of current biological interest. We also assess the biological quality of the ILP solutions: some of the problems, although not all, solve with excellent quality. These results give a practical way to solve instances of some central, hard biological problems, and give practical ways to assess how well certain natural objective functions reflect complex biological phenomena. Perl code to generate the ILPs (for input to CPLEX) is on the web at www.csif.cs.ucdavis.edu/~gusfield.

1 Introduction

Several well-studied problems in computational phylogenetics and population genetics have efficient, elegant solutions when the data is “simple” or “ideal”, but lack efficient solutions when the data is more complex, due to recombination, homoplasy, missing entries, site incompatibility, or the need to use genotypic rather than haplotypic data. Similarly, optimization variants of many decision problems are often more difficult to solve. In this paper we discuss integer linear

programming (ILP) formulations for several such problems and report on empirical investigations done with these formulations. For most of the problems, the concept of incompatibility is fundamental.

Definition: Given a matrix M whose entries are 0's and 1's, two sites (or columns) p and q in M are said to be *incompatible* if and only if there are four rows in M where columns p and q contain all four of the ordered pairs 0,1; 1,0; 1,1; and 0,0. The test for the existence of all four pairs is called the “four-gamete test” in population genetics.

The concept of incompatibility is central to many questions concerning phylogenetic trees and population histories [14,25,7] for the following reason. Considering each row of M as a binary sequence, the classic Perfect Phylogeny Theorem says that there is a rooted phylogenetic tree that derives the sequences in M , starting from some unspecified root sequence and using only one mutation per site, if and only if *no* pair of sites of M are incompatible. Moreover, if the root sequence is specified, then the phylogenetic tree is *unique*. For expositions of this classic result, see [9,25]. The assumption of one mutation per site is called the “infinite sites” assumption in population genetics.

2 Missing Data Problems

When there are no missing values in M , the decision question of whether there are incompatible pairs of sites can be answered in linear time [8], and of course the number of incompatible pairs can trivially be computed in polynomial time. However, the situation is more interesting and realistic when some entries of M are missing, and this leads to three natural, biologically motivated problems, which we call M1, S1, and R1.

2.1 Imputing Values to Minimize Incompatibility

Problem M1: Given a binary matrix M with some entries missing, fill in (impute) the missing values so as to *minimize* the number of incompatible pairs of sites in the resulting matrix M' .

Problem M1 generalizes the following decision question: Can the missing values be imputed so that the the sequences in M' can be generated on a perfect phylogeny? That decision question has an efficient, elegant solution [21] when a required root sequence of the unknown phylogeny is specified as input, but is NP-complete when the root sequence is not specified [27]. When the missing values can be imputed so that M' has no incompatible site pairs, the sequences in M are consistent with the hypothesis that they originated from a perfect phylogeny; when the values cannot be imputed with zero incompatibilities, the solution to Problem M1 gives a measure of the deviation of M from the Perfect Phylogeny model.

The ILP Formulation for Problem M1. Our ILP formulation for Problem M1 is direct and simple. Its importance is that it generally solves very quickly,

Table 1. Six rows and two columns in the input M to Problem M1

M	p	q
1	0	0
2	?	1
3	1	0
4	?	?
5	?	0
6	0	?

imputing missing values with high accuracy, and that it can be built upon to address more complex problems. More generally, these formulations and computations illustrate that for applied problems whose range of data is known, the fact that a problem is NP-hard does not necessarily imply that exact solutions cannot be efficiently obtained for that data.

The ILP for problem M1 has one binary variable $Y(i, j)$ for each cell (i, j) in M that is missing a value; the value given to $Y(i, j)$ is then the imputed value for $M(i, j)$. The program that creates the ILP for problem M1 identifies all pairs of columns (p, q) of M that are not necessarily incompatible, but can be made incompatible depending on the imputed values are set. We let P be the set of such pairs of columns; for each pair (p, q) in P , the program creates a variable $C(p, q)$ in the formulation, which will be forced to 1 whenever the imputations cause an incompatibility between columns p and q . For each pair in P , the program also determines which of the four binary combinations are not presently found in column pair (p, q) ; let $d(p, q)$ represent those missing (deficient) binary combinations. The program creates a binary variable $B(p, q, a, b)$ for every ordered binary combination a, b in $d(p, q)$; $B(p, q, a, b)$ will be forced to 1 if the combination a, b has been created (through the setting of the Y variables) in *some* row in columns (p, q) . The program next creates inequalities that set a binary variable $C(p, q)$ to 1 if $B(p, q, a, b)$ has been set to 1 for *every* combination a, b in $d(p, q)$. Therefore, $C(p, q)$ is set to 1 if (but not only if) the imputations of the missing values in columns (p, q) cause those sites to be incompatible. To explain the formulation in detail, consider the pair of columns shown in Table 1, where a missing entry is denoted by a “?”. Then $d(p, q)$ is $\{(0, 1), (1, 1)\}$.

For each pair a, b in $d(p, q)$, the program will make one inequality involving $B(p, q, a, b)$ for each row r where the pair a, b can be created in columns p, q . The specific inequality for a pair a, b and a row r depends on the specific pair a, b and whether there is a fixed value in row r in sites p or q . The full details are simple and omitted but explained for variable $B(p, q, 1, 1)$ using the above example. Those inequalities are:

$$Y(2, p) \leq B(p, q, 1, 1) \tag{1}$$

$$Y(4, p) + Y(4, q) - B(p, q, 1, 1) \leq 1, \tag{2}$$

which force the variable $B(p, q, 1, 1)$ to 1 when the missing value in the second row is set to 1 or the two missing values in the fourth row are set to 1; this is the general pattern for this combination.

In the above example, the inequalities to set variable $B(p, q, 0, 1)$ are:

$$Y(2, p) + B(p, q, 0, 1) \geq 1 \tag{3}$$

$$Y(4, q) - Y(4, p) - B(p, q, 0, 1) \leq 0 \tag{4}$$

$$Y(6, q) - B(p, q, 0, l) \leq 0 \tag{5}$$

The ILP for the example has the following inequality to set the value of variable $C(p, q)$ to one, *if* all the combinations in $d(p, q)$ have been created in the column pair (p, q) :

$$C(p, q) \geq B(p, q, 1, 1) + B(p, q, 0, 1) - 1$$

In general, the constant on the right-hand side of the inequality is one less than the number of pairs in $d(p, q)$. These inequalities assure that $C(p, q)$ will be forced to 1 if (but not only if) the missing values in columns (p, q) are imputed (by the setting of the Y variables) in a way that makes site pair (p, q) incompatible.

The overall objective function for the ILP is therefore to Minimize $[|F| + \sum_{(p,q) \in P} C(p, q)]$, where F is the set of pairs of incompatibilities forced by the initial 0 and 1 entries in matrix M . We include $|F|$ in the objective for continuity in a later section.

Because the objective function calls for minimizing, we do not need inequalities to assure that $C(p, q)$ will be set to 1 *only if* the missing values in columns (p, q) are imputed in a way that makes site pair (p, q) incompatible. However, such inequalities are possible, and would be added to (or used in place of) the above inequalities if we want to solve the the problem of imputing missing values in order to *maximize* the resulting number of incompatible pairs. Details are omitted for lack of space. The solution to the maximization problem, along with the solution to Problem M1, bracket the number of incompatible pairs in the true data from which M was derived.

If M is an n by m matrix, the above ILP formulation for problem M1 creates at most nm Y variables, $2m^2$ B variables, $\frac{m^2}{2}$ C variables, and $O(nm^2)$ inequalities, although all of these estimates are worst case and the numbers are typically much smaller. For example, if I is the expected percentage of missing entries (which is as low as 3% in many applications), then the expected number of Y variables is nmI . The formulation as described can create redundant inequalities, but we have left them in our description for conceptual clarity, and in practice we have found that the preprocessor in CPLEX removes such redundancies as effectively as any of our more refined programs do. There are additional practical reductions that are possible that we cannot discuss here due to limited space.

Empirical Results for Problem M1. We extensively tested the ILPs for Problem M1 to answer two questions: 1) How quickly are the ILPs for problem M1 solved when problem instances are generated using data from a population genetic process; 2) When parts of the data are removed, and the missing data

imputed by the ILP solution, how accurately do those imputations reconstruct the original values? In phylogenetic applications, missing data rates of up to 30% are common, but in population genetic applications, rates from one to five percent are more the norm.

We used the program *ms* created by Richard Hudson [15] to generate the binary sequences. That program is the widely-used standard for generating sequences that reflect the population genetic coalescent model of binary sequence evolution. The program allows one to control the level of recombination (defined in Section 2.3) through a parameter r , and a modified version of the program provided by Yun Song, allows one to control the level of *homoplasy* (recurrent or back mutations violating the infinite sites assumption). After a complete dataset was generated, each value in the data was chosen for removal with probability p , varied in the study; we use $I = p \times 100$ to denote the expected percent of missing values. All computations were done on a 1.5 ghz Intel itanium, and the ILPs were solved using CPLEX 9.1. The time needed to generate the ILPs was minimal and only the time used to solve the ILP is reported. We tested our ILP solution to Problem M1 on all combinations of n (# rows) = {30, 60, 90, 120, 150}; m (# columns) = {30, 60, 90}; r = {0, 4, 16, 30}; and I (expected percent missing values) = {5%, 10%, 15%, 20%, 25%, 30%}[1]. We generated and tested fifty datasets for each parameter combination.

The running times were slightly more influenced by m than by n , and mildly influenced by r , but were mostly a function of $n \times m$ and I (increasing with $n \times m$ and I , and decreasing with r). The largest average execution time (averaged over all 50 datasets) occurred when n, m and I , were at their maximum values and r was zero, but that average time was only 3.98 seconds. Figure 1(a) shows average running times as a function of I . Hence, despite being NP-hard, problem M1 can be solved very efficiently on a wide range of data whose parameters reflect datasets of current interest in phylogenetics and population genetics.

The imputation accuracy was also excellent, when the product $n \times m$ is large. Error is the percentage of missing values that were incorrectly imputed. The case of 5% missing data illustrates this. When $n = m = 30$, and $r = 4$, the missing values were imputed with an average error of 6.4%, but when $n = 120, m = 90$ the average error dropped to 1.8%. In general, error increased with increasing r , and fell with increasing $n \times m$. The likely reason for such good results when $n \times m$ is large is the high level of redundancy in the data, and that these redundancies are exploited when the objective is to minimize incompatibilities. Figure 1(b) shows error rates as a function of $n \times m$.

2.2 Imputing Values to Minimize Site-Removals

In this section we discuss another natural objective function related to Problem M1. We first define the **Site-Removal** problem on *complete* data: Given a binary

¹ We had earlier studied the case of $I = 1\%$ using a slower ILP formulation, but almost all of the computations took zero recorded time (to three decimal places) on that data, and so we started with a higher percentage of missing data for this study.

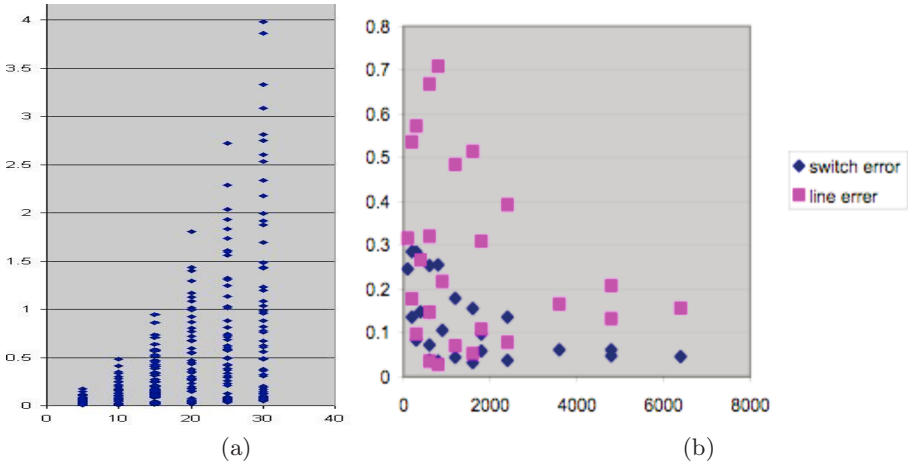


Fig. 1. (a) Average time (in seconds) needed to solve the ILP for Problem M1 as a function of I , the expected percentage of missing data. Each diamond is the average of 50 datasets for a particular combination of the parameters n, m, r and I . (b) Average imputation error rates (percentages) in the solution of the ILP for Problem M1, as a function of $n \times m$. Each diamond is the average of 50 datasets for a particular combination of the parameters n, m, r and I .

matrix M with no missing entries but some pairs of incompatible sites, find the *smallest* set of sites to remove from M so that no remaining pair of sites is incompatible.

Finding a small(est) set of sites to remove so that no remaining pairs are incompatible is often suggested and employed (particularly in phylogenetic studies) as a means to clean up data that does not perfectly conform to the Perfect Phylogeny model. The expectation is that while there may be some sites where homoplasy is observed (due to recurrent or back mutations at that site), a Perfect Phylogeny constructed from the remaining sites will give valid evolutionary information about the taxa. There are similar scenarios in population genetics. Of course, in order to get the most informative phylogeny, we want to remove as few sites as possible, motivating the Site-Removal problem. The Site-Removal Problem is NP-hard and is typically formulated as a Node-Cover problem in a graph where each node represents a site and each edge connects an incompatible pair of sites [25,7].

When M has missing entries, the Site-Removal problem is generalized to: **Problem S1:** Over all matrices created by imputing missing values in M , find the matrix M' to minimize the the solution to the Site-Removal problem on M' .

An ILP formulation for Problem S1 is easily obtained from the ILP formulation for Problem M1 as follows: Let $D(i)$ be a binary variable used to indicate whether or not site i will be removed. Then for each pair $(p, q) \in P$, add the inequality $D(p) + D(q) - C(p, q) \geq 0$, which says that if the missing values are imputed so that site pair (p, q) becomes incompatible, then either site p

or site q must be removed. Also, for each pair $(p, q) \in F$, add the inequality $D(p) + D(q) \geq 1$. Finally, change the objective function in the formulation for M1 to Minimize $\sum_{i=1}^m D(i)$.

Problem S1 can be solved efficiently on the range of data considered in Section 2.1 (with most computations taking less than one second) but slightly slower than for Problem M1, and with a higher error rate. For example, the average time and error of the 50 datasets with $n = 120$, $m = 90$, $r = 16$, $I = 15$, was 0.53 seconds and 3.1% for Problem M1 and 0.75 seconds and 5.4% for Problem S1.

2.3 Estimating Recombination in Data with Missing Values

Recombination (crossing-over) is a fundamental molecular phenomena, where during meiosis two equal length sequences produce a third sequence of the same length consisting of a prefix of one of the sequences followed by a suffix of the other sequence. A central problem is to determine the *minimum* number of recombinations, denoted $Rmin(M)$, needed to generate a set of binary sequences M from some known or unknown ancestral sequence, when the infinite sites assumption applies [16]. There is a large literature on this problem, but there is no known efficient algorithm to exactly compute $Rmin(M)$. However, there are efficient algorithms that give relatively good *lower bounds* on $Rmin$, and there are biological questions concerning recombination (for example, finding recombination hotspots) that have been successfully addressed using lower bounds on $Rmin$ rather than using $Rmin$ itself [1, 5]. The first published, and most basic lower bound, called the HK bound [16], is obtained as follows: Consider the m sites of M to be integer points $1 \dots m$ on the real line and pick a minimum number of non-integer points R so that for every pair of incompatible sites (p, q) in M , there is at least one point in R (strictly) between p and q . It is easy to show that $|R| \leq Rmin(M)$. When the data in M is complete, the HK bound $|R|$ can be computed in polynomial-time by a greedy-like algorithm. However, under the realistic situation that some entries in M are missing, we have **Problem R1**: Over all matrices created by imputing missing values in M , find the matrix M' to *minimize* the resulting HK lower bound on the $Rmin(M')$.

The reason for minimizing is that the result is then a valid lower bound on the number of recombinations needed to generate the true underlying data, when the infinite sites assumption applies.

Problem R1 is NP-hard [30], but an ILP formulation for it can be easily obtained from the formulation for Problem M1: For each c from 1 to $m - 1$, let $R(c)$ be a binary variable used to indicate whether a point in R should be chosen in the open interval $(c, c + 1)$. Then, for every pair of sites (p, q) in $F \cup P$, add the inequality $\sum_{p \leq c < q} R(c) \geq C(p, q)$ to the ILP for Problem M1, and change the objective function to Minimize $\sum_{c=1}^{m-1} R(c)$.

In our computations (details omitted due to space limitations), we have established that Problem R1 can be solved in practice over the same range of data discussed in Section 2.1; the computation times were longer than for Problem M1, but generally not more than twice as long. Of greater interest is the quality of the imputed values obtained in the solution to Problem R1 on data generated

with recombination. Because Problem R1 more explicitly reflects recombination than does Problem M1, we conjectured that the solutions to Problem R1 would impute the original values better than solutions to Problem M1. Surprisingly, the average error for solutions to Problem R1 was somewhat larger than for Problem M1. For example, the average error over all datasets with $n = 150$ was 4% for Problem M1 and 4.75% for problem R1.

3 Haplotyping Problems

One of the key technical problems in the acquisition of variation data in populations is called the ‘‘Haplotype Inference (HI) Problem’’, or the problem of determining the ‘‘phase’’ of unphased genotype data. A very large literature now exists on this problem (see [12] for one survey). Abstractly, input to the HI problem consists of n *genotype* vectors, each of length m , where each value in the vector is either 0,1, or 2. A site with value 2 is called a ‘‘heterozygous’’ site, while the other sites are called ‘‘homozygous’’ sites. In the context of this problem, a vector with only entries of 0 and 1 is called a ‘‘haplotype’’. Given an input set of n genotype vectors, a solution to the HI Problem is a set of n pairs of haplotypes, one pair for each genotype vector. For any genotype vector g , the associated haplotypes v_1, v_2 must both have value 0 (or 1) at any position where g has value 0 (or 1); but for any position where g has value 2, exactly one of v_1, v_2 must have value 0, while the other has value 1. Hence, for an individual with h heterozygous sites there are 2^{h-1} pairs of haplotypes that could appear in a solution to the HI problem. For example, if the observed genotype g is 0212, then the pair of haplotypes 0110, 0011 is one feasible solution out of two feasible solutions. Of course, we want to find the HI solution that is most biologically plausible, and for that we need additional criteria to guide the algorithm solving the HI problem. The goal is to devise criteria that reflect biological reality and yet allow efficient solution to the HI problem. Criteria have been previously proposed that were encoded as optimization problems with precise objective functions. In this paper, we discuss four additional biologically-motivated optimization problems that have practical ILP solutions.

3.1 Haplotyping Versions of M1, S1, R1

Each of the three problems M1, S1 and R1 has a natural analog as a haplotyping problem, and has biological and historical connections to other haplotyping problems.

Problem HM1: Solve the HI problem so that the number of incompatible pairs of sites in the HI solution is *minimized* over all HI solutions.

Problem HM1 is a natural extension of the following ‘‘Perfect Phylogeny Haplotyping (PPH)’’ problem [10]: Find, if possible, a solution to the HI problem so that the haplotypes in the solution can be derived on a perfect phylogeny; in other words, so that there are *no* incompatible pairs of sites in the HI solution. Such an HI solution is called a ‘‘PPH solution’’, and if there is one, it can be

found in linear time [6,22]. See [10] for a discussion of the biological justification of the PPH problem. The PPH model is justified in some applications, but not all, and there are additional applications where the true haplotypes deviate by only a small amount from the PPH model (low recombination “haplotype blocks” are the prime example). In [13], a heuristic approach was developed to handle small deviations from the PPH model. An attempt to more formally model small deviations from the PPH model was explored in [26,23]. Problem HM1 is an alternative way to formalize, and quantify, deviations from the PPH model: a set of genotypes that allow HI solutions with a small number of incompatible pairs deviate less from the PPH model than do genotypes that only allow HI solutions with a large number of incompatible pairs. We were therefore interested in whether the HM1 problem can be solved efficiently in a range of biologically relevant data, and how well the HI solutions obtained this way reconstruct the correct haplotypes.

An ILP formulation for Problem HM1 can be easily obtained by modifying the formulation for Problem M1: First, duplicate each row of M creating matrix \overline{M} , and create the ILP for Problem M1 using matrix \overline{M} , treating each 2 as a “?”. Then for each cell (i, q) where $M(i, q)$ is 2, add the inequality $Y(2i - 1, q) + Y(2i, q) = 1$. This formulation can be further improved, and such improvements have been implemented. For example, if $M(i, p) = 2$ and $M(i, q) = 1$ the binary combinations 0,1 and 1,1 will definitely be generated in columns (p, q) and that information may reduce the elements in $d(p, q)$, and reduce the size of the ILP formulation.

In the same way, we can modify the ILP for Problem S1 to obtain an ILP for **Problem HS1**: Remove the minimum number of columns in the input *genotypes*, so that there is a PPH solution to the HI problem on the remaining data. That is another way to formalize, and quantify, deviation from the PPH model. The same kind of modification also extends the ILP for Problem R1 to an ILP for **Problem HR1**: Solve the HI problem in order to minimize the HK bound on the haplotypes in the solution. That problem has been proposed as a way to search for recombination hotspots in genotypic data rather than haplotypic data [29]. It is interesting to note that Problem HR1 has a polynomial time solution [29], as does the problem of solving the HI problem in order to *maximize* the HK bound [31], even though Problem R1 is NP-hard.

Empirical Results for Problem HM1. We extensively tested instances of Problems HM1, HS1 and HR1 using datasets with $\frac{n}{2}$ genotypes created by pairing n haplotypes output by the program *ms* described in Section 2.1. We tested 50 datasets for each combination of n, m and r that we examined. We observed that we could solve these problems in practical time on a wide range of data, but not as extensive as for Problem M1. Further, the computation times were longer (considerably so for larger instances) for the same parameter combinations of n, m and r . In general, the HI solutions given by Problem HM1 were better than for HS1 and HR1, and so we will only discuss those here, although the running times for HM1 were larger than for HR1. In our experiments, we stopped any computations that exceeded three hours, and considered only combinations of

$n = \{10, 20, 30, 60, 80\}$ haplotypes and $m = \{30, 60, 90\}$ sites. As an example, when $n = 80, m = 60, r = 16$, 94% of the datasets terminated within the three hour limit. However, for most of the parameter choices we examined, all of the datasets terminated within the time limit, and most terminated well below that limit.

In addition to the solution time, we were interested in the quality of the haplotypes produced and how that quality varied depending on whether the deviation from the PPH model was due to recombination or to homoplasy. Datasets with homoplasy were generated with 5, 10 or 20 sites where additional mutations were forced to occur. Over these ranges of n, m and r , we did not see a significant difference between the quality of the haplotypes produced from datasets generated with recombination and those with homoplasy, and so we will discuss only the recombination case.

To assess the quality of the solutions, we used the standard *switch error* [17,19] and the *line error*, comparing the haplotype pairs obtained from solving Problem HM1 with the original pairs used to generate the genotype data. The switch error is the minimum number of runs (blocks) of contiguous sites that need to be exchanged between the computed haplotype pairs in order to make the resulting haplotype pairs agree with the correct pairs, divided by the number of heterozygous sites in the data. The line error is simply the number of haplotype pairs in the solution that do not agree completely with the corresponding correct pair, divided by the number of genotypes. The ILP executions that were terminated after three hours all found HI solutions, and so we could test their quality also. Hence, no datasets were excluded from our accuracy analysis. We also compared the accuracy of the HM1-computed haplotypes with the haplotypes found by program FastPhase [24], the successor program of the widely-used program PHASE [28].

We observed switch errors for the HI solutions produced by solving Problem HM1 that were very good in some parameter ranges, often superior (by a small amount) to the switch error of the solutions produced by FastPhase; in other parameter ranges the observed switch errors were somewhat inferior to those from FastPhase, and to accuracies reported for simulations using HapMap data [20] (although in line with some real data [17]). The line errors of solutions from both FastPhase and Problem HM1 were relatively large, but quite similar to each other. Unlike imputation error, switch accuracy is highly influenced by r , the recombination parameter; consistent with imputation error, all accuracies improved with larger data sets, particularly as the number of rows increase. We should note that in our tests we did not require that the minor allele appear above a minimum frequency as is commonly done; it is well known [19] that accuracies are improved by imposing that requirement.

Figure 2 (a) summarizes the switch and line errors of the HI solutions from obtained Problem HM1 compared to solutions given by FastPhase; Figure 2 (b) shows the time needed to solve Problem HM1.

As an illustration of the influence of n , when $n = 20, m = 30, r = 4$, the average switch errors from HM1 and FastPhase were 0.1189 and 0.136 which are

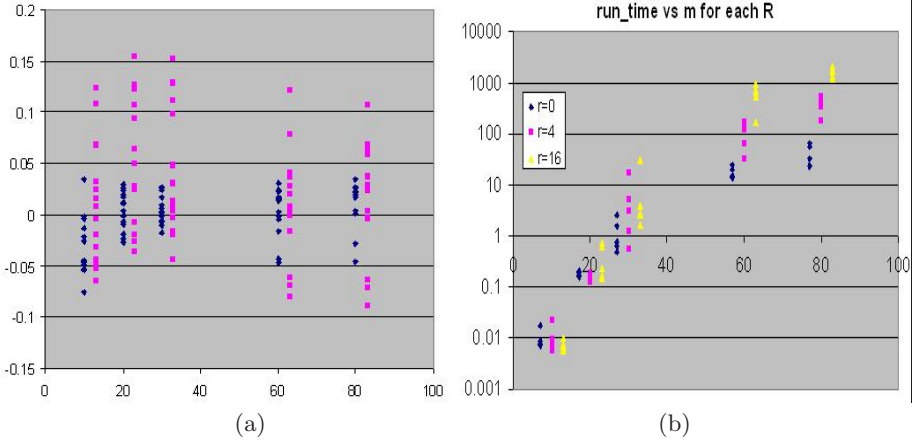


Fig. 2. (a) Comparison of switch and line errors the HI solutions from Problem HM1 and FastPhase, as a function of n . In each simulated dataset the switch and line errors of the HM1 solution were subtracted from switch and line errors of the FastPhase solution. A positive result shows the HM1 solution superior to the FastPhase solution. Each diamond (square) is the average difference of switch (line) errors from the 50 datasets for a particular combination of the parameters n, m and r . (b) Solution times (over the terminating datasets) as a function of m , for three values of parameter r . Each object is the average of 50 datasets with the same values of n .

relatively large, but when n increased to 60, the errors declined to 0.0532 and 0.068, and when n increased to 80, the errors were 0.0452 and 0.062. To see the influence of the recombination parameter r , consider the case of $n = 60, m = 30$, where the switch errors for HM1 were 0.046, 0.0532, 0.0984 with $r = 0, 4$ and 16 respectively, and the errors for FastPhase were 0.063, 0.068, and 0.083.

The comparison of *individual* HI solutions obtained from Problem HM1 and from FastPhase often gave contradictory results, so we averaged the results over all the data examined: the HI solutions from Problem HM1 had an average switch error of 0.13, an average line error of 0.34 and required an average computation time of 186.3 seconds for the terminating computations, while 4% of the computations did not terminate in three hours. The FastPhase solutions had an average switch error of 0.128, an average line error of 0.36 and required an average of 38 seconds to compute. These results suggest that the qualities of the two approaches are very similar. While the time needed to solve Problem HM1 is greater than the time for FastPhase, the main goal in solving HM1 was to see how well this natural extension of perfect phylogeny haplotyping solves the HI problem (although we would have been pleased to report that it solved faster than FastPhase). Having a solvable, simple-to-state objective function allows one to assess the biological fidelity of the model reflected by the objective function, giving much cleaner and clearer semantics compared to more black-box methods whose semantics may be very unclear. We consider the results based on HM1 to be positive and informative.

3.2 The MinPPH Problem

When there is a PPH solution to the HI problem, there may be several solutions, and it is desirable to apply a secondary criterion to choose one. An appealing approach, motivated both by theory and empirical observations, is to solve the following problem called the **MinPPH Problem**: Find a PPH solution that *minimizes* the number of *distinct* haplotypes used in any of the PPH solutions.

The MinPPH Problem is a mixture of the PPH problem and the problem of Haplotype Inference by Pure Parsimony (denoted HIPP) [11,4,18]. The MinPPH problem was defined and justified in [2] where it was shown to be NP-hard. An ILP formulation for MinPPH (different than presented here) was described in [3], but not implemented due to the expectation that it would not solve efficiently.

The idea of our ILP formulation is to modify the formulation for Problem HM1 and combine it with the simplest ILP formulation for the HIPP problem given in [4] (see also [12] for a description of that HIPP formulation, and [18] for a similar formulation). We start with the HIPP formulation from [4], but add to it the inequalities from the HM1 formulation along with the equality $\sum_{(p,q) \in P} C(p,q) = 0$. The end result is an ILP formulation that solves the HI problem using the minimum number of distinct haplotypes possible, subject to the constraint that the HI solution is a PPH solution (assuming a PPH solution exists).

We extensively tested this ILP formulation for solution speed and haplotype accuracy, using genotypic data (created from *ms* with $r = 0$) where PPH solutions were assured. We obtained two striking empirical results. The first result is that the ILPs solve extremely fast (generally less than one second) over a range of data up to 80 rows and 80 columns. This speed is even more notable considering that the HIPP formulation from [4] requires hours or days to solve the HI problem on the smaller instances, and cannot solve the larger instances

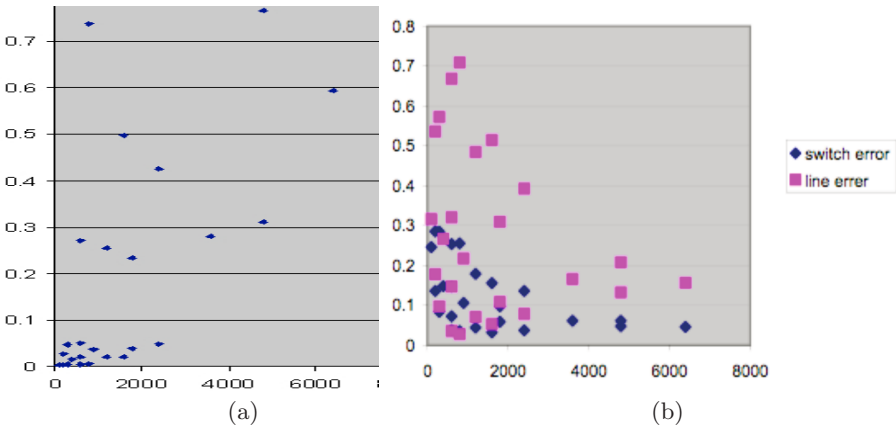


Fig. 3. (a) Average time needed to solve the ILP for Problem MinPPH, as a function of $n \times m$. Each diamond is the average of 50 datasets for a particular combination of the parameters n, m . (b) Average switch and line errors as a function of $n \times m$.

in practical time. Hence, it is the PPH constraints added to the HIPPP formulation that makes the resulting formulation solve so quickly. We also examined the question of how the running times were influenced by the number of PPH solutions, and we saw no clear pattern.

The second striking empirical result is that the accuracy of the HI solutions given by the MinPPH solution is notably better than solutions obtained by FastPhase (except for instances with a very small number of rows), while running considerably faster. For example, when $n = m = 80$, the average MinPPH solution time was 0.59 seconds with a switch-error of 0.045, while the run time for FastPhase was 163 seconds with a switch-error of 0.074. In general, the MinPPH switch and line errors decrease with increasing problem size, as measured either by n or $m \times n$. Figures 3(a) and 3(b) show the MinPPH runtime and switch-error and line-error as a function of $n \times m$.

Acknowledgements

We thank Yun Song for the use of the code to generate SNP sequences with homoplasy events and for helpful conversations and suggestions. We thank Chuck Langley for helpful conversations and suggestions. The research was supported by NSF grants CCF 0515278, IIS 0513910 and REU 0434759.

References

1. Bafna, V., Bansal, V.: Improved recombination lower bounds for haplotype data. In: McLysaght, A., Huson, D.H. (eds.) RECOMB 2005. LNCS (LNBI), vol. 3678, Springer, Heidelberg (2005)
2. Bafna, V., Gusfield, D., Hannenhalli, S., Yooseph, S.: A note on efficient computation of haplotypes via perfect phylogeny. *Journal of Computational Biology* 11(5), 858–866 (2004)
3. Brown, D., Harrower, I.: A new formulation for haplotype inference by pure parsimony. report cs-2005-03. Technical report, University of Waterloo, School of Computer Science (2005)
4. Brown, D.G., Harrower, I.M.: Integer Programming Approaches to Haplotype Inference by Pure Parsimony. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 3(2), 141–154 (2006)
5. International HapMap Consortium. A haplotype map of the human genome. *Nature*, 437 1299–1320 (2005)
6. Ding, Z., Filkov, V., Gusfield, D.: A linear-time algorithm for the perfect phylogeny haplotyping problem. In: Miyano, S., Mesirov, J., Kasif, S., Istrail, S., Pevzner, P., Waterman, M. (eds.) RECOMB 2005. LNCS (LNBI), vol. 3500, pp. 585–600. Springer, Heidelberg (2005)
7. Felsenstein, J.: *Inferring Phylogenies*. Sinauer, Sunderland, MA (2004)
8. Gusfield, D.: Efficient algorithms for inferring evolutionary history. *Networks* 21, 19–28 (1991)
9. Gusfield, D.: *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge (1997)

10. Gusfield, D.: Haplotyping as Perfect Phylogeny: Conceptual Framework and Efficient Solutions (Extended Abstract). In: Proceedings of RECOMB 2002: The Sixth Annual International Conference on Computational Biology, pp. 166–175 (2002)
11. Gusfield, D.: Haplotype inference by pure parsimony. In: Baeza-Yates, R.A., Chávez, E., Crochemore, M. (eds.) CPM 2003. LNCS, vol. 2676, pp. 144–155. Springer, Heidelberg (2003)
12. Gusfield, D., Orzack, S.: Haplotype inference. In: Aluru, S. (ed.) Handbook of Computational Molecular Biology, vol. 18, pp. 1–25. Chapman and Hall/CRC, Boca Raton (2005)
13. Halperin, E., Eskin, E.: Haplotype reconstruction from genotype data using Imperfect Phylogeny. *Bioinformatics* 20, 1842–1849 (2004)
14. Hein, J., Schierup, M., Wiuf, C.: *Gene Genealogies, Variation and Evolution: A primer in coalescent theory*. Oxford University Press, Oxford (2005)
15. Hudson, R.: Generating samples under the Wright-Fisher neutral model of genetic variation. *Bioinformatics* 18(2), 337–338 (2002)
16. Hudson, R., Kaplan, N.: Statistical properties of the number of recombination events in the history of a sample of DNA sequences. *Genetics* 111, 147–164 (1985)
17. Kimmel, G., Shamir, R.: GERBIL: Genotype resolution and block identification using likelihood. *PNAS* 102, 158–162 (2005)
18. Lancia, G., Pinotti, C., Rizzi, R.: Haplotyping populations by pure parsimony: Complexity, exact and approximation algorithms. *INFORMS J. on Computing, special issue on Computational Biology* 16, 348–359 (2004)
19. Lin, S., Cutler, D., Zwick, M., Chakravarti, A.: Haplotype inference in random population samples. *Am. J. of Hum. Genet.* 71, 1129–1137 (2002)
20. Marchini, J., Donnelly, P., et al.: A comparison of phasing algorithms for trios and unrelated individuals. *Am. J. of Human Genetics* 78, 437–450 (2006)
21. Pe’er, I., Pupko, T., Shamir, R., Sharan, R.: Incomplete directed perfect phylogeny. *SIAM J. on Computing* 33, 590–607 (2004)
22. Satya, R.V., Mukherjee, A.: An optimal algorithm for perfect phylogeny haplotyping. In: Proceedings of 4th CSB Bioinformatics Conference, IEEE Computer Society Press, Los Alamitos (2005)
23. Satya, R.V., Mukherjee, A., Alexe, G., Parida, L., Bhanot, G.: Constructing near-perfect phylogenies with multiple homoplasy events. *Bioinformatics* 22, 514–522 (2006) *Bioinformatics Suppl.*, Proceedings of ISMB 2006
24. Scheet, P., Stephens, M.: A fast and flexible statistical model for large-scale population genotype data: applications to inferring missing genotypes and haplotypic phase. *Am. J. Human Genetics* 78, 629–644 (2006)
25. Semple, C., Steel, M.: *Phylogenetics*. Oxford University Press, Oxford (2003)
26. Song, Y.S., Wu, Y., Gusfield, D.: Haplotyping with one homoplasy or recombination event. In: Casadio, R., Myers, G. (eds.) WABI 2005. LNCS (LNBI), vol. 3692, Springer, Heidelberg (2005)
27. Steel, M.: The complexity of reconstructing trees from qualitative characters and subtrees. *J. of Classification* 9, 91–116 (1992)
28. Stephens, M., Smith, N., Donnelly, P.: A new statistical method for haplotype reconstruction from population data. *Am. J. Human Genetics* 68, 978–989 (2001)
29. Wiuf, C.: Inference of recombination and block structure using unphased data. *Genetics* 166, 537–545 (2004)
30. Wu, Y.: Personal Communication
31. Wu, Y., Gusfield, D.: Efficient computation of minimum recombination over genotypes (not haplotypes). In: Proceedings of Life Science Society Computational Systems Bioinformatics (CSB) 2006, pp. 145–156 (2006)

Improved Exact Algorithms for Counting 3- and 4-Colorings

Fedor V. Fomin¹, Serge Gaspers¹, and Saket Saurabh^{1,2}

¹ Department of Informatics, University of Bergen,
N-5020 Bergen, Norway
{fomin,serge,saket}@ii.uib.no

² The Institute of Mathematical Sciences,
Chennai 600 113, India
saket@imsc.res.in

Abstract. We introduce a generic algorithmic technique and apply it on decision and counting versions of graph coloring. Our approach is based on the following idea: either a graph has nice (from the algorithmic point of view) properties which allow a simple recursive procedure to find the solution fast, or the pathwidth of the graph is small, which in turn can be used to find the solution by dynamic programming. By making use of this technique we obtain the fastest known exact algorithms

- running in time $\mathcal{O}(1.7272^n)$ for deciding if a graph is 4-colorable and
- running in time $\mathcal{O}(1.6262^n)$ and $\mathcal{O}(1.9464^n)$ for counting the number of k -colorings for $k = 3$ and 4 respectively.

1 Introduction

The graph coloring problem is one of the oldest and most intensively studied problems in Combinatorics and Algorithms. The problem is to color the vertices of a graph such that no two adjacent vertices are assigned the same color. The smallest number of colors needed to color a graph G is called the *chromatic number*, $\chi(G)$, of G . The corresponding decision version of the coloring problem is k -COLORING, where for a given graph G and an integer k we are asked if $\chi(G) \leq k$. The k -COLORING problem is one of the classical NP-complete problems [12]. In fact it is known to be NP complete for every $k \geq 3$. A lot of effort was also put in designing efficient approximation algorithms for the optimization version of the problem, namely, given a k -colorable graph to try to color it with as few colors as possible. Unfortunately, it has been shown that if certain reasonable complexity conjectures hold then k -COLORING is hard to approximate within $n^{1-\epsilon}$ for any $\epsilon > 0$ [10,13].

The history of exponential time algorithms for graph coloring is rich. Christofides obtained the first non-trivial algorithm computing the chromatic number of a graph on n vertices running in time $n!n^{O(1)}$ in 1971 [6]. In 1976, Lawler [15] devised an algorithm with running time $\mathcal{O}^*(2.4423^n)$ based on dynamic programming over subsets and enumeration of maximal independent sets. Eppstein [7]

reduced the bound to $\mathcal{O}(2.4151^n)$ and Byskov [5] to $\mathcal{O}(2.4023^n)$. In two breakthrough papers last year, Björklund & Husfeldt [3] and Koivisto [14] independently devised $2^n n^{\mathcal{O}(1)}$ algorithms based on a combination of inclusion-exclusion and dynamic programming.

Apart from the general chromatic number problem, the problem of k -COLORING for small values of k like 3, 4 has also attracted a lot of attention. The fastest algorithm deciding if the chromatic number of a graph is at most 3 runs in time $\mathcal{O}(1.3289^n)$ and is due to Beigel & Eppstein [2]. For 4-COLORING Byskov [5] designed the fastest algorithm, running in time $\mathcal{O}(1.7504^n)$.

The counting version of the k -COLORING problem, $\#k$ -COLORING, is to count the number of all possible k -colorings of a given graph. $\#k$ -COLORING (and its generalization known as *Chromatic Polynomial*) are among the oldest counting problem. Recently Björklund & Husfeldt [3] and Koivisto [14] have shown that the chromatic polynomial of a graph can be computed in time $2^n n^{\mathcal{O}(1)}$.

For small k , $\#k$ -COLORING was also studied in the literature. Angelsmark et al. [1] provide an algorithm for $\#3$ -COLORING with running time $\mathcal{O}(1.788^n)$. Fürer and Kashiviswanathan [11] show how to solve $\#3$ -COLORING with running time $\mathcal{O}(1.770^n)$. No algorithm faster than $2^n n^{\mathcal{O}(1)}$ for $\#4$ -COLORING was known in the literature [13, 11, 14].

Our Results. In this paper we introduce a generic technique to obtain exact algorithms for coloring problems and its different variants. This technique can be seen as a generalization of the technique introduced in [8] for a different problem. The technique is based on the following combinatorial property which is proved algorithmically and which is interesting in its own: *Either* a graph G has a nice “algorithmic” property which (very sloppily) means that when we apply branching or a recursive procedure to solve a problem then the branching procedure on subproblems of a smaller size works efficiently, *or* (if branching is not efficient) the pathwidth of the graph is small. This type of technique can be used for a variety of problems (not only coloring and its variants) where sizes of the subproblems on which the algorithm is called recursively decrease significantly by branching on vertices of high degrees.

In this paper we use this technique to obtain exact algorithms for different coloring problems. We show that $\#3$ -COLORING and $\#4$ -COLORING can be solved in time $\mathcal{O}(1.6262^n)$ and $\mathcal{O}(1.9464^n)$ respectively. We also solve 4-COLORING in time $\mathcal{O}(1.7272^n)$. These improve the best known results for each of the problems.

2 Preliminaries

In this paper we consider simple undirected graphs. Let $G = (V, E)$ be a graph and let n denote the number of vertices and m the number of edges of G . We denote by $\Delta(G)$ the maximum vertex degree in G . For a subset $V' \subseteq V$, $G[V']$ is the graph induced by V' , and $G - V' = G[V \setminus V']$. For a vertex $v \in V$ we denote the set of its neighbors by $N(v)$ and its *closed neighborhood* by $N[v] = N(v) \cup \{v\}$.

Similarly, for a subset $D \subseteq V$, we define $N[D] = \cup_{v \in D} N[v]$. An *independent set* in G is a subset of pair-wise non-adjacent vertices. A subset of vertices $S \subseteq V$ is a *vertex cover* in G if for every edge e of G at least one endpoint of e is in S .

Major tools of our paper are tree and path decompositions of graphs. A *tree decomposition* of G is a pair $(\{X_i : i \in I\}, T)$ where each X_i , $i \in I$, is a subset of V , called a *bag* and T is a tree with elements of I as nodes such that we have the following properties :

1. $\cup_{i \in I} X_i = V$;
2. for all $\{u, v\} \in E$, there exists $i \in I$ such that $\{u, v\} \subseteq X_i$;
3. for all $i, j, k \in I$, if j is on the path from i to k in T then $X_i \cap X_k \subseteq X_j$.

The width of a tree decomposition is equal to $\max_{i \in I} |X_i| - 1$. The *treewidth* of a graph G is the minimum width over all its tree decompositions and it is denoted by $\mathbf{tw}(G)$. We speak of a *path decomposition* when the tree T in the definition of a tree decomposition is restricted to be a path. The *pathwidth* of G is defined similarly to its *treewidth* and is denoted by $\mathbf{pw}(G)$.

We need the following bound on the pathwidth of graphs with small vertex degrees.

Proposition 1 ([8]). *For any $\varepsilon > 0$, there exists an integer n_ε such that for every graph G with $n > n_\varepsilon$ vertices,*

$$\mathbf{pw}(G) \leq \frac{1}{6}n_3 + \frac{1}{3}n_4 + \frac{13}{30}n_5 + \frac{23}{45}n_6 + n_{\geq 7} + \varepsilon n,$$

where n_i is the number of vertices of degree i in G for any $i \in \{3, \dots, 6, \geq 7\}$. Moreover, a path decomposition of the corresponding width can be constructed in polynomial time.

In our algorithms we also use the following results.

Proposition 2 ([7]). *The number of maximal independent sets of size k in a graph on n vertices is at most $3^{4k-n}4^{n-3k}$ and can be enumerated with polynomial time delay.*

Our \mathcal{O}^* notation suppresses polynomial terms. Thus we write $\mathcal{O}^*(T(x))$ for a time complexity of the form $\mathcal{O}(T(x) \cdot |x|^{\mathcal{O}(1)})$ where $T(x)$ grows exponentially with $|x|$, the input size.

3 Framework for Combining Enumeration and Pathwidth Arguments

Let us assume that we have a graph problem for which

- (a) we know how to solve it by enumerating independent sets, or maximal independent sets, of the input graph (for an example to check whether a graph G is 3-colorable, one can enumerate all independent sets I of G and for each independent set I can check whether $G - I$ is bipartite), and

- (b) we also know how to solve the problem using dynamic programming over the path decomposition of the input graph.

For some instances, the first approach might be faster and for other instances, the path decomposition algorithm might be preferable. One method to get the best of both algorithms would be to compute a path decomposition of the graph using Proposition [11](#), and choose one of the two algorithms based on the width of this path decomposition. Unfortunately, this method is not very helpful in obtaining better worst case bounds on the running time of the algorithm.

Here in our technique we start by enumerating (maximal) independent sets and based on the knowledge we gain on the graph by this enumeration step, we prove that either the enumeration algorithm is fast, or the pathwidth of the graph is small. This means that either the input graph has a good algorithmic property, or it has a good graph-theoretic property.

To enumerate (maximal) independent sets of the input graph G , we use a very standard approach. Two sets I and C are constructed by a recursive procedure, where I is the set of vertices in the independent set and C the set of vertices not in the independent set. Let v be a vertex of maximum degree in $G - I - C$, the algorithm makes one recursive call where it adds v to I and all its neighbors to C and another recursive call where it adds v to C . This branching into two subproblems decreases the number of vertices in $G - I - C$ according to the following recurrence

$$T(n) \leq T(n - d(v) - 1) + T(n - 1).$$

From this recurrence, we see that the running time of the algorithm depends on how often it branches on a vertex of high degree. This algorithmic property is reflected by the size of C : frequent branchings on vertices of high degree imply that $|C|$ grows fast (in one branch).

On the other hand we can exploit a graph-theoretic property if C is small and there are no vertices of high degree in $G - I - C$. Based on the work of Monien and Preis on the bisection width of 3-regular graphs [16](#), small upper bounds on the pathwidth of the input graph G depending on their maximum degree have been obtained [8,9](#) (also see Proposition [11](#)). If a path decomposition of $G - I - C$ of size $\beta_d|V(G) - I - C|$ can be computed, then a path decomposition of G of size $\beta_d|V(G) - I - C| + |C|$ can be computed easily. Here β_d is a constant strictly less than 1 depending on the maximum degree of the graph. If it turns out that a path decomposition of small width can be computed, the algorithm enumerating (maximal) independent sets is completely stopped without any further backtracking and an algorithm based on this path decomposition is executed on the original input graph.

In the rest of this section, we give a general framework combining

- algorithms based on the enumeration of maximal independent sets, and
- algorithms based on path decompositions of small width,

Input: A graph G , an independent set I of G and a set of vertices C such that $N(I) \subseteq C \subseteq V(G) - I$.

Output: An optimal solution which has the problem-dependent properties.

if $(\Delta(G - I - C) \geq a)$ *or*

$(\Delta(G - I - C) = a - 1$ *and* $|C| > \alpha_{a-1}|V(G)|)$ *or*

$(\Delta(G - I - C) = a - 2$ *and* $|C| > \alpha_{a-2}|V(G)|)$ *or*

\dots *or*

$(\Delta(G - I - C) = 3$ *and* $|C| > \alpha_3|V(G)|)$

then

 choose a vertex $v \in V(G) - I - C$ of maximum degree in $G - I - C$

$S_1 \leftarrow \text{enumISPw}(G, I \cup \{v\}, C \cup N(v))$

$S_2 \leftarrow \text{enumISPw}(G, I, C \cup \{v\})$

return **combine** (S_1, S_2)

R1

R2

else if $\Delta(G - I - C) = 2$ *and* $|C| > \alpha_2|V(G)|$ **then**

return **enumIS** (G, I, C)

else

 Stop this algorithm and **run** **Pw** (G, I, C) instead.

Fig. 1. Algorithm **enumISPw** (G, I, C)

and discuss the running time of the algorithms based on this framework. This framework is not problem-dependent and it relies on two black boxes that have to be replaced by appropriate procedures to solve a specific problem.

Algorithm **enumISPw** (G, I, C) in Figure 1 is invoked with the parameters $(G, \emptyset, \emptyset)$, where G is the input graph, and the algorithms **enumIS** and **Pw** are problem-dependent subroutines. The function **combine** takes polynomial time and it is also a problem-dependent subroutine. The values for a , α_a , \dots , and α_2 ($0 = \alpha_a \leq \alpha_{a-1} \leq \dots \alpha_2 < 1$) are carefully chosen constants to balance the time complexities of enumeration and path decomposition based algorithms and to optimize the overall running time of the combined algorithm.

Algorithm **enumIS** (G, I, C) is problem-dependent and returns an optimal solution *respecting* the choice for I and C , where I is an independent set and C is a set of vertices not belonging to the independent set (set of discarded vertices). The sets I and C are supposed to be completed into a (maximal) independent set and a (minimal) vertex cover for G by enumerating (maximal) independent sets of $G - I - C$, before the problem-specific treatment is done.

Algorithm **Pw** (G, I, C) first computes a path decomposition based on G, I and C and the maximum degree of $G - I - C$. It then calls a problem-dependent algorithm based on this path decomposition of G .

Let n denote the number of vertices of G , $T(n)$ be the running time of Algorithm **enumISPw** on G , $T_e(n, i, c)$ be the running time of Algorithm **enumIS** and $T_p(n, i, c)$ be the running time of Algorithm **Pw** with parameters G, I, C where $i = |I|$ and $c = |C|$. We also assume that for any graph with n vertices and maximum degree d , a path decomposition of width at most $\beta_d n$ can be computed.

The following lemma is used by Algorithm **Pw** to compute a path decomposition of G of small width.

Lemma 1. *Suppose that given a graph H with $\Delta(H) \leq d$, a path decomposition of width at most $\beta_d|H|$ can be computed where $\beta_d < 1$ is a constant depending on d alone. Then a path decomposition of width at most $\beta_d|V(G) - I - C| + |C|$ can be computed for a graph G if I is an independent set in G , $N(I) \subseteq C \subseteq V(G)$ and $\Delta(G - I - C) \leq d$.*

Proof. As I is an independent set in G and C separates I from $G - I - C$, every vertex in I has degree 0 in $G - C$. Thus, a path decomposition of $G - C$ of size at most $\beta_d|V(G) - I - C|$ can be computed. Adding C to each bag of this path decomposition gives a path decomposition of width at most $\beta_d|V(G) - I - C| + |C|$ of G . \square

Given the conditions under which **Pw** is executed, the following lemma upper bounds its running time.

Lemma 2. *If the considered problem can be solved on G in time $O^*(t_{pw}^\ell)$, given a path decomposition of width ℓ of G , then*

$$T_p(n, i, c) = O^* \left(\max_{d \in \{2, 3, \dots, a-1\}} \left(t_{pw}^{(\beta_d + (1-\beta_d)\alpha_d)n} \right) \right).$$

Proof. The proof follows from Lemma \square and the conditions on $|C|$ and $\Delta(G - I - C)$ under which Algorithm **Pw** is executed. \square

To estimate the size of the search tree we assume that Algorithm **Pw** is not executed. We denote $(\alpha_{d-1} - \alpha_d)n$ by $\Delta_{\alpha_d}n$. Let t_n, t_i and t_c be constants such that $T_e(n, i, c) = O^*(t_n^n t_i^i t_c^c)$. The next lemma bounds the size of the search tree when the algorithm based on path decomposition is not used.

Lemma 3. *If Algorithm **Pw** is not executed, then*

$$T(n) = O^* \left(t_n^n t_c^{2n} \prod_{d=3}^a t_d^{\Delta_{\alpha_d}n} \right)$$

where $t_d = (1 + r_d)$ and r_d is the minimum positive root of

$$(1 + r)^{-(d-1)} \cdot r^{-1} \cdot t_i - 1.$$

Proof. We divide $T(n)$ into $T_d(n, i, c)$ for $d \in \{2, 3, \dots, a\}$ where d corresponds to the maximum degree of $G - I - C$ if $d < a$ and $T_a(n, 0, 0) = T(n)$ if Algorithm **Pw** is not executed. Clearly, $T_2(n, i, c) = T_e(n, i, c)$. Let us now express $T_d(n, i, c)$ in terms of $T_{d-1}(\cdot, \cdot, \cdot)$ for $d \in \{3, \dots, a\}$. Consider the part of the search tree with branchings on vertices of degree d (or at least d if $d = a$). Observe that

$|C|$ increases in the worst case by at most $(\alpha_{d-1} - \alpha_d)n = \Delta\alpha_d n$ in this part of the search tree. In each branch of the type **R1**, $|C|$ increases by d and in each branch of the type **R2**, $|C|$ increases by 1. Let $r \in [0, \Delta\alpha_d n/d]$ be the number of times the algorithm branches according to **R1**, then it branches $\Delta\alpha_d n - dr$ times according to **R2**. We get that

$$T_d(n, i, c) = O^* \left(\sum_{r=0}^{\Delta\alpha_d n/d} \binom{\Delta\alpha_d n - (d-1)r}{r} T_{d-1}(n, i+r, c + \Delta\alpha_d n) \right).$$

In general situation the degree d may not change to $d-1$ but rather jump to something smaller. But the worst case bounds on the size of the search tree are achieved when d decreases progressively as considered above.

To prove the lemma, it is sufficient to expand $T_a(n, 0, 0)$ and to prove that

$$\sum_{r=0}^{\Delta\alpha_d n/d} \binom{\Delta\alpha_d n - (d-1)r}{r} t_i^r \leq t_d^{\Delta\alpha_d n}.$$

The sum over binomial coefficients $\sum_{r=0}^{\Delta\alpha_d n/d} \binom{\Delta\alpha_d n - (d-1)r}{r} t_i^r$ is bounded by $(\Delta\alpha_d n/d)B$ where B is the maximum term in this sum. Let us assume that $B = \binom{\Delta\alpha_d n - (d-1)j}{j} t_i^j$ for some $j \in \{0, 1, \dots, \Delta\alpha_d n/d\}$.

$$B = \binom{\Delta\alpha_d n - (d-1)j}{j} t_i^j \leq \frac{(1+r_i)^{\Delta\alpha_d n - (d-1)j}}{r_i^j} t_i^j.$$

Here we use the well known fact that for any $x > 0$ and $0 \leq k \leq n$,

$$\binom{n}{k} \leq \frac{(1+x)^n}{x^k}.$$

By choosing r_i to be the minimum positive root of $\frac{(1+r)^{-(d-1)}}{r} t_i - 1$, we arrive at $B < (1+r_i)^{\Delta\alpha_d n} = t_d^{\Delta\alpha_d n}$. \square

The following theorem combines Lemmas [2](#) and [3](#) to upper bound the overall running time of the algorithms resulting from this framework.

Theorem 1. *The running time of Algorithm `enumISPw` on a graph on n vertices is*

$$T(n) = O^* \left(t_n^n t_c^{\alpha_2 n} \prod_{d=3}^a t_d^{\Delta\alpha_d n} + \max_{d \in \{2, 3, \dots, a-1\}} \left(t_{pw}^{(\beta_d + (1-\beta_d)\alpha_d)n} \right) \right)$$

where $t_d = (1+r_d)$ and r_d is the minimum positive root of

$$(1+r)^{-(d-1)} \cdot r^{-1} \cdot t_i - 1.$$

The current best values for $\beta_d, d = \{2, \dots, 6\}$ are obtained from Proposition [11](#).

4 Applications

In this section we use the framework of the previous section to derive improved algorithms for #3-COLORING, #4-COLORING and 4-COLORING.

4.1 Counting 3-Colorings

We first describe the problem-dependent subroutines we need to use in our Algorithm `enumISPw` to solve #3-COLORING in time $\mathcal{O}(1.6262^n)$.

Algorithm `enumISPw` returns here an integer, I corresponds to the color class C_1 and C to the remaining two color classes C_2 and C_3 . Algorithm `enumIS` with parameters G, I, C enumerates all independent sets of $G - I - C$ and for each, adds this independent set to I , then checks if $G - I$ is bipartite. If $G - I$ is bipartite, then a counter counting the independent sets is incremented. This takes time $T_e(n, i, c) = 2^{n-i-c}$. Thus, $t_n = 2, t_i = 1/2$ and $t_c = 1/2$.

The function `combine` corresponds in this case to the plus-operation. The running time of Algorithm `Pw` is based on the following lemma.

Lemma 4. *Given a graph $G = (V, E)$ with a path decomposition of G of width ℓ , # k -COLORING can be solved in time $\mathcal{O}(k^\ell n^{\mathcal{O}(1)})$.*

Now we use Theorem [1](#) and Proposition [1](#) to evaluate the overall complexity of our #3-COLORING algorithm.

Theorem 2. *The #3-COLORING problem can be solved in time $\mathcal{O}(1.6262^n)$ for a graph on n vertices.*

Proof. We use Theorem [1](#) and Lemma [1](#) with $\mathbf{a} = 5, \alpha_2 = 0.44258, \alpha_3 = 0.33093$ and $\alpha_4 = 0.16387$. The pathwidth part of the algorithm takes time

$$\begin{aligned} O^* \left(\max \left(3^{\alpha_2 n}, 3^{(1+5\alpha_3)n/6}, 3^{(1+2\alpha_4)n/3} \right) \right) \\ = \mathcal{O}(1.62617^n). \end{aligned}$$

The branching part of the algorithm takes time

$$\begin{aligned} O^* \left(2^n \cdot (1/2)^{\alpha_2 n} \cdot 1.29716^{(\alpha_2 - \alpha_3)n} \cdot 1.25373^{(\alpha_3 - \alpha_4)n} \cdot 1.22329^{\alpha_4 n} \right) \\ = \mathcal{O}(1.62617^n). \quad \square \end{aligned}$$

4.2 Counting 4-Colorings

To solve #4-COLORING, Algorithm `enumIS` with parameters G, I, C enumerates all independent sets of $G - I - C$ and for each, adds this independent set to I , then counts the number of 3-colorings of $G - I$ using the previous algorithm. This takes time $T_e(n, i, c) = \sum_{\ell=0}^{n-i-c} 1.62617^{\ell+c}$. Thus, $t_n = 2.62617, t_i = 1/2.62617$ and $t_c = 1.62617/2.62617$. We evaluate the running time as previously.

Theorem 3. *The #4-COLORING problem can be solved in time $\mathcal{O}(1.9464^n)$ for a graph on n vertices.*

Proof. We use Theorem [1](#) and Lemma [1](#) with $\mathbf{a} = 6, \alpha_2 = 0.480402, \alpha_3 = 0.376482, \alpha_4 = 0.220602$ and $\alpha_5 = 0.083061$. The pathwidth part of the algorithm takes time

$$\begin{aligned} O^* \left(\max \left(4^{\alpha_2 n}, 4^{(1+5\alpha_3)n/6}, 4^{(1+2\alpha_4)n/3}, 4^{(13+17\alpha_5)n/30} \right) \right) \\ = \mathcal{O}(1.9464^n). \end{aligned}$$

The branching part of the algorithm takes time

$$\begin{aligned} O^* \left(2.62617^n \cdot (1.62617/2.62617)^{\alpha_2 n} \cdot 1.24548^{(\alpha_2 - \alpha_3)n} \cdot 1.21324^{(\alpha_3 - \alpha_4)n} \cdot \right. \\ \left. 1.18993^{(\alpha_4 - \alpha_5)n} \cdot 1.17212^{\alpha_5 n} \right) = \mathcal{O}(1.9464^n). \quad \square \end{aligned}$$

4.3 4-Coloring

A well known technique [\[15\]](#) to check if a graph is k -colorable is to check for all maximal independent sets I of size at least $\lceil n/k \rceil$ whether $G - I$ is $(k - 1)$ -colorable. In the analysis, we use Proposition [2](#) to bound the number of maximal independent sets of a given size.

We also need the current best algorithm deciding 3-COLORING.

Theorem 4 ([\[2\]](#)). *The 3-COLORING problem can be solved in time $\mathcal{O}(1.3289^n)$ for a graph on n vertices.*

In Algorithm `enumISPw`, which returns here a boolean, I corresponds to the color class C_1 and C to the remaining three color classes C_2, C_3 and C_4 . Algorithm `enumIS` with parameters G, I, C enumerates all maximal independent sets of $G - I - C$ of size at least $\lceil n/k \rceil - |I|$ and for each, adds this independent set to I , then checks if $G - I$ is 3-colorable using Theorem [4](#). If yes, then G is 4-colorable. This takes time

$$T_e(n, i, c) = \sum_{\ell=\lceil n/4 \rceil - i}^{n-i-c} 3^{4\ell-n+c+i} 4^{n-c-i-3\ell} 1.3289^{n-i-\ell}.$$

As $\sum_{\ell=0}^{\lfloor 3n/4 \rfloor - c} 3^{4\ell} 4^{-3\ell} 1.3289^{-\ell}$ is upper bounded by a constant, $t_n = 4^{1/4} 1.3289^{3/4}$, $t_i = 4^2/3^3$ and $t_c = 3/4$.

Theorem 5. *The 4-COLORING problem can be solved in time $\mathcal{O}(1.7272^n)$ for a graph on n vertices.*

Proof. We use Theorem [1](#) and Lemma [1](#) with $\mathbf{a} = 5, \alpha_2 = 0.39418, \alpha_3 = 0.27302$ and $\alpha_4 = 0.09127$, and the pathwidth algorithm of Lemma [4](#). \square

References

1. Angelsmark, O., Jonsson, P.: Improved Algorithms for Counting Solutions in Constraint Satisfaction Problems. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 81–95. Springer, Heidelberg (2003)
2. Beigel, R., Eppstein, D.: 3-coloring in time $O(1.3289^n)$. Journal of Algorithms 54(2), 168–204 (2005)
3. Björklund, A., Husfeldt, T.: Inclusion–Exclusion Algorithms for Counting Set Partitions. In: The Proceedings of FOCS 2006, pp. 575–582 (2006)
4. Byskov, J.M.: Exact Algorithms for Graph Colouring and Exact Satisfiability. PhD Dissertation (2004)
5. Byskov, J.M.: Enumerating Maximal Independent Sets with Applications to Graph Colouring. Operations Research Letters 32(6), 547–556 (2004)
6. Christofides, N.: An algorithm for the chromatic number of a graph. Computer J. 14, 38–39 (1971)
7. Eppstein, D.: Small Maximal Independent Sets and Faster Exact Graph Coloring. J. Graph Algorithms Appl. 7(2), 131–140 (2003)
8. Fomin, F.V., Gaspers, S., Saurabh, S., Stepanov, A.A.: On Two Techniques of Combining Branching and Treewidth. Report No 337, Department of Informatics, University of Bergen, Norway (December 2006)
9. Fomin, F.V., Høie, K.: Pathwidth of cubic graphs and exact algorithms. Information Processing Letters 97(5), 191–196 (2006)
10. Feige, U., Kilian, J.: Zero Knowledge and the Chromatic Number. Journal of Computer and System Sciences 57(2), 187–199 (1998)
11. Fürer, M., Kasiviswanathan, S.P.: Algorithms for counting 2-SAT solutions and colorings with applications. In: ECCV 33 (2005)
12. Garey, M.R., Johnson, D.S.: Computer and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, San Francisco, CA (1979)
13. Khot, S., Ponnuswami, A.K.: Better Inapproximability Results for MaxClique, Chromatic Number and Min-3Lin-Deletion. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 226–237. Springer, Heidelberg (2006)
14. Koivisto, M.: An $O(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion-exclusion. In: Proceedings of FOCS 2006, pp. 583–590 (2006)
15. Lawler, E.L.: A Note on the Complexity of the Chromatic Number. Information Processing Letters 5(3), 66–67 (1976)
16. Monien, B., Preis, R.: Upper bounds on the bisection width of 3- and 4-regular graphs. Discrete Algorithms 4(3), 475–498 (2006)

Connected Coloring Completion for General Graphs: Algorithms and Complexity*

Benny Chor¹, Michael Fellows^{2,3}, Mark A. Ragan⁴, Igor Razgon⁵,
Frances Rosamond², and Sagi Snir⁶

¹ Computer Science Department, Tel Aviv University, Tel Aviv, Israel
benny@cs.tau.ac.il

² University of Newcastle, Callaghan NSW 2308, Australia
{michael.fellows, frances.rosamond}@newcastle.edu.au

³ Durham University, Institute of Advanced Study,
Durham DH1 3RL, United Kingdom

⁴ Institute for Molecular Biosciences, University of Queensland, Brisbane, QLD 4072 Australia
m.ragan@imb.uq.edu.au

⁵ Computer Science Department, University College Cork, Ireland
i.razgon@cs.ucc.ie

⁶ Department of Mathematics, University of California, Berkeley, USA
ssagi@math.berkeley.edu

Abstract. An r -component connected coloring of a graph is a coloring of the vertices so that each color class induces a subgraph having at most r connected components. The concept has been well-studied for $r = 1$, in the case of trees, under the rubric of *convex coloring*, used in modeling perfect phylogenies. Several applications in bioinformatics of connected coloring problems on general graphs are discussed, including analysis of protein-protein interaction networks and protein structure graphs, and of phylogenetic relationships modeled by splits trees. We investigate the r -COMPONENT CONNECTED COLORING COMPLETION (r -CCC) problem, that takes as input a partially colored graph, having k uncolored vertices, and asks whether the partial coloring can be completed to an r -component connected coloring. For $r = 1$ this problem is shown to be NP-hard, but fixed-parameter tractable when parameterized by the number of uncolored vertices, solvable in time $O^*(8^k)$. We also show that the 1-CCC problem, parameterized (only) by the treewidth t of the graph, is fixed-parameter tractable; we show this by a method that is of independent interest. The r -CCC problem is shown to be $W[1]$ -hard, when parameterized by the treewidth bound t , for any $r \geq 2$. Our proof also shows that the problem is NP-complete for $r = 2$, for general graphs.

Topics: Algorithms and Complexity, Bioinformatics.

1 Introduction

The following two problems concerning colored graphs can be used to model several different issues in bioinformatics.

* This research has been supported by the Australian Research Council through the Australian Centre in Bioinformatics. The second and fifth authors also acknowledge the support provided by a William Best Fellowship at Grey College, Durham, while the paper was in preparation.

r-COMPONENT CONNECTED RECOLORING (*r*-CCR)

Instance: A graph $G = (V, E)$, a set of colors \mathcal{C} , a coloring function $\Gamma : V \rightarrow \mathcal{C}$, and a positive integer k .

Parameter: k

Question: Is it possible to modify Γ by changing the color of at most k vertices, so that the modified coloring Γ' has the property that each color class induces a subgraph with at most r components?

In the case where G is a tree and $r = 1$, the problem is of interest in the context of maximum parsimony approaches to phylogenetics [17][13]. A connected coloring corresponds to a perfect phylogeny, and the recoloring number can be viewed as a measure of distance from perfection. The problem was introduced by Moran and Snir, who showed that CONVEX RECOLORING FOR TREES (which we term 1-CCR) is NP-hard, even for the restriction to colored paths. They also showed that the problem is fixed-parameter tractable, and described an FPT algorithm that runs in time $O(k(k/\log k)^k n^4)$ for colored trees [17]. Subsequently, Bodlaender *et al.* have improved this to an FPT algorithm that runs in linear time for every fixed k , and have described a polynomial-time kernelization to a colored tree on at most $O(k^2)$ vertices [3].

Here we study a closely related problem.

r-COMPONENT CONNECTED COLORING COMPLETION (*r*-CCC)

Instance: A graph $G = (V, E)$, a set of colors \mathcal{C} , a coloring partial function $\Gamma : V \rightarrow \mathcal{C}$ where there are k uncolored vertices.

Parameter: k

Question: Is it possible to complete Γ to a total coloring function Γ' such that each color class induces a subgraph with at most r components?

The problem is of interest in the following contexts.

(1) Protein-protein interaction networks. In a protein-protein interaction network the vertices represent proteins and edges model interactions between that pair of proteins [22][7][20][21]. Biologists are interested in analyzing such relationship graphs in terms of cellular location or function (either of which may be represented by vertex coloring) [16]. Interaction networks colored by cellular location would be expected to have monochrome subgraphs representing localized functional subnetworks. Conversely, interaction networks colored by function may also be expected to have monochrome connected subgraphs representing cellular localization. The issue of error makes the number of recolorings (corrections) needed to attain color-connectivity of interest [17], and the issue of incomplete information may be modeled by considering uncolored vertices that are colored to attain color-connectivity, the main combinatorial problem we are concerned with here. Protein-protein interaction graphs generally have bounded treewidth.

(2) Phylogenetic networks. Phylogenetic relationships can be represented not only as trees, but also as networks, as in the *splits trees* models of phylogenetic relationships that take into account such issues as evidence of lateral genetic transfer, inconsistencies

in the phylogenetic signal, or information relevant to a specific biological hypothesis, e.g., host-parasite relationships [15,14]. Convex colorings of splits trees have essentially the same modeling uses and justifications as in the case of trees [17,4,13]. Splits trees for natural datasets have small treewidth.

Our main results are summarized as follows:

1. 1-CCC is NP-hard for general colored graphs, even if there are only two colors.
2. 1-CCC for general colored graphs, parameterized by the number k of uncolored vertices, is fixed-parameter tractable, and can be solved in time $O^*(8^k)$.
3. 1-CCC is in XP for colored graphs of treewidth at most t , parameterized by t . (That is, it is solvable in polynomial time for any fixed t . Note that under this parameterization the number of uncolored vertices is unbounded.)
4. 1-CCC is fixed-parameter tractable when parameterized by treewidth.
5. For all $r \geq 2$, r -CCC, parameterized by a treewidth bound t , is hard for $W[1]$.

For basic background on parameterized complexity see [10,11,19].

2 Connected Coloring Completion Is NP-Hard

Theorem 1. *The 1-CCC problem is NP-hard, even if there are only two colors.*

Proof. (Sketch.) The reduction from 3SAT can be inferred from Figure 1 (the details are omitted due to space limitations). The two colors are T and F.

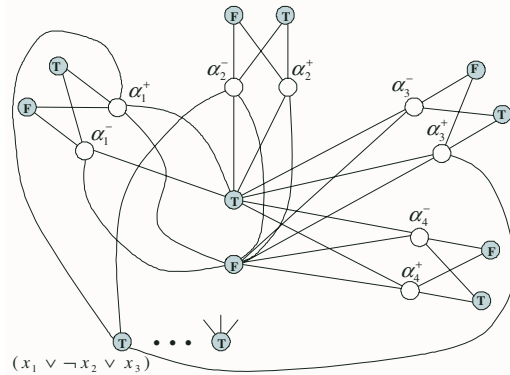


Fig. 1. The reduction from 3SAT

3 1-CCC for k Uncolored Vertices Is Fixed-Parameter Tractable

The input to the problem is a partially colored graph $G = (V, E)$, and the parameter is the number of uncolored vertices.

Soundness for the following reduction rules is easy to verify.

Rule 1. A maximal connected monochromatic subgraph (of colored vertices) can be collapsed to a single vertex. The parameter is unchanged.

Rule 2. If a color occurs on only a single vertex, then that vertex can be deleted. The parameter is unchanged.

Rule 3. An edge between two colored vertices of different color can be deleted. The parameter is unchanged.

Suppose that a partially colored graph G is reduced with respect to the above three reduction rules. The situation can be represented by a bipartite *model graph* that on one side (let us say, the *left side*), has vertices representing the vertices created by Rule 1, but not deleted by Rule 2. On the *right side* are the k uncolored vertices (and their adjacencies), and between the two sides are edges that represent an incidence relationship. Clearly, if in this representation, there are more than k vertices on the left, then the answer is NO. Thus, there are at most k colors represented on the left, and an FPT algorithm that runs in time $O^*(k^k)$ follows by exploring all possibilities of coloring the k uncolored vertices (on the right) with the k colors represented on the left of the model. We can do better than this.

Theorem 2. *1-CCC parameterized by the number of uncolored vertices is fixed-parameter tractable, solvable in time $O^*(8^k)$.*

Proof. We use the *model graph* described above. Instead of the brute-force exploration of all possibilities of coloring the right-side vertices with the left-side colors, we employ a dynamic programming algorithm. Let H denote the set of at most k uncolored vertices, and let \mathcal{C} denote the set of colors represented on the left of the model graph. Create a table of size $2^{\mathcal{C}} \times 2^H$. We employ a table T to be filled in with 0/1 entries. The entry $T(\mathcal{C}', H')$ of the table T indexed by (\mathcal{C}', H') , where $\mathcal{C}' \subseteq \mathcal{C}$ and $H' \subseteq H$, represents whether (1) or not (0), it is possible to solve the connectivity problem for the colors in \mathcal{C}' by assigning colors to the vertices of H' . (The only way this can happen, is that, for each color $c \in \mathcal{C}'$, the (single) connected component of c -colored vertices on the right, dominates all the c -colored vertices on the left.) We have the following recurrence relationship for filling in this table:

$$T(\mathcal{C}', H') = 1 \iff \exists(c, H''), c \in \mathcal{C}', H'' \subset H', \text{ such that } T(\mathcal{C}' - c, H'') = 1$$

and $H' - H''$ induces a connected subgraph that dominates the vertices of color c . The table T has at most $2^k \times 2^k = 4^k$ entries, and computing each entry according to the recurrence requires time at most $O(k \cdot 2^k)$, so the total running time of the dynamic programming algorithm is $O^*(8^k)$.

4 Bounded Treewidth

Most natural datasets for phylogenetics problems have small bounded treewidth. 1-CCR is NP-hard for paths (and therefore, for graphs of treewidth one) [17]. Bodlaender

and Weyer have shown that 1-CCR parameterized by (k, t) , where k is the number of vertices to be recolored, and t is a treewidth bound, is fixed-parameter tractable, solvable in linear time for all fixed k [6].

4.1 1-CCC Parameterized by Treewidth is Linear-Time FPT

We describe an algorithm for the 1-CCC problem that runs in linear time for any fixed treewidth bound t , and we do this by using the powerful machinery of Monadic Second Order (MSO) logic, due to Courcelle [9] (also [15]). At first glance, this seems either surprising or impossible, since MSO does not provide us with any means of describing families of colored graphs, where the number of colors is unbounded. We employ a “trick” that was first described (to our knowledge) in a paper in these proceedings [3]. Further applications of what appears to be more a useful new method, rather than just a trick, are described in [12].

The essence of the trick is to construct an auxiliary graph that consists of the original input, augmented with additional *semantic vertices*, so that the whole ensemble has — or can safely be assumed to have — bounded treewidth, and relative to which the problem of interest *can* be expressed in MSO logic.

Let $G = (V, E)$ be a graph of bounded treewidth, and $\Gamma : V' \rightarrow \mathcal{C}$ a vertex-coloring function defined on a subset $V' \subseteq V$. (Assume each color in \mathcal{C} is used at least once.) We construct an auxiliary graph G' from G in the following way: for each color $c \in \mathcal{C}$, create a new *semantic vertex* v_c (these are all of a second *type* of vertex, the vertices of V are of the first type). Connect v_c to every vertex in G colored c by Γ .

Consider a tree decomposition Δ for G , witnessing the fact that it has treewidth at most t . This can be computed in linear time by Bodlaender’s algorithm.

Say that a color $c \in \mathcal{C}$ is *relevant* for a bag B of Δ if either of the following holds:

- (1) There is a vertex $u \in B$ such that $\Gamma(u) = c$. (When this holds, say that c is *present* in B .)
- (2) There are bags B' and B'' of Δ such that B is on the unique path from B' to B'' relative to the tree that indexes Δ , and there are vertices $u' \in B'$ and $u'' \in B''$ such that $\Gamma(u') = \Gamma(u'') = c$, and furthermore, c is not present in B . (When this holds, say that c is *split* by the bag B .)

Lemma 1. *If the colored graph G is a yes-instance for 1-CCC, then for any bag B , there are at most $t + 1$ relevant colors.*

Proof. Suppose that a bag B has more than $t + 1$ relevant colors, and that p of these are present in B . If s denotes the number of colors split by B , then $s > t + 1 - p$. Since B contains at most $t + 1$ vertices, the number of colors split by B exceeds the number of uncolored vertices in B , and because each bag is a cutset of G , it follows that G is a no-instance for 1-CCC.

Lemma 2. *If the colored graph G is a yes-instance for 1-CCC, then the auxiliary graph G' has treewidth at most $2t + 1$.*

Proof. Consider a tree decomposition Δ for G witnessing that the treewidth of G is at most t . By the above lemma, if we add to each bag B of Δ all those vertices v_c for

colors c that are relevant to B , then (it is easy to check) we obtain a tree-decomposition Δ' for G' of treewidth at most $2t + 1$.

Theorem 3. *The 1-CCC problem, parameterized by the treewidth bound t , is fixed-parameter tractable, solvable in linear time for every fixed t .*

Proof. The algorithm consists of the following steps.

Step 1. Construct the auxiliary graph G' .

Step 2. Compute in linear time, using Bodlaender's algorithm, a tree-decomposition for G' of width at most $2t + 1$, if one exists. (If not, then correctly output NO.)

Step 3. Otherwise, we can express the problem in MSO logic. That this is so, is not entirely trivial, and is argued as follows (sketch).

The vertices of G' can be considered to be of three types: (i) the original colored vertices of G (that is, the vertices of V'), (ii) the uncolored vertices of G (that is, the vertices of $V - V'$), and (iii) the color-semantic vertices added in the construction of G' . (The extension of MSO Logic to accomodate a fixed number of vertex types is routine.)

If G is a yes-instance for the problem, then this is witnessed by a set of edges F between vertices of G (both colored and uncolored) that provides the connectivity for the color classes. In fact, we can choose such an F so that it can be partitioned into classes F_c , one for each color $c \in \mathcal{C}$, such that the classes are disjoint: no vertex $v \in V$ has incident edges $e \in F_c$ and $e' \in F_{c'}$ where $c \neq c'$.

The following are the key points of the argument:

(1) Connectivity of a set of vertices, relative to a set of edges, can be expressed by an MSO formula.

(2) We assert the existence of a set of edges F of $G \subseteq G'$, and of a set of edges F' between uncolored vertices of G and color-semantic vertices of G' such that:

- Each uncolored vertex of G has degree 1 relative to F' . (The edges of F' thus represent a coloring of the uncolored vertices of G .)
- If u and v are colored vertices of G that are connected relative to F , then there is a unique color-semantic vertex v_c such that both u and v are adjacent to v_c .
- If u is a colored vertex of G and v is an uncolored vertex of G that are connected via edges in F , then there is a unique color-semantic vertex v_c such that v is adjacent to v_c by an edge of F' , and u is adjacent to v_c by an edge of G' .
- If u is an uncolored vertex of G and v is an uncolored vertex of G that are connected via edges in F , then there is a unique color-semantic vertex v_c such that v is adjacent to v_c by an edge of F' , and u is adjacent to v_c by an edge of F' .
- If u and v are colored vertices of G that are both adjacent to some color-semantic vertex v_c , then u and v are connected relative to F .

4.2 r -CCC Parameterized by Treewidth is $W[1]$ -Hard for $r \geq 2$

In view of the fact that 1-CCC is fixed-parameter tractable for bounded treewidth, it may be considered surprising that this does not generalize to r -CCC for any $r \geq 2$.

Theorem 4. *The 2-CCC problem, parameterized by the treewidth bound t , is hard for $W[1]$.*

Proof. (Sketch.) The proof is an FPT Turing reduction, based on color-coding [2]. We reduce from the $W[1]$ -hard problem of k -CLIQUE. Let $(G = (V, E), k)$ be an instance of the parameterized CLIQUE problem. Let \mathcal{H} be a suitable family of hash functions $h : V \rightarrow \mathcal{A} = \{1, \dots, k\}$.

If G is a yes-instance for the k -CLIQUE problem, then for at least one $h \in \mathcal{H}$, the coloring function h is injective on the vertices of a witnessing k -clique in G (that is, each vertex of the k -clique is assigned a different color).

We describe a Turing reduction to instances $G'(h)$ of the 2-CCC problem, one for each $h \in \mathcal{H}$, such that G is a yes-instance for the k -CLIQUE problem if and only at least one $G'(h)$ is a yes-instance for the 2-CCC problem. Each $G'(h)$ has treewidth $t = O(k^2)$.

The construction of $G'(h)$ is based on an *edge-representation of the clique* strategy. We will describe the construction of $G'(h)$ in stages, building up in a modular fashion. A *module* of the construction will be a subgraph that occurs in $G'(h)$ as an induced subgraph, except for a specified set of boundary vertices of the module. These boundary sets will be identified as the various modules are “plugged together” to assemble $G'(h)$. In the figures that illustrate the construction, each kind of module is represented by a symbolic *schematic*, and the modules are built up in a hierarchical fashion. Square vertices represent uncolored vertices.

Figure 2 illustrates the *Choice Module* that is a key part of our construction, and its associated schematic representation. A Choice Module has four “output” boundary vertices, labeled c_1, \dots, c_4 in the figure. It is easy to see that the module admits a partial solution coloring that “outputs” any one of the (numbered) colors occurring in the module depicted, in the sense that the vertices c_1, \dots, c_4 are assigned the output color (which is *unsolved*, that is, this color class is not connected in the module), and that the other colors are all solved internally to the module, in the sense that there is (locally) only one connected component of the color class.

A *Co-Incident Edge Set Module* is created from the disjoint union of $k - 1$ Choice Modules, as indicated in in Figure 3. The boundary of the module is the union of the boundaries of the constituent Choice Modules.

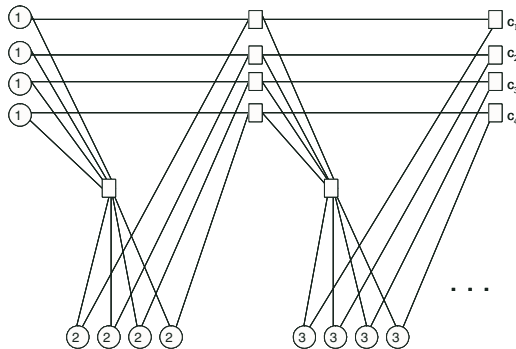


Fig. 2. A Choice Module of size 3

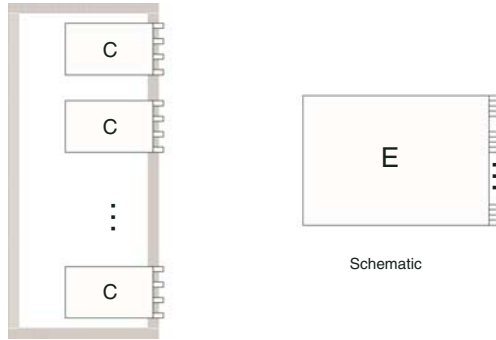


Fig. 3. A Co-Incident Edge Choice Module of size s is the disjoint union of s Choice Modules

An *XOR Stream Module* is shown in Figure 4. This has two “input” boundary sets, each consisting of four uncolored vertices, and one “output” boundary set of four uncolored vertices.

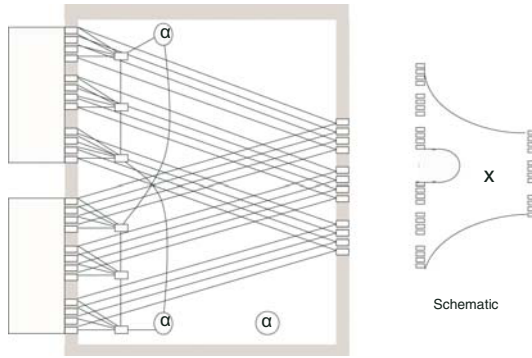


Fig. 4. An XOR Stream Module

Figure 5 shows how a *Tree of Choice Module* is assembled from Co-Incident Edge Set modules and XOR Stream modules. By the *size* of a Tree of Choice module we refer to the number of Co-Incident Edge Set modules occurring as leaves in the construction.

The overall construction of $G'(h)$ for $k = 5$ is illustrated in Figure 6. Suppose the instance graph $G = (V, E)$ that is the source of our reduction from CLIQUE has $|V| = n$ and $|E| = m$. Then each Tree of Choice module $T(h, i)$ has size $n(h, i)$, where $n(h, i)$

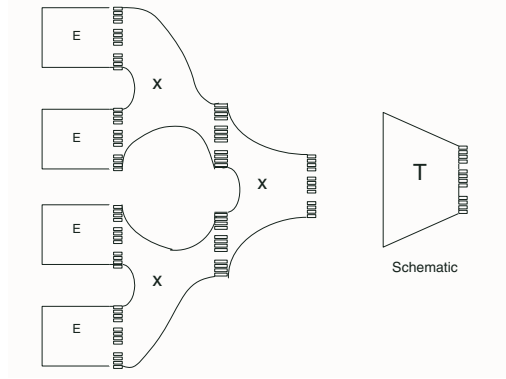


Fig. 5. A Tree of Choice Module of size 4

is the number of vertices colored $i \in \{1, \dots, k\}$ by h . Let $V(h, i)$ denote the subset of vertices of V colored i by h .

In the coloring of G by h , if it should happen that a vertex v colored i has no neighbors of color $j, j \neq i$, then v cannot be part of a multicolored k -clique in G , and can be deleted. We consider only colorings of G that are *reduced* in this sense.

The “leaves” of $T(h, i)$ consist of Co-Incident Edge Set modules $E(h, i, u)$, one for each $u \in V(h, i)$. The Co-Incident Edge Set module $E(h, i, u)$ consists of $k - 1$ Choice modules $C(h, i, u, j), j \in \{1, \dots, k\}$ and $j \neq i$, and each of these has size equal to the number of edges uv incident to u in G , where v is colored j by h .

The colors used in the construction of $E(h, i, u)$ are in 1:1 correspondence with the edges incident to v in G . Overall, the *colors* of the colored vertices in G' occurring in the Choice modules represent, in this manner, *edges* of G . Each XOR module M has three vertices colored $\alpha = \alpha(M)$ where this color occurs nowhere else in G' (see Figure 4). One of these vertices colored α is an isolated vertex.

Verification that if G has a k -clique, then G' admits a solution to the CCC problem is relatively straightforward. It is important to note that if uv is an edge of G , where $h(u) = i$ and $h(v) = j$, then the color corresponding to uv occurs in exactly two Choice modules: in $C(h, i, u, j)$ and in $C(h, j, v, i)$. If uv is “not selected” (with respect to a coloring completion) then the two local connectivities yield two components of that color.

The argument in the other direction is a little more subtle. First of all, one should verify that the gadgets enforce some restrictions on any solution for G' :

- (1) Each Choice module necessarily “outputs” one unsolved color (that is, a color not connected into a single component locally), and thus a Co-Incident Edge Set module $E(v)$ “outputs” $k - 1$ colors representing edges incident on v .
- (2) Each XOR module forces the “output” stream of unsolved colors to be one, or the other, but not a mixture, of the two input streams of unsolved colors.

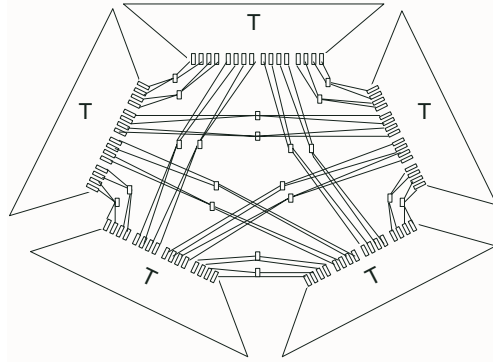


Fig. 6. The modular design of G' for $k = 5$

(3) The unsolved colors that are presented to the central gadget (see Figure 7) can be solved only if these unsolved colors occur in pairs.

The treewidth of G' is easily seen to be $O(k^2)$.

References

1. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. *J. Algorithms* 12, 308–340 (1991)
2. Alon, N., Yuster, R., Zwick, U.: Color-coding. *Journal of the ACM* 42, 844–856 (1995)
3. Bodlaender, H.L., Fellows, M., Langston, M., Ragan, M.A., Rosamond, F., Weyer, M.: Quadratic kernelization for convex recoloring of trees. *Proceedings COCOON 2007, these proceedings* (2007)
4. Bar-Yehuda, R., Feldman, I., Rawitz, D.: Improved approximation algorithm for convex recoloring of trees. In: Erlebach, T., Persinao, G. (eds.) *WAOA 2005*. LNCS, vol. 3879, pp. 55–68. Springer, Heidelberg (2006)
5. Borie, R.B., Parker, R.G., Tovey, C.A.: Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively generated graph families. *Algorithmica* 7, 555–581 (1992)
6. Bodlaender, H.L., Weyer, M.: Convex and connected recolourings of trees and graphs. *Manuscript* (2005)
7. Bu, D., Zhao, Y., Cai, L., Xue, H., Zhu, X., Lu, H., Zhang, J., Sun, S., Ling, L., Zhang, N., Li, G., Chen, R.: Topological structure analysis of the protein-protein interaction network in budding yeast. *Nucleic Acids Res.* 31(9), 2443–2450 (2003)
8. Chen, J., Chor, B., Fellows, M., Huang, X., Juedes, D., Kanj, I., Xia, G.: Tight lower bounds for certain parameterized NP-hard problems. In: *Proceedings of the 19th Annual IEEE Conference on Computational Complexity*, pp. 150–160. IEEE Computer Society Press, Los Alamitos (2004)
9. Courcelle, B.: The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation* 85, 12–75 (1990)
10. Downey, R.G., Fellows, M.R.: *Parameterized Complexity*. Springer, Heidelberg (1999)

11. Flum, J., Grohe, M.: *Parameterized Complexity Theory*. Springer, Heidelberg (2006)
12. Fellows, M., Giannopoulos, P., Knauer, C., Paul, C., Rosamond, F., Whitesides, S., Yu, N.: The lawnmower and other problems: applications of MSO logic in geometry, Manuscript (2007)
13. Gramm, J., Nickelsen, A., Tantau, T.: Fixed-parameter algorithms in phylogenetics. Manuscript (2006)
14. Huson, D.H., Bryant, D.: Application of phylogenetic networks in evolutionary studies. *Mol. Biol. E* 23, 254–267 (2006)
15. Huson, D.H.: SplitsTree: a program for analyzing and visualizing evolutionary data. *Bioinformatics* 14, 68–73 (1998)
16. Kelley, B.P., Sharan, R., Karp, R.M., Sittler, T., Root, D.E., Stockwell, B.R., Ideker, T.: Conserved pathways within bacteria and yeast as revealed by global protein network alignment. *Proc. Natl. Acad. Sci. USA* 100, 11394–11399 (2003)
17. Moran, S., Snir, S.: Convex recolorings of strings and trees: definitions, hardness results and algorithms. To appear in *Journal of Computer and System Sciences*. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 218–232. Springer, Heidelberg (2005) A preliminary version appeared
18. Moran, S., Snir, S., Sung, W.: Partial convex recolorings of trees and galled networks. Manuscript (2006)
19. Niedermeier, R.: *Invitation to Fixed Parameter Algorithms*. Oxford University Press, Oxford (2006)
20. Ramadan, E., Tarafdar, A., Pothén, A.: A hypergraph model for the yeast protein complex network. Fourth IEEE International Workshop on High Performance Computational Biology, Santa Fe, NM, April 26, 2004. IEEE Computer Society Press, Los Alamitos (2004)
21. Rual, J.F., Venkatesan, K., Hao, T., Hirozane-Kishikawa, T., Dricot, A., Li, N., Berriz, G.F., Gibbons, F.D., Dreze, M., Ayivi-Guedehoussou, N., Klitgord, N., Simon, C., Boxem, M., Milstein, S., Rosenberg, J., Goldberg, D.S., Zhang, L.V., Wong, S.L., Franklin, G., Li, S., Albala, J.S., Lim, J., Fraughton, C., Llamas, E., Cevik, S., Bex, C., Lamesch, P., Sikorski, R.S., Vandenhaute, J., Zoghbi, H.Y., Smolyar, A., Bosak, S., Sequerra, R., Doucette-Stamm, L., Cusick, M.E., Hill, D.E., Roth, F.P., Vidal, M.: *Nature*, 437, 1173–1178 (2005)
22. Schwikowski, B., Uetz, P., Fields, S.: A network of protein-protein interactions in yeast. *Nature Biotechnology* 18(12), 1257–1261 (2000)
23. Viveshwara, S., Brinda, K.V., Kannan, N.: Protein structure: insights from graph theory. *J. Theoretical and Computational Chemistry* 1, 187–211 (2002)

Quadratic Kernelization for Convex Recoloring of Trees^{*}

Hans L. Bodlaender¹, Michael R. Fellows^{2,3}, Michael A. Langston⁴,
Mark A. Ragan^{3,5}, Frances A. Rosamond^{2,3}, and Mark Weyer⁶

¹ Department of Information and Computing Sciences, Utrecht University,
Utrecht, The Netherlands

`hansb@cs.uu.nl`

² Parameterized Complexity Research Unit, Office of the DVC(Research),
University of Newcastle, Callaghan NSW 2308, Australia

`{michael.fellows, frances.rosamond}@newcastle.edu.au`

³ Australian Research Council Centre in Bioinformatics

⁴ Department of Computer Science, University of Tennessee,

Knoxville TN 37996-3450 and Computer Science and Mathematics Division,
Oak Ridge National Laboratory, Oak Ridge, TN 37831-6164 USA

`langston@cs.utk.edu`

⁵ Institute for Molecular Bioscience, University of Queensland,
Brisbane, QLD 4072 Australia

`m.ragan@imb.uq.edu.au`

⁶ Institut für Informatik, Humboldt-Universität zu Berlin, Berlin, Germany

`mark.weyer@informatik.hu-berlin.de`

Abstract. The CONVEX RECOLORING (CR) problem measures how far a tree of characters differs from exhibiting a so-called “perfect phylogeny”. For input consisting of a vertex-colored tree T , the problem is to determine whether recoloring at most k vertices can achieve a convex coloring, meaning by this a coloring where each color class induces a connected subtree. The problem was introduced by Moran and Snir, who showed that CR is NP-hard, and described a search-tree based FPT algorithm with a running time of $O(k(k/\log k)^k n^4)$. The Moran and Snir result did not provide any nontrivial kernelization. Subsequently, a kernelization with a large polynomial bound was established. Here we give the strongest FPT results to date on this problem: (1) We show that in polynomial time, a problem kernel of size $O(k^2)$ can be obtained, and (2)

^{*} This research has been supported by the Australian Research Council Centre in Bioinformatics, by the U.S. National Science Foundation under grant CCR-0311500, by the U.S. National Institutes of Health under grants 1-P01-DA-015027-01, 5-U01-AA-013512-02 and 1-R01-MH-074460-01, by the U.S. Department of Energy under the EPSCoR Laboratory Partnership Program and by the European Commission under the Sixth Framework Programme. The second and fifth authors have been supported by a Fellowship to the Institute of Advanced Studies at Durham University, and hosted by a William Best Fellowship to Grey College during the preparation of the paper.

We prove that the problem can be solved in linear time for fixed k . The technique used to establish the second result appears to be of general interest and applicability for bounded treewidth problems.

Topics: Algorithms and Complexity.

1 Introduction

The historically first and most common definition of *fixed-parameter tractability* for parameterized problems is solvability in time $O(f(k)n^c)$, where n is the input size, k is the parameter, and c is a constant independent of n and k . Background and motivation for the general subject of parameterized complexity and algorithmics can be found in the books [9,10,14]. This basic view of the subject is by now well-known.

Less well-known is the point of view that puts an emphasis on FPT kernelization. Kernelization is central to FPT as shown by the lemma:

Lemma 1. *A parameterized problem Π is in FPT if and only if there is a transformation from Π to itself, and a function g , that reduces an instance (x, k) to (x', k') such that:*

1. *the transformation runs in time polynomial in $|(x, k)|$,*
2. *(x, k) is a yes-instance of Π if and only if (x', k') is a yes-instance of Π ,*
3. *$k' \leq k$, and*
4. *$|(x', k')| \leq g(k)$.*

The lemma is trivial, but codifies a shift of perspective. To see how this works, consider the Moran and Snir FPT result for CONVEX RECOLORING, with the running time of $O^*(k/\log k)^k$. When $n > (k/\log k)^k$, then the Moran and Snir algorithm runs in polynomial time, in fact, in time $O(n^5)$. (So we can run the algorithm and determine the answer, and “transform” the input to either a canonical small NO instance or a canonical small YES instance accordingly.) If $n \leq (k/\log k)^k$ then we simply do nothing; we declare that the input is “already” kernelized to the bound $g(k) = (k/\log k)^k$. If a parameterized problem is FPT, that is, solvable in time $f(k)n^c$, then the lemma above gives us the trivial P-time kernelization bound of $g(k) = f(k)$. Normally for FPT problems, $f(k)$ is some exponential function of k , so the subclasses of FPT

$$\text{Lin}(k) \subseteq \text{Poly}(k) \subseteq \text{FPT}$$

of parameterized problems that admit P-time kernelization to kernels of size bounded by *linear* or *polynomial* functions of k would seem to be severe restrictions of FPT.

It is surprising that so many parameterized problems in FPT belong to these much more restrictive subclasses. The connection to heuristics goes like this: *FPT kernelization* is basically a systematic approach to *polynomial-time* preprocessing. Preprocessing plays a powerful role in practical approaches to solving

hard problems. For any parameterized problem in FPT there are now essentially two different algorithm design competitions with independent payoffs:

- (1) to find an FPT algorithm with the best possible exponential cost $f(k)$, and
- (2) to find the best possible polynomial-time kernelization for the problem.

A classic result on FPT kernelization is the VERTEX COVER $2k$ kernelization due to Nemhauser and Trotter (see [14]). The linear kernelization for PLANAR DOMINATING SET is definitely nontrivial [1]. The question of whether the FEEDBACK VERTEX SET (FVS) problem for undirected graphs has a $Poly(k)$ kernelization was a noted open problem in parameterized algorithmics, only recently solved. The first $Poly(k)$ kernelization for FVS gave a bound of $g(k) = O(k^{11})$ [6]. Improved in stages, the best kernelization bound is now $O(k^3)$ [4].

We prove here a polynomial time kernelization for the CONVEX RECOLORING problem, to a reduced instance that has $O(k^2)$ vertices. The basic problem is defined as follows.

CONVEX RECOLORING

Instance: A tree $F = (V, E)$, a set of colors \mathcal{C} , a coloring function $\Gamma : V \rightarrow \mathcal{C}$, and a positive integer k .

Parameter: k

Question: Is it possible to modify Γ by changing the color of at most k vertices, so that the modified coloring Γ' is *convex* (meaning that each color class induces a single monochromatic subtree)?

The CONVEX RECOLORING problem is of interest in the context of maximum parsimony approaches to evaluating phylogenetic trees [12, 11]. Some further applications in bioinformatics are also described in [12]; see also [13].

2 Preliminaries

Our kernelization algorithm is shown for the following generalized problem.

ANNOTATED CONVEX RECOLORING

Instance: A forest $F = (V, E)$ where the vertex set V is partitioned into two types of vertices, $V = V_0 \cup V_1$, a set of colors \mathcal{C} , a coloring function $\Gamma : V \rightarrow \mathcal{C}$, and a positive integer k .

Parameter: k

Question: Is it possible to modify Γ by changing the color of at most k vertices in V_1 , so that the modified coloring Γ' is *convex*?

A *block* in a colored forest is a maximal set of vertices that induces a monochromatic subtree. For a color $c \in \mathcal{C}$, $\beta(\Gamma, c)$ denotes the number of blocks of color c with respect to the coloring function Γ . A color c is termed a *bad color* if $\beta(\Gamma, c) > 1$, and c is termed a *good color* if $\beta(\Gamma, c) = 1$.

A recoloring Γ' of coloring Γ is *expanding*, if for each block of Γ' , there is at least one vertex in the block that keeps its color, i.e., that has $\Gamma'(v) = \Gamma(v)$.

Moran and Snir [12] have shown that there always exists an optimal expanding convex recoloring. It is easy to see that this also holds for the variant where we allow annotations and a forest, as above.

As in [3], we define for each $v \in V$ a set of colors $S(v)$: $c \in S(v)$, if and only if there are two distinct vertices $w, x, w \neq v, x \neq v$, such that w and x have color c (in Γ), and v is on the path from w to x .

We introduce another special form of convex recoloring, called *normalized*. For each vertex $v \in V$, we add a new color c_v . We denote $\mathcal{C}' = \mathcal{C} \cup \{c_v \mid v \in V\}$. A recoloring $\Gamma' : V \rightarrow \mathcal{C}'$ of coloring $\Gamma : V \rightarrow \mathcal{C}$ is *normalized*, if for each $v \in V$: $\Gamma'(v) \in \{\Gamma(v), c_v\} \cup S(v)$.

Lemma 2. *There is an optimal convex recoloring that is normalized.*

Proof. Take an arbitrary optimal convex recoloring Γ' . Define Γ'' as follows. For each $v \in V$, if $\Gamma'(v) \in \{\Gamma(v)\} \cup S(v)$, then set $\Gamma''(v) = \Gamma'(v)$. Otherwise, set $\Gamma''(v) = c_v$. One can show that $\Gamma''(v)$ is convex. Clearly, it is normalized, and recolors at most as many vertices as $\Gamma'(v)$. \square

3 Kernelizing to a Linear Number of Vertices Per Color

In this section, we summarize a set of rules from [6] that ensure that for each color, there are at most $O(k)$ vertices with that color. There are two trivial rules: if F is an empty forest and $k \geq 0$, then we return YES; if $k < 0$, then we return NO if and only if the given coloring is not already convex.

Rule 1. *Suppose there are $\alpha \geq 2k + 3$ vertices with color c . Let v be a vertex, such that each connected component of $F - v$ contains at most $\frac{\alpha}{2}$ vertices of color c . Assume that $v \notin V_0$, or $\Gamma(v) \neq c$. Now*

- *If v has a color, different from c , and $v \in V_0$, then return NO.*
- *If v has a color, different from c , and $v \notin V_0$, then set $\Gamma(v) = c$ and decrease k by one.*
- *Fix the color of v : put $v \in V_0$.*

Rule 2. *Suppose $v \in V_0$ has color c . Suppose one of the connected components of F contains at least $k + 1$ vertices with color c , but does not contain v . Then return NO.*

Rule 3. *Let v be a vertex of color c . Suppose $F - v$ has a component with vertex set W that contains at least $k + 1$ vertices with color c , that contains a neighbor of v . Assume that $w \notin V_0$ or $\Gamma(w) \neq c$. Then*

- *If w has a color, different from c , and $w \in V_0$, then return NO.*
- *If w has a color, different from c , and $w \notin V_0$, then give w color c , and decrease k by one.*
- *Fix the color of w : put $w \in V_0$.*

Rule 4. *If there is a tree T in the colored forest that contains two distinct vertices u and v , both of which have fixed color c , then modify T by contracting the path between u and v (resulting in a single vertex of fixed color c). If r denotes the number of vertices of this path that do not have color c , then $k' = k - r$.*

Rule 5. *If there are two distinct trees in the colored forest that both contain a vertex of fixed color c , then return NO.*

Let $v \in V_0$ have color c . A connected component of $F - v$ with vertex set W is said to be *irrelevant*, if both of the following two properties hold: (1) The vertices with color c in W form a connected set that contains a neighbor of v , and (2) For each color $c' \neq c$, if there are vertices with color c' in W , then c' is not bad. If a component is not irrelevant, it is said to be *relevant*.

Rule 6. *Let $v \in V_0$. Let W be the vertex set of an irrelevant connected component of $F - v$. Then remove W , and all edges incident to vertices in W , from the tree.*

Rule 7. *Suppose $F - v$ contains at least $2k + 1$ components, and each component of $F - v$ is relevant. Then return NO.*

Rule 8. *Let $v \in V_0$ be a vertex with fixed color c . Let W be the vertex set of a relevant component of $F - v$ that contains vertices with color c . Suppose there are vertices with color c , not in $\{v\} \cup W$. Then*

- *Take a new color c' , not yet used, and add it to \mathcal{C} .*
- *Take a new vertex v' , and add it to F .*
- *Set $v' \in V_0$. Color v' with c' .*
- *For all $w \in W$ with color c , give w color c' . (Note that k is not changed.)*
- *For each edge $\{v, w\}$ with $w \in W$: remove this edge from F and add the edge $\{v', w\}$.*

Theorem 1. *An instance that is reduced with respect to the above Rules has at most $2k + 2$ vertices per color.*

4 Kernelizing to a Quadratic Number of Vertices

In this section, we work in a series of steps towards a kernel with $O(k^2)$ vertices. A number of new structural notions are introduced.

4.1 Bad Colors

Rule 9. *If there are more than $2k$ bad colors, then return NO.*

The colors that are not bad are distinguished into three types: gluing, stitching, and irrelevant.

4.2 Gluing Colors

A vertex v is *gluing* for color c , if it is adjacent to two distinct vertices x and y , both of color c , but the color of v is unequal to c . Note that if v is gluing for color c , then v separates (at least) two blocks of color c : x and y belong to different blocks. When we recolor v with color c , then these blocks become one block (are 'glued' together). A vertex v is *gluing*, if v is gluing for some color, and if the color of v is not bad. A color c is *gluing*, if there is a gluing vertex with color c , and c is not bad. (Note that the vertex will be gluing for some other color $c' \neq c$.)

For a color c , let W_c be the set of vertices that have color c or are gluing for color c . A *bunch* of vertices of color c is a maximal connected set of vertices in W_c , i.e., a maximal connected set of vertices that have either color c or are gluing for color c . The following lemma is not difficult, and establishes the soundness of the next Rule.

Lemma 3. *Suppose W is a bunch of color c that contains ℓ vertices that are gluing for color c . Then in any convex recoloring, at least ℓ vertices in W are recolored, and for each, either the old, or the new color is c .*

Rule 10. *If there are more than $2k$ vertices that are gluing, then return NO.*

As a result, there are $O(k)$ colors that are gluing, and thus $O(k^2)$ vertices that have a gluing color. We next bound the number of bunches.

Lemma 4. *Suppose there are ℓ bunches with color c . Then the number of recolored vertices whose old or new color equals c , is at least $\ell - 1$.*

Proof. Omitted due to space limitations. □

Rule 11. *Suppose that for each bad color c , there are ℓ_c bunches. If the sum over all bad colors c of $\ell_c - 1$ is at least $2k + 1$, then return NO.*

4.3 Stitching Colors

We now define the notion of stitching vertices. To arrive at an $O(k^2)$ kernel, we have to define this notion with some care.

We first define the *cost* of a path between two vertices with the same color that belong to different bunches. Let v and w be two vertices with color c , in the same subtree of F , but in different bunches of color c . The *cost* of the path from v to w is the sum of the number of vertices with color unequal to c on this path, and the sum over all colors $c' \neq c$ such that

- c' is not bad and c' is not gluing, and
- there is at least one vertex with color c' on the path from v to w in T

of the following term: consider the blocks with color c' in the forest, obtained by removing the path from v to w from T . If one of these blocks contains a

vertex in V_0 (i.e., it has fixed color c'), then sum the sizes of all other blocks. Otherwise, sum the sizes of the all blocks except one block with the largest number of vertices.

A *stitch* of color c is a path between two vertices v, w , both of color c , such that: (1) v and w are in different bunches, (2) all vertices except v and w have a color, different from c , and do not belong to V_0 , and (3) the cost of the path is at most k . A vertex v is *stitching* for color c , if: (1) the color of v is different from c , (2) v is not gluing for color c , and (3) v is on a stitch of color c . A vertex is *stitching*, if it is stitching for at least one color. (Note that this vertex will be stitching for a bad color.) A color c is *stitching* if there is at least one vertex with color c that is stitching, and it is not bad or gluing. A color is *irrelevant*, if it is neither bad, gluing, or stitching. Summarizing, we have the following types of vertices: (a) vertices with a bad color, (b) vertices with a gluing color, (c) vertices with a stitching color that belong to a stitch, (d) vertices with a stitching color, that do not belong to a stitch — these will be partitioned into pieces in Section 4.4 and (e) vertices with an irrelevant color.

Proofs of the next two lemmas can be found in the full paper, and establish the soundness of the next two rules.

Lemma 5. *Suppose there is a convex recoloring with at most k recolored vertices. Then there is an optimal convex recoloring such that for each bad color c , all vertices that receive color c have color c in the original coloring or are gluing for c or are stitching for c .*

Lemma 6. *Suppose there is a convex recoloring with at most k recolored vertices. Then there is an optimal convex recoloring such that no vertex with an irrelevant color is recolored.*

Rule 12. *Let $v \in V_1$ be a vertex with an irrelevant color. Put v into V_0 (i.e., fix the color of v .)*

Rule 13. *Let $v \in V_1$ be a vertex with a stitching color. Suppose v is not vulnerable. Then put v into V_0 .*

The following rule, in combination with earlier rules, helps us to get a reduced instance without any vertex with an irrelevant color.

Rule 14. *Let $v \in V_0$ be the only vertex with some color c . Then remove v and its incident edges.*

Soundness is easy to see: the same recolorings in the graph with and without v are convex. The combination of Rules 4, 12, and 14 causes the deletion of all vertices with an irrelevant color: all such vertices first get a fixed color, then all vertices with the same irrelevant color are contracted to one vertex, and then this vertex is deleted. So, we can also use instead the following rule.

Rule 15. *Suppose c is an irrelevant color. Then remove all vertices with color c .*

4.4 Pieces of a Stitching Color

For a kernel of size $O(k^2)$, we still can have too many vertices with a stitching color. In order to reduce the number of such vertices we introduce the concept of a *piece of color c* . Consider a stitching color c , and consider the subforest of the forest, induced by the vertices with color c . If we remove from this subtree all vertices that are on a stitch, then the resulting components are the pieces, i.e., a *piece of color c* is a maximal subtree of vertices with color c that do not belong to a stitch. Assume that we have exhaustively applied all of the rules described so far, and therefore we do not have vertices with an irrelevant color. The next lemma, given here without proof, shows that the next Rule is sound.

Lemma 7. *Suppose W is the vertex set of a piece of stitching color c . Suppose there is a vertex $v \in W \cap V_0$. Then if there is a convex recoloring with at most k recolored vertices, then there is a convex recoloring that does not recolor any vertex in W .*

Rule 16. *Let W be a vertex set of a piece of stitching color c , that contains at least one vertex in V_0 . Then put all of the vertices of W into V_0 .*

As a result of this rule and Rule [4](#), a piece containing a vertex with a fixed color will contain only one vertex. We omit the soundness proof for the following rule from this version. If there is a large piece, found by Rule [17](#), then it will be contracted to a single vertex by Rule [4](#).

Rule 17. *Suppose c is a stitching color, and there are α vertices with color c . Suppose there is no vertex in V_0 with color c . If W is the vertex set of a piece of color c , and $|W| > \alpha/2$, then put all of the vertices of W into V_0 .*

4.5 Kernel Size

We now *tag* some vertices that have a stitching color. A tag is labeled with a bad color. Basically, when we have a stitch for a bad color c , we tag the vertices that count for the cost of the stitch with color c . Tagging is done as follows. For each stitch for bad color c , and for each stitching color c' with a vertex on the stitch with color c' , consider the blocks of color c' obtained by removing the vertices on the stitch. If there is no vertex with color c' in V_0 , then take a block with vertex set W such that the number of vertices in this piece is at least as large as the number of vertices in any other piece. Tag all vertices in $Q - W$ with color c . If there is a vertex v with color c' in V_0 , then tag all vertices with color c' , except those that are in the same block as v .

Comparing the tagging procedure with the definition of the cost of a stitch, we directly note that the number of vertices that is tagged equals the cost of the stitch, i.e., for each stitch of bad color c , we tag at most k vertices with c .

We now want to count the number of vertices with a stitching color. To do so, we first count the number of vertices with a stitching color that are tagged. To do this, we consider a bad color c , and count the number of vertices, with a stitching color, that are tagged with c .

The following three lemmas are proved in the full paper.

Lemma 8. *Let c be a bad color with ℓ_c bunches. A reduced instance has at most $2k(\ell_c - 1)$ vertices that are tagged with c .*

Lemma 9. *Let c be a stitching color. There is at most one piece of color c that contains a vertex that is not tagged with a bad color.*

Lemma 10. *In a reduced instance, there are at most $8k^2$ vertices with a stitching color.*

Theorem 2. *In time polynomial in n and k , a kernel can be found with $O(k^2)$ vertices.*

Proof. With standard techniques, one can observe that all rules can be carried out in time polynomial in n and k .

In the reduced instance, there are at most $2k$ bad colors, and at most $2k$ gluing colors. For each of these colors, there are at most $2k + 2$ vertices with that color. So, in total at most $4k^2 + 4k$ vertices have a bad or gluing color. There are at most $8k^2$ vertices with a stitching color, and no vertices with an irrelevant color, so in total we have at most $12k^2 + 4k$ vertices. \square

5 Linear Time FPT with Treewidth and MSOL

We describe an algorithm that solves the CONVEX RECOLORING problem in $O(f(k) \cdot n)$ time. There are four main ingredients of the algorithm: (1) the construction of an auxiliary graph, the *vertex-color* graph, (2) the observation that for yes-instances of the CONVEX RECOLORING problem this graph has bounded treewidth, (3) a formulation of the CONVEX RECOLORING problem for fixed k in Monadic Second Order Logic, and (4) the use of a famous result of Courcelle [8], see also [2][5]. For ease of presentation, we assume that there are no vertices with a fixed color, and that the input is a tree. The “trick” of using such an auxiliary graph seems to be widely applicable. For another example of this new technique, see [7].

Suppose we have a tree $T = (V, E)$, and a coloring $\Gamma : V \rightarrow \mathcal{C}$. The *vertex-color graph* has as vertex set $V \cup \mathcal{C}$, i.e., we have a vertex for each vertex in T , and a vertex for each color. The edge set of the vertex-color graph is $E \cup \{\{v, c\} \mid \Gamma(v) = c\}$, i.e., there are two types of edges: the edges of the tree T , and an edge between a vertex of T and the vertex representing its color. Recall that $S(v)$ denotes the set of colors c for which there are vertices w and x , distinct from v , with v on the path from w to x in T , and $\Gamma(w) = \Gamma(x) = c$.

Lemma 11. *Suppose there is a convex recoloring with at most k recolored vertices. Then for each $v \in V$, $|S(v)| \leq k + 1$.*

Proof. Suppose Γ' is a convex recoloring with at most k recolored vertices of Γ . Consider a vertex $v \in V$, and a color $c \in S(v)$. Suppose $c \neq \Gamma'(v)$. There are vertices w, x with $\Gamma(w) = \Gamma(x) = c$, and the path from w to x uses v . As v is not recolored to c , either w or x must be recolored. So, there can be at most k colors $\neq \Gamma'(v)$ that belong to $S(v)$. \square

Lemma 12. *Suppose there is a convex recoloring with at most k recolored vertices. Then the treewidth of the vertex-color graph is at most $k + 3$.*

Proof. Without loss of generality, we suppose that Γ is surjective, i.e., for all $c \in \mathcal{C}$, there is a $v \in V$ with $\Gamma(v) = c$. (If Γ is not surjective, then colors c with $\Gamma^{-1}(c) = \emptyset$ are isolated vertices in the vertex-color graph, and removing these does not change the treewidth.)

Take an arbitrary vertex $r \in V$ as root of T . For each $v \in V$, set $X_v = \{v, p(v), \Gamma(v)\} \cup S(v)$, where $p(v)$ is the parent of v . For $v = r$, we take $X_v = \{v, \Gamma(v)\} \cup S(v)$.

It is easy to verify directly that $(\{X_v \mid v \in V\}, T)$ is a tree decomposition of the vertex-color graph of width at most $k + 3$. \square

Theorem 3. *For each fixed k , there is a linear time algorithm that, given a vertex-colored tree T , decides if there is a convex recoloring of T with at most k recolored vertices.*

Proof. We show that for a fixed number of recolored vertices k , the CONVEX RECOLORING problem can be formulated as a property of the vertex-color graph that can be stated in Monadic Second Order Logic. The result then directly follows by the result of Courcelle [8], that each such property can be decided in linear time on graphs with bounded treewidth. (See also [25]. We use Lemma [2].)

We assume we are given the vertex-color graph $(V \cup \mathcal{C}, E')$, and have sets V and \mathcal{C} distinguished. A recoloring Γ' that differs from Γ at most k vertices can be represented by these vertices v_1, \dots, v_k and by the corresponding values $c_1 := \Gamma'(v_1), \dots, c_k := \Gamma'(v_k)$. Then, for a given color c , consider the set V_c defined as follows:

- If $v = v_i$ for some $1 \leq i \leq k$, then $v \in V_c$ if and only if $c_i = c$.
- If $v \notin \{v_1, \dots, v_k\}$, then $v \in V_c$ if and only if (v, c) is an edge of the vertex-color graph.

Note, that V_c is the color class of c in the coloring Γ' . Hence Γ' is convex if and only if V_c is connected for each $c \in \mathcal{C}$. It is a standard fact, that MSOL can express connectedness, say by a formula $\varphi(X)$, where the set variable X represents V_c . Furthermore, the definition of V_c given above can be expressed even in first-order logic, say by a formula $\psi(x_1, \dots, x_k, y_1, \dots, y_k, z, X)$, where additionally x_1, \dots, x_k represent v_1, \dots, v_k , y_1, \dots, y_k represent c_1, \dots, c_k , and z represents c . Then, for this k , the CONVEX RECOLORING problem can be expressed by the formula

$$\exists x_1, \dots, x_k \exists y_1, \dots, y_k \left(\bigwedge_{1 \leq i \leq k} V x_i \wedge \bigwedge_{1 \leq i < j \leq k} x_i \neq x_j \wedge \bigwedge_{1 \leq i \leq k} \mathcal{C} y_i \wedge \forall z \forall X (\psi \rightarrow \varphi) \right).$$

The algorithm works as follows. Given a colored tree (T, Γ) , it constructs the vertex-color graph and computes a tree-decomposition of width at most $k + 3$. If this fails, the algorithm returns NO, in accordance with Lemma [2]. Otherwise, using the tree-decomposition, it evaluates the above formula on the vertex-color graph. Each step uses at most linear time. \square

References

1. Alber, J., Fellows, M.R., Niedermeier, R.: Polynomial-time data reduction for dominating sets. *J. ACM* 51, 363–384 (2004)
2. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. *J. Algorithms* 12, 308–340 (1991)
3. Bar-Yehuda, R., Feldman, I., Rawitz, D.: Improved approximation algorithm for convex recoloring of trees. In: Erlebach, T., Persinao, G. (eds.) WAOA 2005. LNCS, vol. 3879, pp. 55–68. Springer, Heidelberg (2006)
4. Bodlaender, H.L.: A cubic kernel for feedback vertex set. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 320–331. Springer, Heidelberg (2007)
5. Borie, R.B., Parker, R.G., Tovey, C.A.: Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica* 7, 555–581 (1992)
6. Burrage, K., Estivill-Castro, V., Fellows, M.R., Langston, M.A., Mac, S., Rosamond, F.A.: The undirected feedback vertex set problem has a $\text{poly}(k)$ kernel. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 192–202. Springer, Heidelberg (2006)
7. Chor, B., Fellows, M., Ragan, M., Rosamond, F., Razgon, I., Snir, S.: Connected coloring completion for general graphs: algorithms and complexity. In: Proceedings COCOON (2007)
8. Courcelle, B.: The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation* 85, 12–75 (1990)
9. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
10. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
11. Gramm, J., Nickelsen, A., Tantau, T.: Fixed-parameter algorithms in phylogenetics. To appear in *The Computer Journal*.
12. Moran, S., Snir, S.: Convex recolorings of strings and trees: Definitions, hardness results, and algorithms. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 218–232. Springer, Heidelberg (2005)
13. Moran, S., Snir, S.: Efficient approximation of convex recolorings. In: Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX 2005 and RANDOM 2005. LNCS, vol. 3624, pp. 192–208. Springer, Heidelberg (2005)
14. Niedermeier, R.: Invitation to fixed-parameter algorithms. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, Oxford (2006)

On the Number of Cycles in Planar Graphs

Kevin Buchin¹, Christian Knauer¹, Klaus Kriegel¹, André Schulz¹, and Raimund Seidel²

¹ Freie Universität Berlin, Institute of Computer Science,
Takustr. 9, 14195 Berlin, Germany
{buchin, knauer, kriegel, schulz}@inf.fu-berlin.de

² Universität des Saarlandes, Institute of Computer Science, PO Box 151150,
66041 Saarbrücken, Germany
rseidel@cs.uni-sb.de

Abstract. We investigate the maximum number of simple cycles and the maximum number of Hamiltonian cycles in a planar graph G with n vertices. Using the transfer matrix method we construct a family of graphs which have at least 2.4262^n simple cycles and at least 2.0845^n Hamilton cycles.

Based on counting arguments for perfect matchings we prove that 2.3404^n is an upper bound for the number of Hamiltonian cycles. Moreover, we obtain upper bounds for the number of simple cycles of a given length with a face coloring technique. Combining both, we show that there is no planar graph with more than 2.8927^n simple cycles. This reduces the previous gap between the upper and lower bound for the exponential growth from 1.03 to 0.46.

1 Introduction

In this paper we consider the following question:

How many simple cycles and how many Hamiltonian cycles can there be in a planar graph with n vertices?

Since the determination of the exact numbers seems to be out of reach, our goal is to learn more about the asymptotic behavior of these numbers. Denoting by $C_s(G)$ and $C_h(G)$ the numbers of simple cycles and of Hamiltonian cycles in a graph G we define

$$C_s(n) = \max \{C_s(G) \mid G \text{ is a planar graph on } n \text{ vertices}\}, \text{ and} \\ C_h(n) = \max \{C_h(G) \mid G \text{ is a planar graph on } n \text{ vertices}\}.$$

It is easy to observe that both $C_s(n)$ and $C_h(n)$ grow exponentially and thus we are interested in describing this *exponential growth rate* by constants $c, d \in \mathbb{R}$ such that $c^n \leq C_s(n) \leq d^n$, and analogously for $C_h(n)$.

The lower bound $C_s(n) = \Omega(2.259^n)$ was obtained in [1] and is based on counting the number of simple paths connecting two adjacent vertices in a special planar graph on 29 vertices and joining $n/28$ copies of it in a cyclic way.

An $O(3.363^n)$ upper bound was proved in the same paper by a probabilistic argument. Here we extend the original problem setting to Hamiltonian cycles.

The problem has gained new attention by some recent results of Sharir and Welzl [2], [3]. They investigate the numbers of several geometric objects on a point set in the plane, among them triangulations and crossing-free spanning cycles. In particular they note that an upper bound on the number of crossing-free spanning cycles can be obtained by combining an upper bound on the number of triangulations with an upper bound on the number of Hamiltonian cycles in planar graphs. Here, we will present the proof to the $\sqrt{6}^n$ upper bound for the number of Hamiltonian cycles, which is quoted in [2] as a personal communication, along with an improvement to $\sqrt[4]{30}^n$.

In [2] Sharir and Welzl prove a bound of $O(86.81^n)$ for the number of crossing-free spanning cycles on n points with an alternative approach. This bound is better than the combined bound and it remains better even if the improved bound for Hamiltonian cycles presented in our paper is used. In fact, the lower bound on the number of Hamiltonian cycles presented in our paper shows that a better combined bound cannot be obtained without improving the bound on the number of triangulations.

The paper is organized as follows: In Section 2 we present new lower bounds for $C_s(n)$ and for $C_h(n)$. We prove $C_s(n) = \Omega(2.4262^n)$ and $C_h(n) = \Omega(2.0845^n)$. Both bounds are based on the so-called transfer matrix method applied on a twisted cylinder.

In Section 3 we prove first the $O(\sqrt[4]{30}^n)$ upper bound on $C_h(n)$. Next we present a new technique for upper bounds on the number of simple cycles with a given length k in planar graphs on n vertices. Combining both we will obtain a new $O(2.8928^n)$ upper bound for $C_s(n)$.

2 Lower Bounds

We will present a new lower bound for $C_s(n)$ by counting cycles on the *twisted cylinder*. We use the technique of the *transfer matrix method* (see [4,5,6]).

The twisted cylinder describes a graph which is parametrized by a *width* w and a *length* l . We will describe the graph by the following construction: Consider an $\lfloor l/w \rfloor \times w$ integer lattice with the upper leftmost point $(0, 0)$ and the lower rightmost point (r, w) . Furthermore we attach $(l \bmod w)$ squares at the right end, starting from the top. As a next step we triangulate each square of the lattice by adding diagonals $((x, y), (x+1, y+1))$ for all appropriate values x and y . Finally we identify all edges $((x, w), (x+1, w))$ with the edges $((x+1, 0), (x+2, 0))$ for all x smaller than $\lfloor l/w \rfloor$. Observe that this graph is planar since it can be embedded as the graph of a 3-polytope. Figure 1 shows a twisted cylinder of length 41 and width 5, and Figure 2 shows a planar embedding of a twisted cylinder of length 12 and width 6. To count the cycles, we construct the cylinder by increasing its length consecutively. We name the cylinder of width w after k rounds Z_k^w and call the last inserted $w + 1$ points its *border*. During the construction we have to deal with unfinished cycles. These cycles are represented

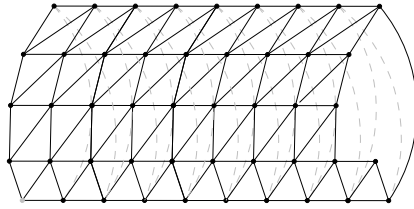


Fig. 1. The graph of a twisted cylinder

as non-intersecting paths which start and end at the border of Z_k^w . To complete a cycle we need the information which of the points at the border belong to the same path. We will store this information in a string of length $w + 1$ which we call the *signature*. The last point introduced corresponds to the first character of the signature, its predecessor to the second character and so on. Every path has a start and an end point on the border. The point that was introduced later is considered as the start point, the other as the end point of a path. We associate a start point at the border with an A . The position of the end point of a path will be marked in the signature as B . Interior points of paths at the border are denoted by X . A point at the border that is not used from any path will be represented as O in the signature. Thus we get as signature a string from $\{A, B, X, O\}^{w+1}$. Figure 2 shows an example which has the signature $AXOAOBB$. Notice that

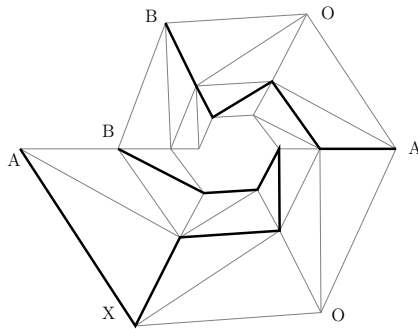


Fig. 2. Partially constructed cycle with signature $AXOAOBB$

the signatures can be represented as 2-Motzkin paths [7, Exercise 6.38.], which are one of the numerous incarnations of Catalan structures.

We come back to the counting procedure. During the construction we trace the number of completed and uncompleted cycles. We count the different ways of generating an uncompleted cycle by a variable indexed by its signature. The completed cycles are stored in a distinct variable. All variables are stored in a vector which we call the *state vector* S_k .

Going from cylinder Z_k^w to Z_{k+1}^w will change the state vector. We introduce three new edges and therefore have at most 7 ways to continue uncompleted cycles (choosing all 3 edges will not give a valid successor state). Not all of these choices will produce a valid signature for the successor state. For example the signature $AXOAOBB$ (Figure 2) has four successors, depicted in Figure 3.

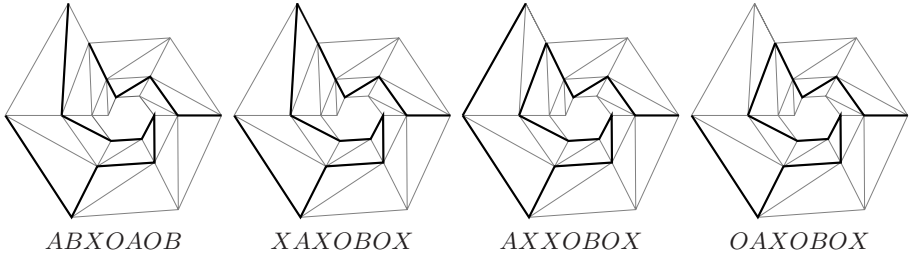


Fig. 3. The successor states from Figure 2

It is not hard to see that every entry of the “new” state vector S_{k+1} is a non-negative linear combination of the entries of the “old” state vector S_k . Thus $S_{k+1} = T \cdot S_k$, where T is a square matrix with non-negative entries, which is called the *transfer matrix*. By construction this matrix T does not depend on k , and thus for every $k \geq 0$ we have $S_k = T^k \cdot S_0$. The entry in S_k for the completed cycles in Z_k^w is then a lower bound for $C_s(k+w)$.

This entry can be represented as $e^t \cdot S_k$ for the appropriate unit basis vector e . Thus we are interested in the asymptotic behavior of a sequence with elements $p_k = a^t \cdot T^k \cdot b$, where a and b are vectors and T is a square matrix. By considering the Jordan canonical form $T = X^{-1} \cdot J \cdot X$ it is easy to see that $p_k = O(q(k)c^k)$, where c is the largest absolute value of any eigenvalue of T and q is a polynomial of degree smaller than the multiplicity of any eigenvalue of maximal absolute value.

If in our case we remove all unreachable configurations from consideration then the resulting transfer matrix T will be primitive in the sense that for some $\ell > 0$ all entries of T^ℓ are strictly positive. In this case the Perron-Frobenius Theorem [8] guarantees that the eigenvalue of largest absolute value is real and unique. Thus $S_k = O(c^k)$, where c is the largest real eigenvalue of the transfer matrix T .

The generation of T and the computation of its eigenvalues was done with the help of computer programs. We omit the details of the computation of T . The correctness of the calculations was checked by two independent implementations. We observed that larger widths will result in larger growth. The largest width our implementations could handle is 13. The largest absolute eigenvalue for T was computed as 2.4262. The computation was done on a AMD Athlon 64 with 1.8 GHz and 1 GB of RAM. It required 5 days of CPU-time and 550 MB of memory. The implementation can be found under [9]. The results of the computation are listed in Table 1.

Theorem 1. *The maximal number of simple cycles in a planar graph G with n vertices is bounded from below by $\Omega(2.4262^n)$.*

We can also construct a lower bound for Hamiltonian cycles with the method from above. To this end we restrict the state transitions in such a way that if a vertex vanishes from the border, it is guaranteed to be on some path. We forbid all sequences which contain a O as character and calculate the modified transfer matrix.

The largest eigenvalue for the modified transfer matrix is 2.0845. It is obtained for a twisted cylinder of width 13. See Table 1 for the results of the computation.

Table 1. Eigenvalues λ_T of the transfer matrix T , generated for Hamiltonian cycles (H. cyc.) and simple cycles (simple cyc.) depending on the width of the twisted cylinder

w	λ_T H. cyc.	λ_T simple cyc.	w	λ_T H. cyc.	λ_T simple cyc.
2	1.8124	1.9659	8	2.0688	2.4078
3	1.9557	2.2567	9	2.0740	2.4139
4	2.0022	2.3326	10	2.0777	2.4183
5	2.0335	2.3654	11	2.0805	2.4217
6	2.0507	2.3858	12	2.0827	2.4242
7	2.0614	2.3991	13	2.0845	2.4262

Theorem 2. *The maximal number of Hamiltonian cycles in a planar graph G with n vertices is bounded from below by $\Omega(2.0845^n)$.*

3 Upper Bounds

3.1 Hamiltonian Cycles

In this section G denotes a planar graph with n vertices, e edges, and f faces. Since additional edges cannot decrease the number of cycles, we focus on triangulated planar graphs. In this case we have $3f = 2e$, which leads to $e = 3n - 6$ and $f = 2n - 4$.

Let us assume first that n is even and let $M(G)$ denote the number of perfect matchings in G . By a theorem of Kasteleyn, c.f. [10], there is an orientation of the edges of G such that the corresponding skew symmetric adjacency matrix A characterizes $M(G)$ in the following way:

$$(M(G))^2 = |\det(A)|.$$

Note that all but $6n - 12$ entries of A are zero and the nonzero entries are 1 or -1 . In this situation we can apply the Hadamard bound for determinants and we obtain $|\det(A)| \leq \sqrt{6}^n$.

In this way we obtain an $\sqrt[4]{6}^n$ upper bound on the number of perfect matchings in G , which improves the $O(\sqrt{3}^n)$ bound from [11]. Moreover, our bound can be improved for graphs with few edges.

Theorem 3. *The number of perfect matchings in a planar graph G with n vertices is bounded from above by $\sqrt[4]{6^n}$.*

The number of perfect matchings in a planar graph G with n vertices and at most kn edges is bounded from above by $\sqrt[4]{2k^n}$.

Our first bound on the number of Hamiltonian cycles follows from Theorem 3 by an easy observation.

Theorem 4. $C_h(n) = O(\sqrt[4]{30^n}) = O(2.3404^n)$.

Proof. Any Hamiltonian cycle in a graph G with an even number of vertices splits into two perfect matchings, which implies $C_h(G) \leq (M(G))^2 \leq \sqrt{6^n}$. The following modification of the arguments above results in a slight improvement of that bound:

Splitting a Hamiltonian cycle into two perfect matchings, we fix the matching with the lexicographically smallest edge as the first matching m_1 and the other one as the second matching m_2 . It follows that if m_1 is fixed, m_2 is a matching in a graph with $2.5n - 6$ edges. Repeating the above observations for both matchings, we get

$$M_1(G) \leq \sqrt[4]{6^n}, \quad M_2(G) \leq \sqrt[4]{5^n} \quad \text{and together} \quad C_h(G) \leq \sqrt[4]{30^n}.$$

Finally we study the case that n is odd. We choose in G a vertex v of degree at most 5, and for each e incident to v we consider the Graph G_e obtained from G by contracting e . Any Hamiltonian cycle in G contains two edges e and e' incident to v and hence induces a Hamiltonian cycle in G_e and $G_{e'}$. On the other hand, any Hamiltonian cycle in some G_e may be extended in up to two ways to a Hamiltonian cycle in G . Thus we obtain an upper bound on $C_h(G)$ by adding the number of Hamiltonian cycles in the at most five planar graphs G_e , leading to a bound of $C_h(G) \leq 5\sqrt[4]{30^{n-1}}$. \square

3.2 Simple Cycles

We start with a new upper bound for the number of cycles in planar graphs and successively improve the bound.

Instead of counting cycles we count paths on G , which can be completed to a simple cycle. We call these paths *cycle-paths*. Their number is an upper bound for the number of cycles. The number of cycle-paths is maximized when G is triangulated. Therefore we assume that G is triangulated.

There exist n paths of length 0. The number of all cycle-paths in G of nonzero length is at most the number of edges e times the maximum number of cycle-paths starting from an arbitrary edge. Thus the exponential growth of the number of cycle-paths is determined by the number of cycle-paths starting from an edge.

Lemma 1. *The maximum number of cycle-paths on G starting from an edge is bounded by $O(n) \cdot 3^n$.*

Proof. We give the starting edge an orientation. We consider only paths in the direction induced by this orientation. The total number of cycle-paths starting from this edge is at most twice the number of cycle-paths with the chosen orientation.

We associate cycle-paths with the nodes of a tree. The root of the tree contains the path of length one corresponding to the starting edge. The children of a tree node contain paths starting with the path stored in the predecessor plus an additional edge. Every cycle-path is only stored in one tree node.

Every cycle-path in G corresponds to a partial red-blue coloring of the faces of G . The coloring is defined as follows: The faces right of the oriented path will be colored blue the faces left of the oriented path red (see Figure 4). We color all faces incident to an inner vertex or the starting edge of the path. The coloring is consistent, because we consider only paths which can be extended to cycles. Therefore the colors correspond to a part of the interior or exterior region induced by the cycle.

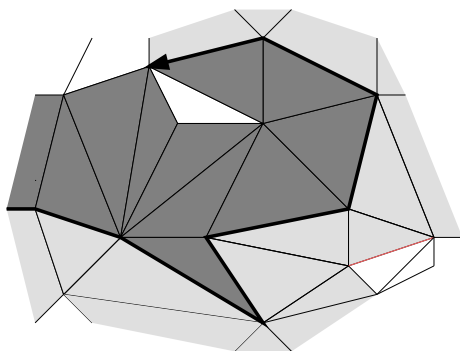


Fig. 4. Example of an induced red-blue coloring by a path (light gray corresponds to red, dark gray to blue)

We construct the tree top down. When we enter a new tree node, the color of at least two faces incident to the last vertex v_i of the path is given. It might be that other faces incident to v_i have been colored before. In that case we color the faces incident to v_i which lie in between two red faces red. The faces which are located in between two blue faces will be colored blue. Observe that at most one non-colored connected region incident to v_i remains. Otherwise it is not possible to extend the path to a cycle and therefore the path stored in this tree node is not a cycle-path. Figure 5 illustrates this procedure. Let k_v be the number of faces of the non-colored region incident to v . We have $k_v + 1$ different ways to continue the path and therefore $k_v + 1$ children of its tree node. No matter which child we choose, we will color all faces incident to v .

It remains to analyze the number of nodes in the tree. A bound on the number of nodes can be expressed by the following recurrence:

$$P(n, f) \leq (k_v + 1)P(n - 1, f - k_v) + 1.$$

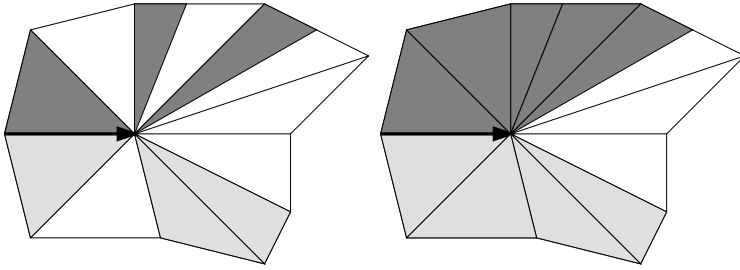


Fig. 5. Completing the red-blue coloring when entering a new vertex (light gray corresponds to red, dark gray to blue)

Because we want to maximize the number of nodes in the tree, we can assume that the k_v s for all v within a level l of the tree are equal. This holds for the root and by an inductive argument for the whole tree. Let κ_l denote the number k_v for the vertices v on level l .

$P := P(n - 2, 2n + 2)$ will give us the number of nodes in the tree. We know that $P(0, \cdot) = P(\cdot, 0) = 1$. All κ_l s have to be non-negative numbers. The κ_l along a path of length L have to fulfill the condition $\sum_{l \leq L} \kappa_l \leq 2n + 2$. A path is of length at most $n - 2$, and therefore we can bound \bar{P} by

$$1 + \sum_{L=1}^{n-2} \prod_{l \leq L} (\kappa_l + 1). \tag{1}$$

We are interested in a set κ_l which maximizes (1). Due to the convexity of (1) the maximum will be obtained when all κ_l are equal. Thus (1) is bounded by $1 + \sum_{i \leq n} (\frac{2n+2}{n-2} + 1)^i$. Therefore the exponential growth of the maximum number of cycle-paths is $O(n) \cdot 3^n$. \square

This already yields an improvement of the best known upper bound for cycles in planar graphs.

Observation 1. *The number of simple cycles on a planar graph with n vertices is bounded from above by $O(n) \cdot 3^n$.*

We improve the obtained upper bound further. For this we go back to the proof of Lemma 1. Instead of considering cycle-paths of length n , we focus on shorter cycle-paths of length αn , where $\alpha \in [0, 1]$.

Lemma 2. *Let $C_s^\alpha(n)$ be the number of simple cycles of length αn in a planar graph with n vertices and f faces. Then we have*

$$C_s^\alpha(n) \leq \left(\frac{f}{\alpha n} + 1 \right)^{\alpha n}. \tag{2}$$

Proof. We reconsider the argumentation which led to Observation 1 and notice that $P(k, f)$ will be maximized by equally distributed values of κ_l . Therefore we set $\kappa_l = f/(\alpha n)$, which proves the Lemma. \square

As final step we combine the result from Lemma 2 with the results of Section 3.1.

Theorem 5. *The number of simple cycles in a planar graph G with n vertices is bounded from above by $O(2.89278^n)$.*

Proof. An upper bound ν^n for the number of Hamiltonian cycles will always imply an upper bound for $C_s(G)$ since every simple cycle is an Hamiltonian cycle on a subgraph of G . This leads to

$$C_s(n) \leq \sum_{t \leq n} \binom{n}{t} \nu^t = (1 + \nu)^n.$$

Plugging in our bound of $\sqrt[4]{30}$ for ν yields $C_s(n) \leq 3.3404^n$, which is larger than 3^n . Responsible for this are cycles with small length. When choosing a small subset of vertices, it is unlikely that they are connected in G . Therefore the Hamiltonian cycles counted for this subset will not correspond to cycles in G . Thus we overestimate the number of small cycles.

We modify the upper bound induced by the Hamiltonian cycles such that they can express $C_s^\alpha(n)$. Every αn -cycle is a Hamiltonian cycle on a subgraph of size αn . Thus

$$C_s^\alpha(n) \leq \binom{n}{\alpha n} \nu^{\alpha n} \leq 5 \binom{n}{\alpha n} (\sqrt[4]{30})^{\alpha n}$$

Since $\sum_i \binom{n}{i} \alpha^i (1 - \alpha)^{n-i} = 1$, for $0 \leq \alpha \leq 1$ every summand of this sum is at most 1. Considering the summand for $i = \alpha n$ yields $\binom{n}{\alpha n} \leq (1/(\alpha^\alpha (1 - \alpha)^{1-\alpha}))^n$ and therefore

$$C_s^\alpha(n) = O \left(\left(\frac{\sqrt[4]{30}^\alpha}{\alpha^\alpha (1 - \alpha)^{(1-\alpha)}} \right)^n \right). \tag{3}$$

So far we know two upper bounds for $C_s^\alpha(n)$. The two bounds are shown in Figure 6. The graph of (3) is represented as dashed gray curve, whereas the graph of (2) is depicted solid black. Clearly the maximum of the lower envelope of the two functions induces an upper bound for the exponential growth of cycles in G .

One can observe that the two functions intersect in the interval $[0, 1]$ in only one point, which is approximately $\tilde{\alpha} = 0.91925$. The maximal exponential growth is realized for this α . Evaluating $C_s^{\tilde{\alpha}}(n)$ yields a bound of 2.89278^n on the number of cycles. \square

At its core the bound of $\left(\frac{f}{\alpha n} + 1\right)^{\alpha n}$ in Lemma 2 comes from *consuming* f faces in αn steps. A similar bound can be obtained by consuming edges instead. In this case we do not need a coloring scheme. In each step we get as many ways

to continue the cycle-path as the number of edges consumed in the step. This yields a bound of $(\frac{e}{\alpha n})^{\alpha n}$. For $e = 3n - 6$, $f = 2n - 4$, and $\alpha < 1$ the bound obtained by considering faces is stronger.

For this counting argument the graph does not need to be planar. With $\alpha = 1$ it yields a bound of $(\frac{e}{n})^n$ on the number of cycles in the graph. This bound has been independently observed by Sharir and Welzl [2].

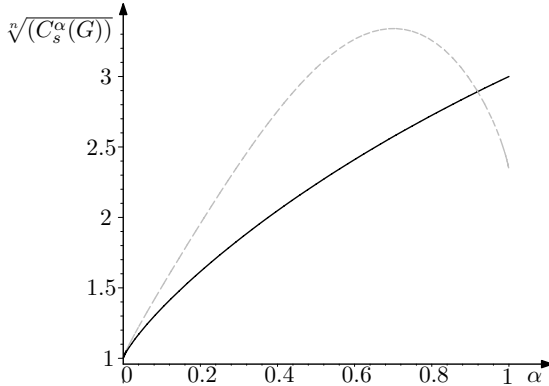


Fig. 6. Plot of the two bounds for $C_s^\alpha(G)$

4 Discussion

We improved the lower and upper bounds for the number of simple cycles in planar graphs. This reduces the gap between the upper and lower bound for the exponential growth from 1.03 to 0.46. We believe that the truth is closer to the lower bound. This is indicated by the technique sketched in the following which might further improve the upper bound.

For Observation 1 the worst case scenario is the situation where there are three possible ways to continue the cycle-path. However it is clear that this situation will not constantly occur during the construction of the cycle-paths. To use this fact we compute recurrences for the number of cycle-paths by simultaneously analyzing two or more consecutive levels of the tree which stores the cycle-paths. A careful analysis reveals other effects in this setting which also reduce the number of cycle-paths. In particular, vertices and faces will be surrounded and absorbed by the colored regions. The main part of the analysis is an intricate case distinction for which we have not checked all cases yet.

Furthermore we used the transfer matrix approach on the twisted cylinder to obtain lower bounds for other structures (for instance perfect matchings) on planar graphs. Moreover we adapted the counting procedure for sub-classes of planar graphs (for instance grid graphs). The results of these computations have not yet been double-checked and we therefore do not include them in this extended abstract.

Acknowledgments

We would like to thank Günter Rote for fruitful discussions on the subject and Andreas Stoffel for providing his implementation [\[9\]](#) of the transfer matrix method.

References

1. Alt, H., Fuchs, U., Kriegel, K.: On the number of simple cycles in planar graphs. *Combinatorics, Probability & Computing* 8(5), 397–405 (1999)
2. Sharir, M., Welzl, E.: On the number of crossing-free matchings (cycles, and partitions). In: *Proc. 17th ACM-SIAM Sympos. Discrete Algorithms*, pp. 860–869. ACM Press, New York (2006)
3. Sharir, M., Welzl, E.: Random triangulations of planar point sets. In: *22nd Annu. ACM Sympos. Comput. Geom.*, pp. 273–281. ACM Press, New York (2006)
4. Barequet, G., Moffie, M.: The complexity of Jensen’s algorithm for counting polyominoes. In: *Proc. 1st Workshop on Analytic Algorithmics and Combinatorics*, pp. 161–169 (2004)
5. Conway, A., Guttmann, A.: On two-dimensional percolation. *J. Phys. A: Math. Gen.* 28(4), 891–904 (1995)
6. Jensen, I.: Enumerations of lattice animals and trees. *J. Stat. Phys.* 102(3–4), 865–881 (2001)
7. Stanley, R.P.: *Enumerative combinatorics*, vol. 2. Cambridge University Press, Cambridge (1999)
8. Graham, A.: *Nonnegative Matrices and Applicable Topics in Linear Algebra*. John Wiley & Sons, New York (1987)
9. Stoffel, A.: Software for Counting Cycles on the Twisted Cylinder, <http://page.mi.fu-berlin.de/~schulza/cyclecount/>
10. Lovasz, L., Plummer, M.: *Matching theory*. Elsevier, Amsterdam (1986)
11. Aichholzer, O., Hackl, T., Vogtenhuber, B., Huemer, C., Hurtado, F., Krasser, H.: On the number of plane graphs. In: *Proc. 17th ACM-SIAM Sympos. Discrete Algorithms*, pp. 504–513. ACM Press, New York (2006)

An Improved Exact Algorithm for Cubic Graph TSP

Kazuo Iwama* and Takuya Nakashima

School of Informatics, Kyoto University, Kyoto 606-8501, Japan
{iwama,tnakashima}@kuis.kyoto-u.ac.jp

Abstract. It is shown that the traveling salesman problem for graphs of degree at most three with n vertices can be solved in time $O(1.251^n)$, improving the previous bound $O(1.260^n)$ by Eppstein.

1 Introduction

The CNF satisfiability (SAT) and the traveling salesman (TSP) problems are probably the two most fundamental NP-hard problems. However, there are major differences between those two problems, which seems to be especially important when we develop their exact algorithms. For example, SAT is a so-called *subset* problem whose basic search space is 2^n for n variables, while TSP is a *permutation* problem whose search space is $n!$ for n vertices. For SAT, it is enough to find *any* feasible solution (satisfiable assignment), while it is not enough to find a Hamiltonian cycle in the case of TSP; we have to find an *optimal* one. Thus TSP is intuitively much harder than SAT. Not surprisingly, compared to the rich history of exact SAT algorithms (see , e.g. [6]), TSP has a very small amount of literature. For more than four decades, we had nothing other than the simple dynamic programming by Held and Karp [11], which runs in time $O(2^n)$.

In 2003, Eppstein considered this problem for *cubic graphs*, graphs with maximum degree three [8]. His basic idea is to use the fact that if the degree is three then a selection of an edge as a part of a Hamiltonian cycle will implicationally force several edges to be in the cycle or not. He also maintained several nontrivial algorithmic ideas, e.g., the one for how to treat 4-cycles efficiently. As a result, his algorithm runs in $O(2^{n/3}) \approx 1.260^n$ time. He also showed that there is a cubic graph having $O(2^{n/3})$ different Hamiltonian cycles. Hence his algorithm cannot be improved if one tries to enumerate all Hamiltonian cycles.

Our Contribution. In this paper we give an improved time bound, $O(2^{31n/96}) \approx 1.251^n$. First of all, the above lower bound for the number of Hamiltonian cycles is not harmful since such a graph includes a lot of 4-cycles for which the special treatment is already considered in [8].

Our new algorithm is similar to Eppstein's and is based on a branching search which tries to enumerate all Hamiltonian cycles (and even more, i.e., some cycle

* Supported in part by Scientific Research Grant, Ministry of Japan, 1609211 and 16300003.

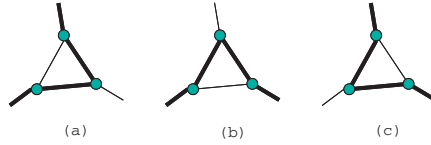


Fig. 1. Treatment of 3-cycles

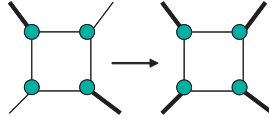


Fig. 2. Treatment of 4-cycles

decompositions). Here is our idea of the improvement: Suppose that at each branch there are two possibilities and the corresponding recurrence relations which look like

$$T(n) \leq 2T(n-1) \text{ and } T(n) \leq 3T(n-2).$$

The second equation is obviously more desirable for us, but we often have to assume the worst case (i.e., the first equation and $T(n) = 2^n$ as a result) due to the lack of specific information. If we do have useful information, for example, the information that the first case can happen at most $3n/4$ time, then it obviously helps. Note that such information can be incorporated into analysis by introducing new parameters and relations among them. In the above case, we have

$$\begin{aligned} T(n, m) &\leq 2T(n-1, m-1), \\ T(n, m) &\leq 3T(n-2, m), \\ m &\leq 3n/4 \end{aligned}$$

and it is easy to conclude $T(n) \approx 1.931^n$ by setting $T(n, m) = 2^{\alpha n + \beta m}$ and solving equations on α and β . In our analysis, we need to introduce three new parameters other than the main parameter n .

Related Work. If a given graph has a small separator, then as with many other NP-hard problems, TSP has efficient algorithms. For example the planar graphs have a $O(\sqrt{n})$ size separator [3] and the planar graph TSP can be solved in time $O(2^{10.8224\sqrt{n}}n^{3/2} + n^3)$ [9]. Euclidean TSP can also be solved in time $O(n^{\sqrt{n}})$ [5] similarly.

2 Eppstein's Algorithm

For a given cubic graph G of n vertices, we first remove all 3-cycles by merging corresponding three vertices into a single vertex and add the cost of each triangle edge to the opposite non-triangle edge. Since a Hamiltonian cycle must go

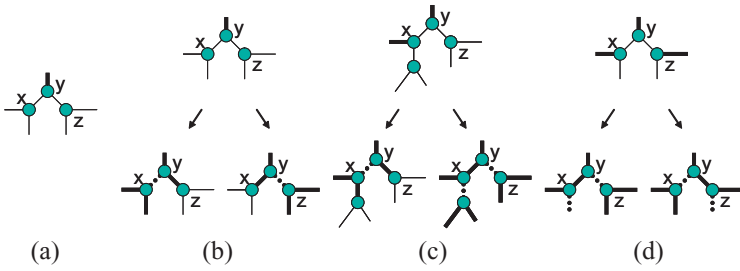


Fig. 3. Single branching of the algorithm

like Fig. 1 (a), (b), or (c) on a 3-cycle, this preprocessing works. As mentioned previously, 4-cycles also receive a special treatment: As shown in Fig. 2 if two diagonal edges attached to the cycle are selected, then the other two are automatically selected and then further decisions (whether two horizontal edges or vertical edges are selected) are postponed until the last step of the algorithm (see 8 for details).

The main part of the algorithm is illustrated in Fig. 3. Suppose that we have a vertex y such that one of three neighboring edges is already selected as a part of the solution (see Fig. 3 (a) where the selected edge is denoted by a thick line) and we try to extend the solution by selecting an edge yx or yz . Then there are three cases (b), (c) and (d) due to the state of the vertices x and z . Suppose that both x and z are *free*, i.e., neither is adjacent to already selected edges as in Fig. 3 (b). Then either selecting yx or yz to extend the solution, we can force at least three more edges to be in the solution. Suppose that one of them, say x , is not free as in (c). Then by a similar extension, we can force at least two and five edges. If neither is free as in (d), we can force at least four and four edges. In this last case, however, we have to be careful since we can only force three edges if this portion is a part of a 6-cycle as shown in Fig. 4

Thus our recurrence equation can be written as follows using the number n of edges which should be selected from now until the end of the algorithm.

$$T(n) \leq 2T(n - 3) \tag{1}$$

$$T(n) \leq T(n - 2) + T(n - 5) \tag{2}$$

$$T(n) \leq 2T(n - 4) \tag{3}$$

It is easy to see that (1) is the worst case and by solving it we have $T(n) = 2^{n/3}$. Because the length of a hamiltonian cycle of a n vertices graph is n , the number of edges which should be selected is also n . Thus, TSP for a cubic graph with n vertices can be solved in time $O(2^{n/3})$, which is the main result of 8.

3 New Algorithm

Recall that one of the worst cases happens for the case of Fig. 3 (b). Notice, however, that if this case happens once then the number of free vertices decreases

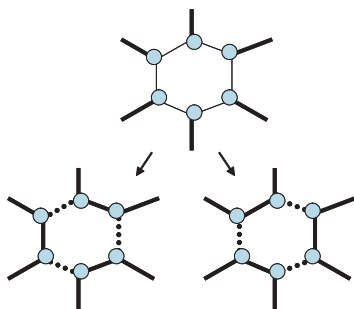


Fig. 4. 6-cycles

by four. Therefore, in each path of the backtrack tree from the root to a leaf, this worst case can happen at most $n/4$ times. Without considering this property, we could only say that the case can happen up to $n/3$ times, since three edges are selected at each step and the backtrack path ends when n edges have been selected. Thus the number of worst-case branches reduces by the new property, which should contribute to a better time complexity.

Unfortunately this idea does not work by itself, since we also have to consider the case of a 6-cycle in Fig. 4: if the six edges attaching (radiating from) the 6-cycle are all selected, then the further extension selects only three edges (every other ones in the six cycle edges) in a single branch. To make things worse, this 6-cycle case and the worst case of Fig. 3 (b) can happen alternatively without any other in between.

Now here is a natural idea. Since a 6-cycle plays a bad role only when all six attaching edges are selected *before* the cycle edges, we should select (some of) the cycle edges before the attaching edges have been all selected. This can be done by placing a priority to edges constituting 6-cycles when we extend the solution.

A 6-cycle is called *live* if none of its six cycle edges are selected. A 6-cycle which is not live is called *dead*. Fig. 9 shows the original algorithm in [8] where F shows the set of edges already selected. Our algorithm differs only in 3 (b), which is replaced by the following (b1) and (b2):

- (b1) If there is no such 4-cycle and if $G \setminus F$ contains a live 6-cycle with a vertex y which has a neighboring edge in F (that is not a cycle edge but an attaching one), let z be one of y 's neighboring vertices (on the cycle). If two or more such live 6-cycles exist, then select a 6-cycle such that most attaching edges are already selected.
- (b2) If there is no such 4-cycle or 6-cycle, but F is nonempty, then let uy be any edge in F such that y is adjacent to at least one vertex which is not free. Let yz be an edge in $G \setminus F$ between y and such a vertex ($= z$). If there is no such uy , then let uy be any edge in F and yz be any adjacent edge in $G \setminus F$.

4 Analysis of the Algorithm

Let $C(i), 0 \leq i \leq 6$, be a live 6-cycle such that i edges of its six attached edges have already been selected. Also, we call the branch shown in Fig. 3 (b) and Fig. 4 *A-branch* and *B-branch*, respectively, and all the other branches *D-branch*. Recall that A- and B-branches are our worst cases. (Note that Fig. 3 (b) may select more than three new edges if there are already selected edges near there. If that is the case, then this branch is not an A-branch any more but a D-branch.) Here is our key lemma:

Lemma 1. *Let P be a single path of the backtrack tree and suppose that we have a $C(6)$, say Q , somewhere on P . Then at least three attached edges of Q have been selected by D-branches.*

Proof. Q is of course $C(0)$ at the beginning of the algorithm, so it becomes $C(6)$ by changing its state like, for instance, $C(0) \rightarrow C(3) \rightarrow C(5) \rightarrow C(6)$. If it once becomes $C(3)$, then the lemma is true since the change from $C(3)$ to $C(6)$ is caused by a branch associated with other 6-cycle(s) which is at least $C(3)$ (see 3 (b1) of our algorithm given in the previous section). One can easily see that only D-branches can be applied to $C(3), C(4)$ and $C(5)$. Furthermore, if Q changes from $C(2)$ to $C(6)$, then it must be done by a D-branch and the lemma is true. Hence we can assume that Q becomes $C(6)$ through $C(4)$ or $C(5)$. In the following we only discuss the case that Q once becomes $C(4)$; the other case is much easier and omitted.

Now Q is $C(4)$. Its previous state is $C(0), C(1)$ or $C(2)$ (not $C(3)$ as mentioned above). If Q changes from $C(0)$ to $C(4)$ directly, then that branch must be a D-branch (A-branch can select only three edges) and we are done. The case that Q changes from $C(1)$ to $C(4)$ directly is also easy and omitted. As a result, we shall prove the following statement: Suppose that Q changes from $C(2)$ to $C(4)$ directly and finally becomes $C(6)$. Then the change from $C(2)$ to $C(4)$ must be by a D-branch and hence the lemma is true.

Suppose for contradiction that Q changes from $C(2)$ to $C(4)$ by other than D-branch. Since B-branch is obviously impossible (it never increase selected attached edges), the branch must an A-branch. Furthermore, we can assume without loss of generality that when Q was $C(2)$, there were no 6-cycles which were already $C(4)$ or more. (Namely, we are now considering such a moment that a “bad” $C(4)$ has appeared for the first time in the path of the algorithm. $C(4)$ ’s that previously appeared should have been processed already and should be dead at this moment.) Under such a situation, our first claim is that when Q becomes $C(4)$, there must be at least one other 6-cycle, Q' , which becomes also $C(4)$ at the same time. The reason is easy: If Q would be only one $C(4)$, then it must be processed at the next step and would become dead, meaning it can never be $C(6)$.

Now Suppose that a single A-branch has created two $C(4)$ ’s Q and Q' . First of all, this is impossible if Q and Q' are *disjoint* or do not share cycle edges for the following reason. If they are disjoint, we need to select four attached edges by a single A-branch. Since A-branch selects only three new edges, this would

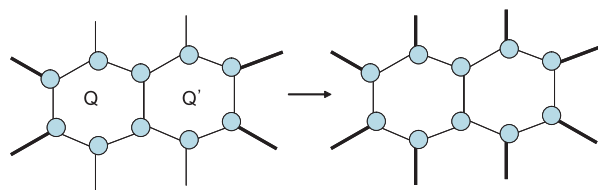


Fig. 5. Case 1

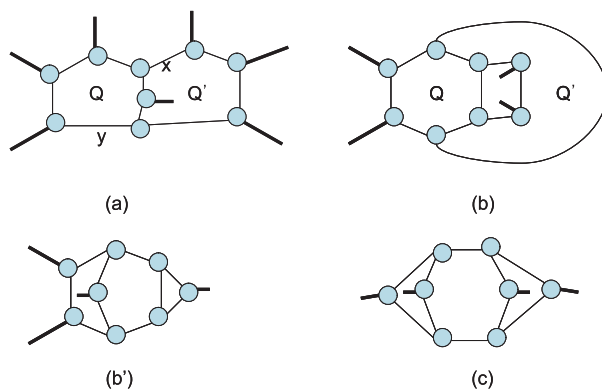


Fig. 6. Case 2

be possible only if some single edge is an attached edge of both Q and Q' . This means that this common edge was selected but no adjacent edges (i.e., cycle edges of Q and Q') are selected, which is obviously impossible. Thus Q and Q' must share some cycle edges.

Case 1. Q and Q' share one cycle edge. See Fig. 5. To change both Q and Q' from $C(2)$ to $C(4)$, we need to select four edges at a single step (recall that we have no common attaching edges), which is impossible by an A-branch.

Case 2. Q and Q' share two cycle edges. See Fig. 6 (a), (b), (b') and (c). In the case of (a), after making two $C(4)$'s, neither can be $C(6)$. (One can see that if we select x then y must be selected also.) In the case (b), we even cannot make both Q and Q' $C(4)$. (b') includes a 3-cycle, which cannot happen in our algorithm. In (c), we have a 4-cycle whose diagonal attaching edges are selected. Then, in the next step, 3 (a) of the algorithm applies and both Q and Q' become dead.

Case 3. Q and Q' share three cycle edges. See Fig. 7 (a), (b), (b') and (c), where shared edges are given by x , Q edges by straight lines and Q' edges by dotted lines. The case (a) is obviously impossible since this subgraph is completely disconnected from the rest of the graph. In the case (b), it is impossible to make Q and Q' both $C(4)$. (b') includes a 3-cycle. (c) shows the situation before the branch is made, i.e., Q and Q' are both $C(2)$. Notice that we have the third 6-cycle, say Q'' other than Q and Q' . Now Q and Q' are $C(2)$. But to be so, one can see that at least one of Q , Q' and Q'' has two attached edges such that

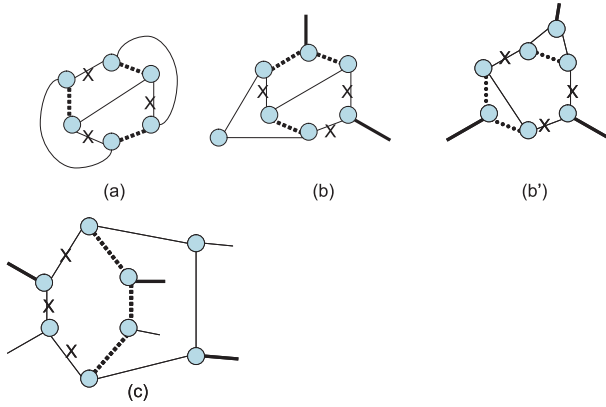


Fig. 7. Case 3

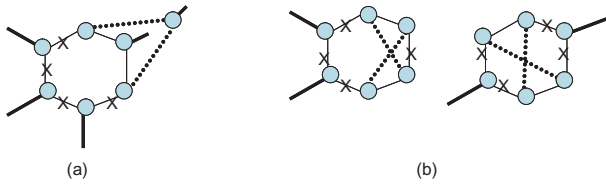


Fig. 8. Case 4

their end points apart only at most distance two on the cycle. Then, by 3 (b2) of our algorithm (see section 3), that cycle must be processed in the next step, which is not an A-branch.

Case 4. Q and Q' share four cycle edges. See Fig. 8 (a) and (b). (a) is only one possibility to make two $C(4)$'s. Then notice that we have a 4-cycle whose two diagonal attached edges are selected, which is the same as (c) of Case 2. In (b), it is obviously impossible to make two $C(4)$'s.

Case 5. Q and Q' share five cycle edges. This is impossible since our graph does not have parallel edges.

Thus we have contradictions in all the cases. □

Now we are ready to prove our main theorem. As for the correctness of the algorithm, note that our modification from [8] is very small and it is not hard to verify that the modification does not affect its correctness. Therefore we can use the analysis of [8] as it is, and so, we are only interested in its time complexity.

Theorem 1. *Our algorithm runs in time $O(n^{31n/96})$.*

Proof. Since the recurrence relation is most important, we once again summarize several different cases. (All other parts of the algorithm including the treatment of 4-cycles, runs in a polynomial time, see [8] for details.)

1. Repeat the following steps until one of the steps returns or none of them applies:
 - (a) If G contains a vertex with degree zero or one, return None.
 - (b) If G contains a vertex with degree two, add its incident edges to F .
 - (c) If F consists of a Hamiltonian cycle, return the cost of this cycle.
 - (d) If F contains a non-Hamiltonian cycle, return None.
 - (e) If F contains three edges meeting at a vertex, return None.
 - (f) If F contains exactly two edges meeting at some vertex, remove from G that vertex and any other edge incident to it; replace the two edges by a single forced edge connecting their other two endpoints, having as its cost the sum of the costs of the two replaced edges costs.
 - (g) If G contains two parallel edges, at least one of which is not in F , and G has more than two vertices, then remove from G whichever of the two edges is unforced and has larger cost.
 - (h) If G contains a self-loop which is not in F , and G has more than one vertex, remove the self-loop from G .
 - (i) If G contains a triangle xyz , then for each non-triangle edge e incident to a triangle vertex, increase the cost of e by the cost of the opposite triangle edge. Also, if the triangle edge opposite e belongs to F , add e to F . Remove from G the three triangle edges, and contract the three triangle vertices into a single supervertex.
 - (j) If G contains a cycle of four unforced edges, two opposite vertices of which are each incident to a forced edge outside the cycle, then add to F all non-cycle edges that are incident to a vertex of the cycle.
2. If $G \setminus F$ forms a collection of disjoint 4-cycles, perform the following steps.
 - (a) For each 4-cycle C_i in $G \setminus F$, let H_i consist of two opposite edges of C_i , chosen so that the cost of H_i is less than or equal to the cost of $C_i \setminus H_i$.
 - (b) Let $H = \cup_i H_i$. Then $F \cup H$ is a degree-two spanning subgraph of G , but may not be connected.
 - (c) Form a graph $G' = (V', E')$, where the vertices of V' consist of the connected components of $F \cup H$. For each set H_i that contains edges from two different components K_j and K_k , draw an edge in E' between the corresponding two vertices, with cost equal to the difference between the costs of C_i and of H_i .
 - (d) Compute the minimum spanning tree of (G', E') .
 - (e) Return the sum of the costs of $F \cup H$ and of the minimum spanning tree.
3. Choose an edge yz according to the following cases:
 - (a) If $G \setminus F$ contains a 4-cycle, two vertices of which are adjacent to edges in F , let y be one of the other two vertices of the cycle and let yz be an edge of $G \setminus F$ that does not belong to the cycle.
 - (b) If there is no such 4-cycle, but F is nonempty, let xy be any edge in F and yz be an adjacent edge in $G \setminus F$.
 - (c) If F is empty, let yz be any edge in G .
4. Call the algorithm recursively on $G, F \cup \{yz\}$.
5. Call the algorithm recursively on $G \setminus \{yz\}, F$.
6. Return the minimum of the set of at most two numbers returned by the two recursive calls.

Fig. 9. Eppstein's algorithm

- (1) Fig. 3 (b) happens. In either branch, three new edges are selected and four free vertices disappear. (As mentioned before, it may happen that more than three edges are selected due to the existence of nearby selected edges. But this is obviously more desirable for us and can be omitted.)
- (2) Fig. 4 happens. In either branch, three new edges are selected and the number of free vertices does not change at all.
- (3) Fig. 3 (c) happens. In one branch, two new edges are selected and in the other branch five. At least two free vertices disappear in either branch.
- (4) Fig. 3 (d) happens. In either branch, four new edges are selected and at least two free vertices disappear. However, we have to be careful here again. Now we have to consider an 8-cycle such that its attaching edges have been all selected and none of its cycle edges have been selected. In this case, by selecting one edge in the cycle, four edges are forced to be selected in either branch. However, the number of free vertices does not change and so this case is worse than Fig. 3 (d).

Now let $T(n, a, b, f)$ be the number of nodes that appear in the backtrack tree for a graph G such that in each path starting from this node to a leaf, (i) we further need to select n edges, (ii) the extension of Fig. 3 (b) type happens a times from now on, (iii) the extension of Fig. 4 happens b times from now on, and (iv) G has f free vertices. Then by the case (1) to case (4) above, we have the following recurrence relation:

$$T(n, a, b, f) \leq \max \begin{cases} 2T(n-3, a-1, b, f-4) \\ 2T(n-3, a, b-1, f) \\ T(n-5, a, b, f-2) + T(n-2, a, b, f-2) \\ 2T(n-4, a, b, f) \end{cases}$$

Note that in the case of (1), we have now used this type of extension once, we can decrease the value of a by one, and similarly for (2). Also, at each leaf, we have $T(0,0,0,0) = 1$.

Now let $T(n, a, b, f) = 2^{\frac{n+\frac{1}{2}(a+2b)+f/8}{4}}$. Then as shown below, this function satisfies all the recurrence formulas above:

$$2T(n-3, a-1, b, f-4) = 2^{\frac{(n-3)+\frac{1}{2}((a-1)+2b)+(f-4)/8}{4}+1} = 2^{\frac{n+\frac{1}{2}(a+2b)+f/8}{4}}$$

$$2T(n-3, a, b-1, f) = 2^{\frac{(n-3)+\frac{1}{2}(a+2(b-1))+f/8}{4}+1} = 2^{\frac{n+\frac{1}{2}(a+2b)+f/8}{4}}$$

$$2T(n-4, a, b, f) = 2^{\frac{(n-4)+\frac{1}{2}(a+2b)+f/8}{4}+1} = 2^{\frac{n+\frac{1}{2}(a+2b)+f/8}{4}}$$

$$T(n-5, a, b, f-2) + T(n-2, a, b, f-2) = 2^{\frac{(n-5)+\frac{1}{2}(a+2b)+(f-2)/8}{4}} + 2^{\frac{(n-2)+\frac{1}{2}(a+2b)+(f-2)/8}{4}}$$

$$= (2^{\frac{-5-2/8}{4}} + 2^{\frac{-2-2/8}{4}})2^{\frac{n+\frac{1}{2}(a+2b)+f/8}{4}}$$

$$> (1.07)2^{\frac{n+\frac{1}{2}(a+2b)+f/8}{4}} > 2^{\frac{n+\frac{1}{2}(a+2b)+f/8}{4}}$$

$$T(0, 0, 0, 0) = 2^0 = 1$$

Here we use Lemma 1, which says that if type (2) happens b times in the path, then at least $3b$ edges are selected by neither type (1) or type (2). Since we select $3a + 3b$ edges by types (1) and (2) and the total number of selected edges is n , we have

$$3a + 3b + 3b \leq n,$$

which means $a + 2b \leq n/3$. Also $f \leq n$ obviously. Using these two inequalities, we have

$$2^{\frac{n + \frac{1}{2}(a+2b) + n/8}{4}} \leq 2^{\frac{n + \frac{1}{2}n/3 + n/8}{4}} = 2^{31n/96}. \quad \square$$

References

1. Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. *SIAM Journal on Applied Mathematics* 10, 196–210 (1962)
2. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman (1979)
3. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics* 36, 177–189 (1979)
4. Lipton, R.J., Tarjan, R.E.: Applications of a planar separator theorem. *SIAM Journal on Computing* 9, 615–627 (1980)
5. Hwang, R.Z., Chang, R.C., Lee, R.C.T.: The Searching over Separators Strategy To Solve Some NP-Hard Problems in Subexponential Time. *Algorithmica* 9, 398–423 (1993)
6. Iwama, K., Tamaki, S.: Improved Upper Bounds for 3-SAT, 15th annual ACM-SIAM Symposium on Discrete Algorithms. In: *Proc. SODA*, January 2004, pp. 328–329 (2004)
7. Woeginger, G.J.: 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)
8. Eppstein, D.: The Traveling Salesman Problem for Cubic Graphs. In: Dehne, F., Sack, J.-R., Smid, M. (eds.) *WADS 2003*. LNCS, vol. 2748, pp. 307–318. Springer, Heidelberg (2003)
9. Dorn, F., Penninkx, E., Bodlaender, H.L., Fomin, F.V.: Efficient Exact Algorithms on Planar Graphs: Exploiting Sphere Cut Branch Decompositions. In: Brodal, G.S., Leonardi, S. (eds.) *ESA 2005*. LNCS, vol. 3669, pp. 95–106. Springer, Heidelberg (2005)

Geometric Intersection Graphs: Do Short Cycles Help? (Extended Abstract)

Jan Kratochvíl* and Martin Pergel**

Department of Applied Mathematics, Charles University, Malostranské nám. 25,
118 00 Praha 1, Czech Republic
{honza, perm}@kam.mff.cuni.cz

Abstract. Geometric intersection graphs are intensively studied both for their practical motivation and interesting theoretical properties. Many such classes are hard to recognize. We ask the question if imposing restrictions on the girth (the length of a shortest cycle) of the input graphs may help in finding polynomial time recognition algorithms. We give examples in both directions. First we present a polynomial time recognition algorithm for intersection graphs of polygons inscribed in a circle for inputs of girth greater than four (the general recognition problem is NP-complete). On the other hand, we prove that recognition of intersection graphs of segments in the plane remains NP-hard for graphs with arbitrarily large girth.

1 Introduction

Intersection graphs are defined as graphs with representations by set-systems of certain types. Each set corresponds to a vertex and two vertices are adjacent iff the corresponding sets have nonempty intersection. Any graph can be represented by some set-system, but interesting and nontrivial graph classes are obtained when further restrictions are imposed on the sets representing the vertices. Especially geometrical representations are popular and widely studied. They stem from practical applications, have many interesting structural and algorithmic properties, and often serve as a method of graph visualization. Numerous examples include interval graphs, circle graphs, circular arc graphs, boxicity two graphs, and many others, cf. e.g., [16, 22]. (In all examples that are mentioned in this paper, representations are assumed in the Euclidean plane.)

Many of these classes are NP-hard to recognize (e.g., boxicity two graphs [12], intersection graphs of segments, convex sets or curves in the plane [11], intersection graphs of unit and general disks in the plane [11, 13], contact graphs of curves and unit disks [8, 11], and others, see [2]). It has been observed in [13] that triangle-free disk graphs are planar and hence recognizable in polynomial

* Support of Institute for Theoretical Computer Science (ITI), a project of Czech Ministry of Education 1M0021620808 is gladly acknowledged.

** Supported by Project 201/05/H014 of the Czech Science Foundation (GAČR).

time (in fact, the observation was done for a larger class of pseudodisk graphs), but that forbidding triangles does not help recognizing intersection graphs of curves (so called string graphs). In the present paper we propose a more thorough study of this phenomenon and we prove two results showing that forbidding short cycles does help sometimes, but not always. We first introduce the relevant graph classes in more detail.

Polygon-circle graphs (shortly *PC-graphs*) are intersection graphs of convex polygons inscribed into a circle. They extend e.g., interval graphs, circular-arc graphs, circle graphs, and chordal graphs. M. Fellows observed that this class is closed under taking induced minors [personal communication, 1988]. These graphs are also interesting because the Clique and Independent Set problems can be solved in polynomial time [6]. On the other hand, determining their chromatic number is NP-hard, since PC-graphs contain circle graphs [7], but the graphs are near-perfect in the sense that their chromatic number is bounded by a function of their clique number [10]. The recognition problem for PC-graphs is in the class NP, since every PC-graph with n vertices has a representation whose all polygons have at most n corners each, and an asymptotically tight bound $n - \log n + o(\log n)$ on the maximum number of corners needed in a representation is presented in [15]. A polynomial time recognition algorithm was announced in [9], but a full paper containing the algorithm was never published. On the contrary, the recognition has recently been shown NP-complete [18]. We show that restricting the girth (the length of a shortest cycle) of the input graphs does make the recognition easier.

Theorem 1. *PC-graphs of girth greater than four can be recognized in polynomial time.*

Segment (or SEG-) graphs are intersection graphs of segments in the plane. These graphs were first considered in [5] where it is shown that determining the chromatic number of these graphs is NP-hard. The near-perfectness of these graphs is a well known open problem going back to Erdős. The recognition of SEG-graphs is shown NP-hard in [11] in a uniform reduction which also shows NP-hardness of recognition of *string graphs* (intersection graphs of curves), *1-string graphs* (intersection graphs of nontangent curves such that each two curves intersect in at most one point) and *CONV-graphs* (intersection graphs of convex sets). SEG- and CONV-graphs have been further studied in [14] where their recognition is shown to be in the class PSPACE. Membership in the class NP is still open. On the contrary, a surprising development occurred for the most general class of string graphs. This class, also mentioned in [5], was introduced already in 1966 in [21]. For decades no recursive algorithm for its recognition was known, until 2001 when two bounds on the number of crossing points needed in their representations were proved independently in [17][19] (yielding recursive recognition algorithms), followed by the proof of NP-membership in 2002 in [20]. The complexity of recognition is connected to the question of maximum size of a representation. The construction of [14] showing that there are SEG-graphs requiring representations of double exponential size (the size of a representation by segments with integral coordinates of endpoints is the maximum

absolute value of the endpoint coordinates) shows that the would-be-straightforward ‘guess-and-verify’ algorithm is not in NP. Another long-standing open problem is whether every planar graph is a SEG-graph (it is easy to show that every planar graph is a string graph, and this was recently improved in [3] showing that every planar graph is a 1-string graph). Regarding the question of recognizing graphs of large girth, it was noted in [13] that triangle-free string graphs are NP-hard to recognize, but all known reductions hinge on the presence of cycles of length 4. We show that for SEG-graphs, this is not the case, and in fact no girth restriction would help.

Theorem 2. *For every k , recognition of SEG-graphs of girth greater than k is NP-hard.*

2 Polygon-Circle Graphs

2.1 Preliminaries

We first explain the terminology. All graphs considered are finite and undirected. For a graph G and two disjoint subsets U and H of its vertex set, we denote by $G[U]$ the subgraph induced by U , and we denote by $E_G(U, H)$ the edges of G between U and H , i. e., $E_G(U, H) = E(G[U \cup H]) \setminus (E(G[U]) \cup E(G[H]))$.

Given a PC-graph G and a PC-representation, we refer to *vertices* of the graph and to *corners* of the polygons (these are their vertices lying on the geometrical bounding circle). For a vertex v of the graph, we use P_v to denote the corresponding polygon representing v (but sometimes and namely in figures, we avoid multiple subscripts by denoting also the polygon simply by v). Given a representation R of G with $G \subseteq H$, we say that R is *extended into* a representation S of H if S is obtained from R by adding new corners to existing polygons and by adding new polygons representing the vertices of $V(H) \setminus V(G)$.

In a PC representation, we describe the relative position of a set of polygons in terms of “visibility” from one polygon to another. The corners of any polygon R divide the bounding circle into circular arcs referred to as *R-segments*. If P, Q, R are disjoint polygons and Q lies in a different P -segment than R , we say that Q is *blocked* from R by P . A set A of polygons is said to *lie around the circle* if for no triple of disjoint polygons $P, Q, R \in A$, the polygon Q is blocked by R from P .

It is easy to check that we may assume that the corners of all polygons are distinct. By cutting the bounding circle in an arbitrary point and listing the names of the vertices in the order as the corners of their polygons appear along the circle we obtain the so called *alternating sequence* of the representation. To be more precise, we say that two symbols a and b *alternate* in a sequence S , if S contains a subsequence of the form $\dots a \dots b \dots a \dots b \dots$ or $\dots b \dots a \dots b \dots a \dots$. It is a well-known fact that $G = (\{v_1, \dots, v_n\}, E)$ is a PC-graph if and only if there exists an alternating sequence S over the alphabet v_1, \dots, v_n such that $v_i v_j \in E(G)$ if and only if v_i and v_j alternate in S .

Though polygon-circle representations lie behind the original geometric definition of PC-graphs and are visually more accessible, precise proofs of several

observations and auxiliary technical lemmas are easier to formulate in the language of alternating sequences. Thus in the paper we often switch between these two equivalent descriptions of PC-representations.

2.2 PC-Graphs of Low Connectivity and Their Decompositions

We first observe that we may restrict our attention to bi-connected (vertex-2-connected) graphs.

Lemma 1. *A graph G is a PC-graph if and only if all its connected components are PC-graphs.*

Proof. We just take alternating representations of individual components and place one after another.

Lemma 2. *A graph G is a PC-graph if and only if all its biconnected components are PC-graphs.*

Proof. Similar to the proof of Lemma [1](#) □

From now on we assume that our input graphs are bi-connected. It turns out that vertex cuts containing two nonadjacent vertices do not create any problems, but those containing two adjacent vertices do. We formalize this in two definitions of decompositions in the next subsection.

Definition 1. *Let $G = (V, E)$ be a graph. An edge ab whose endpoints form a cut in G is called a cutting edge. Let C_1, C_2, \dots, C_k be the connected components of $G[V \setminus \{a, b\}]$ for a cutting edge ab . The pair-cutting decomposition of G based on ab is the collection of graphs $G_i^{ab} = (C_i \cup \{a, b, c\}, E(G[C_i \cup \{a, b\}]) \cup \{ac, bc\})$, where c is a new extra vertex, $i = 1, 2, \dots, k$.*

A graph is called pc-prime (pair-cutting-prime) if it has no nontrivial pair-cutting decomposition (i.e., if every cutting edge divides the graph into one connected component and a single vertex).

Proposition 1. *A biconnected graph G with a cutting edge ab is a PC-graph if and only if all the graphs in the pair-cutting decomposition based on ab are PC-graphs.*

Proof. We start with alternating representations S_1, \dots, S_k of the graphs $G_1^{ab}, \dots, G_k^{ab}$ in the pair-cutting decomposition. Without loss of generality we may assume that each of them begins with the newly added vertex c . Let \tilde{S}_i be obtained from S_i by removing all occurrences of c and adding an occurrence of a to the beginning and an occurrence of b to the end of the sequence. Then the concatenation $\tilde{S}_1 \dots \tilde{S}_k$ is an alternating representation of G .

Obviously a and b alternate. By our transformation we could not remove any alternation among vertices of G . Thus it suffices to check that we did not add new ones. For $i \neq j$, any symbol from \tilde{S}_i distinct from a and b cannot alternate with any symbol from \tilde{S}_j (distinct from a and b). Suppose that by replacing the first occurrence of c by a we have created a new alternation of a and $x \neq b$ in \tilde{S}_i . Thus in S_i all occurrences of a lie between two occurrences of x . But a and c alternated in S_i ,

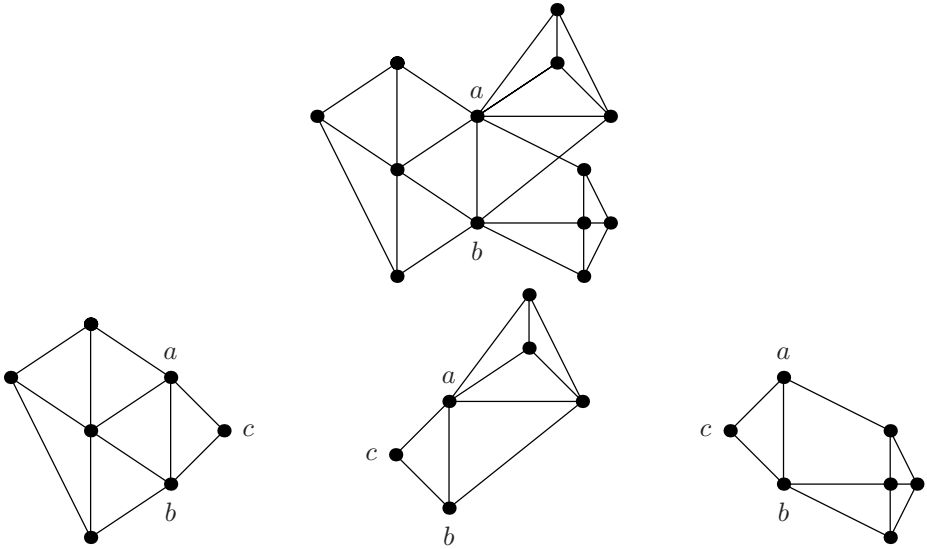


Fig. 1. An example of the pair-cutting decomposition. At the top of the figure is the original graph with vertices a and b forming a cutting edge. Below are depicted the three elements of its pair-cutting decomposition based on ab . Note that the left one and the right one are pc-primes, but the middle one is not.

thus between these two occurrences of x there is an occurrence of c and thus x and c alternated. Hence $x = b$ (c alternated only with a and b). A contradiction.

The converse is clear from the fact that the class of PC-graphs is closed under taking induced minors and the fact that each G_i^{ab} is an induced minor of G . Indeed, for some $j \neq i$, contract all vertices of C_j into a single vertex c and contract all vertices of all remaining C_l 's for $l \neq i, j$ into the vertex a . Since G was biconnected and vertices a and b formed a cut, c is adjacent exactly to a and b .

The previous proposition is general in the sense that it does not require large girth of G . However, as we aim at graphs of large girth, the reduction is inconvenient because it creates triangles in the blocks of the decomposition. For this sake we introduce the following adjustment. In each element of the pair-cutting decomposition $G[V(C_i) \cup \{a, b\}]$ we mark the edge ab "red" instead of adding the triangle abc . We will talk about the *red-edged decomposition* and each element of this decomposition will be called a *red-edged graph*. It is not true anymore that a biconnected graph is in PC if and only if each red-edged component of the decomposition is a PC-graph. However, we can still control how to glue representations together if G has no short cycles.

In the sense of Definition \square we denote by G^e the graph obtained from G by adding a new vertex adjacent to the endvertices of e . Similarly, G^{e_1, \dots, e_k} is the graph obtained by adding k new vertices, each connected to the endpoints of one of the edges e_i .

Lemma 3. *Let R be an alternating representation of a graph G . If this representation can be extended to a representation of G^{ab} , it can be done so by adding the subsequence $cabc$ between some two consecutive symbols in R .*

Proof. Will be presented in the journal version.

Proposition 2. *Let G be a connected red-edged graph of girth at least 4 with $Q = \{e_1, \dots, e_k\}$ being the set of its red edges. Let R be a PC-representation of G . If for any $e \in Q$, the representation R can be extended into a representation of G^e , then R can be extended into a representation of G^{e_1, \dots, e_k} .*

Proof. Will be presented in the journal version.

2.3 Pseudoears

A well-known characterization of biconnected graphs is given by the *ear-decomposition lemma*:

Lemma 4. *A biconnected graph can be constructed from any of its cycles by consecutive addition of paths (with endpoints in the already constructed subgraph, so called ears) and/or single edges.*

However, we need a version where the constructed graph is an induced subgraph, and therefore we need to avoid adding edges. Thus we define the technical notion of a pseudoear and introduce a special version of this lemma for K_3 -free graphs.

Definition 2. *Let H be an induced subgraph of a graph G , and let $U = a_1 \dots a_3$ be an induced path in H of length at least 2. A pseudoear attached along U is an induced path $P = p_1 \dots p_k$ in $G \setminus H$ of length at least 1 such that either*

- I. H has length at least 3 and $E(P, H) = \{p_1 a_1, p_k a_3\}$, or*
- II. $U = a_1 a_2 a_3$ has length 2 and $E(p_1, H) = \{p_1 a_1\}$, $E(p_k, H) = \{p_k a_3\}$, and $E(\{p_2, \dots, p_{k-1}\}, H) \subseteq \{p_i a_2 \mid i = 2, \dots, k - 1\}$.*

Lemma 5. *Any biconnected K_3 -free graph without cutting edges can be constructed from any of its induced cycles by consecutive addition of pseudoears and/or single vertices adjacent to at least two vertices of the so far constructed*

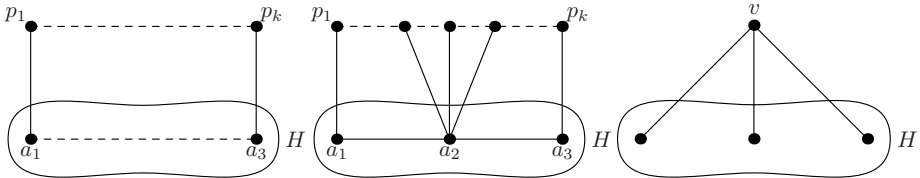


Fig. 2. The first two pictures show pseudoears of type I and II, the third one is an example of a vertex we operate with in our version of ear-decomposition lemma. Solid lines denote edges, dashed lines represent paths.

subgraph. On the other hand, any K_3 -free graph constructed in this way is bi-connected and has no cutting edges.

Proof. Will be presented in the journal version. We only note here that the proof is constructive and yields a polynomial time algorithm.

2.4 Minimal Representations of Biconnected Pc-Prime Graphs

We call an alternating representation of a PC-graph G *minimal* if deleting any occurrence of any symbol from the sequence results in a sequence that does not represent G anymore. For instance, in every minimal alternating representation of a cycle of length n , every symbol occurs exactly twice and the representation is unique (up to rotation and symmetrical flip). The core theorem our algorithm is based on says that this is also true for bi-connected pc-prime graphs of large girth.

Theorem 3. *Every bi-connected pc-prime graph with girth at least 5 has at most 1 minimal PC-representation. This representation (if it exists) can be found in polynomial time.*

Proof. Will be presented in journal version. It uses the Pseudoear lemma and by induction on the number of pseudoears shows how to find this unique representation (when it exists).

2.5 Algorithms

In this subsection we formalize our algorithm and estimate its running time.

Algorithm 1:

Input: A biconnected graph G

Output: Decision whether G has a PC-representation.

Auxiliary variables: \mathcal{H} – set of graphs, \mathcal{S} – set of representations, initially empty

Add G to set \mathcal{H} .

```

while exists  $F \in \mathcal{H}$  with a cutting-edge  $ab$  do
    remove  $F$  from  $\mathcal{H}$  and replace it by the elements of the red-edged
    decomposition of  $F$  with respect to  $ab$ .
done
forall  $F \in \mathcal{H}$  do
    if Algorithm2( $F$ )='false' then return 'false'
    else
        add the representation of  $F$  provided by
        Algorithm2( $F$ ) into  $\mathcal{S}$ 
done
forall  $R \in \mathcal{S}$  do
    for  $e \in \text{red\_edges\_of}(\text{graph\_of}(R))$  do
        if Algorithm3( $R, e$ )='false' then return 'false';
return 'true';

```

The correctness of Algorithm 1 follows from Lemmas 1, 2 and Propositions 1, 2.

Next we present the algorithm finding a representation of a pc-prime graph as the rest is either brute-force or obvious (and was described above). In Algorithm 2 we call an arc of a bounding circle between two corners of a polygon a *sector*.

Algorithm 2:

Input: pc-prime graph G .

Output: PC-representation or **false**.

Auxiliary variables: Graph H , initially empty.

```

while ( $|V(H)| < |V(G)|$ ) do
  find a pseudoear or a single vertex  $P$  attachable to  $H$ ;
  by brute force represent it;
  if the representation is impossible then return false;
  add  $P$  to  $H$ ;
done

```

When looking for pseudoears (or single vertices), we implement the (constructive) proof of Lemma 5, while finding a suitable representation follows the proof of Proposition 3 (none of these proofs is included in this extended abstract). Note that looking for a pseudoear or a single vertex is (naively) possible in $O(n^2)$, representing a pseudoear is possible in $O(n^4)$ (looking for feasible sector, trying different placements of p_1 and p_k), representing of single vertex is possible in $O(n^5)$. Pseudoears (or single vertices) are added only linearly many times. Thus the complexity of Algorithm 2 is $O(n^6)$.

Algorithm 3:

Input: Alternating representation \mathcal{R} of a graph G and a red edge $ab = e$ in G (c is not a vertex of G)

Output: Decision whether G^e has PC-representation.

By brute force try to add cab between all consecutive pairs of symbols into the alternating representation and check whether a correct representation is obtained.

If we at least once succeed, return representation; else return 'false';

The algorithm is just brute force, but polynomial ($O(n^4)$ as the length of representation is at most n^2). Its correctness is obvious from Observation 1. The total complexity of all algorithms together is therefore $O(n^7)$.

3 Segment Intersection Graphs

To prove the desired NP-hardness result, we show a polynomial reduction of the problem P3CON3SAT(4) to the recognition problem. P3CON3SAT(4) is the SATISFIABILITY problem restricted to formulas Φ with variables v_1, \dots, v_n and clauses C_1, \dots, C_m in CNF such that each clause contains exactly 3 literals,

each variable occurs in Φ at most 4 times (positively or in negation) and the bipartite graph

$$G(\Phi) = \left(\bigcup_{i=1}^n \{v_i\} \cup \bigcup_{j=1}^m \{C_j\}, \{\{v_i, C_j\} | v_i \in C_j\} \right)$$

is planar and (vertex) 3-connected. This variant of SATISFIABILITY is shown NP-complete in [12] and we will use it in the proof of Theorem 2.

We use a construction similar to [11]. Given a formula Φ , we use $G(\Phi)$ to construct a graph H with girth at least k such that H is a SEG-graph if and only if Φ is satisfiable. We modify the graph $G(\Phi)$ in the following way. Each vertex corresponding to a variable is replaced by a circle of length $\max\{16, k\}$. We replace each edge of $G(\Phi)$ by a pair of long-enough paths, and pairs of paths leaving the same variable gadget are linked by so called cross-over gadgets. Each vertex corresponding to a clause is replaced by a clause gadget. The assignment to variables is obtained from the orientation of the circle representing the particular variable. The "orientation" of pair of paths from the vertex gadget to a clause gadget describes whether the respective occurrence in the clause is positive or negative. By "orientation" of a pair of paths we mean that one path is "to the left" to the other. This has now become a standard trick used in NP-hardness reductions of geometric flavor.

The variable gadgets do not differ much from those used in [11], only the bounding cycle of the variable gadget may need to be artificially extended to make it meet the girth requirement. The clause gadget is slightly modified and it is depicted in Fig. 4. The main novelty of the present proof is the cross-over gadget depicted in Fig. 3.

The arguments are based on the fact that two intersecting segments share only one common point. The details of the proofs will be presented in the journal version.

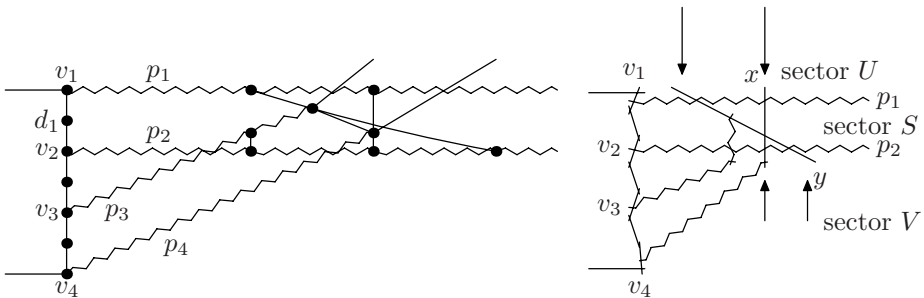


Fig. 3. The cross-over gadget and a representation starting from the variable gadget. Broken lines denote sequences of arbitrarily many segments (depending on the required girth). Arrows in the representation propose two possibilities for the respective paths – either to cross or to touch only. Note that the clause gadget is surrounded by a cycle, therefore a *sector S* is well-defined as a place surrounded by p_1, p_2, v_1, d_1, v_2 and the corresponding boundary-part of the clause gadget. As $G(\Phi)$ is 3-connected, even the sectors U and V are well-defined as the place above p_1 (distinct from S) and below p_2 , respectively. Moreover S, U and V describe disjoint regions.

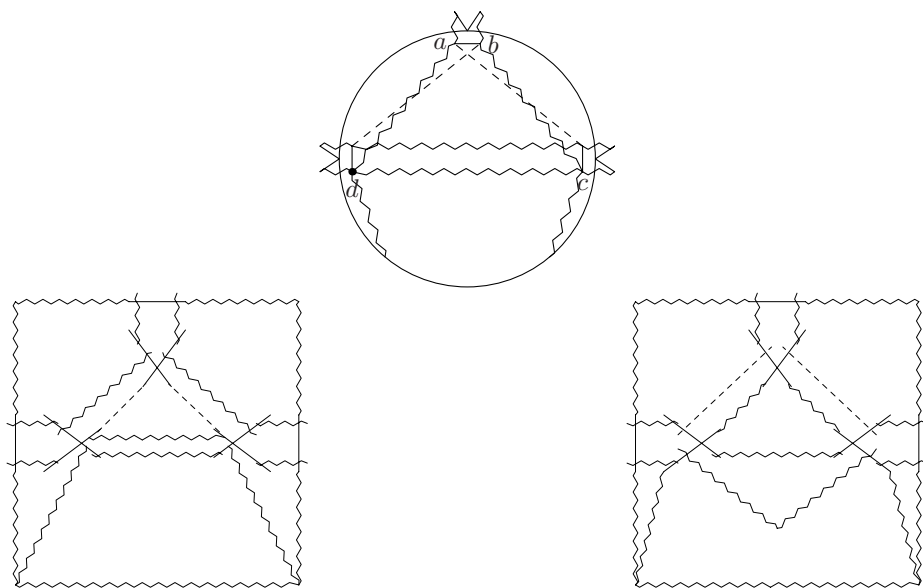


Fig. 4. The clause gadget (at the top, depicting the “all-true” position of the literals involved in it). Dashed and broken lines denote paths of lengths depending on k . The bottom two figures are examples of representations, the left one corresponds to the “false-true-false” valuation of the literals, the right one corresponds to “true-false-true” (the order of literals being left-top-right). Other cases can be represented similarly. The “all-false” position has no representation, even by curves with 1 crossing per pair.

4 Conclusion

We have shown that restricting the girth of the input graph helps with recognizing polygon-circle graphs, but that recognition of segment intersection graphs remains NP-hard. We believe that this phenomenon is worth of studying for other graph classes and that interesting results may be obtained. Also our results leave some room for strengthening. Several particular presently open cases are

- String graphs of girth at least k (known NP-hard only for $k = 4$).
- Polygon-circle graphs of girth 4.
- Intersection graphs of convex sets of girth at least k .

References

1. Breu, H., Kirkpatrick, D.G.: Unit Disk Graph Recognition is NP-Hard. *Comput. Geom. Theory and Applications* 9(1-2), 3-24 (1998)
2. Brandstaedt, A., Le, V.B., Spinrad, J.: *Graph Classes: A Survey*. SIAM (1999)

3. Chalopin, J., Goncalves, D., Ochem, P.: Planar graphs are in 1-STRING, to appear in: Proceedings of SODA 2007
4. Dangelmayr, C., Felsner, S.: Chordal Graphs as Intersection Graphs of Pseudosegments. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 208–219. Springer, Heidelberg (2007)
5. Ehrlich, G., Even, S., Tarjan, R.E.: Intersection graphs of curves in the plane. *J. Combin. Theory Ser. B.* 21, 8–20 (1976)
6. Gavril, F.: Maximum weight independent sets and cliques in intersection graphs of filaments. *Information Processing Letters* 73(5–6), 181–188 (2000)
7. Garey, M.R., Johnson, D.S., Miller, G.L., Papadimitriou, C.H.: The complexity of coloring circular arcs and chords. *SIAM Journal of Algebraic and Discrete Methods*, vol. 1(2) (1980)
8. Hliněný, P.: Classes and recognition of curve contact graphs. *J. Combin. Theory ser. B.* 74, 87–103 (1998)
9. Koebe, M.: On a New Class of Intersection Graphs. In: Proceedings of the Fourth Czechoslovak Symposium on Combinatorics Prachatice, pp. 141–143 (1990)
10. Kostochka, A., Kratochvíl, J.: Covering and coloring polygon-circle graphs. *Discrete Math.* 163, 299–305 (1997)
11. Kratochvíl, J.: String graphs. II. Recognizing string graphs is NP-hard. *Journal of Combinatorial Theory, Series B* 52, 67–78 (1991)
12. Kratochvíl, J.: A special planar satisfiability problem and a consequence of its NP-completeness. *Discr. Appl. Math.* 52, 233–252 (1994)
13. Kratochvíl, J.: Intersection Graphs of Noncrossing Arc-Connected Sets in the Plane. In: North, S.C. (ed.) GD 1996. LNCS, vol. 1190, pp. 257–270. Springer, Heidelberg (1997)
14. Kratochvíl, J., Matoušek, J.: Intersection Graphs of Segments. *Journal of Combinatorial Theory, Series B* 62, 289–315 (1994)
15. Kratochvíl, J., Pergel, M.: Two Results on Intersection Graphs of Polygons. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 59–70. Springer, Heidelberg (2004)
16. McKee, T.A., McMorris, F.R.: Topics on Intersection Graphs, SIAM (1999)
17. Pach, J., Tóth, G.: Recognizing string graphs is decidable. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 247–260. Springer, Heidelberg (2002)
18. Pergel, M.: Recognition of Polygon-circle Graphs and Graphs of Interval Filaments is NP-complete, in preparation
19. Schaefer, M., Štefankovič, D.: Decidability of string graphs, In: STOC, pp. 241–246 (2001)
20. Schaefer, M., Sedgwick, E., Štefankovič, D.: Recognizing String Graphs in NP. In: STOC 2002, pp. 1–6 (2002)
21. Sinden, F.W.: Topology of thin film RC-circuits. *Bell System Technological Journal*, pp. 1639–1662 (1966)
22. Spinrad, J.: Efficient Graph Representations, Fields Institute Monographs 19, American Mathematical Society (2003)

Dimension, Halfspaces, and the Density of Hard Sets*

Ryan C. Harkins and John M. Hitchcock

Department of Computer Science, University of Wyoming

Abstract. We use the connection between resource-bounded dimension and the online mistake-bound model of learning to show that the following classes have polynomial-time dimension zero.

1. The class of problems which reduce to nondense sets via a majority reduction.
2. The class of problems which reduce to nondense sets via an iterated reduction that composes a bounded-query truth-table reduction with a conjunctive reduction.

As corollary, all sets which are hard for exponential time under these reductions are exponentially dense. The first item subsumes two previous results and the second item answers a question of Lutz and Mayordomo. Our proofs use Littlestone's Winnow2 algorithm for learning r -of- k threshold functions and Maass and Turán's algorithm for learning halfspaces.

1 Introduction

Recent work has found applications of computational learning theory to the resource-bounded measure [10] and dimension [12] of complexity classes. Lindner, Schuler, and Watanabe [8] studied connections between computational learning theory and resource-bounded measure [10], primarily focusing on the PAC (probably approximately correct) model. They also observed that any admissible subclass of P/poly that is learnable in Littlestone's online mistake-bound model [9] has p-measure 0. This observation was later developed into a general tool for resource-bounded dimension [6]. To show that a class has p-dimension 0, it suffices to show that it is reducible to a learnable concept class family. This idea was used to show that the following classes have p-dimension 0.

- (1) $P_{\text{ctt}}(\text{DENSE}^c)$.
- (2) $P_{\text{dtt}}(\text{DENSE}^c)$.
- (3) $P_{n^{\alpha-\text{T}}}(\text{DENSE}^c)$, for all $\alpha < 1$.

Here $P_r(\text{DENSE}^c)$ is the class of all problems which reduce to nondense sets under \leq_r^p reductions, where a problem is nondense if its census function is subexponential. The result for (3) improved previous work [4,13,15] and solved one of Lutz and Mayordomo's twelve problems in resource-bounded measure [14].

* This research was supported in part by NSF grant 0515313.

The classes in (2) and (3) were reduced to disjunctions, which can be learned by Littlestone’s Winnow algorithm [9]. We obtain further results in this direction using more sophisticated learning algorithms and concept classes that generalize disjunctions.

In our first result we show that the class

$$(4) \text{P}_{\text{maj}}(\text{DENSE}^c)$$

of problems which reduce to nondense sets via majority reductions has p-dimension 0. Our proof gives a reduction to r -of- k threshold functions and applies Littlestone’s Winnow2 algorithm. This subsumes the results about (1) and (2) above and answers a question of Fu [5].

Our second result concerns iterated reductions and answers the following question of Lutz and Mayordomo [13]:

$$(Q) \text{ Does the class } \text{P}_{\text{btt}}(\text{P}_{\text{ctt}}(\text{DENSE}^c)) \text{ have measure 0 in } E?$$

Agrawal and Arvind [1] showed that $\text{P}_{\text{btt}}(\text{P}_{\text{ctt}}(\text{SPARSE})) \subseteq \text{P}_{\text{m}}(\text{LT}_1)$, where LT_1 is the class of problems that have a nonuniform family of depth-1 weighted linear threshold circuits. Equivalently, LT_1 is the class of problems where each input length is a halfspace. We use their technique to reduce

$$(5) \text{P}_{\alpha \log n - \text{tt}}(\text{P}_{\text{ctt}}(\text{DENSE}^c)), \text{ for all } \alpha < 1$$

to a subexponential-size family of halfspaces. We then apply the online learning algorithm of Maass and Turán [16] to learn these halfspaces and conclude that the classes in (5) have p-dimension 0. This strongly answers (Q) in the affirmative.

This paper is organized as follows. Section 2 contains preliminaries about halfspaces, learning, and dimension. The majority reductions result is in section 3 and the iterated reductions result is in section 4. Section 5 concludes with some observations for NP and directions for further work.

2 Preliminaries

A language L is a subset of $\{0, 1\}^*$. For the length of a string x , we write $|x|$. By $L_{=n}$ we denote the set all strings in L of length n , and by $L_{\leq n}$ we denote the set of all strings in L with length at most n . Let L be a language.

- L is *sparse* if for all $n \in \mathbb{N}$, $|L_{\leq n}| \leq p(n)$, where $p(n)$ is a polynomial.
- L is *dense* if for some $\epsilon > 0$, for all but finitely many n , $|L_{\leq n}| > 2^{n^\epsilon}$.
- L is *io-dense* if for some $\epsilon > 0$, for infinitely many n , $|L_{\leq n}| > 2^{n^\epsilon}$.

We write SPARSE , DENSE , and $\text{DENSE}_{\text{i.o.}}$ for the classes of sparse, dense, and io-dense languages, respectively. Note that $L \in \text{DENSE}^c$ if for all $\epsilon > 0$, for infinitely many n , $|L_{\leq n}| < 2^{n^\epsilon}$, and $L \in \text{DENSE}_{\text{i.o.}}^c$ if for all $\epsilon > 0$, for all sufficiently large n , $|L_{\leq n}| < 2^{n^\epsilon}$.

We assume the reader is familiar with the various notions of polynomial-time reductions. If a reduction $g(x)$ produces a single query, then $|g(x)|$ refers to the size of that query. If it produces multiple queries, then $|g(x)|$ is the number of queries produced.

2.1 Threshold Circuits

A *weighted linear threshold gate* with n inputs is determined by a weight vector $\hat{w} \in \mathbb{Q}^n$ and a threshold $T \in \mathbb{Q}$ such that on inputs $x \in \{0, 1\}^n$, where x is considered an n -valued vector (x_1, x_2, \dots, x_n) , the gate will output 1 if and only if $\sum_{1 \leq i \leq n} w_i x_i > T$. An *exact weighted linear threshold gate* is defined similarly, except that the gate will output 1 if and only if $\sum_{1 \leq i \leq n} w_i x_i = 0$. As this is an inner product on vectors, we use the notation $\hat{w} \cdot \hat{x}$ for $\sum_{1 \leq i \leq n} w_i x_i$.

A linear threshold circuit has a linear threshold gate at its root. A language L is in the class LT_1 if there exists a family of nonuniform, depth-1 weighted linear threshold circuits defined by a family of weight vectors $\{\hat{w}_n\}_{n \geq 0}$ such that for all $x \in \{0, 1\}^*$, $x \in L$ if and only if $\hat{w}_{|x|} \cdot \hat{x} > 0$. Similarly, L is in the class ELT_1 if there exists a family of nonuniform, depth-1 exact weighted linear threshold circuits defined by a family of weight vectors $\{\hat{w}_n\}_{n \geq 0}$ such that for all $x \in \{0, 1\}^*$, $x \in L$ if and only if $\hat{w}_{|x|} \cdot \hat{x} = 0$.

Topologically, a linear threshold gate on n inputs describes a halfspace S in $\{0, 1\}^n$, and an exact linear threshold gate describes a hyperplane H in $\{0, 1\}^n$, where strings in $\{0, 1\}^n$ are viewed as binary vectors.

For more information on LT_1 and ELT_1 , we refer the reader to Agrawal and Arvind [1], from which we will make several useful extensions in Section 4.

2.2 Dimension and Learning

Resource-bounded dimension was introduced by Lutz [12] as a refinement of resource-bounded measure [10]. Each class X of languages has a p-dimension $\text{dim}_p(X) \in [0, 1]$, and if $\text{dim}_p(X) < 1$, then X has p-measure 0. In this paper we do not use original definition of p-dimension but instead the result that if X reduces to a learnable concept class family, then $\text{dim}_p(X) = 0$ [6]. For more information on measure and dimension we refer to [3, 7, 11, 14].

A concept is a set $C \subseteq U$ for some universe U . A concept class is a set \mathcal{C} of concepts. An online learner, given a concept class \mathcal{C} and a universe U , attempts to learn a target concept $C \in \mathcal{C}$. Given a sequence of examples x_1, x_2, \dots in U , the learner must predict whether $x_i \in C$. The answer is then revealed, the learner adjusts its strategy, and the next concept is presented for classification. The learner makes a mistake if it incorrectly classifies an example. The *mistake bound* of a learning algorithm for a concept class \mathcal{C} is the maximum over all $C \in \mathcal{C}$ of the number of mistakes made when learning C , over all possible sequences of examples. The running time of the learner is the time required to predict the classification of an example.

Let $L \subseteq \{0, 1\}^*$ and let $\mathcal{C} = (\mathcal{C}_n | n \in \mathbb{N})$ be a sequence of concept classes. For a time bound $r(n)$, we say L reduces to \mathcal{C} in $r(n)$ time if there is a reduction f computable in $O(r(n))$ time such that for infinitely many n , there is a concept $C_n \in \mathcal{C}_n$ such that for all $x \in \{0, 1\}^{\leq n}$, $x \in L$ if and only if $f(0^n, x) \in C_n$. Note that the reduction is not required to hold for all n , but only infinitely many n .

Let $\mathcal{L}(t, m)$ be the set of all sequences of concept classes \mathcal{C} such that for each $C_n \in \mathcal{C}$, there is an algorithm that learns C_n in $O(t(n))$ time with mistake bound

$m(n)$. Then the class $\mathcal{RL}(r, t, m)$ is the class of languages that reduce to some sequence of concept classes in $\mathcal{L}(t, m)$ in $r(n)$ time.

Theorem 2.1 (Hitchcock [6]). *For every $c \in \mathbb{N}$, the class $\mathcal{RL}(2^{cn}, 2^{cn}, o(2^n))$ has p-dimension 0.*

Because $X \subseteq Y$ implies $\dim_p(X) \leq \dim_p(Y)$, the task of proving that a class has p-dimension 0 can be reduced to showing the class is a subset of $\mathcal{RL}(2^{cn}, 2^{cn}, o(2^n))$ for some constant c .

2.3 Learning Algorithms

We make use of two learning algorithms. The first is the second of Littlestone's Winnow algorithms [9], which can be used to learn Boolean r -of- k functions on n variables. In an r -of- k function there is a subset V of the n variables with $|V| = k$ such that the function evaluates to 1 if at least r of the variables in V are set to 1. Winnow2 has two parameters α , a weight update multiplier, and θ , a threshold value. Initially, each of the variables x_i has a weight $w_i = 1$. Winnow2 operates by predicting that an example x is in the concept if and only if $\sum_i w_i x_i > \theta$. The weights are updated following each mistake by the following rubric:

- If Winnow2 incorrectly predicts that x is in the target concept, then for each x_i such that $x_i = 1$, set $w_i = w_i/\alpha$.
- If Winnow2 incorrectly predicts that x is not in the target concept, then for each x_i such that $x_i = 1$, set $w_i = \alpha \cdot w_i$.

Littlestone showed that for $\alpha = \frac{1}{2r}$ and $\theta = n$, Winnow2 has a mistake bound on learning r -of- k functions of $8r^2 + 5k + 14kr \ln n$. Winnow2 also classifies examples in polynomial time.

The second learning algorithm we use is Maass and Turán's [16] algorithm for learning halfspaces. They first describe the Convex Feasibility Problem: given a separation oracle and a guarantee r for an unknown convex body P , find a point in P . By a guarantee, they mean a number such that the volume of the convex body P (in d dimensions) within the ball of radius r around \hat{o} is at least r^{-d} .

Theorem 2.2 (Maass and Turán [16]). *Assume that there is an algorithm A^* solving the Convex Feasibility Problem with query complexity $q(d, \log r)$ (where q is a function of both the dimension d and the guarantee r) and time complexity $t(d, \log r)$. Then there is a learning algorithm A for learning a halfspace in d dimensions and n values such that the mistake bound of A is $q(d, 4d(\log d + \log n + 3)) + 1$ and the running time is at most $t(d, 4d(\log d + \log n + 3)) + q(d, 4d(\log d + \log n + 3)) \cdot p(d, \log n)$ for some polynomial p .*

Using Vaidya's algorithm for learning convex bodies [17], which is an algorithm for the Convex Feasibility Problem, they show that learning a halfspace on d dimensions and n values (in our case, with the binary alphabet, $n = 2$) has a mistake bound of $O(d^2(\log d + \log n))$ and a polynomial running time.

3 Majority Reductions

We say that $A \leq_{\text{maj}}^P B$ if there is a polynomial-time computable function $f : \{0, 1\}^* \rightarrow \mathcal{P}(\{0, 1\}^*)$ such that for all $x \in \{0, 1\}^*$, $x \in A$ if and only if

$$|f(x) \cap B| \geq \frac{|f(x)|}{2}.$$

The following lemma says that if B is nondense, then we can assume that the majority reduction makes the same number of queries for all inputs of each length.

Lemma 3.1. *Let $A \in P_{\text{maj}}(\text{DENSE}^c)$. Then there exists a $B \in \text{DENSE}^c$, a majority reduction f computable in polynomial time, and a polynomial q such that for all $x \in \{0, 1\}^*$, $|f(x)| = q(|x|)$.*

We now prove our first main result.

Theorem 3.2. $P_{\text{maj}}(\text{DENSE}^c)$ has p -dimension 0.

Proof. It suffices to show that there is a concept class family $\mathcal{CF} \in \mathcal{L}(2^{cn}, o(2^n))$ and a reduction g computable in 2^{cn} time such that for all $A \in P_{\text{maj}}(\text{DENSE}^c)$, A reduces to \mathcal{CF} by g .

Let $A \in P_{\text{maj}}(\text{DENSE}^c)$. Then there is a $p(n)$ -time-bounded majority reduction f that makes exactly $r(n)$ queries for each n , and a set $B \in \text{DENSE}^c$ such that for all $x \in \{0, 1\}^*$, $x \in A$ if and only if $|f(x) \cap B| \geq \frac{|f(x)|}{2}$.

Let $Q_n = \bigcup_{|x| \leq n} f(x)$ be the set of all queries made by f up through length n . Then $|Q_n| \leq 2^{n+1}p(n)$. Enumerate Q_n as q_1, \dots, q_N . Then each subset $R \subseteq Q_n$ can be identified with its characteristic string $\chi_R \in \{0, 1\}^N$ according to this enumeration.

Let $\delta \in (0, 1)$. Then $|B_{\leq p(n)}| < 2^{n^\delta}$ for infinitely many n because B is nondense. Thus $M = |Q_n \cap B| \leq 2^{n^\delta}$. Since for all $x \in \{0, 1\}^*$, $f(x)$ makes exactly $r(|x|)$ number of queries and $x \in A$ if and only if $|f(x) \cap B| \geq \frac{r(|x|)}{2}$, our target concept is a $\frac{r(n)}{2}$ -of- M threshold function h , such that $x \in A$ if and only if $h(\chi_{f(x)}) = 1$, which can be learned by Winnow2.

Given x , $\chi_{f(x)}$ can be computed in $O(2^{2n})$ time, and thus Winnow2 can classify examples in $O(2^{2n})$ time, making only $2r^2(n) + 5 \cdot 2^{n^\delta} + 7 \cdot 2^{n^\delta} r(n) \ln 2^{n+1}p(n)$ mistakes, which is $o(2^{2n})$. Thus $P_{\text{maj}}(\text{DENSE}^c) \subseteq \mathcal{RL}(2^{2n}, 2^{2n}, o(2^{2n}))$ and the theorem follows by Theorem 2.1. □

We remark that as r -of- k threshold functions are a special case of halfspaces, we could also use the halfspace learning algorithm instead of Winnow2 to prove Theorem 3.2. As $P_{\text{dtt}}(\text{DENSE}^c) \subseteq P_{\text{maj}}(\text{DENSE}^c)$ and $P_{\text{ctt}}(\text{DENSE}^c) \subseteq P_{\text{maj}}(\text{DENSE}^c)$, Theorem 3.2 subsumes two results from [6]. We also have the following corollary about hard sets for exponential time.

Corollary 3.3. $E \not\subseteq P_{\text{maj}}(\text{DENSE}^c)$. *That is, every \leq_{maj}^P -hard set for E is dense.*

4 Iterated Reductions

Our proof that $P_{\alpha \log n - \text{tt}}(P_{\text{ctt}}(\text{DENSE}^c))$ has p-dimension zero follows the proof technique of Agrawal and Arvind [11] that

$$P_{\text{btt}}(P_{\text{ctt}}(\text{SPARSE})) \subseteq P_{\text{m}}(\text{LT}_1)$$

to reduce the class to a family of halfspaces. We then use Maass and Turán’s [16] learning algorithm to learn these halfspaces. As long as the reduction runs in 2^{n^α} time for some $\alpha < 1$, the halfspaces have subexponential size and the mistake bound is $2^{o(n)}$. Therefore by Theorem [2.1]

$$\dim_{\text{p}}(R_{\text{m}}^{2^{n^\alpha}}(\text{LT}_1)) = 0, \tag{1}$$

where $R_{\text{m}}^{t(n)}$ denotes many-one reductions that run in time $t(n)$.

Instead of $P_{\alpha \log n - \text{tt}}(P_{\text{ctt}}(\text{DENSE}^c))$ we will focus on the smaller class

$$P_{\alpha \log n - \text{tt}}(P_{\text{ctt}}(\text{DENSE}_{\text{i.o.}}^c)).$$

The benefit is that the nondense set will be small almost everywhere rather than infinitely often, which simplifies the arguments. Our proofs can be adapted to the infinitely-often case. First we need to extend some of Agrawal and Arvind’s results. They proved the following technical lemma.

Lemma 4.1. *Let $A \in P_{\text{m}}(\text{ELT}_1)$ (resp. $A \in P_{\text{m}}(\text{LT}_1)$). Then there exist $L \in \text{ELT}_1$ ($L \in \text{LT}_1$), an FP function f , and a polynomial r , such that for every x , for every $n \geq r(|x|)$, $x \in A$ iff $f(x, 1^n) \in H(\hat{w}_n)$ ($f(x, 1^n) \in S_+(\hat{w}_n)$), where \hat{w}_n are the weight vectors associated with L .*

We use the following extension.

Lemma 4.2. *Let Δ be a family of computable functions that is closed under multiplication and composition with polynomials, and let $\text{F}\Delta$ be the functional class with bounds in Δ . Then Lemma [4.1] holds with $f \in \text{F}\Delta$ and $r \in \Delta$.*

We say that a time bound $t(n)$ is *subexponential* if for all $\epsilon > 0$, $t(n) < 2^{n^\epsilon}$ for all sufficiently large n . We write se for the class of all subexponential time bounds.

Corollary 4.3. *Let $A \in R_{\text{m}}^{\text{se}}(\text{ELT}_1)$. Then there is a subexponential-time function f and a language $B \in \text{ELT}_1$ such that for every $x \in \{0, 1\}^*$, for all $q_i, q_j \in f(x)$, $|q_i| = |q_j|$.*

Agrawal and Arvind showed that $\text{SPARSE} \subseteq P_{\text{m}}(\text{ELT}_1)$. We extend this in the following lemma.

Lemma 4.4. $\text{DENSE}_{\text{i.o.}}^c \subseteq R_{\text{m}}^{\text{se}}(\text{ELT}_1)$.

Proof. Let $S \in \text{DENSE}_{1,0}^c$. Then for all but finitely many n , $|S_{=n}| \leq 2^{n^\epsilon}$ for all $\epsilon > 0$. Let $S_{=n} = \{s_{n,1}, s_{n,2}, \dots, s_{n,m(n)}\}$, where $m(n) \leq f(n)$ and f is a subexponential function. Let the string $s_{n,i}$ also stand for the natural number representing the lexicographic rank of the string $s_{n,i}$ in $\{0, 1\}^n$ for every n and $1 \leq i \leq m(n)$. Define $T_n(z)$ to be the polynomial $\prod_{i=1}^{m(n)} (z - s_{n,i})$. Clearly, $T_n(z)$ is a polynomial in z of degree bounded by $f(n)$. Letting z represent both a string in $\{0, 1\}^n$ as well as the lexicographic rank of z in $\{0, 1\}^n$, then $z \in S$ iff $T_n(z) = 0$.

Rewriting $T_n(z)$, we have $T_n(z) = \sum_{1 \leq j \leq f(n)} a_j z^j$. For $1 \leq j \leq f(n)$, we can write z^j as $\sum_{1 \leq r \leq n \cdot f(n)} 2^r y_{j,r}$, where the $y_{j,r}$ essentially denotes the bits in the binary representation of z^j . Thus it follows that $T_n(z)$ can be rewritten as a linear combination $\sum_{1 \leq j \leq f(n)} \sum_{1 \leq r \leq n \cdot f(n)} w_{j,r} y_{j,r}$ of the bits $y_{j,r}$ defined above.

Now we can define a language $L \in \text{ELT}_1$ using these linear functions to define the corresponding weighted exact threshold gates in the circuit family accepting L . As there will be $n \cdot f(n)^2$ variables, which is subexponential, we have that $S \leq_m^{\text{se}} L$. □

Agrawal and Arvind showed that $\text{P}_{\text{ctt}}(\text{ELT}_1) = \text{P}_m(\text{ELT}_1)$ and $\text{P}_{\text{btt}}(\text{ELT}_1) \subseteq \text{P}_m(\text{LT}_1)$. This allows them to show $\text{P}_{\text{btt}}(\text{P}_{\text{ctt}}(\text{SPARSE})) \subseteq \text{P}_m(\text{LT}_1)$ through the following steps:

$$\begin{aligned} \text{P}_{\text{btt}}(\text{P}_{\text{ctt}}(\text{SPARSE})) &\subseteq \text{P}_{\text{btt}}(\text{P}_{\text{ctt}}(\text{P}_m(\text{ELT}_1))) \\ &\subseteq \text{P}_{\text{btt}}(\text{P}_{\text{ctt}}(\text{ELT}_1)) \\ &\subseteq \text{P}_{\text{btt}}(\text{P}_m(\text{ELT}_1)) \\ &\subseteq \text{P}_{\text{btt}}(\text{ELT}_1) \\ &\subseteq \text{P}_m(\text{LT}_1). \end{aligned}$$

We will adapt this proof to our setting. Agrawal and Arvind made use of the following technical lemma.

Lemma 4.5. *Let $\{F_n(\hat{x})\}_{n \geq 1}$, $F_n(\hat{x})$ defined over \mathbb{Q}^n , be a family of degree k multinomials (for a constant $k > 0$). Let the family of weight vectors $\{\hat{c}_n\}_{n > 0}$ and the FP function f be such that for every $\hat{x} \in \mathbb{Q}^n$, $F_n(\hat{x}) = \hat{c}_m \cdot f(\hat{x})$ where $f(\hat{x}) \in \mathbb{Q}^m$. Then the function f reduces the set*

$$A = \bigcup_{n \geq 1} \{x \in \{0, 1\}^* \mid F_n(x) = 0\}$$

to the set in ELT_1 defined by weight vectors $\{\hat{c}_n\}_{n > 0}$. Also, f reduces the set

$$B = \bigcup_{n \geq 1} \{x \in \{0, 1\}^* \mid F_n(x) > 0\}$$

to the set in LT_1 defined by weight vectors $\{\hat{c}_n\}_{n > 0}$ (where a string x of length n is interpreted as an n -dimensional 0-1 vector when it is an argument to F_n).

Lemma 4.6. $R_{\text{ctt}}^{\text{se}}(\text{ELT}_1) = R_{\text{m}}^{\text{se}}(\text{ELT}_1)$.

Proof. Let A be a set that is conjunctively reducible to some set $B \in \text{ELT}_1$. Then there is an se-computable function f such that for every $x \in \{0, 1\}^*$, $f(x)$ is a list of queries such that $x \in A$ iff for every q in $f(x)$, $q \in B$. Using Corollary 4.3, there exist $B' \in \text{ELT}_1$ defined by a family of weight vectors $\{\hat{c}_n\}_{n \geq 1}$, an se-computable function g , and a subexponential function r such that for every x , for every $j \geq r(|x|)$, $x \in A$ iff $g(x, 1^j) \subseteq H(\hat{c}_j)$.

Since f is a conjunctive reduction, there is a subexponential p such that for every x , $g(x, 1^{r(|x|)})$ has exactly $p(|x|)$ queries (this can be achieved simply by repeating the last query a suitable number of times). Define

$$F_{p(n)r(n)}(\hat{q}_1, \hat{q}_2, \dots, \hat{q}_{p(n)}) = \sum_{i=1}^{p(n)} (\hat{c}_{r(n)} \cdot \hat{q}_i)^2$$

where $\hat{q}_i \in \{0, 1\}^{r(n)}$ for $1 \leq i \leq p(n)$. The set L is defined as

$$L = \bigcup_{n \geq 1} \{x \in \{0, 1\}^{*p(n)r(n)} \mid F_{p(n)r(n)}(x) = 0\}.$$

Note that as an argument to $F_{p(n)r(n)}$, x is interpreted as a 0-1 vector.

It is easy to see that $x \in A$ iff $(\hat{q}_1, \hat{q}_2, \dots, \hat{q}_{p(|x|)}) \in L$ where $g(x, 1^{r(|x|)}) = \{\hat{q}_1, \dots, \hat{q}_{p(|x|)}\}$. Lemma 4.5 implies that L is in $R_{\text{m}}^{\text{se}}(\text{ELT}_1)$. \square

To show $P_{\text{btt}}(\text{ELT}_1) \subseteq P_{\text{m}}(\text{LT}_1)$, Agrawal and Arvind divide a k -tt reduction into each separate condition τ , and then note that

$$P_{\tau}(A) \subseteq P_{\text{b}\oplus}(P_{\text{bc}}(P_{1\text{-tt}}(A))),$$

where $P_{\text{b}\oplus}$ is the closure under the bounded parity reduction and P_{bc} is the closure under the bounded conjunctive reduction. They then show that

$$P_{\text{b}\oplus}(P_{\text{bc}}(P_{1\text{-tt}}(\text{ELT}_1))) \subseteq P_{\text{m}}(\text{LT}_1),$$

and finish their proof by showing $P_{\text{m}}(\text{LT}_1)$ is closed under the join operation, i.e. it is possible to create a linear threshold circuit from all 2^{2^k} k -tt conditions in polynomial time. We proceed in the same fashion.

Lemma 4.7. $R_{\text{b}\oplus}^{\text{se}}(\text{LT}_1) = R_{\text{m}}^{\text{se}}(\text{LT}_1)$.

Lemma 4.8. $R_{\text{bd}}^{\text{se}}(\text{ELT}_1) = R_{\text{m}}^{\text{se}}(\text{ELT}_1)$.

Lemma 4.9. $R_{\tau}^{\text{se}}(\text{LT}_1) \subseteq R_{\text{m}}^{\text{se}}(\text{LT}_1)$.

Lemma 4.10. $P_{\text{bc}}(R_{1\text{-tt}}^{\text{se}}(\text{ELT}_1)) \subseteq R_{\tau}^{\text{se}}(\text{ELT}_1)$.

Proof. For $A \in P_{\text{bc}}(R_{1\text{-tt}}^{\text{se}}(\text{ELT}_1))$, there is a reduction f to $B \in \text{ELT}_1$ with k queries such that $x \in A$ iff m of the k queries are in B and $k - m$ are not

in B . We label these queries $\hat{q}_1, \hat{q}_2, \dots, \hat{q}_m$ and $\hat{r}_1, \hat{r}_2, \dots, \hat{r}_{k-m}$. Thus $x \in A$ iff $\hat{q}_i \in B$ for all $1 \leq i \leq m$ and no $\hat{r}_j \in B$ for all $1 \leq j \leq k-m$. We can look at this as the combination of a bounded conjunctive reduction and a bounded disjunctive reduction, both requiring subexponential time. By Lemma 4.6 and Lemma 4.8, we can alter these reductions to a single query each to a language $B' \in \text{ELT}_1$. Call these single queries \hat{q} and \hat{r} . Then $x \in A$ iff $\hat{q} \in B'$ and $\hat{r} \notin B'$. This transformation can be carried out in subexponential time, so the lemma follows. \square

We are ready to prove the simplified version of our main result.

Theorem 4.11. $P_{\alpha \log n\text{-tt}}(\text{P}_{\text{ctt}}(\text{DENSE}_{i.o.}^c))$ has p -dimension 0.

Proof. Through Lemma 4.4, Lemma 4.6, Lemma 4.10, and Lemma 4.7 respectively, the following holds for each truth-table condition τ :

$$\begin{aligned} P_{\tau}(\text{P}_{\text{ctt}}(\text{DENSE}_{i.o.}^c)) &\subseteq P_{\text{b}\oplus}(P_{\text{bc}}(P_{1\text{-tt}}(\text{P}_{\text{ctt}}(\text{DENSE}_{i.o.}^c)))) \\ &\subseteq P_{\text{b}\oplus}(P_{\text{bc}}(P_{1\text{-tt}}(\text{P}_{\text{ctt}}(\text{R}_m^{\text{se}}(\text{ELT}_1)))) \\ &\subseteq P_{\text{b}\oplus}(P_{\text{bc}}(P_{1\text{-tt}}(\text{R}_{\text{ctt}}^{\text{se}}(\text{ELT}_1)))) \\ &\subseteq P_{\text{b}\oplus}(P_{\text{bc}}(P_{1\text{-tt}}(\text{R}_m^{\text{se}}(\text{ELT}_1)))) \\ &\subseteq P_{\text{b}\oplus}(P_{\text{bc}}(\text{R}_{1\text{-tt}}^{\text{se}}(\text{ELT}_1))) \\ &\subseteq P_{\text{b}\oplus}(\text{R}_{\tau}^{\text{se}}(\text{ELT}_1)) \\ &\subseteq \text{R}_m^{\text{se}}(\text{LT}_1). \end{aligned}$$

With $\alpha \log n$ queries for $|x| = n$, there are 2^{n^α} truth-table conditions. Through the reduction above, each condition corresponds to a different set of weights $\hat{c}_{n,j}$, $1 \leq j \leq 2^{n^\alpha}$, defining a threshold circuit that is subexponential in size. Let us say this size is $s(n)$. Thus we can create a single linear threshold circuit with weights $\hat{d}_n = (\hat{c}_{n,1}, \hat{c}_{n,2}, \dots, \hat{c}_{n,2^{n^\alpha}})$ as the join of all these individual circuits. The size of this new circuit is $2^{n^\alpha} \cdot s(n) \leq 2^{n^\delta}$ for some δ such that $0 < \delta < 1$. Let $L \in \text{LT}_1$ be the set defined by the weight vectors $\{\hat{d}_n\}$.

Let $A \in P_{\alpha \log n\text{-tt}}(\text{P}_{\text{ctt}}(\text{DENSE}_{i.o.}^c))$. Let g be the $\alpha \log n\text{-tt}$ reduction, and suppose that $g(x)$ uses the condition which corresponds to $\hat{c}_{|x|,j}$. Let \hat{q}_x be the many-one query corresponding to that condition produced by the reduction above. Then the reduction f mapping A to L is defined by

$$f(x) = (\hat{0}_{(j-1)s(|x|)}, \hat{q}_x, \hat{0}_{(2^{|x|^\alpha} - (j+1))s(|x|)}).$$

Then $x \in A$ iff $f(x) \in L$. It follows that

$$P_{\alpha \log n\text{-tt}}(\text{P}_{\text{ctt}}(\text{DENSE}_{i.o.}^c)) \subseteq \text{R}_m^{2^{n^\alpha}}(\text{LT}_1),$$

which yields the theorem by (III). \square

A similar argument yields our main result.

Theorem 4.12. $P_{\alpha \log n\text{-tt}}(\text{P}_{\text{ctt}}(\text{DENSE}^c))$ has p -dimension 0.

Corollary 4.13. $E \not\subseteq P_{\alpha \log n\text{-tt}}(P_{\text{ctt}}(\text{DENSE}^c))$.

Theorem 4.12 gives the answer to Lutz and Mayordomo's question [13].

Corollary 4.14. $P_{\text{btt}}(P_{\text{ctt}}(\text{DENSE}^c))$ has p-measure 0.

5 Conclusion

We conclude with a brief remark about the density of hard sets for NP. If NP has positive p-dimension, then it follows from our results that

$$\text{NP} \not\subseteq P_{\text{maj}}(\text{DENSE}^c)$$

and

$$\text{NP} \not\subseteq P_{\alpha \log n\text{-tt}}(P_{\text{ctt}}(\text{DENSE}^c))$$

for all $\alpha < 1$. These conclusions are stronger than what is known from the hypothesis $P \neq \text{NP}$. If $P \neq \text{NP}$, then $\text{NP} \not\subseteq P_{\text{btt}}(P_{\text{ctt}}(\text{SPARSE}))$ [2], but nothing is known about majority reductions.

One direction for further research is to improve the $\alpha \log n\text{-tt}$ bound in Theorem 4.12. Can the bound be improved to $n^\alpha\text{-tt}$? Or ideally, to subsume the main result in [6], can it be improved to $n^\alpha\text{-T}$? A more basic direction is to find more applications of learning algorithms in resource-bounded measure and dimension.

References

1. Agrawal, M., Arvind, V.: Geometric sets of low information content. *Theoretical Computer Science* 158(1–2), 193–219 (1996)
2. Arvind, V., Han, Y., Hemachandra, L., Köbler, J., Lozano, A., Mundhenk, M., Ogiwara, A., Schöning, U., Silvestri, R., Thierauf, T.: Reductions to sets of low information content. In: Ambos-Spies, K., Homer, S., Schöning, U. (eds.) *Complexity Theory: Current Research*, pp. 1–45. Cambridge University Press, Cambridge (1993)
3. Athreya, K.B., Hitchcock, J.M., Lutz, J.H., Mayordomo, E.: Effective strong dimension in algorithmic information and computational complexity. *SIAM Journal on Computing* (to appear)
4. Fu, B.: With quasilinear queries EXP is not polynomial time Turing reducible to sparse sets. *SIAM Journal on Computing* 24(5), 1082–1090 (1995)
5. Fu, B.: Personal communication (2006)
6. Hitchcock, J.M.: Online learning and resource-bounded dimension: Winnow yields new lower bounds for hard sets. *SIAM Journal on Computing* 36(6), 1696–1708 (2007)
7. Hitchcock, J.M., Lutz, J.H., Mayordomo, E.: The fractal geometry of complexity classes. *SIGACT News* 36(3), 24–38 (2005)
8. Lindner, W., Schuler, R., Watanabe, O.: Resource-bounded measure and learnability. *Theory of Computing Systems* 33(2), 151–170 (2000)
9. Littlestone, N.: Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning* 2(4), 285–318 (1987)

10. Lutz, J.H.: Almost everywhere high nonuniform complexity. *Journal of Computer and System Sciences* 44(2), 220–258 (1992)
11. Lutz, J.H.: The quantitative structure of exponential time. In: Hemaspaandra, L.A., Selman, A.L. (eds.) *Complexity Theory Retrospective II*, pp. 225–254. Springer, Heidelberg (1997)
12. Lutz, J.H.: Dimension in complexity classes. *SIAM Journal on Computing* 32(5), 1236–1259 (2003)
13. Lutz, J.H., Mayordomo, E.: Measure, stochasticity, and the density of hard languages. *SIAM Journal on Computing* 23(4), 762–779 (1994)
14. Lutz, J.H., Mayordomo, E.: Twelve problems in resource-bounded measure. *Bulletin of the European Association for Theoretical Computer Science* 68, 64–80 (1999). *Current Trends in Theoretical Computer Science: Entering the 21st Century*, pp. 83–101. World Scientific Publishing, Singapore (2001)
15. Lutz, J.H., Zhao, Y.: The density of weakly complete problems under adaptive reductions. *SIAM Journal on Computing* 30(4), 1197–1210 (2000)
16. Maass, W., Turán, G.: How fast can a threshold gate learn? In: Hanson, S.J., Drastal, G.A., Rivest, R.L. (eds.) *Computational Learning Theory and Natural Learning Systems. Constraints and Prospects*, vol. I, pp. 381–414. MIT Press, Cambridge (1994)
17. Vaidya, P.M.: A new algorithm for minimizing convex functions over convex sets. In: *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, pp. 338–349. IEEE Computer Society Press, Los Alamitos (1989)

Isolation Concepts for Enumerating Dense Subgraphs

Christian Komusiewicz*, Falk Hüffner**, Hannes Moser***,
and Rolf Niedermeier

Institut für Informatik, Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
{ckomus,hueffner,moser,niedermr}@minet.uni-jena.de

Abstract. In a graph $G = (V, E)$, a vertex subset $S \subseteq V$ of size k is called c -isolated if it has less than $c \cdot k$ outgoing edges. We repair a nontrivially flawed algorithm for enumerating all c -isolated cliques due to Ito et al. [European Symposium on Algorithms 2005] and obtain an algorithm running in $O(4^c \cdot c^4 \cdot |E|)$ time. We describe a speedup trick that also helps parallelizing the enumeration. Moreover, we introduce a more restricted and a more general isolation concept and show that both lead to faster enumeration algorithms. Finally, we extend our considerations to s -plexes (a relaxation of the clique notion), pointing out a W[1]-hardness result and providing a fixed-parameter algorithm for enumerating isolated s -plexes.

1 Introduction

Finding and enumerating cliques and clique-like structures in graphs has many applications ranging from technical networks [9] to social and biological networks [1–3]. Unfortunately, clique-related problems are known to be notoriously hard for exact algorithms, approximation algorithms, and fixed-parameter algorithms [7, 5]. Ito et al. [9] introduced an interesting way out of this quandary by restricting the search to *isolated* cliques. Herein, given a graph $G = (V, E)$, a vertex subset $S \subseteq V$ of size k is called c -isolated if it has less than $c \cdot k$ outgoing edges. As their main result, Ito et al. [9] claimed an $O(4^c \cdot c^5 \cdot |E|)$ time algorithm for enumerating all c -isolated cliques in a graph. In particular, this means linear time for constant c and fixed-parameter tractability with respect to the parameter c . Unfortunately, the algorithm proposed by Ito et al. [9] suffers from serious deficiencies¹.

* Partially supported by the Deutsche Forschungsgemeinschaft, project OPAL (optimal solutions for hard problems in computational biology), NI 369/2.

** Supported by the Deutsche Forschungsgemeinschaft, Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4.

*** Supported by the Deutsche Forschungsgemeinschaft, project ITKO (iterative compression for solving hard network problems), NI 369/5.

¹ A later manuscript [8] does not fundamentally resolve the problem.

We start with describing the algorithm of Ito et al. [9] and show that it does not fulfill the claimed running time bound. Then, we present some new results that eventually help us to repair Ito et al.'s approach, ending up with an algorithm that enumerates all c -isolated cliques in $O(4^c \cdot c^4 \cdot |E|)$ time. We also observe a speedup trick which seems to have high practical potential and which allows to parallelize the so far purely sequential enumeration algorithm.

Next, inspired by Ito et al.'s isolation concept, we propose two further isolation definitions, a weaker (less demanding) and a stronger concept, both practically motivated. Somewhat surprisingly, we can show that *both* concepts lead to faster enumeration algorithms for isolated cliques, improving the exponential factor from 4^c to 2^c and 2.44^c , respectively.

Finally, we show how to adapt the isolation scenario to the concept of s -plexes, a relaxation of cliques occurring in social networks analysis [15, 1]. In a graph $G = (V, E)$, a vertex subset $S \subseteq V$ of size k is called an s -plex if the minimum degree in $G[S]$ is at least $k - s$. First, strengthening an NP-hardness result of Balasundaram et al. [1], we point out that the problem of finding s -plexes is W[1]-hard with respect to the parameter k ; that is, the problem seems as (parameterized) intractable as CLIQUE is. This motivates our final result, a fixed-parameter algorithm (the parameter is the isolation factor) for constant s that enumerates all of one type of maximal isolated s -plexes. As a side result, here we improve a time bound for a generalized vertex cover problem first studied by Nishimura et al. [12].

Preliminaries. We consider only undirected graphs $G = (V, E)$ with $n := |V|$, $m := |E|$, $V(G) := V$, and $E(G) := E$. Let $N(v) := \{u \in V \mid \{u, v\} \in E\}$ and $N[v] := N(v) \cup \{v\}$. For $v \in V$, let $\deg_G(v) := |N(v)|$. For $A, B \subseteq V$, $A \cap B = \emptyset$, let $E(A, B) := \{\{u, v\} \mid u \in A, v \in B\}$. For $V' \subseteq V$, let $G[V']$ be the subgraph of G induced by V' and $G \setminus V' := G[V \setminus V']$. For $v \in V$, let $G - v := G[V \setminus \{v\}]$. A set S with property P is called *maximal* if no proper superset of S has property P , and *maximum* if no other set with property P has higher cardinality.

Parameterized complexity [5, 11] is an approach to finding optimal solutions for NP-hard problems. The idea is to accept the seemingly inevitable combinatorial explosion, but to confine it to one aspect of the problem, the *parameter*. If for relevant inputs this parameter remains small, then even large problems can be solved efficiently. More precisely, a problem of size n is *fixed-parameter tractable* (FPT) with respect to a parameter k if there is an algorithm solving it in $f(k) \cdot n^{O(1)}$ time.

Due to the lack of space, most proofs will appear in the full version of this paper. Some material also appears in [10].

2 Enumerating Isolated Cliques

We begin with describing Ito et al.'s algorithm for enumerating maximal c -isolated cliques [9]. Given a graph $G = (V, E)$ and an isolation factor c , first the vertices are sorted by their degree such that $u < v \Rightarrow \deg(u) \leq \deg(v)$. The

index of a vertex is its position in this sorted order. For a vertex set $C \subseteq V$, an *outgoing edge* is an edge $\{u, v\}$ with $u \in C$ and $v \notin C$, and for a vertex $v \in C$, its outgoing edges are the outgoing edges of C that are incident on v . Let $N_+[v] := \{u \in N[v] \mid u \geq v\}$ and $N_-(v) := \{u \in N(v) \mid u < v\}$.

In a c -isolated clique, the vertex with the lowest index is called the *pivot* of the clique. Clearly, a pivot has less than c outgoing edges. Since every c -isolated clique has a pivot, we can enumerate all maximal c -isolated cliques of a graph by enumerating all maximal c -isolated cliques with pivot v for each $v \in V$ and then removing those c -isolated cliques with pivot v that are a subset of a c -isolated clique with another pivot.

The enumeration of maximal c -isolated cliques with pivot v for each $v \in V$ is the central part of the algorithm. We call this the *pivot procedure*. It comprises three successive stages.

Trimming stage. In this stage, we build a candidate set C that is a superset of all c -isolated cliques with pivot v . The candidate set C is initialized with $N_+[v]$, and then vertices that obviously cannot be part of a c -isolated clique with pivot v are removed from C . We refer to Ito et al. [9] for details.

Enumeration stage. In this stage, all maximal c -isolated cliques with pivot v are enumerated. Let C be the candidate set after the trimming stage, which deleted d vertices from $N_+[v]$. In total, we can delete only less than c vertices from $N_+[v]$, since otherwise v obtains too many outgoing edges. Therefore, $\tilde{c} := c - d - 1$ is the number of vertices that we may still remove from C . We can enumerate cliques $C' \subseteq C$ of size *at least* $|C| - \tilde{c}$ by enumerating vertex covers of size *at most* \tilde{c} in the complement graph $\overline{G[C]}$. Ito et al. propose to enumerate *all* vertex covers of size at most \tilde{c} [9]. We point to problems with this approach in Sect. 2.1.

Screening stage. In the screening stage, all cliques that are either not c -isolated or that are c -isolated but not maximal are removed. First the c -isolation is checked. Then those cliques that pass the test for isolation are compared pairwise, and we only keep maximal cliques. Finally, we check each clique that is left for pivot v against each clique obtained during calls to $\text{pivot}(u)$ with $u \in N_-(v)$, since these are the only cliques that can be superset of a clique obtained for pivot v . The claimed overall running time, in the exponential part dominated by this last step, is then $O(4^c \cdot c^5 \cdot |E|)$ [9].

2.1 Problems with the Algorithm

The crucial part of the algorithm is the enumeration stage, in which the algorithm enumerates *all* vertex covers of size less than \tilde{c} . The authors argued that for a graph of size $|C|$, this can be done in $O(1.39^{\tilde{c}} \cdot \tilde{c}^2 + |C|^2)$ time. In contrast, Fernau [6] showed that the vertex covers of size *exactly* k in a graph can be enumerated in time $O(2^k k^2 + kn)$ if and only if k is the size of a minimum vertex cover of the graph and that otherwise no algorithm of running time $f(k) \cdot n^{O(1)}$ that enumerates all of these vertex covers exists, simply because there are too many. But in the course of the pivot procedure it may happen that we have to do

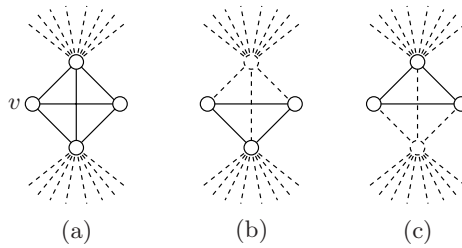


Fig. 1. Example for the enumeration stage with pivot v . *Solid lines* are edges between members of the clique; *dashed lines* are outgoing edges.

just that: enumerate all vertex covers of size \tilde{c} or less, where \tilde{c} is not the size of a minimum vertex cover of $\overline{G[C]}$. Since this cannot be done in time $f(c) \cdot n^{O(1)}$, the algorithm does not yield fixed-parameter tractability with respect to the parameter c .

Figure 1 (a) illustrates such a situation. Consider the case $c = 4$ with v as pivot. No trimming takes place. This means that at the beginning of the enumeration stage, we may still remove up to $\tilde{c} = c - 1 = 3$ vertices from C to obtain a c -isolated clique. Since $C = N_+[v]$ forms a clique, the graph $\overline{G[C]}$ has only one minimum vertex cover, namely the empty set. This means that all subsets of $C \setminus \{v\}$ (v as pivot must not be eliminated from C) of size 3 or less are vertex covers that we would have to enumerate. Clearly, the number of such vertex covers is not only dependent on the size of the covers, but also on the size of $N_+[v]$. In our example, there are 8 such covers, and it is easy to see that we can increase the number of vertex covers simply by increasing the size of $N_+[v]$.

In contrast, enumeration of minimal vertex covers was shown to be *inclusion-minimally fixed parameter enumerable* [4]; in particular, all minimal solutions of size at most c can be enumerated in $O(2^c c^2 + m)$ time. So running time is not a problem here; however, we miss some c -isolated cliques when only considering minimal vertex covers. This is because we cannot simply discard a maximal clique that violates the isolation condition; it might have some subsets that *are* c -isolated. As an example, in Fig. 1 (a), the clique has 4 vertices and 16 outgoing edges and is thus not 4-isolated. However, two subsets ((b) and (c)) are cliques with 3 vertices and 11 outgoing edges, and thus are 4-isolated.

2.2 Repairing the Enumeration Stage

To cope with the problems described in Sect. 2.1, we propose a two-step approach for enumerating all maximal c -isolated cliques. First, we enumerate all *minimal* vertex covers and thus obtain maximal cliques in the candidate set C . Then, to also capture c -isolated cliques that are subsets of non- c -isolated cliques enumerated this way, for each of these cliques, we enumerate all maximal subsets that fulfill the isolation condition. The problem ISOLATED CLIQUE SUBSET of finding these c -isolated subsets is then: given a graph $G = (V, E)$ and a clique $C \subseteq V$, find a set $C' \subseteq C$ that forms a c -isolated vertex set, that is, a set

procedure isolated-subset(C, c, x_{\min})

Input: A clique $C = \{v_1, v_2, \dots, v_k\}$ with vertices sorted by degree, an isolation factor c and a minimum number x_{\min} of outgoing edges from each vertex.

Output: The set of maximal c -isolated cliques \mathcal{C} in C .

1: **foreach** $v \in C$: $x(v) := \deg(v) - |C| - 1 - x_{\min}$
2: $\hat{c} := c - x_{\min}$
3: $e(C) := (\sum_{v \in C} x(v)) - \hat{c} \cdot |C| + 1$
4: $\mathcal{D} := \{\emptyset\}$, $\mathcal{C} := \emptyset$
5: **repeat** \hat{c} **times**
6: **foreach** $D \in \mathcal{D}$
7: **if** $C \setminus D$ is a c -isolated clique **then** $\mathcal{C} := \mathcal{C} \cup \{C \setminus D\}$
8: **else**
9: **if** $D = \emptyset$ **then** $i := k + 1$ **else** $i := \min_{v_l \in D} \{l\}$
10: $\mathcal{D} := \mathcal{D} \cup \{D \cup \{v_j\} \mid k - \hat{c} < j < i\}$
11: $\mathcal{D} := \mathcal{D} \setminus \{D\}$
12: **return** \mathcal{C}

Fig. 2. Algorithm for enumerating maximal c -isolated subsets of a clique C

with less than $c \cdot |C'|$ outgoing edges. The difficulty is in doing this fast enough, in particular with the running time depending only polynomially on $|C|$. For this (Theorem 1), the key is the following lemma, which reduces the choices of which vertices to omit from C .

Lemma 1. *Given a clique C with $|C| = k$, every maximal c -isolated subset of C is a superset of C^{k-c+1} , where C^{k-c+1} is the set of the $k - c + 1$ vertices with lowest index in C .*

According to Lemma 1, we may only remove vertices from the $c - 1$ vertices in $C \setminus C^{k-c+1}$ to obtain maximal c -isolated subsets of C . Hence, there are 2^{c-1} subsets of $C \setminus C^{k-c+1}$, and we enumerate maximal c -isolated subsets of C by generating the subsets of $C \setminus C^{k-c+1}$ in order of increasing cardinality and testing for each generated set whether its removal from C yields a maximal c -isolated subset. In this way, we can avoid examining supersets of removal sets for which a c -isolated clique was already output, since they would yield non-maximal cliques. The algorithm is shown in Fig. 2. Note that in lines 1–2 we compute an equivalent instance of ISOLATED CLIQUE SUBSET with isolation factor \hat{c} by decreasing the number of outgoing edges of each vertex by x_{\min} , where x_{\min} is the minimum number of outgoing edges from each vertex in C . The computation of line 3 derives the number of outgoing edges above the threshold allowed by the isolation condition. With this, the condition in line 7 can be tested in constant time. This is needed to obtain the running time as claimed by the following theorem.

Theorem 1. *Given an instance of ISOLATED CLIQUE SUBSET with at least x_{\min} outgoing edges from each vertex, all of the at most $O(2^{c-x_{\min}})$ maximal solutions can be enumerated in $O(2^{c-x_{\min}} + |C|)$ time.*

We now describe how to use Theorem 1 to obtain a correct pivot procedure. Our modified pivot procedure differs from the original procedure only in the enumeration stage and in the screening stage. The enumeration stage is divided into two steps: the enumeration of maximal cliques and the enumeration of maximal subsets that fulfill the isolation condition for each of those cliques. Using Theorem 1, we can upper-bound the running time of the enumeration stage.

Lemma 2. *Given a graph $G = (V, E)$, a vertex $v \in V$, a set $C \subseteq N_+[v]$, and an isolation factor c , there are at most $2^{c-1} \cdot c$ maximal c -isolated cliques with pivot v , and they can be enumerated in $O(2^c \cdot c^2 \cdot m(C))$ time, where $m(C)$ is the number of edges in $G[C]$.*

In the screening stage, we filter non-maximal cliques by $O(4^c \cdot c^3)$ pairwise comparisons. Since the cliques obtained in the enumeration stage have size at most $\deg_G(v)$, these comparisons can be performed in $O(4^c \cdot c^3 \cdot \deg_G(v))$ time. With the running times of the stages of the pivot procedure for pivot v we can upper-bound the running time of the whole algorithm:

Theorem 2. *All maximal c -isolated cliques of a graph can be enumerated in $O(4^c \cdot c^3 \cdot m)$ time.*

2.3 Improved Screening of Cliques

In addition to fixing Ito et al.'s algorithm [9], we present an improved screening stage. While we can improve the asymptotic running time derived in Theorem 2 only slightly, the improvement facilitates parallelization of the enumeration algorithm and allows an exponential speedup for a variant of isolated clique enumeration to be presented in Sect. 3.2. More precisely, instead of a brute-force all-pairwise comparison, we achieve a simple and efficient test for checking whether an enumerated clique is subset of a clique with a different pivot.

Lemma 3. *A c -isolated clique C with pivot v is subset of a c -isolated clique C' with pivot $u \neq v$ iff $u \in N_-(v)$ and $N(u) \supseteq C$.*

Proof. We prove both directions separately. If $C' \supseteq C$ is a clique with pivot u , then u must be adjacent to all vertices in C , in particular $u \in N_+[v]$. Since u is the pivot of C' , it has lower index than v and thus $u \in N_-(v)$.

If there is a vertex $u \in N_-(v)$ that is adjacent to all vertices in C , then $C \cup \{u\}$ is a clique and a superset of C . It is furthermore c -isolated, since with u we have added a vertex with less than c outgoing edges (because $u < v$). Also, u is its pivot, again because $u < v$. \square

According to Lemma 3, we can replace the pairwise comparisons between cliques enumerated in previous calls of the pivot procedure and those of the current call for pivot v with a simple test that looks for vertices in $N_-(v)$ that are adjacent to all vertices of an enumerated clique. This test takes $O(c \cdot |C|)$ time. Since the enumerations of cliques for different pivots now run completely independent from

each other, we can parallelize our algorithm by executing the pivot procedures for different pivot vertices on up to n different processors. Unfortunately, the asymptotic running time derived in Theorem 2 remains largely unchanged, since there are still $O(4^c c^2)$ pairwise comparisons between cliques for a single pivot; however, we save a factor of c and there is also a conceivable speedup in practice since we significantly reduce the number of brute-force set comparisons.

3 Alternative Isolation Concepts

Since isolation is not merely a means of developing efficient algorithms for the enumeration of cliques but also a trait in its own right, it makes sense to consider varying degrees of isolation. For instance, this is useful for the enumeration of isolated dense subgraphs for the identification of communities, which play a strong role in the analysis of biological and social networks [13].

In this context, the definition of c -isolation is not particularly tailored to these applications and we propose two alternative isolation concepts. One of them, min- c -isolation, is a weaker notion than c -isolation and the other, max- c -isolation, is a stronger notion than c -isolation. For both isolation concepts, we achieve a considerable speedup in the exponential part of the running time.

3.1 Minimum Isolation

Min- c -isolation is a weaker concept of isolation than the previously defined c -isolation, since we only demand that a set contains at least one vertex with less than c outgoing edges.

Definition 1. *Given a graph $G = (V, E)$ and a vertex set $S \subseteq V$ of size k , S is min- c -isolated when there is at least one vertex in S with less than c neighbors in $V \setminus S$.*

Obviously, every c -isolated set is also min- c -isolated. The enumeration of maximal min- c -isolated cliques consequently yields sets that are at least as large and often larger than c -isolated cliques.

The algorithm for the enumeration of maximal min- c -isolated cliques is mainly a simplification of the algorithm from Sect. 2. However, we lose linear-time solvability in the case of constant isolation factors c —the running time then becomes $O(n \cdot m)$. We use the same pivot definition and enumerate cliques for each possible pivot; from our definition of min- c -isolation it follows directly that the pivot of a min- c -isolated clique must have less than c neighbors outside of the clique. Subsequently, we point out the differences in the three main stages of the pivot procedure.

In the trimming stage, we start with $C := N[v]$ as candidate set. After trimming, we can assume that every vertex u that was not removed has at least $|C| - c$ neighbors in C . In the enumeration stage, we simply enumerate minimal vertex covers in $\overline{G[C]}$ of size at most \tilde{c} , where \tilde{c} is the number of vertices that can still be removed from the candidate set C . For each enumerated minimal vertex

cover D , the set $C \setminus D$ is a maximal min- c -isolated clique. Hence, we need not test for maximality, but the enumerated cliques might contain a vertex with lower index than v , since we have not necessarily removed all vertices from $N_-(v)$. If a clique C' features a vertex with lower index than v , then C' is removed from the output. Compared to Theorem 2, the fact that we do not perform any maximality test results in an improved exponential part of the running time.

Theorem 3. *All maximal min- c -isolated cliques of a graph can be enumerated in $O(2^c \cdot c \cdot m + n \cdot m)$ time.*

3.2 Maximum Isolation

Compared to c -isolation, max- c -isolation is a stronger notion. This results in most cases in the enumeration of smaller cliques for equal values of c .

Definition 2. *Given a graph $G = (V, E)$ and a vertex set $S \subseteq V$ of size k , S is max- c -isolated if every vertex $v \in S$ has less than c neighbors in $V \setminus S$.*

This isolation concept is especially useful for graphs where the vertices have similar degrees. Consider for example a graph in which all vertices have the same degree. Here, the notions c -isolation and max- c -isolation become equivalent for cliques, but max- c -isolation allows a better worst-case running time.

We apply the algorithm scheme presented in Sect. 2, that is, for every vertex $v \in V$ we enumerate all maximal max- c -isolated cliques with pivot v .

Trimming Stage. We compute a candidate set $C \subseteq N_+[v]$ by removing every vertex from $N_+[v]$ that cannot be in a maximum max- c -isolated clique with pivot v .

Enumeration Stage. In this stage, we enumerate max- c -isolated cliques $C' \subseteq C$ with pivot v . As in Sect. 2, we first enumerate maximal cliques in C via enumeration of minimal vertex covers of size at most \tilde{c} in $\overline{G[C]}$, where \tilde{c} is the number of vertices that can still be removed from the candidate set C . The cliques thus obtained may violate the isolation condition, since they may contain vertices with too many outgoing edges. We can restore the isolation condition for each enumerated clique by simply removing these vertices. This is done until the resulting clique is either max- c -isolated or we have removed more than \tilde{c} vertices. In the latter case we discard the clique. The remaining enumerated cliques are not necessarily maximal, and therefore non-maximal cliques must be removed from the output in the screening stage.

Screening Stage. There are two possibilities for an enumerated clique C to be non-maximal. First, it can be proper subset of another max- c -isolated clique with pivot v . Second, it can be proper subset of a max- c -isolated clique with pivot $u < v$. For the first possibility, we test whether there is a set of vertices $D \subseteq N_+[v] \setminus C$ such that $C \cup D$ is a max- c -isolated clique. Clearly, D has to form a clique and all its vertices have to be adjacent to all vertices in C . Furthermore, whenever D contains a vertex u with degree $|C| + c + x$, then $|D|$ must have size

at least $x + 1$. Otherwise, $C \cup D$ is not max- c -isolated, because u has at least c outgoing edges from $C \cup D$. Hence, we test for all $0 \leq x < c - 1$ whether the set

$$D^x := \{w \in N_+[v] \setminus C \mid C \subseteq N(w) \wedge \deg(w) \leq |C| + c + x\}$$

contains a clique of size at least $x + 1$. If this is not the case for any x , then C is a maximal max- c -isolated clique for pivot v . Otherwise, C is removed from the output.

It remains to check whether C is a proper subset of a clique with another pivot $u < v$. This can be tested in the manner described in Sect. 2.3. The running time of the pivot procedure is dominated by the first maximality test of the screening stage. For each of the $O(2^c)$ enumerated cliques, we have to solve MAXIMUM CLIQUE up to c times. Since $|D^x| < c$ for all $0 \leq x < c - 1$, this can be done in $O(1.22^c)$ time [14]. The overall running time of this test is then

$$O(2^c \cdot 1.22^c \cdot c) = O(2.44^c \cdot c).$$

The running time of the whole enumeration can be bounded in a similar way as in Sect. 2.2.

Theorem 4. *All maximal max- c -isolated cliques of a graph can be enumerated in $O(2.44^c \cdot c \cdot m)$ time.*

4 Enumerating Isolated s -Plexes

In many applications such as social network analysis, cliques have been criticized for their overly restrictive nature or modelling disadvantages. Hence, more relaxed concepts of dense subgraphs such as s -plexes [15, 1] are of interest.

An s -plex is a degree-based relaxation of the clique concept. In a graph $G = (V, E)$, a subset of vertices $S \subseteq V$ of size k is called an s -plex if the minimum degree in $G[S]$ is at least $k - s$. It has been shown that the problem of deciding whether a graph has an s -plex of size k is NP-complete [1]. We strengthen this by the corresponding parameterized hardness result. The parameter-preserving reduction from CLIQUE is given in the full version of this paper. It shows that MAXIMUM s -PLEX is W[1]-hard with respect to the combined parameter (s, k) and thus also if parameterized only by either one of s and k . Therefore, as for CLIQUE, we rather consider isolation as parameter in terms of studying fixed-parameter tractability.

We present an algorithm for the enumeration of maximal min- c -isolated s -plexes that runs in FPT time with respect to parameter c for any constant s . In this paper, we have chosen to consider only min- c -isolation, since the enumeration algorithm is easier to describe with this isolation concept. A min- c -isolated s -plex S contains at least one vertex that has less than c neighbors in $V \setminus S$. Compared to the enumeration of maximal min- c -isolated cliques, we face two obstacles when enumerating maximal min- c -isolated s -plexes. First, we cannot use the algorithm for the enumeration of minimal vertex covers, since an s -plex does

not necessarily induce an independent set in the complement graph. Instead, since in an s -plex S of size k every vertex $v \in S$ is adjacent to at least $k - s$ vertices, the subgraph induced by S in the complement graph $\overline{G[S]}$ is a graph with maximum degree at most $s - 1$. Consider therefore the following generalization of a vertex cover:

Definition 3. *Given a graph G and a nonnegative integer d , we call a subset of vertices $S \subseteq V$ a max-deg- d deletion set if $G[V \setminus S]$ has maximum degree at most d .*

The idea is to enumerate maximal s -plexes in G by enumerating minimal max-deg- d deletion sets in \overline{G} . We present a fixed-parameter algorithm for the enumeration of minimal max-deg- d deletion sets that uses the size of the solution sets as parameter.

Finding a minimum max-deg- d deletion set was also considered by Nishimura et al. [12], who presented an $O((d+k)^{k+3} \cdot k + n(d+k))$ time algorithm for the decision version. We improve the exponential part of this running time while also covering the enumeration version. The idea is to pick a vertex v with more than d neighbors and then branch into $d+2$ cases corresponding to the deletion of v or the deletion of one of the $d+1$ first neighbors of v :

Lemma 4. *Given a graph G and an integer k , all minimal max-deg- d deletion sets of size at most k can be enumerated in $O((d+2)^k \cdot (k+d)^2 + m)$ time.*

The second obstacle lies in the fact that given a pivot vertex v , maximal min- c -isolated s -plexes with pivot v are not necessarily a subset of $N_+[v]$, since they can contain up to $s-1$ vertices that are not adjacent to v . We deal with this by enumerating all maximal min- c -isolated s -plexes for a given pivot set instead of a single pivot. The *pivot set* of a min- c -isolated s -plex is defined as the set that contains the pivot vertex v of the s -plex and those vertices that belong to the s -plex but are not adjacent to v . The *pivot vertex* is defined as the vertex with lowest index among the vertices with less than c neighbors outside of the s -plex. There has to be at least one such vertex, since otherwise the condition of min- c -isolation would be violated, but it does not necessarily have to be the vertex with the lowest index of all vertices in the s -plex.

The enumeration algorithm also consists of the three stages. In the trimming stage, we build a candidate set C by removing vertices from $N(v)$ that obviously cannot belong to a min- c -isolated s -plex with pivot v . For each possible pivot set P with pivot v , we independently enumerate the maximal min- c -isolated s -plexes. This is done in the enumeration stage by first building the complement graph $\overline{G[C \cup P]}$ and then enumerating minimal max-deg- $(s-1)$ deletion sets of size at most $c-1$ in $\overline{G[C \cup P]}$. In the screening stage, we first test whether any of the enumerated min- c -isolated s -plexes contains a vertex $u < v$, where u has less than c neighbors outside of the s -plex. Then this s -plex has pivot u and not v and is therefore removed from the output. Finally we perform a maximality test and remove non-maximal min- c -isolated s -plexes from the output.

Theorem 5. *All maximal min- c -isolated s -plexes of a graph can be enumerated in $O((s+1)^c \cdot (s+c) \cdot n^{s+1} + n \cdot m)$ time.*

Thus, for every fixed s , we obtain a fixed-parameter algorithm for enumerating all maximal \min - c -isolated s -plexes with respect to the parameter c .

Acknowledgement. We thank H. Ito and K. Iwama (Kyoto) for making their manuscript [8] available to us and J. Guo (Jena) for the idea for Lemma 4.

References

1. Balasundaram, B., Butenko, S., Hicks, I.V., Sachdeva, S.: Clique relaxations in social network analysis: The maximum k -plex problem. Manuscript (2006)
2. Balasundaram, B., Butenko, S., Trukhanovzu, S.: Novel approaches for analyzing biological networks. *Journal of Combinatorial Optimization* 10(1), 23–39 (2005)
3. Butenko, S., Wilhelm, W.E.: Clique-detection models in computational biochemistry and genomics. *European Journal of Operational Research* 173(1), 1–17 (2006)
4. Damaschke, P.: Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *Theoretical Computer Science* 351(3), 337–350 (2006)
5. Downey, R.G., Fellows, M.R.: *Parameterized Complexity*. Springer, Heidelberg (1999)
6. Fernau, H.: On parameterized enumeration. In: Ibarra, O.H., Zhang, L. (eds.) COCOON 2002. LNCS, vol. 2387, pp. 564–573. Springer, Heidelberg (2002)
7. Hästad, J.: Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica* 182(1), 105–142 (1999)
8. Ito, H., Iwama, K.: Enumeration of isolated cliques and pseudo-cliques. Manuscript (August 2006)
9. Ito, H., Iwama, K., Osumi, T.: Linear-time enumeration of isolated cliques. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 119–130. Springer, Heidelberg (2005)
10. Komusiewicz, C.: Various isolation concepts for the enumeration of dense subgraphs. Diplomarbeit, Institut für Informatik, Friedrich-Schiller Universität Jena (2007)
11. Niedermeier, R.: *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, Oxford (2006)
12. Nishimura, N., Ragde, P., Thilikos, D.M.: Fast fixed-parameter tractable algorithms for nontrivial generalizations of vertex cover. *Discrete Applied Mathematics* 152(1–3), 229–245 (2005)
13. Palla, G., Derényi, I., Farkas, I., Vicsek, T.: Uncovering the overlapping community structure of complex networks in nature and society. *Nature* 435(7043), 814–818 (2005)
14. Robson, J.M.: Algorithms for maximum independent sets. *Journal of Algorithms* 7(3), 425–440 (1986)
15. Seidman, S.B., Foster, B.L.: A graph-theoretic generalization of the clique concept. *Journal of Mathematical Sociology* 6(1), 139–154 (1978)

Alignments with Non-overlapping Moves, Inversions and Tandem Duplications in $O(n^4)$ Time

Christian Ledergerber and Christophe Dessimoz

ETH Zurich, Institute of Computational Science, Switzerland
ledergec@student.ethz.ch, cdessimoz@inf.ethz.ch

Abstract. Sequence alignment is a central problem in bioinformatics. The classical dynamic programming algorithm aligns two sequences by optimizing over possible insertions, deletions and substitution. However, other evolutionary events can be observed, such as inversions, tandem duplications or moves (transpositions). It has been established that the extension of the problem to move operations is NP-complete. Previous work has shown that an extension restricted to non-overlapping inversions can be solved in $O(n^3)$ with a restricted scoring scheme. In this paper, we show that the alignment problem extended to non-overlapping moves can be solved in $O(n^5)$ for general scoring schemes, $O(n^4 \log n)$ for concave scoring schemes and $O(n^4)$ for restricted scoring schemes. Furthermore, we show that the alignment problem extended to non-overlapping moves, inversions and tandem duplications can be solved with the same time complexities. Finally, an example of an alignment with non-overlapping moves is provided.

1 Introduction

In computational biology, alignments are usually performed to identify the characters that have common ancestry. More abstractly, alignments can also be represented as edit sequences that transform one sequence into the other under operations that model the evolutionary process. Hence, the problem of aligning two sequences is to find the most likely edit sequence, or equivalently, under an appropriate scoring scheme, the highest scoring edit sequence.

Historically, the only edit operations allowed were insertions, deletions and substitutions of characters, which we refer to as standard edit operations. The computation of the optimal alignment with respect to standard edit operations is well understood [1], and commonly used. But in some cases, standard edit operations are not sufficient to accurately model gene evolution. To take into account observed phenomena such as inversions, duplications or moves (intra-genic transpositions) of blocks of sequences [2], the set of edit operations must be extended correspondingly. Such extensions have been studied in the past and a number of them turned out to be hard [3]. In particular an extension to general move operations was shown to be NP-complete [4]. For the simple greedy

algorithm presented in [4] it was shown by [5] that $O(n^{0.69})$ is an upper bound for the approximation factor. An efficient $O(\log^* n \log n)$ factor approximation algorithm for the general problem is presented in [6].

A number of results for the extension of the standard alignment including block inversions have been achieved. It is shown in [7] that sorting by reversals is NP-hard, but the complexity of alignment with inversions is still unknown. A restricted problem of non-overlapping inversions was proposed by [8] who found an $O(n^6)$ algorithm. This result was then further improved by [9,10,11,12] where [12] obtained an $O(n^3)$ algorithm for a restricted scoring scheme.

In this paper, we show that the alignment problem extended with non-overlapping moves and non-overlapping tandem duplications can be solved exactly in polynomial time, and provide algorithms with time complexity of $O(n^5)$ for general scoring schemes, $O(n^4 \log n)$ for concave scoring schemes and $O(n^4)$ for restricted scoring schemes.

Since the probability that k independent and uniformly distributed moves be non-overlapping decreases very rapidly¹, this restriction is only of practical interest for small k , that is, if such events are very rare. Convincing evidence that this is indeed the case can be found in [13]. They show that protein domain order is highly conserved during evolution. It is established in [13] that most domains cooccur with only zero, one or two domain families. Since a move operation of the more elaborate type such as $ABCD \rightarrow ACBD$ immediately implies that B cooccurs with three other domains, we conclude that move operations have to be rare. Furthermore, exon shuffling is highly correlated to domain shuffling [14,15,16] and hence cannot lead to a large amount of move operations. Finally, a number of move operations can be found in the literature [17,18]. As for tandem duplication events, articles on domain shuffling reveal that the most abundant block edit operations are tandem duplications where the duplicate stays next to the original [19,20].

In the next section, we present a rigorous definition of the two alignment problems solved here: an extension to non-overlapping moves and an extension to non-overlapping moves, inversions and tandem duplications. Then, we provide solutions to both problems. The last section presents the experimental results using the extension to non-overlapping moves.

2 Definition of the Problems and Preliminaries

2.1 Notation and Definitions

In the following, we will denote the two strings to be aligned with $S = s_1 \dots s_n$ and $T = t_1 \dots t_m$ where $|S| = n$ and $|T| = m$. The i -th character of S is $S[i]$ and $S[i..j] = s_{i+1} \dots s_j$ (note the indices). Thus, if $j \leq i$, $S[i..j] = \lambda$. By this definition, $S[i..j]$ and $S[j..k]$ are disjoint. $\overline{S} = s_n \dots s_1$ denotes the reverse of S and $\overline{S}[i..j] = \overline{S}[n - j..n - i]$ is the reverse of a substring, the substring of the reverse respectively (note the extension of the bar). Let us denote the score of

¹ For long sequences, this probability converges to $\frac{1}{(2k-1)!!} = \frac{1}{1 \cdot 3 \cdot 5 \dots 2k-1}$.

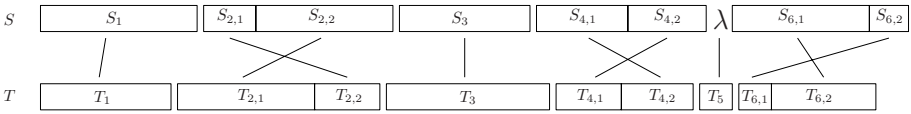


Fig. 1. Example of a non-overlapping move alignment of S with T

the standard alignment of S with T with $\delta(S, T)$. The score for substituting a character a with character b is denoted by an entry in the scoring matrix $\sigma(a, b)$. To simplify the definition of the alignment problems we introduce the concept of d -decompositions:

Definition 1. Let a d -decomposition of a string S be a decomposition of S in d substrings such that the concatenation of all the substrings is equal to S . E.g. $S = S_1 \cdots S_d$. Let $\mathcal{M}_d(S)$ be the set of all d -decompositions of S .

Note that S_i denotes a substring of a d -decomposition while s_i denotes a character. Let us further define the cyclic string to string comparison problem as introduced by [21]:

Definition 2. The cyclic string comparison problem is to find the 2-decomposition $S_1S_2 \in \mathcal{M}_2(S)$ and $T_1T_2 \in \mathcal{M}_2(T)$ such that the score $\delta(S_1, T_2) + \delta(S_2, T_1)$ is maximal. The optimal score is denoted by $\delta_c(S, T)$.

Due to the symmetry of the problem there exists always a two decomposition of $S = S_1S_2$ such that $\delta_c(S, T) = \delta(S_2S_1, T)$ as proven by [21].

Finally, we assume that the reader is familiar with the concept of edit graphs as defined for instance in [22] or [23].

2.2 Definition of Alignment with Non-overlapping Moves

Using d -decompositions and the cyclic string to string comparison problem we can now define the alignment with non-overlapping moves as follows.

Definition 3. The problem of aligning S and T with non-overlapping moves is to find $d \in \mathbb{N}$ and d -decompositions of S and T such that the score $\sum_{i=1}^d \max\{\delta(S_i, T_i), \delta_c(S_i, T_i) + \sigma_c(l_{S_{i1}}) + \sigma_c(l_{S_{i2}}) + \sigma_c(l_{T_{i1}}) + \sigma_c(l_{T_{i2}})\}$ is maximal for all $d \in \mathbb{N}$, $S_1 \dots S_d \in \mathcal{M}_d(S)$ and $T_1 \dots T_d \in \mathcal{M}_d(T)$. Where $l_{S_{i1}}, l_{S_{i2}}, l_{T_{i1}}$ and $l_{T_{i2}}$ are the lengths of the blocks involved in the move operation and $\sigma_c(l)$ is a penalty function for move operations. The optimal score is denoted by $\delta_m(S, T)$.

Note that substrings S_i, T_i may be empty. However, a substring needs to have a length of at least 2 to contain a move. In other words, we align d pairs of substrings of S and T and allow for each aligned pair of substrings at most one swap of a prefix with a suffix as defined by the cyclic string comparison problem. $\sigma_c(l_{S_1}) + \sigma_c(l_{S_2}) + \sigma_c(l_{T_1}) + \sigma_c(l_{T_2})$ is a penalty function for such a move operation and depends on the lengths of the four substrings involved in the move operation.

This decomposition in a sum will be required in the algorithm. An example of a non-overlapping move alignment is shown in Fig. 11. We now introduce different scoring schemes that will influence the time complexity of the results.

Definition 4. General scoring scheme: *the standard alignment of substrings is done with affine gap penalties, $\sigma_c(l)$ is an arbitrary function and the scoring matrix $\sigma(a, b)$ is arbitrary.* Concave scoring scheme: *the standard alignment of substrings is done with constant indel penalties, $\sigma_c(l)$ is a concave function and the scoring matrix $\sigma(a, b)$ is arbitrary.* Restricted scoring scheme: *the standard alignment of substrings is done with constant indel penalties and $\sigma_c(l)$ is a constant. The scoring matrix $\sigma(a, b)$ is selected such that the number of distinct values of $DIST[i, j] - DIST[i, j - 1]$ is bounded by a constant ψ . For more details on the restricted scoring scheme, we refer to [24].*

2.3 Definition of Alignment with Non-overlapping Moves, Inversions and Tandem-Duplications

In favor of simplicity, we assume constant indel penalties and constant penalties for block operations in the treatment of this problem. However, the scoring schemes of section 2.2 could be used here as well.

Definition 5. *The problem of aligning S and T with non-overlapping moves, reversals and tandem duplications is to find $d \in \mathbb{N}$ and d -decompositions of S and T such that the score $\sum_{i=1}^d \max\{\delta(S_i, T_i), \delta_c(S_i, T_i) + \sigma_c, \delta_d(S_i, T_i) + \sigma_d, \delta_r(S_i, T_i) + \sigma_r\}$ is maximal for all $d \in \mathbb{N}, S_1 \dots S_d \in \mathcal{M}_d(S)$ and $T_1 \dots T_d \in \mathcal{M}_d(T)$, where $\delta_d(A, B) = \max\{\delta(AA, B), \delta(A, BB)\}$ and $\delta_r(A, B) = \delta(A, \overline{B})$. Where $\sigma_c, \sigma_d, \sigma_r$ are penalties for move operations, duplications or reversals respectively. The optimal score is denoted by $\delta_{drm}(S, T)$.*

2.4 Other Preliminaries

The notion of $DIST[i, j]$ arrays as used in [24,25] can be defined as follows.

Definition 6. *Let $DIST_{S,T}[i, j], 0 \leq i \leq j \leq m$ denote the score of the optimal alignment of $T[i..j]$ with S .*

Let us further introduce input vectors I , output vectors O and a matrix OUT .

Definition 7. *Let $OUT_{S,T}[i, j] = I[i] + DIST_{S,T}[i, j] + \sigma_c(j - i), 0 \leq i \leq j \leq m$. Then I is an arbitrary vector called input vector and $O[j] = \max_i OUT[i, j]$ is called output vector containing all the column maxima of OUT .*

Lemma 1. *$DIST_{S,T}[i, j]$ arrays are inverse monge arrays.*

The following lemma will become useful in the selection of the parameters.

Lemma 2. *If $f(l)$ is concave then $f_i(j', j) := f(j - j'), 0 \leq j' \leq m, 0 \leq j \leq m$ is inverse Monge.*

A proof of these lemmas can be found with the definition of inverse Monge in the *Appendix*.

Corollary 1. $OUT_{S,T}[i, j] = DIST_{S,T}[i, j] + f(j - i) + I[i]$ is inverse Monge for f concave and constant indel penalties.

Proof. Due to lemma 1 $DIST_{S,T}$ arrays with constant indel penalties are inverse Monge. The rest follows from definition 8 and lemma 2 (in *Appendix*).

Using our observations and the results from [24,25], we can conclude with the following results: (i) For arbitrary penalty functions σ_c and affine gap penalties as in the general scoring scheme, we can compute $DIST_{S[0..i],T}$ from $DIST_{S[0..i-1],T}$ in $O(m^2)$ as indicated in the *Appendix*. Then we can trivially compute the output vector O as in definition 7 in $O(m^2)$ time by inspecting all entries. (ii) For concave functions σ_c and constant indel penalties as in the concave scoring scheme, we can compute a representation of $DIST_{S[0..i],T}$ from $DIST_{S[0..i-1],T}$ in $O(m \log m)$ time using the data structure of [25]. Then since OUT is inverse Monge, we can compute the output vector O by applying the algorithm of [26] for searching all column maxima in a Monge array to OUT . This algorithm will access $O(m)$ entries of the array and hence the computation of O will take $O(m \log m)$ time since we can access an entry of $DIST$ in the data structure of [25] in $O(\log m)$ time. (iii) For constant functions σ_c , constant indel penalties and a restricted scoring matrix as in the restricted scoring scheme, we can compute a representation of $DIST_{S[0..i],T}$ from $DIST_{S[0..i-1],T}$ in $O(m)$ time due to section 6 of [25] and then compute the output vector O using the algorithm of [24] in $O(m)$ time.

Note that the $O(m \log m)$ and $O(m)$ results rely heavily on the fact that $DIST$ arrays are Monge. Since this is not true for affine gap penalties these results cannot be easily extended to affine gap penalties.

3 Algorithms

3.1 Alignment with Non-overlapping Moves

Let $SCO_{S,T}[i, j]$ be the score of the optimal alignment of $S[0..i]$ and $T[0..j]$ with non-overlapping moves. Then the following recurrence relation and initialization of the table will lead to a dynamic programming solution for the problem.

Base Case: $SCO_{S,T}[i, 0] = i \cdot \sigma_I$ and $SCO_{S,T}[0, j] = j \cdot \sigma_I$

$$\text{Recurrence: } SCO_{S,T}[i, j] = \max \begin{cases} SCO_{S,T}[i, j - 1] + \sigma_I \\ SCO_{S,T}[i - 1, j - 1] + \sigma(S[i], T[j]) \\ SCO_{S,T}[i - 1, j] + \sigma_I \\ MOVE \end{cases}$$

where

$$MOVE = \max_{0 \leq i' < i, 0 \leq j' < j} \{ SCO_{S,T}[i', j'] + \delta_c(S[i'..i], T[j'..j]) + \sigma_c(l_{S_{d_1}}) + \sigma_c(l_{S_{d_2}}) + \sigma_c(l_{T_{d_1}}) + \sigma_c(l_{T_{d_2}}) \}$$

Proof. Let us consider an optimal non-overlapping move alignment of $S[0..i]$ with $T[0..j]$. Let S_d and T_d be the last substrings of the optimal d -composition of $S[0..i]$ and $T[0..j]$. Then there are two cases: (1) S_d and T_d are aligned using the cyclic string comparison or (2) S_d and T_d are aligned by the standard alignment. In case (1), we know that $SCO_{S,T}[i, j] = SCO_{S,T}[i', j'] + \delta_c(S_d, T_d)$ which is considered in $MOVE$. In case (2), we are in the usual standard alignment cases. Hence, we consider all the cases and therefore find the optimal solution.

With the goal of economizing the computation of the table let us rewrite $MOVE$ as

$$\begin{aligned} & \max_{\substack{0 \leq i' < i'' < i \\ 0 \leq j' < j'' < j}} \{ SCO_{S,T}[i', j'] + DIST_{S[i''..i], T[j', j'']} + \sigma_c(j'' - j') + \sigma_c(i - i'') \\ & \quad + DIST_{S[i'..i''], T[j'', j]} + \sigma_c(j - j'') + \sigma_c(i'' - i') \}. \end{aligned}$$

To compute $MOVE$ for a given i' and i'' we can first maximize over j' and then over j'' . That is, we can first compute the output row of the first $DIST_{S[i''..i], T}$ array and then, given that output, compute the output of the second $DIST_{S[i'..i''], T}$ array. This leads to the following definitions (illustrated in Fig. 2).

$$O_1[j''] = \max_{0 \leq j' < j''} SCO_{S,T}[i', j'] + DIST_{S[i''..i], T[j', j'']} + \sigma_c(j'' - j') + \sigma_c(i - i'') \quad (1)$$

$$O_2[j] = \max_{0 \leq j'' < j} O_1[j''] + DIST_{S[i'..i''], T[j'', j]} + \sigma_c(j - j'') + \sigma_c(i'' - i') \quad (2)$$

Given $DIST_{S[i''..i], T[j', j'']}$ and $DIST_{S[i'..i''], T[j'', j]}$, $O_1[j'']$ and $O_2[j]$ can be computed efficiently using the results from section 2.4 since both of them are output vectors as in definition 7.

DP_MOVE

- 1: **for all** i, j **such that** $0 \leq i \leq n, 0 \leq j \leq m$ **do**
- 2: {base case}
- 3: $SCO[i, 0] := i \cdot \sigma_I$
- 4: $SCO[0, j] := j \cdot \sigma_I$
- 5: $SCO[i, j] := -\infty$ if $i \neq 0, j \neq 0$
- 6: **end for**
- 7: **for** i **from** 0 **to** n **do**
- 8: **if** $i \geq 1$ **then**
- 9: **for** j **from** 1 **to** m **do**
- 10: {standard alignment recurrence}
- 11: $SCO[i, j] := \max\{SCO[i, j], SCO[i - 1, j] + \sigma_I, SCO[i, j - 1] + \sigma_I, SCO[i - 1, j - 1] + \sigma(S[i], T[j])\}$

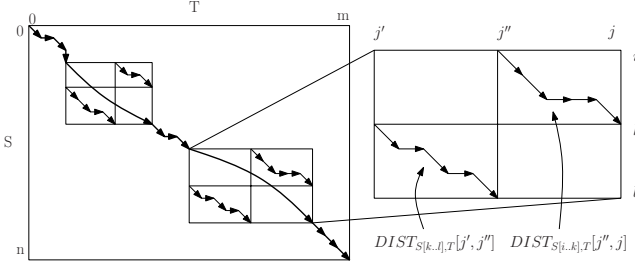


Fig. 2. An illustration of the computation of a move operation in DP_MOVE. Since the scores are additive: $SCO[l, j] = SCO[i, j'] + DIST_{S[k..l], T}[j'', j''] + DIST_{S[i..k], T}[j'', j]$. In DP_MOVE this is maximized for all $i < k < l, j' < j'' < j$.

```

12:   end for
13: end if
14: for k from i to n do
15:   {move operations}
16:    $DIST_{S[i..k], T} := calcDist(DIST_{S[i..k-1], T})$ 
17:   for l from k to n do
18:      $DIST_{S[k..l], T} := calcDist(DIST_{S[k..l-1], T})$ 
19:      $O_1 := calcOutput(OUT[j', j''] = SCO[i, j'] + DIST_{S[k..l], T}[j', j''] +$ 
20:        $\sigma_c(j'' - j') + \sigma_c(l - k))$ 
21:      $O_2 := calcOutput(OUT[j'', j] = O_1[j''] + DIST_{S[i..k], T}[j'', j] + \sigma_c(j -$ 
22:        $j'') + \sigma_c(k - i))$ 
23:     for j from 0 to m do
24:        $SCO[l, j] := \max\{SCO[l, j], O_2[j]\}$ 
25:     end for
26:   end for
27: end for

```

Where $calcDist(DIST_{S[0..l-1], T})$ computes $DIST_{S[0..l], T}$ from $DIST_{S[0..l-1], T}$ and $calcOutput(OUT[i, j])$ computes O as in definition [7](#).

Correctness. To show the correctness of the algorithm it suffices to show that we process all edges in the edit graph and whenever we process an edge $(u, v) \in E$ we have completed the computation of the score of u and any of its predecessors in topological order [22](#). The computation of the score of a node u is completed iff all the incoming edges of u have been processed. This can be proven by induction. In our edit graph, the only edges are either due to the standard alignment, or due to move operations, as can be seen in the recurrence. Assuming that when computing the i -th row of SCO , all edges due to move operations starting in a row $i' < i$ have already been processed and the computation of any node (i', j) with $i' < i$ has been completed, we can see that the processing of the edges due to the standard alignment recurrence ending in the i -th row as done on line [9](#) to [12](#) is legitimate. After having processed those edges, we have completed

the computation of all edges ending on any node in the i -th row and hence can compute any edge due to move operations starting on that row which is done on line 14 to 24. We compute all such edges. Consequently, when we advance to the computation of row $i + 1$ the assumption is again true.

Using the results from section 2.4 we can analyze the runtime of the algorithm and conclude with the following theorem.

Theorem 1. *The problem of aligning S and T , $|S| = n, |T| = m$, with non-overlapping moves can be solved in $O(n^3m^2)$ time and $O(nm + m^2)$ space for general scoring schemes, in $O(n^3m \log m)$ time and $O(nm + m^2)$ space for concave scoring schemes and in $O(n^3m)$ time and $O(nm)$ space for restricted scoring schemes.*

3.2 Alignment with Non-overlapping Moves, Inversions, and Tandem Duplications

The dynamic programming recurrence of non-overlapping move operations extends nicely to this problem.

$$\text{Base Case: } SCO_{S,T}[i, 0] = i \cdot \sigma_I \text{ and } SCO_{S,T}[0, j] = j \cdot \sigma_I$$

$$\text{Recurrence: } SCO_{S,T}[i, j] = \max \begin{cases} SCO_{S,T}[i, j - 1] + \sigma_I \\ SCO_{S,T}[i - 1, j - 1] + \sigma(S[i], T[j]) \\ SCO_{S,T}[i - 1, j] + \sigma_I \\ MOVE + \sigma_c \\ S_DUPLICATE + \sigma_d \\ T_DUPLICATE + \sigma_d \\ REVERSE + \sigma_r \end{cases}$$

where

$$\begin{aligned} MOVE &= \max_{0 \leq i' < i, 0 \leq j' < j} \{SCO_{S,T}[i', j'] + \delta_c(S[i'..i], T[j'..j])\} \\ S_DUPLICATE &= \max_{0 \leq i' < i, 0 \leq j' < j'' < j} \{SCO_{S,T}[i', j'] + \delta(S[i'..i], T[j'..j'']) \\ &\quad + \delta(S[i'..i], T[j''..j])\} \\ T_DUPLICATE &= \max_{0 \leq i' < i'', 0 \leq j' < j} \{SCO_{S,T}[i', j'] + \delta(S[i'..i''], T[j'..j]) \\ &\quad + \delta(S[i''..i], T[j'..j])\} \\ REVERSE &= \max_{0 \leq i' < i, 0 \leq j' < j} \{SCO_{S,T}[i', j'] + \delta(\overline{S[i'..i]}, T[j'..j])\} \end{aligned}$$

A proof of this recurrence is analogous to the proof for non-overlapping moves and is omitted.

We have split tandem duplication into tandem duplication of a substring of S and tandem duplication of a substring of T . We have already shown how $MOVE$ can be treated and in 11 it is shown how to handle $REVERSE$. $S_DUPLICATE$ can be done as follows. We calculate $DIST_{S[i..k]}$. Then, we

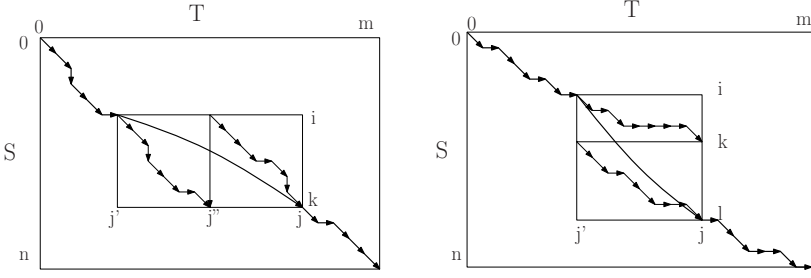


Fig. 3. An illustration on how duplications are treated

first use $SCO_{S,T}[i, j']$ as input vector for $DIST_{S[i..k]}$ to get $O_1[j']$ and then use $O_1[j']$ as input for $DIST_{S[i..k]}$ to get $O_2[j]$. $T_DUPLICATE$ can be computed by computing the output vector of $DIST_{S[i..k], T}[j', j] + DIST_{S[k..l], T}[j', j]$ for the input vector $SCO_{S,T}[i, j']$. Note, that this array is well defined and is again inverse Monge because it is a sum of two inverse Monge arrays. Using these observations which are illustrated in Fig. 3 we can now present our algorithm for this problem.

DP_MOVE_INV_DUPL

```

1: {initialize the table as in DP_MOVE}
2: for  $i$  from 0 to  $n$  do
3:   {compute REVERSE as done in [12]}
4:   {compute the standard alignment recurrence as in DP_MOVE}
5:   {treat MOVE as done in DP_MOVE}
6:   for  $k$  from  $i$  to  $n$  do
7:     {duplication}
8:      $DIST_{S[i..k], T} := calcDist(DIST_{S[i..k-1], T})$ 
9:      $O_1 := calcOutput(OUT[j', j''] = SCO[i, j'] + DIST_{S[i..k], T}[j', j''])$ 
10:     $O_2 := calcOutput(OUT[j'', j] = O_1[j''] + DIST_{S[i..k], T}[j'', j])$ 
11:    for  $l$  from  $k$  to  $n$  do
12:       $DIST_{S[k..l], T} := calcDist(DIST_{S[k..l-1], T})$ 
13:       $O := calcOutput(OUT[j', j] = SCO[i, j'] + DIST_{S[i..k], T}[j', j] +$ 
14:         $DIST_{S[k..l], T}[j', j])$ 
15:      for  $j$  from 0 to  $m$  do
16:         $SCO[l, j] := \max\{SCO[l, j], O[j] + \sigma_d, O_2[j] + \sigma_d\}$ 
17:      end for
18:    end for
19:  end for

```

A proof of the correctness of the algorithm is analogous to the proof for DP_MOVE. This proof however reveals that it is important to process edges due to REVERSE *before* the standard alignment recurrence.

Theorem 2. *The problem of aligning S and T with non-overlapping moves, reversals and tandem duplications can be solved in $O(n^3m^2)$ time and $O(nm + m^2)$ space for general scoring schemes, in $O(n^3m \log m)$ time and $O(mn + m^2)$ space for concave scoring schemes and in $O(n^3m)$ time and $O(nm)$ space for restricted scoring schemes, where $n = |S|$ and $m = |T|$.*

4 Implementation and Experiments

We have implemented an $O(n^5)$ version of the algorithm with constant gap penalties in [2]. This implementation has proven useful for aligning sequences of up to about 400 AA, taking a few hours to compute the alignment. We have tested the algorithm on real data and were able to confirm a number of examples found in [2]. In addition we run the algorithm on an example found in [18]. This alignment is shown in figure 4 and is compared with a standard alignment obtained from Darwin [27].

```

DP_MOVE
Seq1: RPSTVPLP_NTQ__A_LAMA_[GTAYKGYVKVP_KPTGVK_KGWQRAYAVVCDCKFLFYLDLPEGK_STQPGVIASQVLDLRDDEFAVSSVLA
Seq2: LSSADNDPEDSQHSSLLSLTQ[DSVFEGLWSVFNKQNRRRHGHWKRQYVIVSSRKIIIFYNSDIDKHNTTDAVL___ILLD_SKVYHVRVSTQ

Seq1: SDVIHATRRDIPICIFRV_T_ASLLG_S__PSKTSSL_L_ILTENENEKRR|GP_KPKAHQF_SIKSPSPPTQCSHCTSLMVGLLR__QGYACE
Seq2: GDVIRADAKEIPRIFQLLYAGE_GASHRPDEQSQLDVSVLHGNCNEERP|GTIVHKGHEFVHI_TYHMPTACEVCPKPLWHMFKPPAAEYCK

Seq1: VCAPFSCHVSKCDS_APQV__PIPPE_QSKRP___LGVDVQ_RGI|WVGILEGLQAILHKNRRLRSQVV_HVAQEAYD_S_SLPLI
Seq2: RCRNKIHKEHVDKHDPLAP^KLNHDPRSARDMLLLAATPEDQSL]WVARL___LKRI_QKSGYKAAASYNNNSTDGSKISPSQSTR

DARWIN
Seq1: RPSTVPLPNTQALAMAGPKPKA^HQFSIKSPSPPTQCSHCTSLMVGLLRQGYACEVCAPFSCHVSKCDSAPQV^PIPPEQSKRPLGVDVQRGIG
Seq2: _____LSSADNDPEDSQHS__SLLSLTQ_____D

Seq1: TAYKGYVKVPKPTGVKK__GWQRAYAVVCDCKFLFY__DLPEGKSTQPGVIASQVLDLRDDEFAVSSVSLASDVIHATRRDIPICIFRV_____
Seq2: SVFEGWLSVFNKQNRRRHGHWKRQYVIVSSRKIIIFYNSDIDKHNTTDAVLILLD_SKVYHVRVSTQGDVIRADAKEIPRIFQLLYAGE

Seq1: _____TASLLGS
Seq2: GASHRPDEQSQLDVSVLHGNCNEERP^GTIVHKGHEFVHITYHMPTACEVCPKPLWHMFKPPAAEYCKRCRNKIHKEHVDKHDPLAP^KLNHDP

Seq1: PSKTSSLLILTENENEKRRKWGIL_____EGLQAILHKNRRLRSQVVHVAQEAYDSSSLPLI
Seq2: PRSARDMLLLAATPEDQSLWVARLLKRIQKSGYKAAASYNNN_____STDGSKISPSQSTR
    
```

Fig. 4. An example found in [18]. In this figure we compare an alignment computed with our new algorithm and an alignment done with Darwin [27]. The brackets '[' and ']' indicate the boundary of the substrings containing a move operation and '^' marks the position of the split. Seq1: Q7TT49 AA 1005-1241; Seq2: Q9VXE3 AA 1112-1367. Using the annotation of SMART we have marked the domains involved in the move operation. Pleckstrin homology phospholipid binding domain is shown in blue, protein kinase C-type diacylglycerol binding domain is shown in yellow.

5 Conclusions

In this paper, we have presented a number of new alignment problems extending the notion of non-overlapping inversions to non-overlapping moves and tandem duplications. For all of them we found algorithms that solve the problems exactly and can be implemented to run in $O(n^2)$ space and $O(n^5)$, $O(n^4 \log n)$ or $O(n^4)$

² Available from the authors upon request.

time depending on the scoring scheme used. We believe that this approach may yield new insights by finding the best alignment of two sequences, and think that it is justifiable due to the rarity of such events in nature. Using the implementation of the $O(n^5)$ variant of the algorithm, we were able to align previously identified cases of pairs of sequences with move operations. Furthermore, these experiments also showed the necessity of an $O(n^4 \log n)$ implementation to be applicable to large sequences, which are more likely to contain a move.

Acknowledgments

We thank Manuel Gil, Gaston H. Gonnet, Gina M. Cannarozzi and three anonymous referees for helpful critiques and discussions.

References

1. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* 48(3), 443–453 (1970)
2. Fliess, A., Motro, B., Unger, R.: Swaps in protein sequences. *Proteins.* 48(2), 377–387 (2002)
3. Lopresti, D., Tomkins, A.: Block edit models for approximate string matching. *Theor. Comput. Sci.* 181(1), 159–179 (1997)
4. Shapira, D., Storer, J.A.: Edit distance with move operations. In: Apostolico, A., Takeda, M. (eds.) *CPM 2002. LNCS*, vol. 2373, pp. 85–98. Springer, Heidelberg (2002)
5. Chrobak, M., Kolman, P., Sgall, J.: The greedy algorithm for the minimum common string partition problem. *ACM Trans. Algorithms* 1(2), 350–366 (2005)
6. Cormode, G., Muthukrishnan, S.: The string edit distance matching problem with moves. In: *SODA '02. Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, Philadelphia, PA. Society for Industrial and Applied Mathematics, pp. 667–676. ACM Press, New York (2002)
7. Caprara, A.: Sorting by reversals is difficult. In: *RECOMB '97. Proceedings of the first annual international conference on Computational molecular biology*, pp. 75–83. ACM Press, New York (1997)
8. Schoeninger, M., Waterman, M.S.: A local algorithm for dna sequence alignment with inversions. *Bull. Math. Biol.* 54(4), 521–536 (1992)
9. Chen, Z.Z., Gao, Y., Lin, G., Niewiadomski, R., Wang, Y., Wu, J.: A space-efficient algorithm for sequence alignment with inversions and reversals. *Theor. Comput. Sci.* 325(3), 361–372 (2004)
10. do Lago, A.P., Muchnik, I.: A sparse dynamic programming algorithm for alignment with non-overlapping inversions. *Theoret. Informatics Appl.* 39(1), 175–189 (2005)
11. Alves, C.E.R., do Lago, A.P., Vellozo, A.F.: Alignment with non-overlapping inversions in $o(n^3 \log n)$ time. In: *Proceedings of GRACO 2005. Electronic Notes in Discrete Mathematics*, vol. 19, pp. 365–371. Elsevier, Amsterdam (2005)
12. Vellozo, A.F., Alves, C.E.R., do Lago, A.P.: Alignment with non-overlapping inversions in $o(n^3)$ -time. In: Bücher, P., Moret, B.M.E. (eds.) *WABI 2006. LNCS (LNBI)*, vol. 4175, pp. 186–196. Springer, Heidelberg (2006)

13. Apic, G., Gough, J., Teichmann, S.A.: Domain combinations in archaeal, eubacterial and eukaryotic proteomes. *J. Mol. Biol.* 310(2), 311–325 (2001)
14. Kaessmann, H., Zöllner, S., Nekrutenko, A., Li, W.H.: Signatures of domain shuffling in the human genome. *Genome Res.* 12(11), 1642–1650 (2002)
15. Liu, M., Walch, H., Wu, S., Grigoriev, A.: Significant expansion of exon-bordering protein domains during animal proteome evolution. *Nucleic Acids Res.* 33(1), 95–105 (2005)
16. Vibanovski, M.D., Sakabe, N.J., de Oliveira, R.S., de Souza, S.J.: Signs of ancient and modern exon-shuffling are correlated to the distribution of ancient and modern domains along proteins. *J. Mol. Evol.* 61(3), 341–350 (2005)
17. Bashton, M., Chothia, C.: The geometry of domain combination in proteins. *J. Mol. Biol.* 315(4), 927–939 (2002)
18. Shandala, T., Gregory, S.L., Dalton, H.E., Smallhorn, M., Saint, R.: Citron kinase is an essential effector of the pbl-activated rho signalling pathway in drosophila melanogaster. *Development.* 131(20), 5053–5063 (2004)
19. Andrade, M.A., Perez-Iratxeta, C., Ponting, C.P.: Protein repeats: structures, functions, and evolution. *J. Struct. Biol.* 134(2-3), 117–131 (2001)
20. Marcotte, E.M., Pellegrini, M., Yeates, T.O., Eisenberg, D.: A census of protein repeats. *J. Mol. Biol.* 293(1), 151–160 (1999)
21. Maes, M.: On a cyclic string-to-string correction problem. *Inf. Process. Lett.* 35(2), 73–78 (1990)
22. Myers, E.W.: An overview of sequence comparison algorithms in molecular biology. Technical Report 91-29, Univ. of Arizona, Dept. of Computer Science (1991)
23. Gusfield, D.: Algorithms on Strings, Trees, and Sequences: computer science and computational biology. Press Syndicate of the University of Cambridge, Cambridge (1997/1999)
24. Landau, G.M., Ziv-Ukelson, M.: On the common substring alignment problem. *J. Algorithms* 41(2), 338–354 (2001)
25. Schmidt, J.P.: All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings. *SIAM J. Comput.* 27(4), 972–992 (1998)
26. Aggarwal, A., Klawe, M.M., Moran, S., Shor, P., Wilber, R.: Geometric applications of a matrix-searching algorithm. *Algorithmica* 2(1), 195–208 (1987)
27. Gonnet, G.H., Hallett, M.T., Korostensky, C., Bernardin, L.: Darwin v. 2.0: An interpreted computer language for the biosciences. *Bioinformatics* 16(2), 101–103 (2000)
28. Monge, G.: Déblai et remblai. Mémoires de l'Académie Royale des Sciences (1781)

Appendix

Monge Properties

For a proof of lemma [11](#) and lemma [2](#) we have to define the notion of Monge arrays [28](#) first:

Definition 8. A matrix $M[0 \dots n; 0 \dots m]$ is Monge if for all $i = 1 \dots n, j = 1 \dots m$

$$M[i, j] + M[i - 1, j - 1] \leq M[i - 1, j] + M[i, j - 1]$$

and it is called inverse Monge if for all $i = 1 \dots n, j = 1 \dots m$

$$M[i, j] + M[i - 1, j - 1] \geq M[i - 1, j] + M[i, j - 1]$$

Then the proof for lemma 2 goes as follows.

Proof

$$\begin{aligned} f_l(j' - 1, j - 1) + f_l(j', j) &= 2f(j - j') \\ &\geq 2 \frac{f(j - j' - 1) + f(j - j' + 1)}{2} \\ &= f_l(j' - 1, j) + f_l(j', j - 1) \end{aligned}$$

Where the inequality follows from the definition of concave. A function f is concave iff $f(tx + (1 - t)y) \geq tf(x) + (1 - t)f(y)$ holds for all $x, y \in \mathbb{R}, t \in [0, 1]$. In other words every point on a secant is below the function. In the proof we used $t = 1/2$ as shown in Fig. 5.

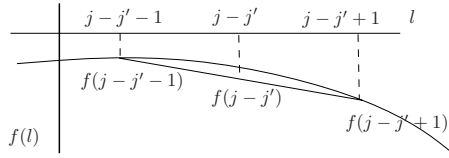


Fig. 5. For concave functions any point on a secant is below the function

Note that if any three points $f(j - 1), f(j), f(j + 1), 0 < j < m$ are not in concave position the resulting array will not be inverse Monge. That is, lemma 2 holds with equivalence if we restrict the definition of concave to values in $\{0 \dots m + 1\} \subseteq \mathbb{N}$.

The proof for Lemma 1 is analogous to the proof in 25.

Proof The paths represented by $DIST_{S,T}[i - 1, j]$ and $DIST_{S,T}[i, j - 1]$ have to cross properly in a vertex v as shown in figure 6. Therefore, we have

$$\begin{aligned} DIST_{S,T}[i - 1, j] + DIST_{S,T}[i, j - 1] &= (a + b) + (c + d) \\ &= (a + d) + (b + c) \\ &\leq g + f \\ &= DIST_{S,T}[i, j] + DIST_{S,T}[i - 1, j - 1] \end{aligned}$$

where the inequality follows from $a + d \leq f$ and $b + c \leq g$. Where $a + d \leq f$ holds since $a + d$ is the length of a path from $(0, i - 1)$ to $(n, j - 1)$ and the optimal path of length f can only be longer and analogously $b + c \leq g$.

In figure 6(2) a counter example for affine gap penalties is given. This is a counter example because the total number of gaps on the paths from $(0, i)$ to $(n, j - 1)$ and $(0, i - 1)$ to (n, j) is smaller than the total number of gaps on the paths from $(0, i)$ to (n, j) and $(0, i - 1)$ to $(n, j - 1)$. Hence $DIST_{S,T}$ arrays cannot be monge for large initial penalties.

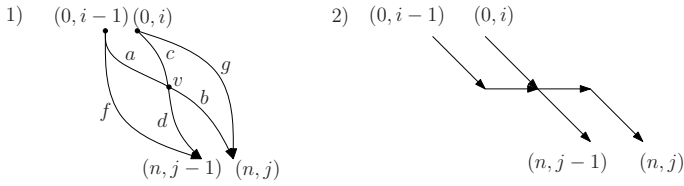


Fig. 6. 1) illustrates that the path from vertex $(0, i)$ to $(n, j - 1)$ and the path from vertex $(0, i - 1)$ to (n, j) in the grid graph have to cross in a common vertex v . 2) gives a counter example for affine gap penalties.

Extension of *DIST* Arrays

This simple algorithm is inspired by [25]. It is repeated here to provide an idea on how to extend our algorithms to affine gap penalties.

For the base cases we observe that $DIST_{S,T}[i, j] = B_{S,T[i..j]}[n, j - i]$ in particular for constant indel penalties $DIST_{S[0..0],T}[i, j] = (j - i) \cdot \sigma_I$ and $DIST_{S[0..l],T}[i, i] = l \cdot \sigma_I$.

By mapping the standard alignment recurrence to *DIST* arrays we obtain:

$$DIST_{S[0..l],T}[i, j] = \max \begin{cases} DIST_{S[0..l-1],T}[i, j] + \sigma_I \\ DIST_{S[0..l-1],T}[i, j - 1] + \sigma(S[l], T[j]) \\ DIST_{S[0..l],T}[i, j - 1] + \sigma_I \end{cases}$$

Therefore, we can compute $DIST_{S[0..l],T}$ given $DIST_{S[0..l-1],T}$ in $O(m^2)$ time. This recurrence can be extended to include affine gap penalties by mapping the more complicated recurrence for the standard alignment with affine gap penalties to *DIST* arrays.

Counting Minimum Weighted Dominating Sets

Fedor V. Fomin and Alexey A. Stepanov

Department of Informatics, University of Bergen, PO Box 7803, 5020 Bergen, Norway
fomin@ii.uib.no, ljosha@ljosha.org

Abstract. We show how to count all minimum weighted dominating sets of a graph on n vertices in time $\mathcal{O}(1.5535^n)$. Our algorithm is a combination of branch and bound approach along with dynamic programming on graphs with bounded treewidth. To achieve $\mathcal{O}(1.5535^n)$ bound we introduce a technique of measuring running time of our algorithm by combining *measure and conquer* approach with linear programming.

1 Introduction

The story of exact (exponential-time) algorithms for hard problems dates back to the sixties and seventies but especially the last decade has seen a growing interest in new techniques for such type of algorithms. We refer to the following recent surveys [8,21,23] for an overview of the field.

Counting problem is a natural extension of a decision problem, where instead of deciding on the existence of a solution, the task is to find the number of solutions. Valiant [22] defined the class #P and showed that computing the permanent is #P-complete. Many NP-complete as well as some problems in P can have their counting versions to be #P-hard. In particular, counting minimum dominating set is #P-hard [14] even when restricted to planar instances. There is a lot of research going on counting problems and the complexity class #P. (See the book by Jerrum [15] for an introduction.)

While exact algorithms for many NP-complete decision problems were studied quite intensively, there are not so many results on exact algorithms for #P-complete problems in the literature. For a long time the only known exact algorithm for an #P-complete problem was time $2^n \cdot n^{\mathcal{O}(1)}$ counting perfect matching in bipartite graph algorithm due to Ryser [20]. There are known exact algorithms for different counting versions of satisfiability problem like counting maximum weighted models for 2SAT and 3SAT [5,11,24], CSP [1], and counting maximum weighted independent sets [4,6].

The principle of inclusion and exclusion was used by Karp [16] to solve many counting problems such as Hamiltonian path, sequencing and bin packing. Lately, Bax and Franklin [2] generalized this technique to the finite-difference approach and obtained algorithms for counting paths and cycles of a given length in a directed graph. Very recently time $2^n \cdot n^{\mathcal{O}(1)}$ algorithm computing chromatic polynomial (counting all k -colorings for each k) was obtained in [3,17].

Previous results. The dominating set problem is one of the classical NP-complete graph optimization problem which fits into the broader class of domination and covering problems. Hundreds of papers have been written on them (see e.g. the survey [13] by Haynes et al.). Recently, several groups of authors independently obtained exact algorithms that solve minimum dominating set problem in a graph on n vertices faster than the trivial $2^n \cdot n^{O(1)}$ -time brute force algorithm [10,12,19]. The fastest known algorithm computes a minimum dominating set of a graph in time $\mathcal{O}(1.5137^n)$ [7]. The algorithm from [7] cannot be used to compute a dominating set of minimum weight (in a weighted graph), however, as it was observed in [9], the problem can be solved in time $\mathcal{O}(1.5780^n)$ by similar techniques. In the same paper it was shown that all minimal dominating sets can be listed in time $\mathcal{O}(1.7697^n)$ (later improved to $\mathcal{O}(1.7170^n)$), which implies that minimum dominating sets can be counted in this time.

Our results. In this paper we give an algorithm that counts minimum weight dominating sets in a weighted graph on n vertices in time $\mathcal{O}(1.5535^n)$. The basic idea is as follows: First we turn the instance of the dominating set problem to the instance of a set cover problem (this is the idea used by Grandoni [12] for decision problem) and perform branching on large sets and sets of size three containing elements of high degree. When branching is complete, we turn the instance of set cover into an instance of red/blue domination on bipartite graphs and use dynamic programming to count all solutions.

The novel and the most difficult part of the paper is the analysis of the algorithm. To analyze the running time we need to investigate the behavior of the pathwidth of a graph as a function of the measure of the corresponding set cover instance. The difficulty here is to find the measure of the problem that “balances” branching and dynamic programming parts of the algorithm. To choose the right measure we express the bounds on pathwidth as a linear program.

Combining branching with dynamic programming is a general approach that works for many decision and counting problems [6,18] and our technique can be used to improve the analysis of many algorithms of this type.

Finally, let us remark that the running time $\mathcal{O}(1.5535^n)$ of our algorithm counting weight dominating sets is even (slightly) better than the running time of the minimum weighted dominating set algorithm from [9].

2 Preliminaries

Let $G = (V, E)$ be an n -vertex undirected, simple graph without loops. We denote by $\Delta(G)$ the maximum vertex degree in G . For a vertex $v \in V$ we denote the set of its neighbors by $N(v)$ and its *closed neighborhood* by $N[v] = N(v) \cup \{v\}$.

A set $D \subseteq V$ is called a *dominating set* for G if every vertex from V is either in D , or adjacent to some vertex in D . Given a weight function $w : V \rightarrow \mathbb{R}^+$ the *Minimum Weighted Dominating Set* problem (MWDS) asks to find a dominating set $D \subseteq V$ of minimum weight $w(D) = \sum_{v \in D} w(v)$. We denote by $\#MWDS$ the

counting version of MWDS where the objective is to count all dominating sets of minimum weight.

Let \mathcal{U} be a set of elements and \mathcal{S} be a collection of non-empty subsets of \mathcal{U} . Given a weight function $w : \mathcal{S} \rightarrow \mathbb{R}^+$ the *Minimum Weighted Set Cover* problem (MWSC) asks to find a subset $\mathcal{S}^* \subseteq \mathcal{S}$ of minimum weight $w(\mathcal{S}^*) = \sum_{S \in \mathcal{S}^*} w(S)$ which *covers* \mathcal{U} ; that is,

$$\bigcup_{S \in \mathcal{S}^*} S = \mathcal{U}.$$

We denote by #MWSC the problem of counting set covers of minimum weight.

The *frequency* of an element $u \in \mathcal{U}$ is the number of sets $S \in \mathcal{S}$ in which u is contained. We denote it by $\text{freq}(u)$.

#MWDS can be reduced to #MWSC by imposing $\mathcal{U} = V$ and $\mathcal{S} = \{N[v] \mid v \in V\}$. Given a weight function $w(v)$ for MWDS we define weight for $S \in \mathcal{S}$ as follows.

$$w(S) = w(S = \{N[v] \mid v \in V\}) = w(v).$$

Thus D is a dominating set of G if and only if $\{N[v] \mid v \in D\}$ is a set cover of $\{N[v] \mid v \in V\}$.

We also need a reduction from MWSC to a version of weighted dominating set problem called *minimum red/blue weighted dominating set* (RBWDS) problem. For a bipartite graph $G = (V, E)$ with a bipartition $V = V_{Red} \cup V_{Blue}$ and a weight function $w : V \rightarrow \mathbb{R}^+$, a subset $D \subseteq V_{Red}$ is *red-blue dominating set* if every vertex in V_{Blue} is adjacent to a vertex of D . RBWDS problem is to determine the minimum weight of a red/blue dominating set in G .

With an instance $(\mathcal{U}, \mathcal{S}, w)$ of MWSC one can associate an *incidence graph* $G_{\mathcal{S}}$, which is a bipartite graph on a vertex set $\mathcal{S} \cup \mathcal{U}$ with a bipartition $(\mathcal{S}, \mathcal{U})$, and vertices $S \in \mathcal{S}$ and $u \in \mathcal{U}$ are adjacent if and only if u is an element of S . Let us observe, that \mathcal{S} has a cover of weight k if and only if its incidence graph has a red-blue (with $V_{Red} = \mathcal{S}$ and $V_{Blue} = \mathcal{U}$) dominating set of weight k .

Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair $\langle \{X_i \mid i \in I\}, T \rangle$, where each X_i is a subset of V , called a *bag*, and T is a tree with the elements of I as vertices. The following three properties must hold:

1. $\bigcup_{i \in I} X_i = V$.
2. For every edge $(u, v) \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$.
3. For all $i, j, k \in I$, if j lies on the path between i and k in T , then $X_i \cap X_k \subseteq X_j$.

The *width* of $\langle \{X_i \mid i \in I\}, T \rangle$ equals $\max\{|X_i| \mid i \in I\} - 1$. The *treewidth* of G is the minimum k such that G has a tree decomposition of width k . A tree decomposition is called a *path decomposition* if T is a path. Accordingly, the *pathwidth* of G is the minimum k such that G has a path decomposition of width k . We denote by $\text{pw}(G)$ the pathwidth of G .

We need the following result which can be obtained by a standard dynamic programming techniques.

Lemma 1. *All minimum red/blue weighted dominating sets of a bipartite graph G on n vertices with bipartition (V_{Red}, V_{Blue}) given together with its path decomposition of width at most p can be counted in time $\mathcal{O}(2^p n)$.*

3 Algorithm for Counting Minimum Weighted Set Covers

We consider a recursive algorithm `countMWSC` for solving $\#\text{MWSC}$. The algorithm depends on the following observation.

Lemma 2. *For a given instance of (\mathcal{S}, w) , if there is an element $u \in \mathcal{U}(\mathcal{S})$ that belongs to a unique $S \in \mathcal{S}$, then S belongs to every set cover.*

Input: A collection on sets \mathcal{S} and a weight function $w : \mathcal{S} \rightarrow \mathbb{R}^+$.

Output: A couple $(weight, num)$ where $weight$ is the minimum weight of a set cover of \mathcal{S} and num is the number of different set covers of this weight.

```

1 if  $|\mathcal{S}| = 0$  then
2   return  $(0, 1)$ ;
3 if  $\exists u \in \mathcal{U}(\mathcal{S}) : \text{freq}(u) = 1$ , Let  $u \in S'$  then
4   return countMWSC(remove( $S', \mathcal{S}$ ),  $w$ );
5 Pick  $S \in \mathcal{S}$  of maximum cardinality;
6 if  $|\mathcal{S}| \leq 3$  and for every  $S \in \mathcal{S}$  the degree of all its elements is at most 6 then
7   return countPW( $\mathcal{S}, w$ );
8 else
9    $(w_{in}, n_{in}) = \text{countMWSC}(\text{remove}(S, \mathcal{S}), w)$ ;
10   $(w_{out}, n_{out}) = \text{countMWSC}(\mathcal{S} \setminus \{S\}, w)$ ;
11   $w_{in} = w_{in} + w(S)$ ;
12  if  $w_{in} < w_{out}$  then
13    return  $(w_{in}, n_{in})$ ;
14  else if  $w_{in} = w_{out}$  then
15    return  $(w_{in}, n_{in} + n_{out})$ ;
16  else
17    return  $(w_{out}, n_{out})$ ;

```

Fig. 1. `countMWSC(\mathcal{S}, w)`

Let us go through the algorithm `countMWSC`; see Figure 1. First, if $|\mathcal{S}| = 0$ then the size of MWSC is 0 and the number of such set covers is 1. Otherwise (lines 3–4), the algorithm tries to reduce the size of the problem by checking whether condition of Lemma 2 is applicable. Specifically, if there is an element $u \in S'$ of frequency one, then we should put S' into minimum set cover. Thus we remove S' and all its elements from the other sets $S \in \mathcal{S}$. Namely `remove(S', \mathcal{S}) = $\{Z \mid Z = S \setminus S', S \in \mathcal{S}\}$` .

If the condition of Lemma 2 does not apply, then we take a set $S \in \mathcal{S}$ of maximum cardinality. If $|\mathcal{S}| \leq 3$ and for every $S \in \mathcal{S}$ the degree of all its elements is at most 6 then we solve the problem with the algorithm `countPW`. We discuss this algorithm and its complexity later.

Otherwise we branch on the following two subproblems. First subproblem (`remove(S, \mathcal{S}), w`) corresponds to the case where S belongs to the minimum set cover. Whereas in $(\mathcal{S} \setminus \{S\}, w)$ subproblem S does not belong to the minimum set cover.

And finally we compare the weights of two subproblems and return total weight and number (lines 12–17).

The function `countPW` computes a minimum set cover for a specific instance (\mathcal{S}, w) . Namely, \mathcal{S} consists of sets with cardinalities at most 3 and for every $S \in \mathcal{S}$ the degree of all its elements is at most 6. For such a set \mathcal{S} , the function `countPW` does the following.

- Constructs the incidence graph $G_{\mathcal{S}}$ of \mathcal{S} ;
- Counts the number of minimum red/blue dominating sets in $G_{\mathcal{S}}$ (to perform this step, we construct a path decomposition of $G_{\mathcal{S}}$ and perform dynamic programming algorithm described in Lemma 10);
- Counts the number of minimum set covers of \mathcal{S} from the number of red/blue dominating sets of $G_{\mathcal{S}}$.

4 Analysis of `countMWSC` Algorithm

In this section we show that the running time of the algorithm `countMWSC` is $\mathcal{O}^*(1.2464^{|\mathcal{S}|+|\mathcal{U}|})$. The analysis is based on *Measure & Conquer* technique [8, 7]. The analysis of the branching part of the algorithm is quite similar to analysis from [7].

Let n_i be the number of subsets $S \in \mathcal{S}$ of cardinality i and let m_j be the number of elements $u \in \mathcal{U}$ of frequency j . We use the following measure $k = k(\mathcal{S})$ of the size of \mathcal{S} :

$$k(\mathcal{S}) = \sum_{i \geq 1} w_i n_i + \sum_{j \geq 1} v_j m_j,$$

where the weights $w_i, v_j \in (0, 1]$ will be fixed later. Note that $k \leq |\mathcal{S}| + |\mathcal{U}|$. Let

$$\Delta w_i = w_i - w_{i-1}, \text{ if } i \geq 2 \quad \text{and} \quad \Delta v_i = \begin{cases} v_i - v_{i-1}, & \text{if } i \geq 3, \\ v_2, & \text{if } i = 2. \end{cases}$$

Intuitively, $\Delta w_i (\Delta v_i)$ is a reduction of the size of the problem corresponding to the reduction of the cardinality of a set (the frequency of an element) from i to $i - 1$. Note that this also holds for Δv_2 , because the new element of frequency one introduced is removed before next branching. And thus we get the total reduction $1 - (1 - v_2) = v_2$.

Theorem 1. *Algorithm `countMWSC` solves #MWSC in time $\mathcal{O}^*(1.2464^{|\mathcal{S}|+|\mathcal{U}|})$.*

Proof. In order to simplify the running time analysis, we make the following assumptions:

- $v_1 = 1$;
- $w_i = 1$ for $i \geq 6$ and $v_i = 1$ for $i \geq 6$;
- $0 \leq \Delta w_i \leq \Delta w_{i-1}$ for $i \geq 2$.

Note that this implies $w_i \geq w_{i-1}$ for every $i \geq 2$.

Let $P_h(k)$ be the number of subproblems of size $h \in \{0, \dots, k\}$, solved by countMWSC to solve a problem of size k . Clearly, $P_k(k) = 1$. Consider the case $h < k$ (which implies $|\mathcal{S}| \neq 0$). If one of the condition in line 3 of the algorithm holds, we remove one set from \mathcal{S} . Thus the reduction of size of the problem is at least w_1 (worst case, $|\mathcal{S}| = 1$) and $P_h(k) \leq P_h(k - w_1)$. Otherwise, let S be the subset selected in line 5. If $|\mathcal{S}| \leq 3$ and for every $S \in \mathcal{S}$ the degree of all its elements is at most 6 (line 6), no subproblem is generated. Otherwise, we branch on two subproblems $S_{out} = (w_{out}, n_{out})$ and $S_{in} = (w_{in}, n_{in})$.

Consider subproblem S_{out} . It corresponds to the case where S does not belong to the set cover. The size of S_{out} decreases by $w_{|S|}$ because of the removal of S . Let m_i be the number of elements of S with frequency i . Note that there cannot be elements of frequency 1. Consider an element $u \in S$ with frequency $i \geq 2$. When we remove S , the frequency of u decreases by one. Thus, the size of the subproblem decreases by Δv_i . The overall reduction due to the reduction of the frequencies is at least

$$\sum_{i \geq 2} m_i \Delta v_i = \sum_{i=2}^6 m_i \Delta v_i.$$

Finally, the total reduction of the size of S_{out} is

$$w_{|S|} + \sum_{i=2}^6 m_i \Delta v_i.$$

Now consider the subproblem S_{in} . The size of S_{in} decreases by $w_{|S|}$ because of the removal of S . Since we also remove all elements from S , we also get the reduction of size

$$\sum_{i \geq 2} m_i v_i = \sum_{i=2}^6 m_i v_i + m_{\geq 7}.$$

Here $m_{\geq 7}$ is the number of elements with frequency at least 7. Let S' be the set sharing element u with S ($S' \cap S \neq \emptyset$). Note that $|S'| \leq |S|$. When we remove u , the cardinality of S' is reduced by one. This implies the reduction of size S_{in} by $\Delta w_{|S'|} \geq \Delta w_{|S|}$. Thus the overall reduction of the size of S_{in} due to the reduction of the cardinalities of the sets S' is at least

$$\Delta w_{|S|} \sum_{i \geq 2} (i - 1) m_i \geq \Delta w_{|S|} \left(\sum_{i=2}^6 (i - 1) m_i + 6 \cdot m_{\geq 7} \right).$$

Finally, the total reduction of the size of S_{in} is

$$w_{|S|} + \sum_{i=2}^6 m_i v_i + m_{\geq 7} + \Delta w_{|S|} \left(\sum_{i=2}^6 (i - 1) m_i + 6 \cdot m_{\geq 7} \right).$$

Putting all together, for all possible values of $|S| \geq 3$ and for all values m_i such that

$$\sum_{i=2}^6 m_i + m_{\geq 7} = |S|,$$

(except if $|S| = 3$, then we choose only those sets of m_i where $m_{\geq 7} \neq 0$) we have the following set of recursions

$$P_h(k) \leq P_h(k - \Delta k_{out}) + P_h(k - \Delta k_{in}),$$

where

$$\begin{aligned} \Delta k_{out} &= w_{|S|} + \sum_{i=2}^6 m_i \Delta v_i, \\ \Delta k_{in} &= w_{|S|} + \sum_{i=2}^6 m_i v_i + m_{\geq 7} + \Delta w_{|S|} \left(\sum_{i=2}^6 (i-1)m_i + 6 \cdot m_{\geq 7} \right). \end{aligned}$$

Since $\Delta w_{|S|} = 0$ for $|S| \geq 7$, we have that each recursion with $|S| \geq 8$ is “dominated” by some recurrence with $|S| = 7$. Thus we restrict our attention only to the cases $3 \leq |S| \leq 7$. We need to consider a large number of recursions (1653). For every fixed 9-tuple $(w_1, w_2, w_3, w_4, w_5, v_2, v_3, v_4, v_5)$ the number $P_h(k)$ is upper bounded by α^{k-h} , where α is the largest number from the set of real roots of the set of equations

$$\alpha^k = \alpha^{k-\Delta k_{out}} + \alpha^{k-\Delta k_{in}}$$

corresponding to the different combinations of values $|S|$ and m_i . Thus to estimate $P_h(k)$ we need to choose the weights w_i and v_j minimizing α .

Let K denote the set of the possible sizes of the subproblems solved. Note that $|K|$ is polynomially bounded. Thus the total number $P(k)$ of subproblems is

$$P(k) \leq \sum_{h \in K} P_h(k) \leq \sum_{h \in K} \alpha^{k-h}.$$

After performing branching the algorithm calls `countPW` algorithm. Thus the total running time of the algorithm on an instance of measure k is

$$\mathcal{O}\left(\sum_{h \in K} \alpha^{k-h} \cdot \beta^h\right) = \mathcal{O}(\max\{\alpha, \beta\}^k).$$

Here $\mathcal{O}(\beta^h)$ is the running time of `countPW` algorithm on a problem of size h . So we need to choose the weights w_i and v_j minimizing both α and β . To estimate the running time of `countPW` algorithm we use the idea of measure and conquer applied to linear programming.

Let us remind that `countPW` is called on an instance of `MWSC` problem with all sets of size at most 3 and elements of frequency at most 6. There are no elements of frequency 1. Let \mathcal{S} be an instance of set cover of measure k and let $G_{\mathcal{S}}$ be its incidence graph. Then $G_{\mathcal{S}}$ is a bipartite graph with the bipartition

($\mathcal{X} = \mathcal{S}, \mathcal{Y} = \mathcal{U}(\mathcal{S})$). By Lemma 1, the running time of dynamic programming algorithm on $G_{\mathcal{S}}$ is $\mathcal{O}(\beta^k) = \mathcal{O}(2^{\text{pw}(G_{\mathcal{S}})})$. Let us remind that both constants α and β depend on the choice of the weights in the measure function. In the remaining part of the proof we show how to choose the weights that balance branching and dynamic programming parts of the algorithm.

We denote by \mathcal{X}_i all vertices from \mathcal{X} of degree i . Let $x_i = |\mathcal{X}_i|$. We define $\mathcal{Y}_j \subseteq \mathcal{Y}$ and y_j in the same way for every $j \in \{2, \dots, 6\}$. We need the following lemma. The proof can be obtained by technique used in [6].

Lemma 3. *For any $\varepsilon > 0$, there exists an integer n_ε such that for every graph G with $n > n_\varepsilon$ vertices and maximum degree at most 6,*

$$\text{pw}(G) \leq \frac{1}{6}n_3 + \frac{1}{3}n_4 + \frac{13}{30}n_5 + \frac{23}{45}n_6 + \varepsilon n,$$

where n_i is the number of vertices of degree i in G for any $i \in \{3, \dots, 6\}$. Moreover, a path decomposition of such width can be constructed in polynomial time.

We need to evaluate the running time of `countPW` on an instance of \mathcal{S} of measure k . This gives us the following:

$$k = k(\mathcal{S}) = \sum_{i=1}^3 w_i x_i + \sum_{j=2}^5 v_j y_j + y_6. \tag{1}$$

Here values w_i and v_j are taken from analysis of `countMWSC` algorithm. By counting edges of $G_{\mathcal{S}}$, we arrive at the second condition

$$x_1 + 2x_2 + 3x_3 = \sum_{j=2}^6 j \cdot y_j. \tag{2}$$

By Lemma 3,

$$\text{pw}(G_{\mathcal{S}}) \leq \frac{1}{6}(x_3 + y_3) + \frac{1}{3}y_4 + \frac{13}{30}y_5 + \frac{23}{45}y_6. \tag{3}$$

Combining (1), (2), and (3) we conclude that the pathwidth of $G_{\mathcal{S}}$ is at most the maximum of the following linear function

$$\frac{1}{6}(x_3 + y_3) + \frac{1}{3}y_4 + \frac{13}{30}y_5 + \frac{23}{45}y_6 \rightarrow \max$$

subject to the following constraints:

$$\text{measure: } \sum_{i=1}^3 w_i x_i + \sum_{j=2}^5 v_j y_j + y_6 = k$$

$$\text{edges: } x_1 + 2x_2 + 3x_3 = \sum_{j=2}^6 j \cdot y_j$$

$$\text{variables: } x_i \geq 0, i \in \{1, 2, 3\}$$

$$y_j \geq 0, j \in \{2, \dots, 6\}$$

The running time of `countPW` is $\mathcal{O}^*(2^{\text{PW}(G)})$. Thus everything boils up to finding the measure that minimizes the maximum of α and maximum of LP obtained from pathwidth bounds. Finding of such weights is an interesting (and nontrivial) computational problem on its own. To find the weights we use a modification of random search with plugged LP solver.

We numerically obtained the following values of the weights.

$$w_i = \begin{cases} 0.1039797, & \text{if } i = 1, \\ 0.4664084, & \text{if } i = 2, \\ 0.8288271, & \text{if } i = 3, \\ 0.9429144, & \text{if } i = 4, \\ 0.9899772, & \text{if } i = 5, \end{cases} \quad \text{and} \quad v_i = \begin{cases} 0.5750176, & \text{if } i = 2, \\ 0.7951411, & \text{if } i = 3, \\ 0.9165365, & \text{if } i = 4, \\ 0.9771879, & \text{if } i = 5. \end{cases}$$

With such weights the optimum of LP is obtained on $x_3 = 0.7525\dots$ and $y_6 = 0.3762\dots$ and all other variables equal to 0.

This gives us the total running time $\mathcal{O}(1.2464^{k(\mathcal{S})}) = \mathcal{O}(1.2464^{|\mathcal{U}|+|\mathcal{S}|})$. The exponential space is used by the algorithm during the dynamic programming part and thus is bounded by $\mathcal{O}(1.2464^{|\mathcal{U}|+|\mathcal{S}|})$. This finalizes the proof. \square

As we mentioned already, $\#\text{MWDS}$ can be reduced to $\#\text{MWSC}$ by imposing $\mathcal{U} = V$ and $\mathcal{S} = \{N[v] \mid v \in V\}$. The size of the $\#\text{MWSC}$ instance obtained is at most $2n$, where n is the number of vertices in G . Thus, we have

Corollary 1. *The $\#\text{MWDS}$ problem can be solved in $\mathcal{O}(1.2464^{2n}) = \mathcal{O}(1.5535^n)$ time.*

5 Conclusions and Open Problems

In this paper we have used dynamic programming on bounded treewidth techniques to speed up branching algorithm counting weighted dominating sets. The running time of our algorithm can be slightly improved by considering more detailed bounds on pathwidth of bipartite graphs with different coefficients for vertices adjacent to degree two and three vertices. However in this case the arguments become much more technical and we do not include them in this extended abstract.

Another general technique to speed up branching algorithms (by using exponential space) is *memorization*, see [8]. Both techniques have many similarities and it would be interesting to understand in which cases each of the techniques is more efficient.

The best known algorithm for the decision version of minimum dominating set is not significantly faster than our counting algorithms. Similar strange effect was observed by Magnus Wahlström (private communication) for different SAT problems. Thus despite the fact that $\#\text{P}$ complete problems seems to be much more difficult than NP complete problems, the running time of best known algorithms for many decision and counting problems do not differ too much. Is there any reasonable explanation to this phenomena? Is this because faster

exponential algorithms for decision problems are still to be found or this is because in the world of exponential time algorithms the gaps between different complexity classes are small?

Acknowledgement. We thank Serge Gaspers for helpful comments.

References

1. Angelsmark, O., Jonsson, P.: Improved algorithms for counting solutions in constraint satisfaction problems. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 81–95. Springer, Heidelberg (2003)
2. Bax, E.T., Franklin, J.A.: finite-difference sieve to count paths and cycles by length. *Inf. Process. Lett.* 60(4), 171–176 (1996)
3. Björklund, A., Husfeldt, T.: Inclusion-exclusion algorithms for counting set partitions. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), pp. 575–582. IEEE Computer Society Press, Los Alamitos (2006)
4. Dahllöf, V., Jonsson, P.: An algorithm for counting maximum weighted independent sets and its applications. In: Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002). Society for Industrial and Applied Mathematics, pp. 292–298. ACM Press, New York (2002)
5. Dahllöf, V., Jonsson, P., Wahlström, M.: Counting models for 2 SAT and 3 SAT formulae. *Theoretical Computer Science* 332 332(1-3), 265–291 (2005)
6. Fomin, F.V., Gaspers, S., Saurabh, S.: Branching and treewidth based exact algorithms. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 16–25. Springer, Heidelberg (2005)
7. Fomin, F.V., Grandoni, F., Kratsch, D.: Measure and conquer: Domination – a case study. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 191–203. Springer, Heidelberg (2005)
8. Fomin, F.V., Grandoni, F., Kratsch, D.: Some new techniques in design and analysis of exact (exponential) algorithms. *Bulletin of the EATCS* 87, 47–77 (2005)
9. Fomin, F.V., Grandoni, F., Pyatkin, A.V., Stepanov, A.A.: Bounding the number of minimal dominating sets: a measure and conquer approach. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 573–582. Springer, Heidelberg (2005)
10. Fomin, F.V., Kratsch, D., Woeginger, G.J.: Exact (exponential) algorithms for the dominating set problem. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 245–256. Springer, Heidelberg (2004)
11. Fürer, M., Kasiviswanathan, S.P.: Algorithms for counting 2-SAT solutions and colorings with applications. *Electronic Colloquium on Computational Complexity (ECCC)* 33 (2005)
12. Grandoni, F.: A note on the complexity of minimum dominating set. *Journal of Discrete Algorithms* 4(2), 209–214 (2006)
13. Haynes, T.W., Hedetniemi, S.T.: Domination in graphs (Advanced topics). *Monographs and Textbooks in Pure and Applied Mathematics*, vol. 209. Marcel Dekker Inc., New York (1998)
14. Hunt III, H.B., Marathe, M.V., Radhakrishnan, V., Stearns, R.E.: The complexity of planar counting problems. *SIAM Journal on Computing* 27(4), 1142–1167 (1998)
15. Jerrum, M.: Counting, sampling and integrating: algorithms and complexity. *Lectures in Mathematics ETH Zürich*. Birkhäuser Verlag, Basel (2003)

16. Karp, R.M.: Dynamic programming meets the principle of inclusion and exclusion. *Operations Research Letters* 1 2(1981/82), 49–51 (1981)
17. Koivisto, M.: An $O(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion-exclusion. In: *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006)*, pp. 583–590. IEEE Computer Society Press, Los Alamitos (2006)
18. Mölle, D., Richter, S., Rossmanith, P.: Enumerate and expand: Improved algorithms for connected vertex cover and tree cover. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) *CSR 2006*. LNCS, vol. 3967, pp. 270–280. Springer, Heidelberg (2006)
19. Randerath, B., Schiermeyer, I.: Exact algorithms for MINIMUM DOMINATING SET. Technical Report zaik-469, Zentrum für Angewandte Informatik Köln, Germany (2004)
20. Ryser, H.J.: *Combinatorial mathematics*. The Carus Mathematical Monographs, No. 14. Published by The Mathematical Association of America (1963)
21. Schöningh, U.: Algorithmics in exponential time. In: Diekert, V., Durand, B. (eds.) *STACS 2005*. LNCS, vol. 3404, pp. 36–43. Springer, Heidelberg (2005)
22. Valiant, L.G.: The complexity of computing the permanent. *Theoretical Computer Science* 8(2), 189–201 (1979)
23. Woeginger, G.J.: 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)
24. Zhang, W.: Number of models and satisfiability of sets of clauses. *Theoretical Computer Science* 155(1), 277–288 (1996)

Online Interval Scheduling: Randomized and Multiprocessor Cases^{*}

Stanley P.Y. Fung¹, Chung Keung Poon², and Feifeng Zheng³

¹ Department of Computer Science, University of Leicester, United Kingdom
pyfung@mcs.le.ac.uk

² Department of Computer Science, City University of Hong Kong, China
ckpoon@cs.cityu.edu.hk

³ School of Management, Xi'an JiaoTong University, China
zhengff@mail.xjtu.edu.cn

Abstract. We consider the problem of scheduling a set of equal-length intervals arriving online, where each interval is associated with a weight and the objective is to maximize the total weight of completed intervals. An optimal 4-competitive algorithm has long been known in the deterministic case, but the randomized case remains open. We give the first randomized algorithm for this problem, achieving a competitive ratio of 3.618. We also prove a randomized lower bound of $4/3$, which is an improvement over the previous $5/4$ result, and a lower bound of 2 for a class of barely random algorithms which include our new algorithm. We also show that the techniques can be carried to the deterministic multiprocessor case, giving a 3.618-competitive 2-processor algorithm, a $5/4$ lower bound for any number of processors, and a 2 lower bound for 2 processors.

1 Introduction

We study the problem of scheduling a set of intervals which arrive online. Each interval has a weight and all intervals are of the same length. The objective is to schedule a set of non-overlapping intervals such that the total weight of all these intervals is maximized. Intervals being processed can be interrupted, but the value will be lost. This can also be viewed as a job scheduling problem where each job must be served immediately or else it is lost. This is a fundamental problem in scheduling and has been widely studied, and is also related to a number of online problems such as call control and bandwidth allocation (see e.g. [15, 4, 1]).

Related Work. For the basic problem where intervals are of the same length and with arbitrary weights, Woeginger [15] gave an optimal deterministic 4-competitive algorithm and a matching lower bound. In the paper, the open

^{*} The work described in this paper was fully supported by a grant from City University of Hong Kong (SRG 7001969), and NSFC Grant No. 70471035.

question of whether randomization can help give better algorithms was raised. Miyazawa and Erlebach [12] gave a 3-competitive randomized algorithm for the special case where the weights of the intervals form a non-decreasing sequence. They also gave the first randomized lower bound of 1.25.

Many other variations exist for the problem. One is to allow variable interval lengths: Canetti and Irani [4] gave a randomized lower bound of $\Omega(\sqrt{\log \Delta / \log \log \Delta})$ and a randomized upper bound of $O(\log \Delta)$ where Δ is the ratio of the longest to shortest interval length. The deterministic case (with variable lengths) seems not studied before on its own. However, more general models have been studied. The problem of *online scheduling of broadcasts with restarts* is considered in [8,5,7,16,14]. In this broadcast scheduling problem, a set of pages is stored in a server, and requests for pages arrive online. Each request has a deadline by which the request has to be completed or else no profit is obtained. Different requests for the same page can be served together in one single broadcast. Requests being served can be interrupted but will need to restart from the beginning if it is to be broadcasted again. It can be seen that the interval scheduling problem is a special case of the broadcast scheduling problem (where all requests have tight deadlines and each request asks for a different page). In particular, there are matching upper and lower bounds $\Theta(\Delta / \log \Delta)$ [14,16] for the deterministic case. These bounds also apply to the interval scheduling problem.

Another related problem, the *real-time job scheduling problem*, gives a bound on the interval scheduling problem w.r.t. a different parameter. In this problem, jobs have release times, deadlines and execution times. Preemption is allowed and preempted jobs can be resumed at the point where they were preempted. When all jobs have no laxity (i.e., execution time equals the difference between deadline and release time), this problem reduces to the interval scheduling problem. Matching upper and lower bounds of $(1 + \sqrt{k})^2$ are established in [2,9], where k is the *importance ratio*, defined as the ratio of maximum to minimum weight of the jobs. (Here jobs have different lengths and the weight of a job is the value per unit length of the job.) Both bounds apply to the interval scheduling case.

Yet another case is where intervals can have different lengths and their weights are some function of their lengths. Seiden [13] gave a randomized 3.732-competitive algorithm for the case of *benevolent instances*, where (roughly speaking) the weight of an interval is either a convex or a monotonic decreasing function of its length. If the weight of an interval is equal to its length, the non-preemptive case was considered in [11]. They gave a randomized $O((\log \Delta)^{1+\epsilon})$ -competitive algorithm and a $\Omega(\log \Delta)$ lower bound.

Multiprocessor scheduling of intervals were studied in [6], giving an optimal (1-competitive) algorithm when all intervals have unit weight (and not necessarily equal length). Multiprocessor scheduling of jobs with different lengths and weights were studied in [10], with competitive ratio roughly $\Theta(\log k)$ where k is the importance ratio. If the profit is equal to the length, a tight bound of 2 is known for 2 processors [2].

Our Results. In this paper we consider the case where all intervals are of the same length. In fact our algorithm applies to the more general case where intervals may not have equal lengths but have *agreeable deadlines*, i.e. no interval being strictly contained in another interval. This is because of the result that the class of interval graphs for agreeable-deadlines intervals is equal to the class of interval graphs for equal-length intervals (see e.g. [3]). In Section 3 we give a randomized online algorithm which is 3.618-competitive, which is lower than the optimal deterministic bound. The algorithm only uses a constant number of random bits as it only makes a single random choice when it starts. In Section 4 we consider lower bounds, giving an improved randomized lower bound of $4/3$ on the competitive ratio. For the class of barely random algorithms that choose between two deterministic algorithms with equal probabilities (which includes our proposed algorithm), we show a lower bound of 2.

There is a close relation between the randomized single-processor case and the deterministic multiprocessor case. We show in Section 5 that (with some modifications) our 3.618-competitive algorithm can be applied in the 2-processor case. The lower bounds also apply: we give a $4/3$ lower bound for the competitive ratio of any deterministic or randomized algorithms for any number of processors, and a deterministic 2 lower bound for 2 processors.

Due to space limitations some proofs are omitted and can be found in the full version of the paper.

2 Preliminaries

An interval I is specified by $s(I)$, its arrival time; $\ell(I)$, its length; and $|I|$, its weight. Since we only consider the case where all intervals are of the same length, we can, without loss of generality, assume $\ell(I) = 1$ for all I .

Intervals arrive online and the scheduling algorithm has to make decisions without knowledge of future intervals. When a scheduling algorithm completes an interval, it receives a *profit* equal to the weight of the interval. An interval being scheduled can be aborted (e.g. when an interval of larger weight arrives) but the value of the aborted interval will be lost. The objective of the algorithm is to maximize the total profit obtained by completing the intervals.

Let $A(S)$ denote the profit obtained by algorithm A on input instance S . An online algorithm A is *c-competitive* if for all input instances S , $OPT(S)/A(S) \leq c$ where OPT is the offline optimal algorithm which has knowledge of all future intervals and hence can schedule optimally. When A is a randomized algorithm, the definition of competitive ratio becomes $OPT(S)/E[A(S)] \leq c$ where the expectation is taken over the random choices of A . A randomized algorithm that only uses a constant number of random bits is called *barely random*.

3 A Randomized Algorithm

The Algorithm. Consider the simple deterministic algorithm Greedy_r: when an interval I arrives and the algorithm is currently executing another interval I' ,

it aborts I' and starts I if $|I| \geq r|I'|$. If the machine is idle at that time then $|I'| = 0$, meaning that I will always get started. We call r the *abortion ratio*. This algorithm is 4-competitive when $r = 2$ and is the best possible for deterministic algorithms [15].

Fix constants α, β, p where $1 < \alpha < \beta$ and $0 < p < 1$. The randomized algorithm **RGreedy** $_{\alpha, \beta, p}$ chooses to run one of the two deterministic algorithms Greedy $_{\alpha}$ and Greedy $_{\beta}$ with probability p and $1 - p$ respectively. It is barely random since the choice is only made in the beginning. Below we will analyze the competitive ratio of this algorithm.

Basic subschedules. Let A and B denote the schedules produced by Greedy $_{\alpha}$ and Greedy $_{\beta}$ respectively on a particular input instance. We define a *basic subschedule* to be a sequence of execution of intervals (I_1, \dots, I_k) , for $k \geq 1$, where I_i is aborted by I_{i+1} for $1 \leq i \leq k - 1$ and I_k is completed. That is, each basic subschedule consists of zero or more aborted intervals followed by a completed interval. Each of the two online schedules A and B can then be partitioned into a sequence of basic subschedules. In a basic subschedule (I_1, \dots, I_k) with abortion ratio r , we have $|I_i| \leq |I_{i+1}|/r$ because this is the condition for I_{i+1} to abort I_i .

Profit amortization. Consider a basic subschedule (I_1, \dots, I_k) with abortion ratio r . Only I_k is completed. Therefore the profit received for the whole basic subschedule is $|I_k|$. For the purpose of analysis, we will ‘amortize’ the profit of a basic subschedule to the individual intervals (not just the completed one) as follows. I_k transfers a profit of $|I_{k-1}|$ to I_{k-1} and keep the rest of $|I_k| - |I_{k-1}|$ profit to itself. Inductively, for $i = k - 1, \dots, 2$, I_i receives an amortized profit of $|I_i|$ from I_{i+1} ; it then transfers a profit of $|I_{i-1}|$ to I_{i-1} and keep the rest of $|I_i| - |I_{i-1}|$ to itself. For I_1 it keeps all its received profit $|I_1|$. Obviously, the total profit remains the same. From now on, unless explicitly stated otherwise, we will refer to the amortized profits.

Schedule segments. Consider a basic subschedule, (X_1, \dots, X_k) , of A . Let x_1, x_2, \dots, x_k be the weights of these intervals. Let t_i be the time when X_i is started, and define $t_{k+1} = t_k + 1$. During $[t_i, t_{i+1})$, OPT can start at most one interval, Z_i , with weight z_i . If there is no interval started by OPT during that time period, we simply skip this interval X_i from our consideration; thus X_i and t_i only refer to those intervals in A which have corresponding Z_i ’s. This will only underestimate the profit of the online algorithm. By the property of basic subschedules we have $x_i \geq \alpha x_{i-1}$ for all $1 < i \leq k$. (Note that X_i and X_{i-1} may not be consecutive intervals in the basic subschedule because of the skipping just mentioned. Nevertheless the inequality remains true.)

Let u_i be the time when Z_i starts. At time u_i , Greedy $_{\alpha}$ must be serving some intervals with weight larger than z_i/α or else Greedy $_{\alpha}$ would abort what it is serving and start Z_i instead (which also has weight $> z_i/\alpha$). Thus $x_i > z_i/\alpha$ for all i . Similarly, at time u_i , the other algorithm Greedy $_{\beta}$ (if it is chosen instead) must be serving some interval, Y_i , of weight larger than z_i/β , or else it will abort what it is serving and start Z_i instead. Denote by y_i the weight of this interval that Greedy $_{\beta}$ is serving at time u_i . We have $y_i > z_i/\beta$.

We make two observations about these y_i 's. First, any two Z_i 's must correspond to different Y_i 's. This is because each interval in OPT is completed and thus takes 1 unit of time, so $u_{i+1} \geq u_i + 1$ and hence Y_i and Y_{i+1} cannot be the same interval. Second, two consecutive Y_{i-1} and Y_i may or may not belong to the same basic subschedule in B . If they do, then we have $y_i \geq \beta y_{i-1}$. Note that even if they are in the same basic subschedule, they may not be consecutive intervals (there may be a number of abortions in-between), but even so the previous inequality remains true. If they are not in the same basic subschedule, we cannot say anything about y_i and y_{i+1} .

Therefore, we further split the basic subschedules in A and B into *segments* as follows. Let X_1 and Y_1 be the first interval in A and B respectively after the previous segment (initially they are simply the first intervals). A segment is then the set of intervals (X_1, \dots, X_n) from A and (Y_1, \dots, Y_n) from B such that at least one of X_n or Y_n is completed, and all other X_i and Y_i is aborted (directly or indirectly) by X_{i+1} and Y_{i+1} respectively. For a pair of corresponding segments, at least one of the two ending intervals is completed. In effect, intervals in a segment satisfy $x_i \leq x_{i+1}/\alpha$ (for those in A) and $y_i \leq y_{i+1}/\beta$ (for those in B).

Note that X_i 's and Y_i 's may not be consecutive intervals in a basic subschedule, as explained before. In the analysis we will ignore all those skipped abortions, e.g. in the profit amortization we treat Y_i and Y_{i+1} as if they are consecutive without giving any profit to any aborted intervals in-between, if any.

Bounding the expected profit. We now consider each such segment, where (X_1, \dots, X_n) is a segment from A , (Y_1, \dots, Y_n) is a segment from B , and (Z_1, \dots, Z_n) is the corresponding sequence of intervals in OPT .

The total profit of OPT in this segment is $\sum_{i=1}^n z_i$. As for the online algorithm, A has an amortized profit of at least $x_n - x_1/\alpha$ for this segment: the last interval has profit x_n and subsequently transferred to other intervals in this segment, except x_1 may transfer profit to a previously aborted interval, which has weight at most x_1/α . Similarly B receives an amortized profit of $y_n - y_1/\beta$ for this segment. The expected profit is thus at least $p(x_n - x_1/\alpha) + (1-p)(y_n - y_1/\beta)$. Note that the terms x_1/α and y_1/β would not be there if they are the first interval in a basic subschedule. But at least one of x_1 and y_1 must be such a first interval, since otherwise the segment would be extended to the front. Therefore, we can remove the smaller of these two terms from the expression. Thus the expected profit of the online algorithm is at least $px_n + (1-p)y_n - \max(px_1/\alpha, (1-p)y_1/\beta)$. We call the $\max(px_1/\alpha, (1-p)y_1/\beta)$ term the *amortized term*. The ratio R of optimal profit to the expected online profit in this segment is at most

$$\frac{\sum_{i=1}^n z_i}{px_n + (1-p)y_n - \max(px_1/\alpha, (1-p)y_1/\beta)} \tag{1}$$

We want to upper bound the above ratio, under the following constraints:

$$z_i < \min(\alpha x_i, \beta y_i), \quad x_i \leq x_{i+1}/\alpha, \quad y_i \leq y_{i+1}/\beta$$

Each interval served by OPT must belong to exactly one segment. Therefore, if we can upper bound the ratio of the total OPT profit to the expected online

profit, for all such segments, this gives a bound on the competitive ratio of the randomized algorithm.

For the rest of the paper, we fix $\alpha = \phi \approx 1.618$, $\beta = \phi^2 \approx 2.618$ and $p = 1/2$, where $\phi = (1 + \sqrt{5})/2$ is the golden ratio. We first state a technical lemma which will be required later.

Lemma 1. *Suppose $x_i = 1/\alpha^{n-i}$ and $y_i = y/\beta^{n-i}$ for all i . Then the function*

$$F(y) = \frac{\sum_{i=1}^n \min(\alpha x_i, \beta y_i)}{1 + y - 1/\alpha^n}$$

is increasing in y for $0 \leq y < \alpha/\beta$, and decreasing in y for $y > \alpha/\beta$.

Theorem 1. *The competitive ratio of $RGreedy_{\alpha,\beta,p}$ is $\phi + 2 \approx 3.618$ when $\alpha = \phi$, $\beta = \phi^2$ and $p = 1/2$.*

Proof. Consider a segment with (X_1, \dots, X_n) , (Y_1, \dots, Y_n) , and (Z_1, \dots, Z_n) . Without loss of generality, assume $x_n = 1$ and denote y_n simply as y . To maximize (II), observe that (for a fixed y) we should make x_i and y_i as large as possible, so that z_i 's are large and also x_1 and y_1 are large. This means $x_i = 1/\alpha^{n-i}$ and $y_i = y/\beta^{n-i}$. Together with $p = 1/2$, (II) becomes at most

$$\frac{2 \sum_{i=1}^n \min(\alpha x_i, \beta y_i)}{1 + y - \max(1/\alpha^n, y/\beta^n)} \tag{2}$$

We consider these cases:

Case 1: $y \leq (\beta/\alpha)^n$. In this case the amortized term is $1/\alpha^n$. The ratio (2) is equal to $2F(y)$ in Lemma 1, which we know is maximum when $y = \alpha/\beta$. At this value of y , all min terms in the numerator are βy_i terms and the ratio has maximum value

$$\begin{aligned} \frac{2\beta y(1 + 1/\beta + \dots + 1/\beta^{n-1})}{1 + y - 1/\alpha^n} &= \frac{2\beta y(1 - 1/\beta^n)/(1 - 1/\beta)}{1 + y - 1/\alpha^n} \\ &= \frac{\frac{2\alpha\beta}{\beta-1}(1 - 1/\beta^n)}{1 - 1/\alpha^n + \alpha/\beta} = \frac{2\phi^2(1 - 1/\phi^{2n})}{1 + 1/\phi - 1/\phi^n}. \end{aligned}$$

This is at most $\phi + 2 \approx 3.618$ for any value of n (maximum occurs when $n = 2$).

Case 2: $y > (\beta/\alpha)^n$. In this case all min terms are the αx_i terms and the amortizing term is y/β^n . So the ratio is

$$\begin{aligned} \frac{2(\alpha + 1 + \dots + 1/\alpha^{n-2})}{1 + y - y/\beta^n} &= \frac{2\alpha(1 - 1/\alpha^n)/(1 - 1/\alpha)}{1 + y(1 - 1/\beta^n)} \\ &\leq \frac{2\alpha(1 - 1/\alpha^n)/(1 - 1/\alpha)}{1 + (\beta/\alpha)^n - 1/\alpha^n} = \frac{2\phi^3(1 - 1/\phi^n)}{1 + \phi^n - 1/\phi^n}. \end{aligned}$$

This is at most ϕ^2 for any value of n .

Therefore in either case the ratio is at most $\phi + 2$. □

We can show that there is an instance which actually attains the competitive ratio of 3.618 using our algorithm (with these chosen parameters), so that the analysis is tight.

4 Lower Bounds

4.1 Randomized Algorithms

Theorem 2. *No randomized algorithm for interval scheduling has competitive ratio better than $4/3$.*

Proof. We will use Yao's principle which states that the randomized lower bound can be obtained by bounding $E[OPT]/E[A]$ for any deterministic algorithm A over a probability distribution of input instances. (See for example, [12].) Thus we define an input distribution as follows. Let (r, w) denote the release time and weight respectively of an interval. Fix a large even integer n . Define $n + 1$ intervals, I_0, I_1, \dots, I_n , such that for $0 \leq i \leq n - 1$, $I_i = (i/2, v_i)$ where $v_i = 2^i$, and $I_n = (n/2, v_n)$ where $v_n = 2^{n-1}$. Define n sets of intervals, S_1, S_2, \dots, S_n , such that $S_i = \{I_0, I_1, \dots, I_i\}$ for $1 \leq i \leq n$. Finally, we define our distribution of inputs to be one such that S_i occurs with probability $p_i = 1/2^i$ for $1 \leq i \leq n - 1$ and S_n occurs with probability $p_n = 1/2^{n-1}$.

Since any I_i does not overlap with I_{i+2} , we have $OPT(S_i) = 1 + 4 + \dots + 2^i = \frac{4^{i/2+1}-1}{3}$ if i is even, and $OPT(S_i) = 2 + 8 + \dots + 2^i = \frac{2(4^{(i+1)/2}-1)}{3}$ if i is odd, and $OPT(S_n) = 1 + 4 + \dots + 2^{n-2} + 2^{n-1} = \frac{4^{n/2}-1}{3} + 2^{n-1}$. Hence

$$\begin{aligned} E[OPT] &= \sum_{i=2, i \text{ even}}^{n-2} \frac{4^{i/2+1}-1}{3 \cdot 2^i} + \sum_{i=1, i \text{ odd}}^{n-1} \frac{2(4^{(i+1)/2}-1)}{3 \cdot 2^i} + \frac{4^{n/2}-1}{3 \cdot 2^{n-1}} + 1 \\ &= 4n/3 - o(n). \end{aligned}$$

We now derive an upper bound on the expected profit of an arbitrary deterministic algorithm A on our input distribution. More specifically, for $i = 1, \dots, n$, we let Q_i be the contribution to the expected profit of A on I_{i-1}, \dots, I_n when the input is one of S_i, \dots, S_n .

Consider the case when the input is S_n . This happens with probability p_n . When I_n arrives at time $n/2$, A may or may not be serving another interval. If it does, it must be serving I_{n-1} . Since we choose $v_{n-1} = v_n$, A will obtain at most a profit of v_n whether it aborts I_{n-1} or not. Thus, $Q_n \leq p_n v_n$.

Now, suppose the input is either S_{n-1} or S_n . When I_{n-1} arrives at time $(n-1)/2$, A may or may not be serving I_{n-2} . There are two cases.

Case 1: A is serving I_{n-2} and it continues until its completion. Then A gains a profit of v_{n-2} on I_{n-2} whether the input is S_{n-1} or S_n . Further, it can gain an expected profit of at most Q_n on I_{n-1} and I_n when the input is S_n . Hence, $Q_{n-1} \leq (p_{n-1} + p_n)v_{n-2} + Q_n$.

Case 2: A is not serving I_{n-2} or if it aborts I_{n-2} . Then A may have an expected profit of $p_{n-1}v_{n-1}$ on I_{n-1} when the input is S_{n-1} and an expected profit of Q_n on I_{n-1} and I_n when the input is S_n . Note that the input being S_{n-1} and being S_n are two disjoint events. Thus, $Q_{n-1} \leq p_{n-1}v_{n-1} + Q_n$.

Setting $(p_{n-1} + p_n)v_{n-2} + Q_n = p_{n-1}v_{n-1} + Q_n$ (which is satisfied by requiring $v_{n-2} = \frac{p_{n-1}}{p_{n-1} + p_n}v_{n-1}$), we have $Q_{n-1} \leq p_{n-1}v_{n-1} + Q_n$ no matter what A does.

In general, consider the case when the input is one of S_i, \dots, S_n . When I_i arrives at time $i/2$, A may or may not be serving I_{i-1} and we consider the following cases.

Case 1: A is serving I_{i-1} and it continues with it until completion. Then A gains an expected profit of $(p_i + \dots + p_n)v_{i-1}$ on I_{i-1} (no matter what the true input is) and an expected profit of Q_{i+1} on I_i, \dots, I_n when the input is S_{i+1}, \dots, S_n . Thus, $Q_i \leq (p_i + \dots + p_n)v_{i-1} + Q_{i+1}$.

Case 2: A is not serving I_{i-1} or if it aborts I_{i-1} . Then A gains an expected profit of $p_i v_i$ on I_i when the input is S_i , and an expected profit of Q_{i+1} on I_i, \dots, I_n when the input is one of S_{i+1}, \dots, S_n . Hence $Q_i \leq p_i v_i + Q_{i+1}$.

Setting $v_{i-1} = \frac{p_i}{p_i + \dots + p_n} v_i$, we have $(p_i + \dots + p_n)v_{i-1} + Q_{i+1} = p_i v_i + Q_{i+1}$. So $Q_i \leq p_i v_i + Q_{i+1}$.

One can easily check that setting p_i and v_i as mentioned earlier, the conditions $v_{i-1} = \frac{p_i}{p_i + \dots + p_n} v_i$ for $1 \leq i \leq n$, are satisfied and the total expected profit of A is $Q_1 \leq p_1 v_1 + \dots + p_n v_n = n$.

Therefore, $E[OPT]/E[A] \rightarrow 4/3$ for $n \rightarrow \infty$. □

Remark on benevolent instances. The lower bound construction does not rely on the exact lengths of the intervals. The only requirement on the lengths is that I_i and I_{i+1} intersect while I_i and I_{i+2} do not. Therefore, the lower bound also holds for benevolent instances; we just create the instances with the specified weights and adjust the lengths accordingly.

4.2 Barely Random Algorithms

Our randomized algorithm in Section 3 chooses between two deterministic algorithms with equal probabilities. We next show a lower bound on such algorithms.

Theorem 3. *No barely random algorithms that choose between two deterministic algorithms with equal probabilities can be better than 2-competitive.*

Proof. Suppose on the contrary there exists such a randomized algorithm which is $(2 - \epsilon)$ -competitive for some constant $\epsilon > 0$. Let D1 and D2 be the two deterministic algorithms. We construct an adversarial request sequence to show that this results in a contradiction.

Consider a set of a large number of intervals where each interval differs from the previous one by arriving slightly later and having a slightly larger weight (difference in weight being δ). The minimum weight of intervals in this set is 1 and the maximum weight is α . Here δ is a sufficiently small and α a sufficiently large constant to be chosen later. The last interval arrives before the deadline of the first interval, and hence any algorithm can serve at most one interval in this set. (This is the set of intervals used in [15].) Given this set of intervals, let x and y be the weights of intervals chosen by D1 and D2, where without loss of generality, assume $x \leq y$. We emphasize that the adversary knows the values of x and y . We consider the following cases.

Case 1: $x = y = 1$. Both D1 and D2 obtains a profit of 1 while OPT schedules the heaviest interval giving a profit of α . So the competitive ratio is α .

Case 2: $x = y \neq 1$. One more interval of weight y is released just before the deadline of the y in the set. Both D1 and D2 either continue with the x or y , or abort and switch to the new y . In either case their profit is at most y . The adversary schedules the interval in the set just before y , together with the new y , giving a profit of $(y - \delta) + y$. Hence the competitive ratio is $2 - \delta/y > 2 - \delta$.

Case 3: $1 = x < y$. D1 and D2 gets a profit of 1 and y respectively while OPT gets α . Thus the competitive ratio is $\alpha/((1+y)/2) \geq 2\alpha/(1+\alpha) = 2 - 2/(1+\alpha)$.

Case 4: $1 < x < y$. The adversary releases another interval with weight y just before the deadline of x in the set. We distinguish two subcases.

If D1 does not abort x in favour of the new y , no more intervals are released. (We remark that the adversary knows the response of D1 and can make requests accordingly.) In this case D1 and D2 get a profit of x and y respectively, while OPT gets $x - \delta + y$. Then the competitive ratio $= (x + y - \delta)/((x + y)/2) = 2 - 2\delta/(x + y) > 2 - 2\delta/(1 + 1) = 2 - \delta$.

If D1 aborts x and serves y , then one more interval of weight y arrives just before the deadline of y in the original set. Then both D1 and D2 gets a profit of y no matter what they do, and OPT gets a profit of $y - \delta + y$. The competitive ratio is $(2y - \delta)/y = 2 - \delta/y > 2 - \delta$.

Considering all cases, the competitive ratio is at least $\min\{\alpha, 2 - \delta, 2 - 2/(1 + \alpha)\}$. By choosing $\delta < \epsilon$ and $\alpha > \max(2 - \epsilon, 2/\epsilon - 1) = 2/\epsilon - 1$, we have the competitive ratio being at least $2 - \epsilon$. \square

5 The Multiprocessor Case

In this section we consider the case of using more than one processor to schedule the intervals. We will see that the cases of randomization and multiple processors are closely related. We first show that the idea of the barely random algorithm in Section 3 can be used to give a deterministic 2-processor algorithm with the same competitive ratio. Then we show that the lower bounds in Section 4 can also be carried to the multiprocessor case; namely, that no deterministic or randomized algorithm can be better than $4/3$ -competitive for any number of processors m , and no 2-processor deterministic algorithm can be better than 2-competitive.

5.1 A 2-Processor Algorithm

We consider the following deterministic 2-processor algorithm. Call the two processors P1 and P2. In simple terms, P1 runs Greedy $_{\alpha}$ whereas P2 runs Greedy $_{\beta}$. Specifically, suppose the two processors are running intervals I_1 and I_2 respectively. When a new interval I arrives, if $|I| < \alpha|I_1|$ and $|I| < \beta|I_2|$ then I is rejected. If one of $|I| \geq \alpha|I_1|$ and $|I| \geq \beta|I_2|$ is true, the corresponding I_1 or I_2 is aborted and I is started on that processor. If I is at least as large as both $\alpha|I_1|$ and $\beta|I_2|$, it aborts I_2 and start I on P2. (This is the only difference from the randomization case: since the two processors cannot be doing the same interval, we need some way of tie-breaking.) A processor which has completed its interval will become idle. Note that an idle processor is regarded as executing a weight-0

interval. Therefore if P1 is idle and $|I| \geq \beta|I_2|$, it will still abort I_2 (and P1 remains idle). Again we set $\alpha = \phi$ and $\beta = \phi^2$.

We will separately bound the value of the two optimal offline schedules produced by the two processors, OPT1 and OPT2. As before, we divide the schedule into basic subschedules and segments. With the same notation as in Section 3, consider a segment where OPT1 schedules (Z_1, \dots, Z_n) , P1 schedules (X_1, \dots, X_n) , and P2 schedules (Y_1, \dots, Y_n) . We will show the same bound on the competitive ratio, i.e. $2 \sum z_i / (x_n + y_n - \max(x_1/\alpha, y_1/\beta)) \leq \phi + 2$. Therefore over the whole OPT1, $OPT1 / ((P1 + P2)/2) \leq \phi + 2$. Here OPT1, P1 and P2 represent both the schedules and their profits. Since OPT2 can be analyzed similarly, we have $OPT2 / ((P1 + P2)/2) \leq \phi + 2$. Adding these two together, $OPT1 + OPT2 \leq (\phi + 2)(P1 + P2)$ and therefore the algorithm is $(\phi + 2)$ -competitive. In the analysis below we only consider OPT1.

We call (z_k, x_k, y_k) in a segment a *triplet*. We first make an observation:

Lemma 2. *For any triplet (z_k, x_k, y_k) , one of the following two cases holds: (i) $x_i > z_i/\alpha$ and $y_i > z_i/\beta$, (ii) $z_i = y_i$ and $x_i \leq z_i/\alpha$.*

We call triplets of case (i) *normal triplets* and those of case (ii) *violating triplets*. The main idea of the competitiveness proof is as follows: if there are no violating triplets in a segment, then we are done by the same proof as in the randomized algorithm. If there are violating triplets, we further divide the segment into *subsegments* so that each subsegment has at most one violating triplet at the beginning of the subsegment. We then perform a similar analysis to the randomized algorithm on each subsegment. There is a small difference in the amortized terms: both the x_1/α and y_1/β terms may be subtracted, since the last pair of intervals in the previous subsegment may not be completed, and hence do not have any real profit. So the amortized term may sometimes become $x_1/\alpha + y_1/\beta$ instead of $\max(x_1/\alpha, y_1/\beta)$. We omit the proof to this theorem:

Theorem 4. *The algorithm is $\phi + 2 \approx 3.618$ -competitive for 2 processors.*

5.2 Lower Bounds

The proofs of the following theorems use almost identical constructions to that in Theorems 2 and 3, so we omit the proofs.

Theorem 5. *No deterministic or randomized algorithm for online interval scheduling on m processors is better than $4/3$ -competitive, for any m .*

Theorem 6. *No deterministic algorithm for online interval scheduling on 2 processors is better than 2-competitive.*

6 Conclusion

In this paper we give the first randomized algorithm and improved lower bounds for the online interval scheduling problem. The gap between the upper and lower

bounds remains wide, however. It may be possible to generalize the barely random algorithm to use 3 or more deterministic algorithms but we encounter some technical difficulties in extending the technique here. Algorithms for three or more processors will also yield randomized algorithms for the one-processor case.

References

1. Awerbuch, B., Bartal, Y., Fiat, A., Rosen, A.: Competitive non-preemptive call control. In: Proc. 5th SODA, pp. 312–320 (1994)
2. Baruah, S., Koren, G., Mao, D., Mishra, B., Raghunathan, A., Rosier, L., Shasha, D., Wang, F.: On the competitiveness of on-line real-time task scheduling. *Real-Time Systems* 4, 125–144 (1992)
3. Bogart, K.P., West, D.B.: A short proof that proper = unit. *Discrete Mathematics* 201, 21–23 (1999)
4. Canetti, R., Irani, S.: Bounding the power of preemption in randomized scheduling. *SIAM Journal on Computing* 27(4), 993–1015 (1998)
5. Chan, W.-T., Lam, T.-W., Ting, H.-F., Wong, P.W.H.: New results on on-demand broadcasting with deadline via job scheduling with cancellation. In: Chwa, K.-Y., Munro, J.I.J. (eds.) COCOON 2004. LNCS, vol. 3106, pp. 210–218. Springer, Heidelberg (2004)
6. Faigle, U., Nawijn, W.M.: Greedy k-coverings of interval orders. Technical Report 979, University of Twente (1991)
7. Fung, S.P.Y., Chin, F.Y.L., Poon, C.K.: Laxity helps in broadcast scheduling. In: Coppo, M., Lodi, E., Pinna, G.M. (eds.) ICTCS 2005. LNCS, vol. 3701, pp. 251–264. Springer, Heidelberg (2005)
8. Kim, J.-H., Chwa, K.-Y.: Scheduling broadcasts with deadlines. *Theoretical Computer Science* 325(3), 479–488 (2004)
9. Koren, G., Shasha, D.: D^{over} : An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM Journal on Computing* 24, 318–339 (1995)
10. Koren, G., Shasha, D.: MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling. *Theoretical Computer Science* 128(1-2), 75–97 (1994)
11. Lipton, R.J., Tomkins, A.: Online interval scheduling. In: Proc. 5th SODA, pp. 302–311 (1994)
12. Miyazawa, H., Erlebach, T.: An improved randomized on-line algorithm for a weighted interval selection problem. *Journal of Scheduling* 7(4), 293–311 (2004)
13. Seiden, S.S.: Randomized online interval scheduling. *Operations Research Letters* 22(4–5), 171–177 (1998)
14. Ting, H.-F.: A near optimal scheduler for on-demand data broadcasts. In: Calamoneri, T., Finocchi, I., Italiano, G.F. (eds.) CIAC 2006. LNCS, vol. 3998, pp. 163–174. Springer, Heidelberg (2006)
15. Woeginger, G.J.: On-line scheduling of jobs with fixed start and end times. *Theoretical Computer Science* 130(1), 5–16 (1994)
16. Zheng, F., Fung, S.P.Y., Chan, W.-T., Chin, F.Y.L., Poon, C.K., Wong, P.W.H.: Improved on-line broadcast scheduling with deadlines. In: Chen, D.Z., Lee, D.T. (eds.) COCOON 2006. LNCS, vol. 4112, pp. 320–329. Springer, Heidelberg (2006)

Scheduling Selfish Tasks: About the Performance of Truthful Algorithms

George Christodoulou¹, Laurent Gourvès², and Fanny Pascual³

¹ Max-Planck-Institut für Informatik, Saarbrücken, Germany
gchristo@mpi-inf.mpg.de

² LAMSADE, CNRS UMR 7024, Université de Paris-Dauphine, Paris, France
laurent.gourves@lamsade.dauphine.fr

³ Equipe MOAIS (CNRS-INRIA-INPG-UJF), Grenoble, France
fanny.pascual@imag.fr

Abstract. This paper deals with problems which fall into the domain of selfish scheduling: a protocol is in charge of building a schedule for a set of tasks without directly knowing their length. The protocol gets these informations from agents who control the tasks. The aim of each agent is to minimize the completion time of her task while the protocol tries to minimize the maximal completion time. When an agent reports the length of her task, she is aware of what the others bid and also of the protocol's algorithm. Then, an agent can bid a false value in order to optimize her individual objective function. With erroneous information, even the most efficient algorithm may produce unreasonable solutions. An algorithm is truthful if it prevents the selfish agents from lying about the length of their task. The central question in this paper is: “*How efficient a truthful algorithm can be?*” We study the problem of scheduling selfish tasks on parallel identical machines. This question has been raised by Christodoulou et al [8] in a distributed system, but it is also relevant in centrally controlled systems. Without considering side payments, our goal is to give a picture of the performance under the condition of truthfulness.

Keywords: scheduling, algorithmic game theory, truthful algorithms.

1 Introduction

The Internet is a complex distributed system involving many autonomous entities (*agents*). Protocols organize this network, using the data held by these agents and trying to maximize the social welfare. Agents are often supposed to be trustworthy but this assumption is unrealistic in some settings as they might try to manipulate the protocol by reporting false information in order to maximize their own profit. With false information, even the most efficient protocol may lead to unreasonable solutions if it is not designed to cope with the selfish behavior of the single entities. Then, it is natural to ask the following question: *How efficient a protocol can be if it guarantees that no agent has incentive to lie?*

In this paper, we deal with the problem of scheduling n selfish tasks on m identical parallel machines. We consider two distinct settings in which the aim is to minimize the *makespan*, i.e. the maximum completion time. The first setting is centralized, while the second one is distributed. Both problems share the following characteristics. Each task is owned by an agent¹. The length l_i of a task i is known to its owner only. The agents, considered as players of a non-cooperative game, want to minimize the completion time of their tasks. The protocol builds the schedule with rules known to all players and fixed in advance. In particular, mixing the execution of two jobs (like round-robin) is not allowed. Before the execution begins, the agents report a value representing the length of their tasks. We assume that every agent behaves rationally and selfishly. Each one is aware of the situation the others face and tries to optimize her own objective function. Thus an agent can report a value which is not equal to her real length. Practically, an agent can add “fake” data to artificially increase the length of her task if it decreases her completion time. This selfish behavior can prevent the protocol to produce a reasonable (i.e. close to the social welfare) schedule. Without considering side payments, which are often used with the aim of inciting the agents to report their real value, some algorithmic tools can simultaneously offer a guarantee on the quality of the schedule (its makespan is not arbitrarily far from the optimum) and guarantee that the solution is *truthful* (no agent can lie and improve her own completion time). For both centralized and distributed settings, our goal is to give lower and upper bounds on the performance under the condition of truthfulness. It is important to mention that we do not strictly restrict the study to polynomial time algorithms.

Since the length of a task is private, each agent bids a value which represents the length of her task. We assume that an agent cannot shrink the length of her task (otherwise she will not get her result), but if she can decrease her completion time by bidding a value larger than the real one, then she will do so. We also assume that an agent does not report a distribution on different lengths. A player may play according to a distribution, but she just announces the outcome, so the protocol does not know if she lies.

In the *centralized setting*, the strategy of agent i is a value b_i representing the length of her task. The protocol, called an algorithm, is in charge of indicating when and on which machine a task will be scheduled. An algorithm is *truthful* when no agent has incentive to report a false value. We focus on the performance of truthful algorithms with respect to the makespan of the schedule. In particular, we are interested in giving lower and upper bounds on the *approximation ratio* that a (deterministic or randomized) truthful algorithm can achieve. For example, a truthful algorithm can be obtained by greedily scheduling the tasks following the increasing order of their lengths. This algorithm, known as SPT, produces a $(2 - 1/m)$ -approximate schedule [11]. *Are there truthful algorithms with better approximation guarantee for the considered scheduling problem?*

¹ We equally refer to a task and its owner since we assume that two tasks cannot be held by the same agent.

In the *distributed setting*, the strategy of agent i is a couple (M_i, b_i) , where M_i is the machine which will execute the task and b_i is the length bidden. As opposed to the centralized setting, the agents choose their machine and M_i can be a probability distribution on different machines. The protocol, called a *coordination mechanism* in this context [8], consists in selecting a *scheduling policy* for each machine (e.g. scheduling the tasks in order of decreasing lengths). An important and natural condition is due to the decentralized nature of the problem: the scheduling on a machine should depend only on the tasks assigned to it, and should be independent of the tasks assigned to the other machines. A coordination mechanism is *truthful* when no agent has incentive to lie on the length of her task. Using the *price of anarchy* [14], we study the performance of truthful coordination mechanisms with respect to the makespan. The price of anarchy of a coordination mechanism is, in the context, equal to the largest ratio between the makespan of a schedule where agent's strategies form a *Nash equilibrium*² and the optimal makespan.

Interestingly, it is possible to slightly transform the SPT algorithm in a truthful coordination mechanism, as suggested in [8]: each machine P_j schedules its tasks in order of increasing lengths, and adds at the very beginning of the schedule a small delay equal to $(j - 1)\varepsilon$ times the length of the first task. By this way, and if ε is small enough, the schedule obtained in a Nash equilibrium is similar to the one returned by the SPT algorithm (excepted the small delays at the beginning of the schedule). When ε is negligible, the price of anarchy of this coordination mechanism is $2 - 1/m$. *Are there truthful coordination mechanisms with better price of anarchy for the considered scheduling problem?*

For both centralized algorithms and coordination mechanisms, we consider the two following execution models:

- **Strong model of execution:** If the owner of task i bids $b_i \geq l_i$, then the execution time will still be l_i (i.e. the task will be completed l_i time units after its start).
- **Weak model of execution:** If the owner of task i bids $b_i \geq l_i$, then the execution time will be b_i (i.e. the task will be completed b_i time units after its start).

The strong execution model corresponds to the case where tasks have to be linearly executed – from their beginning to their end –, whereas the weak execution model corresponds to the case where a task can be executed in any order³ (and the “fake” part of the task is not anymore necessarily executed at the end), or when the machine returns the result of the task only at the end of its execution. Depending on the applications of the scheduling problem, either the strong or the weak model of execution will be used.

² Situation in which no agent can unilaterally change her strategy and improve her own completion time. A Nash equilibrium is *pure* if each agent has a pure strategy : each agent chooses only one machine. A Nash equilibrium is *mixed* if the agents give a probability distribution on the machines on which they will go.

³ Nevertheless, the execution of two jobs is never interlaced.

Related Work

The field of *Mechanism Design* can be useful to deal with the selfishness of the agents. Its main idea is to pay the agents to convince them to perform strategies that help the system to optimize a global objective function. The most famous technique for designing truthful mechanisms is perhaps the Vickrey-Clarke-Groves (VCG) mechanism [20,7,12]. However, when applied to combinatorial optimization problems, this mechanism guarantees the truthfulness under the hypothesis that the objective function is *utilitarian* (i.e. the value of the objective function is equal to the sum of the agents individual objective functions) and that the mechanism is able to compute the optimum. Archer and Tardos introduce in [4] a method which allows to design truthful mechanisms for several combinatorial optimization problems to which the VCG mechanism does not apply. However, both approaches cannot be applied to our problem.

Scheduling selfish agents has been intensively studied these last years, started with the seminal work of Nisan and Ronen [17], and followed by a series of papers [12,4,6,9,15,16]. However, all these works differ from ours since in their case, the selfish agents are the machines while here we consider that the agents are the tasks. Furthermore, they use side payments whereas we focus on truthful algorithms without side payments.

A more closely related work is the one of Christodoulou et al [8] who considered the same model but only in the distributed context of coordination mechanisms. They proposed different coordination mechanisms with a price of anarchy better than the one of the SPT coordination mechanism. Nevertheless, these mechanisms are not truthful. In [13], the authors gave coordination mechanisms for the same model for related machines (i.e. machines can have different speeds), but their mechanisms are also not truthful.

In [3], the authors gave a truthful randomized algorithm for the strong model of execution defined before, and they gave, for the weak model of execution, a coordination mechanism which is truthful if there are two machines and if the lengths of the tasks are powers of a certain constant. An optimal (but exponential time) truthful randomized algorithm and a truthful randomized PTAS for the weak model of execution appear in [18,19]. The technique consists in computing an optimal (resp. a $(1 + \varepsilon)$ -approximate) schedule and each machine executes its tasks in a random order (the truthfulness is due to the introduction of fictitious tasks which guarantee that all the machines have the same load).

Another related work is the one of Auletta et al. who considered in [5] the problem of scheduling selfish tasks in a centralized case. Their work differs from ours since they considered that each machine uses a round and robin policy and thus that the completion of each task is the completion time of the machine on which the task is (this model is known as the KP model). They considered that the tasks can lie in both directions, and that there are some payments.

Contribution and Organization of the Article

Sections 3 and 4 are devoted to the centralized setting. In particular, we study the strong (resp. weak) model of execution in Section 3 (resp. Section 4). Results on the distributed setting are presented in Section 5 for both execution models.

Table 1 gives a summary of the bounds that we are aware of (those with a † are presented in this article). LB stands for “Lower bound”, UB for “Upper bound” and NE for “Nash equilibria”. Due to space constraints, some proofs are omitted.

Table 1. Bounds for m identical machines

Strong model of execution:

	Deterministic		Randomized	
	LB	UB	LB	UB
centralized setting	$2 - \frac{1}{m}$ †	$2 - \frac{1}{m}$ [8]	$\frac{3}{2} - \frac{1}{2m}$ †	$2 - \frac{1}{m+1} (\frac{5}{3} + \frac{1}{3m})$ [3]
distributed setting	$2 - \frac{1}{m}$ (pure NE) † $\frac{3}{2} - \frac{1}{2m}$ (mixed NE) †	$2 - \frac{1}{m}$ [8]	$\frac{3}{2} - \frac{1}{2m}$ †	$2 - \frac{1}{m}$

Weak model of execution:

	Deterministic		Randomized	
	LB	UB	LB	UB
centralized setting	$m = 2 : 1 + \frac{\sqrt{105-9}}{12} > 1.1$ † $m \geq 3 : \frac{7}{6} > 1.16$ †	$\frac{4}{3} - \frac{1}{3m}$ †	1 [18,19]	1 [18,19]
distributed setting	$\frac{1+\sqrt{17}}{4} > 1.28$ (pure NE) †	$2 - \frac{1}{m}$	$1 + \frac{\sqrt{13-3}}{4} > 1.15$ † (pure NE)	$2 - \frac{1}{m}$

2 Notations

We are given m machines (or processors) $\{P_1, \dots, P_m\}$, and n tasks $\{1, \dots, n\}$. Let l_i denote the real execution time (or length) of task i . We use the identification numbers to compare tasks of the same (bidden) lengths: we will say that task i , which bids b_i , is larger than task j , which bids b_j , if and only if $b_i > b_j$ or ($b_i = b_j$ and $i > j$). It is important to mention that an agent cannot lie on her (unique) identification number.

A randomized algorithm can be seen as a probability distribution over deterministic algorithms. We say that a (randomized) algorithm is truthful if for every task the expected completion time when she declares her true length is smaller than or equal to her expected completion time in the case where she declares a larger value. More formally, we say that an algorithm is *truthful* if $E_i[l_i] \leq E_i[b_i]$, for every i and $b_i \geq l_i$, where $E_i[b_i]$ is the expected completion time of task T_i if she declares b_i . In order to evaluate the quality of a randomized algorithm, we use the notion of expected approximation ratio.

We will refer in the sequel to the list scheduling algorithms LPT and SPT, where LPT (resp. SPT) [11] is the algorithm which greedily schedules the tasks, sorted in order of decreasing (resp. increasing) lengths: this algorithm schedules, as soon as a machine is available, the largest (resp. smallest) task which has not yet been scheduled. An LPT (resp. SPT) schedule is a schedule returned by the LPT (resp. SPT) algorithm.

3 About Truthful Algorithms for the Strong Model of Execution

3.1 Deterministic Algorithms

We saw that the deterministic algorithm SPT, which is $(2 - \frac{1}{m})$ -approximate, is truthful. Let us now show that there is no truthful deterministic algorithm with a better approximation ratio.

Theorem 1. *Let us consider that we have m identical machines. There is no truthful deterministic algorithm with an approximation ratio smaller than $2 - \frac{1}{m}$.*

Proof. Let us suppose that we have $n = m(m - 1) + 1$ tasks of length 1. Let us suppose that we have a truthful deterministic algorithm \mathcal{A} which has an approximation ratio smaller than $(2 - 1/m)$. Let t be the task which has the maximum completion time, C_t , in the schedule returned by \mathcal{A} . We know that $C_t \geq m$.

Let us now suppose that task t bids m instead of 1. We will show that the completion time of t is then smaller than m . Let OPT be the makespan of an optimal solution where there are $n - 1 = m(m - 1)$ tasks of length 1 and a task of length m . We have: $OPT = m$. Since the approximation ratio of algorithm \mathcal{A} is smaller than $(2 - 1/m)$, the makespan of the schedule it builds with this instance is smaller than $(2 - 1/m)m = 2m - 1$. Thus, the task of length m starts before time $(m - 1)$. Thus, if task t bids m instead of 1, it will start before time $m - 1$ and be completed one time unit after, that is before time m . Thus task t will decrease its completion time by bidding m instead of 1, and algorithm \mathcal{A} is not truthful.

Note that we can generalize this result to the case of related machines: we have m machines P_1, \dots, P_m , such that machine P_i has a speed v_i , $v_1 = 1$, and $v_1 \leq \dots \leq v_m$. By this way, the bound becomes $2 - \frac{v_m}{\sum_{i=1}^m v_i}$.

Concerning the strong model of execution, no deterministic algorithm can outperform SPT in the centralized setting. Then, it is interesting to consider randomized algorithms to achieve a better approximation ratio.

3.2 Randomized Algorithms

In [3], the authors present a randomized algorithm which consists in returning a LPT schedule with a probability $1/(m + 1)$ and a slightly modified SPT schedule with a probability $m/(m + 1)$. They obtain a truthful algorithm whose expected approximation ratio improves $2 - \frac{1}{m}$ but no instance showing the tightness of their analysis is provided. A good candidate should be simultaneously a tight example for both LPT and SPT schedules. We are not aware of the existence of such an instance and we believe in a future improvement of this upper bound. The following Theorem provides a lower bound.

Theorem 2. *Let us consider that we have m identical machines. There is no truthful randomized algorithm with an approximation ratio smaller than $\frac{3}{2} - \frac{1}{2m}$.*

Generalizing this result to the case of related machines, the bound becomes $\frac{3}{2} - \frac{v_m}{2 \sum_{i=1}^m v_i}$.

4 About Truthful Algorithms for the Weak Model of Execution

4.1 A Truthful Deterministic Algorithm

We saw in the Section 3 that SPT is a truthful and $(2 - 1/m)$ -approximate algorithm for the strong model of execution, and that no truthful deterministic algorithm can have a better approximation ratio. If we consider the weak model of execution, we can design a truthful deterministic algorithm, called LPT_{mirror} , with a better performance guarantee. We are given n tasks $\{1, \dots, n\}$ which bid lengths b_1, \dots, b_n . Make a schedule σ_{LPT} with the LPT list algorithm. Let C_{max}^{OPT} be the optimal makespan. Let $p(i)$ be the machine on which the task i is executed in σ_{LPT} . Let C_i be date at which the task i ends in σ_{LPT} . LPT_{mirror} returns the schedule in which task i is executed on machine $p(i)$ and starts at time $(4/3 - 1/(3m))C_{max}^{OPT} - C_i$.

Theorem 3. *LPT_{mirror} is a deterministic, truthful and $(\frac{4}{3} - \frac{1}{3m})$ -approximate algorithm.*

Proof. We are given n tasks with true lengths l_1, \dots, l_n . Let us suppose than each task has bidden a value, and that task i bids $b_i > l_i$. This can make the task i start earlier in σ_{LPT} but never later. In addition, the optimal makespan when i bids $b_i > l_i$ is necessarily larger than or equal to the optimal makespan when task i reports its true length.

Let S_i be the date at which task i starts to be executed in σ_{LPT} . The completion time of task i in LPT_{mirror} is $(4/3 - 1/(3m))OPT - C_i + b_i = (4/3 - 1/(3m))OPT - S_i$ because $S_i = C_i - b_i$. By bidding $b_i > l_i$, task i can only increase its completion time in the schedule returned by LPT_{mirror} because OPT does not decrease and S_i does not increase. Thus task i does not have incentive to lie.

Since the approximation ratio of the schedule obtained with the LPT list algorithm is at most $(4/3 - 1/(3m))$ [11], the schedule returned by LPT_{mirror} is clearly feasible and its makespan is, by construction, $(4/3 - 1/(3m))$ -approximate. Thus LPT_{mirror} is a truthful and $(\frac{4}{3} - \frac{1}{3m})$ -approximate algorithm.

Note that LPT_{mirror} is not a polynomial time algorithm, since we need to know the value of the makespan in an optimal solution, which is an NP-hard problem [10]. However, it is possible to have a polynomial time algorithm which is $(4/3 - 1/(3m))$ -approximate, even if some tasks do not bid their true values. Consider the following simple algorithm: we first compute a schedule σ_{LPT} with the LPT algorithm. Let $p(i)$ be the machine on which the task i is executed in σ_{LPT} , let C_i be the completion time of task i in σ_{LPT} , and let C_{max} be the makespan of

σ_{LPT} . We then compute the final schedule σ' in which task i is scheduled on $p(i)$ and starts at time $C_{max} - C_i$.

We can show that this algorithm is $(4/3 - 1/(3m))$ -approximate (i.e. the schedule returned by this algorithm is at most $(4/3 - 1/(3m))$ times larger than the optimal schedule in which all the tasks bid their true values). We can show this by the following way. We suppose that all the tasks except i have bidden some values. Let $\sigma_{LPT}(b_i)$ be the schedule σ_{LPT} obtained when i bids b_i , let $S_i(b_i)$ be the date at which task i starts to be executed in $\sigma_{LPT}(b_i)$, and let $C_{max}(\sigma_{LPT}(b_i))$ be the makespan of $\sigma_{LPT}(b_i)$. The completion time of task i (which bids b_i) in σ' is equal to $C_{max}(\sigma_{LPT}(b_i)) - S_i(b_i)$. Since with the LPT algorithm, tasks are scheduled in decreasing order of lengths, if $b_i > l_i$ then $S_i(b_i) \leq S_i(l_i)$. Thus, whatever the values bidden by the other tasks are, i has incentive to lie and bid $b_i > l_i$ only if $C_{max}(\sigma_{LPT}(b_i)) < C_{max}(\sigma_{LPT}(l_i))$. Since this is true for each task, no task will unilaterally lie unless this decreases the makespan of the schedule. The makespan of the schedule σ' in which all the tasks bid their true values is $(4/3 - 1/(3m))$ -approximate, and then the solution returned by this algorithm will also be $(4/3 - 1/(3m))$ -approximate.

4.2 Deterministic Algorithms: Lower Bounds

We suppose that the solution returned by an algorithm depends on the length and the identification number of each task, even those which can be identified with their unique length.

Theorem 4. *Let us consider that we have two identical machines. There is no truthful deterministic algorithm with an approximation ratio smaller than $1 + (\sqrt{105} - 9)/12 \approx 1.1039$.*

Theorem 5. *Let us consider that we have $m \geq 3$ identical machines. There is no truthful deterministic algorithm with an approximation ratio smaller than $7/6$.*

The assumption made to derive Theorems 4 and 5 is, in a sense, stronger than the usual one since we suppose that the solution returned by an algorithm for two similar instances (same number of tasks, same lengths but different identification numbers) can be completely different. If we relax this assumption, i.e. if identification numbers are only required for the tasks which have the same length, the bound presented in Theorem 4 can be improved to $7/6$.

Theorem 6. *Let us consider that we have two identical machines. No truthful deterministic algorithm can be better than $7/6$ -approximate if it does not take into account the identification number of tasks whose length is unique.*

5 About Truthful Coordination Mechanisms

Let $\rho \geq 1$. If there is no truthful deterministic algorithm which has an approximation ratio of ρ , then there is no truthful deterministic coordination mechanism which always induce pure Nash equilibria and which has a price of anarchy

smaller than or equal to ρ . Indeed, if this was not the case, then the deterministic algorithm which consists in building the schedule obtained in a pure Nash equilibrium with this ρ -approximate coordination mechanism would be a ρ -approximate truthful deterministic algorithm.

Likewise, if there is no truthful (randomized) algorithm which has an approximation ratio of ρ , then there is no truthful coordination mechanism which has a price of anarchy smaller than or equal to ρ . Indeed, if this was not the case, the algorithm which consists in building the schedule obtained in a Nash equilibrium with this ρ -approximate coordination mechanism would be a ρ -approximate truthful algorithm.

This observation leads us to the following results for the strong model of execution. We deduce from Theorem 1 that there is no truthful deterministic coordination mechanism which always induce pure Nash equilibria and which has a price of anarchy smaller than $2 - 1/m$. Thus there is no truthful coordination mechanism which performs better than the truthful SPT coordination mechanism, whose price of anarchy tends towards $2 - 1/m$. We deduce from Theorem 2 that there is no truthful coordination mechanism which has a price of anarchy smaller than $\frac{3}{2} - \frac{1}{2m}$. We now consider the weak model of execution.

Theorem 7. *If we consider the weak model of execution, there is no truthful deterministic coordination mechanism which induces pure Nash equilibria, and which has a price of anarchy smaller than $\frac{1+\sqrt{17}}{4} \approx 1.28$.*

Proof. Let us first prove this result in the case where there are two machines, P_1 , and P_2 . Let $\varepsilon > 0$. Let us suppose that there exists a truthful coordination mechanism \mathcal{M} with a price of anarchy of $\frac{1+\sqrt{17}}{4} - \varepsilon$. Let us consider the following instance I_1 : three tasks of length 1. Since \mathcal{M} is a deterministic coordination mechanism which induces pure Nash equilibria, there is at least a task in I_1 which has a completion time larger than or equal to 2. Let t be such a task.

Let us first consider this instance I_2 : we have two tasks of length $\frac{-1+\sqrt{17}}{2} \approx 1.56$. Since \mathcal{M} is $(\frac{1+\sqrt{17}}{4} - \varepsilon)$ -approximate, there is one task on each machine, and each task is completed before time $\frac{-1+\sqrt{17}}{2} \times \frac{1+\sqrt{17}}{4} = 2$. Thus, when it has a task of length $\frac{-1+\sqrt{17}}{2}$, each machine must end it before time 2.

Let us now consider the following instance I_3 : two tasks of length 1, and a task of length $\frac{-1+\sqrt{17}}{2}$. Since \mathcal{M} is $(\frac{1+\sqrt{17}}{4} - \varepsilon)$ -approximate, the task of length $\frac{-1+\sqrt{17}}{2}$ is necessarily alone on its machine (without loss of generality, on P_2). As we have seen it, P_2 must schedule this task before time 2. Thus, task t of instance I_1 , has incentive to bid $\frac{-1+\sqrt{17}}{2}$ instead of 1: by this way it will end before time 2, instead of a time larger than or equal to 2.

We can easily extend this proof in the case where there are more than 2 machines, by having $m + 1$ tasks of length 1 in I_1 ; m tasks of length $\frac{-1+\sqrt{17}}{2}$ in I_2 ; and m tasks of length 1 and a task of length $\frac{-1+\sqrt{17}}{2}$ in I_3 .

Theorem 8. *If we consider the weak model of execution, there is no truthful coordination mechanism which induces pure Nash equilibria, and which has a price of anarchy smaller than $1 + \frac{\sqrt{13}-3}{4} \approx 1.15$.*

6 Conclusion

We showed that, in the strong model of execution, the list algorithm SPT, which has an approximation ratio of $2 - 1/m$ is the best truthful deterministic algorithm, and that there is no truthful randomized algorithm which has an approximation ratio smaller than $3/2 - 1/(2m)$. On the contrary, if we relax the constraints on the execution model, i.e. if the result of a task which bid b is given to this task only b time units after its start, then we can obtain better results. In this model of execution, there is a truthful $4/3 - 1/(3m)$ -approximate deterministic algorithm and a truthful optimal randomized algorithm. For both execution models, we also gave lower bounds on the approximation ratios that a truthful coordination mechanism can have.

As a future work, it would be interesting to improve the results for which a gap between the lower and the upper bound exists. For example, we believe that the lower bound $\frac{1+\sqrt{17}}{4}$ (lower bound on the performance of a truthful deterministic coordination mechanism for the weak model of execution) can be improved to $3/2$ for two machines.

Another direction would be to restrict the study to truthful algorithms (or coordination mechanisms) which run in polynomial time. Giving improved lower bounds which rely on a computational complexity argument would be very interesting.

Acknowledgments. We thank Elias Koutsoupias for helpful suggestions and discussions on the problem.

References

1. Ambrosio, P., Auletta, V.: Deterministic Monotone Algorithms for Scheduling on related Machines. In: Persiano, G., Solis-Oba, R. (eds.) WAOA 2004. LNCS, vol. 3351, pp. 267–280. Springer, Heidelberg (2005)
2. 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)
3. Angel, E., Bampis, E., Pascual, F.: Truthful Algorithms for Scheduling Selfish Tasks on Parallel Machines. In: Deng, X., Ye, Y. (eds.) WINE 2005. LNCS, vol. 3828, pp. 698–707. Springer, Heidelberg (2005)
4. Archer, A., Tardos, E.: Truthful Mechanisms for One-Parameter Agents. In: Proc. of FOCS 2001, pp. 482–491 (2001)
5. Auletta, V., Penna, P., De Prisco, R., Persiano, P.: How to Route and Tax Selfish Unsplittable Traffic. In: Proc. of SPAA 2004, pp. 196–204 (2004)

6. Auletta, V., De Prisco, R., Penna, P., Persiano, P.: 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)
7. Clarke, E.: Multipart pricing of public goods. *Public Choices*, pp. 17–33 (1971)
8. Christodoulou, G., Koutsoupias, E., Nanavati, A.: Coordination Mechanisms. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 345–357. Springer, Heidelberg (2004)
9. Christodoulou, G., Koutsoupias, E., Vidali, A.: A lower bound for scheduling mechanisms. In: Proc. of SODA 2007 (2007)
10. Garey, M., Johnson, D.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co (1979)
11. Graham, R.: Bounds on multiprocessor timing anomalies. In: *SIAM Jr. on Appl. Math.* vol. 17(2), pp. 416–429 (1969)
12. Groves, T.: Incentive in teams. *Econometrica* 41(4), 617–631 (1973)
13. Immorlica, N., Li, L., Mirrokni, V.S., Schulz, A.: Coordination Mechanisms for Selfish Scheduling. In: Deng, X., Ye, Y. (eds.) WINE 2005. LNCS, vol. 3828, pp. 55–69. Springer, Heidelberg (2005)
14. Koutsoupias, E., Papadimitriou, C.: Worst Case Equilibria. In: Meinel, C., Tison, S. (eds.) STACS 99. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
15. Kovács, 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)
16. Mu’alem, A., Schapira, M.: Setting lower bounds on truthfulness. In: Proc. of SODA 2007 (2007)
17. Nisan, N., Ronen, A.: Algorithmic mechanism design. In: Proc. STOC 1999, pp. 129–140 (1999)
18. Pascual, F.: *Optimisation dans les réseaux : de l’approximation polynomiale à la théorie des jeux*. Ph.D Thesis, University of Evry, France, 2006 (in french).
19. Tchetchnia, A-A.: *Truthful algorithms for some scheduling problems*. Master Thesis MPRI, École Polytechnique, France (2006)
20. Vickrey, W.: Counterspeculation, auctions and competitive sealed tenders. *J. Finance* 16, 8–37 (1961)

Volume Computation Using a Direct Monte Carlo Method*

Sheng Liu^{1,2}, Jian Zhang¹, and Binhai Zhu³

¹ State Key Laboratory of Computer Science, Institute of Software,
Chinese Academy of Sciences, Beijing 100080, China
{lius,zj}@ios.ac.cn

² Graduate School, Chinese Academy of Sciences, Beijing 100049, China

³ Department of Computer Science, Montana State University, Bozeman, MT
59717-3880, USA
bhz@cs.montana.edu

Abstract. Volume computation is a traditional, extremely hard but highly demanding task. It has been widely studied and many interesting theoretical results are obtained in recent years. But very little attention is paid to put theory into use in practice. On the other hand, applications emerging in computer science and other fields require practically effective methods to compute/estimate volume. This paper presents a practical Monte Carlo sampling algorithm on volume computation/estimation and a corresponding prototype tool is implemented. Preliminary experimental results on lower dimensional instances show a good approximation of volume computation for both convex and non-convex cases. While there is no theoretical performance guarantee, the method itself even works for the case when there is only a membership oracle, which tells whether a point is inside the geometric body or not, and no description of the actual geometric body is given.

1 Introduction

Volume computation is a highly demanding task in software engineering, computer graphics, economics, computational complexity analysis, linear systems modeling, VLSI design, statistics, etc. It has been studied intensively and lots of progress has been made especially in recent decades [7,2,14,13]. However, so far most of the research work is only concerned with the computational complexity aspect. For instance, some researchers tried to obtain a theoretical lower bound at all costs and neglected the practical feasibility of their algorithms. Therefore, although there are some strong theoretical results on this problem, little progress has been made in putting them into practical use. For this reason, this paper focuses on practically usable tools on computing/estimating the volume of a body.

* This work is partially supported by the National Natural Science Foundation (NSFC) under grant number 60673044 and 60633010, and by Montana EPSCOR Visiting Scholar's Program.

Generally speaking, we want to compute an arbitrary body's volume, i.e., in general a body does not even have to be a polyhedron. But for convenience, we will discuss convex polyhedron in most of the paper and in the end we will show some empirical results on some non-convex bodies' volume computation. Specially, we use the *halfspace representation* of a polyhedron. That is to say, a polyhedron is specified by $P = \{X|AX \leq B\}$ for some $m \times n$ matrix A and m -vector B (the pair (A, B) is called a *halfspace representation* of P).

As for two and three dimensional polyhedra, the volume of some basic shape (rectangle for instance) can be computed using some known mathematical formulas. But it is not straightforward to efficiently handle more general and complicated instances using exact analytic method. What is more, as the dimension n increases, the computational effort required arises drastically. Dyer and Frieze show that if P is a polyhedron then it is $\#P$ -hard to evaluate the volume of P [4]. But later on, by introducing randomness into volume computation, Dyer, Frieze and Kannan gave a polynomial time randomized algorithm to estimate the volume to arbitrary accuracy in their pathbreaking paper [6]. That paper triggers the following works in this field, which reduce the complexity of the algorithm from n^{23} down to n^4 [10,9,11,5,11,8,12,13]. The method used is to reduce volume computation to sampling from a convex body, using the Multi-phase Monte Carlo method. It first constructs a sequence of incremental convex body $K_0 \subset K_1 \subset \dots \subset K_m = V$ (the volume of K_0 is easy to compute). Then the volume of K_1 can be estimated by generating sufficiently many independent uniformly distributed random points in K_1 and counting how many of them fall in K_0 , using an equation like equation (1). The other K_i 's can be computed similarly. At last V (K_m) is obtained. When generating random points, the Markov Chain method is adopted.

However, in contrast with the brisk development on the complexity aspect of the randomized algorithms, little attention is paid to bring the theoretical results into practical use. As mentioned in [3], although these randomized algorithms are very interesting, there does not seem to be any implementation of them. That is why the authors of [3] ignored the randomized algorithm in their practical study on volume computation. In this paper, in contrast, we mainly focus on such practical randomized approximate algorithms. Our algorithm is also based on a Monte Carlo sampling method. But, compared with those Markov chain Monte-Carlo-based algorithms, our method is much simpler and much easier to implement. What is more, preliminary experimental results are very promising.

2 The Framework of the Sampling Algorithm

Assume that the volume of the convex polyhedron to be computed is V , our algorithm tries to estimate the value of V by the random sampling method.

We first generate N uniformly distributed random points in the convex polyhedron P , then at step i we build a probing sphere S with radius r_i in the

polyhedron¹. After that we count the number of points that fall into S , denoted by N_p . Since the volume of the sphere (denoted by W) can be obtained immediately², we can easily obtain the volume of the polyhedron V_i from the following formula:

$$\frac{W}{V_i} = \frac{N_p}{N} \quad (1)$$

In particular, for the sake of higher accuracy, we carry out the probing procedure multiple times to obtain the average value (in the algorithm we use an adjustable parameter **Num_Of_Iteration** to denote this number).

Formally, the algorithm can be described as follows:

Algorithm 1. VOL(N)

```

1: generate  $N$  points randomly in  $P$ ;
2:  $sum = 0$ ;
3: for  $i = 1$  to Num_Of_Iteration do
4:   build a probing sphere  $S$  in  $P$  with radius  $r_i$ ;
5:   count the number of points in  $S$ ;
6:   compute the volume  $V_i$  of  $P$  using formula (1);
7:    $sum = sum + V_i$ ;
8: end for
9:  $V = sum / \mathbf{Num\_Of\_Iteration}$ ;
10: return  $V$ ;
```

3 Implementation

In general, the algorithm framework in section 2 is very simple and is easy to understand. But when putting it into practice, we must handle some difficult technical points and some of them need theoretical analysis.

3.1 Generating Random Points

As shown in Algorithm 1, first of all, we have to generate a lot of uniformly distributed points in the convex polyhedron. Generating points in a given (fat) convex body is easy. But it is hard to generate points that are distributed uniformly. Most previous work adopts a Markov Chain method to obtain theoretically uniformly distributed points. The idea is to divide the space into n -dimensional cubes and perform a random walk on all the cubes that lie within the given convex polyhedron. For each walk, one of the cubes that are orthogonally adjacent to the current cube in the convex body is randomly selected. Thus the random walk is ergodic and the stationary distribution is uniform on cubes in the

¹ Formally, an n -dimensional probing sphere S centering at (o_1, o_2, \dots, o_n) is defined as $S = \{(x_1, x_2, \dots, x_n) | \sum_{j=1}^n (x_j - o_j)^2 \leq r_i^2\}$.

² We know that $W = \pi^{n/2} r_i^n / \Gamma(1 + n/2)$, where Γ denotes the gamma function [15].

convex polyhedron. However, this method is too complicated to use in practice. Instead, we use the pseudo-random number generator to generate many points in the polyhedron and assume them to be uniformly distributed. But there are still some uncertainties in our method. For instance, how many random points are needed. Apparently, for the sake of accuracy, the more the better. But on the other hand, more points mean more time and lower speed. So we must take both into consideration and find a proper compromise. In our preliminary implementation, the number of points is defined as an adjustable parameter so that it can be tuned according to different cases.

3.2 On Selecting the Center of the Sphere

Once the sampling problem has been solved, we begin to probe in the polyhedron with a probing sphere. But before probing, an implied prerequisite condition must be fulfilled. That is to say, the probing sphere must perfectly lie in the convex polyhedron. Otherwise, it is easy to see that the result obtained from equation (1) will not be accurate. But how to make sure that the whole probing sphere stays within the convex polyhedron? Strictly speaking, the distance from the center of the sphere to each of the facets of the convex polyhedron should be at least as large as the radius of the sphere. To achieve this goal, we define a new polyhedron contained in the original polyhedron named the *shrunk* polyhedron. The *shrunk* polyhedron has the same number of facets and vertices as the original polyhedron. In fact they should have the same shape except that it is a smaller version of the original polyhedron. Each facet of the *shrunk* convex polyhedron is parallel to its counterpart in the original convex polyhedron and the distance between them should be at least r_i . If we have such a *shrunk* polyhedron, then the problem can be solved easily by restricting the center of the sphere to lie within the *shrunk* polyhedron. If we use the *halfspace representation* of P , the *shrunk* polyhedron can be obtained easily and accurately by simply replacing each linear constraint $\sum_{k=1}^n a_{jk}x_k \leq b_j$ with $\sum_{k=1}^n a_{jk}x_k \leq b_j - r_i * \sqrt{\sum_{k=1}^n a_{jk}^2}$.

However, Algorithm 1 does not state that the body must be represented by linear constraints. In fact, the body can be presented to the algorithm using a very general mechanism called a *membership oracle*, which only tells whether a point is inside the body³. That is to say, the algorithm is also applicable to nonlinear constraints and other complicated constraints. But for these representations, it is hard for us to obtain the *shrunk* body accurately. So we have to use some approximate methods or heuristics. For example, we may adopt a random select-and-test heuristic. First, we randomly select a point in the original body as the center of the probing sphere. Then, given a radius r_i , we randomly choose some points on the surface of the probing sphere and test whether all of these points are also contained in the original body. If some point fails the test, we will try another point as the center of the probing sphere and perform this select-and-test procedure again. Formally, it can be formulated in Algorithm 2.

³ Remember that in general a body may not be a polyhedron.

Algorithm 2. ForCenter(r_i)

```

1: FOUND=0;
2: for  $j = 1$  to Num_Try_Center do
3:   Selecting a point in the body randomly as the center of the sphere;
4:   for  $k = 1$  to Num_Try_Surface do
5:     Generating a point  $x$  on the surface of the sphere with radius  $r_i$ ;
6:     if  $X$  is not in the body then
7:       break;
8:     end if
9:   end for
10:  if  $k >$  Num_Try_Surface then
11:    FOUND=1;
12:    break;
13:  end if
14: end for
15: return FOUND;

```

The select-and-test heuristic is easy to carry out but there are still some details that we need to clarify. For example, how many points on the surface of the sphere should be tested in the testing process. In general, the more the better. But again in practice more points mean more resources and more running time. The number of points should also vary with the dimension of the probing sphere. We again use an adjustable parameter **Num_Try_Surface** to represent the number in our experiments and it turns out that the heuristic works very well.

3.3 On Radius Selection

As described above, when building the probing sphere, we should determine the radius of the sphere beforehand. It is easy to understand that the radius should not be too small so that there is no point falling into the probing sphere at all. That is to say, there should be at least some point in the sphere. Otherwise, that probing sphere is useless. Based on the random sampling method, we have the following probabilistic analysis.

Theorem 1. *Given an n -dimensional polyhedron with volume V and the total number of sampling points N , if the radius r of the probing sphere satisfies $r = \Theta(\sqrt[n]{\frac{V * \ln N}{N}})$ then the probability that there is at least one point in the sphere converges to 1 as N grows to infinity.*

Proof. Let W be the volume of the probing sphere. Let C_1 denote $\pi^{n/2}/\Gamma(1 + n/2)$. Then $W = C_1 * r^n$. Assume that $r = C_2 * \sqrt[n]{\frac{V * \ln N}{N}}$. Let E represent the event that the sphere is empty, then \bar{E} represents the event that there is at least one point in the sphere. Then we have:

$$\begin{aligned}
 P[\bar{E}] &= 1 - P[E] \\
 &\geq 1 - \left(\frac{V - W}{V}\right)^N \\
 &= 1 - \left(1 - \frac{C_1 * r^n}{V}\right)^N \\
 &= 1 - \left(1 - \frac{C_1 * C_2^n * \ln N}{N}\right)^N \\
 &= 1 - e^{-C_1 * C_2^n * \ln N} \\
 &= 1 - \frac{1}{N^{C_1 * C_2^n}}
 \end{aligned}
 \tag{2}$$

Although C_1 varies with dimension n [15], it is clear that given an n -dimensional instance, when N is big enough, $P[\bar{E}]$ will converge to 1. Thus we complete the proof. \square

On the other hand, in theory, the bigger r is, the better the approximation is.

Claim. Convergence is better when r is bigger.

Proof. Let $p = W/V$ denote the probability that a random point from the polyhedron also falls into the probing sphere. Then the distribution of the random variable N_p will conform to a binomial distribution $\mathbf{B}(N, p)$. Thus we have

$$\mathbf{Var}(N_p) = N * p * (1 - p) \qquad \mathbf{Mean}(N_p) = N * p$$

It follows that

$$\mathbf{Var}(N_p/N) = p * (1 - p)/N \qquad \mathbf{Var}(N_p)/\mathbf{Mean}(N_p) = 1 - p$$

The formula on $\mathbf{Var}(N_p/N)$ reveals that the convergence is better when p is near one⁴ than near 1/2. If $\mathbf{Var}/\mathbf{Mean}$ is used as a measure of convergence, we will also find that the smaller $1 - p$ is, the better the convergence is. While small $1 - p$ means big W , so it is easy to see that the claim holds. \square

The above analyses suggest that we had better find a radius that is as big as possible. Theoretically it is indeed the case but in practice a probing sphere with the largest possible radius may have its weakness in the uniformity of probing. Take a triangle for example, the largest probing sphere inside it may be the inscribed circle of it. If we use the largest probing sphere, we will restrict our probing to the sampling points in the inscribed circle only. However, because all the sampling points are simulated by pseudo-random numbers, we cannot guarantee whether they are absolutely uniformly distributed in any part of the polyhedron. Therefore, we make sure that the probing sphere can visit as many parts of the sampling points as possible, so as to make the probing more general. For this reason, a moderately large but not an extremely large radius may be more suitable.

⁴ The case of p near 0 is trivial, so we do not consider it.

Theorem 1 also reveals that r depends on V , which is exactly something we need to compute. In theory we can estimate an upper bound of V by building another convex polyhedron, which contains the original convex polyhedron and has a volume that is easier to compute. For example, the smallest axis-parallel bounding box may be enough for that purpose. This method heavily depends on the ratio between the volume of the polyhedron and the volume of the bounding box. But the ratio can be very small. What is more, for instances with nonlinear constraints, it is not always convenient to obtain the smallest bounding box.

To handle this problem, we adopt a self-adaptive heuristic in our implementation. First, we randomly choose two of the sampling points in the polyhedron and let r be the distance between them. With the current radius r , if we fail to find a proper center of the sphere after **Num_Try_Center** tries using the heuristic method given in the previous subsection, we assign $r \leftarrow r/2$. Once we find a proper sphere center, we stop so as to make r as big as possible. Experiments show that this self-adaptive method not only works on polyhedra with normal shapes, but is also competent for polyhedra of long skinny shapes.

4 Experiments and Analysis

Based on the above observations, a prototype tool is implemented. We experiment on many simple instances to examine its performance. For the sake of comparison, we also test it on instances with known volume (named REAL volume) and examine the ratios of the results computed by our program to the REAL volume. Due to the space limit, we only introduce some simple ones.

4.1 Simple Examples

Example 1 $\begin{cases} -x + 2y \leq 200 \\ -y \leq 0 \\ x - y \leq 0 \\ -x - y \leq 50 \\ x + y \leq 200 \end{cases}$. It is in fact a pentagon with vertices $(0, 0)$, $(-50, 0)$, $(-100, 50)$, $(200/3, 400/3)$, and $(100, 100)$ and its REAL volume (the area of the pentagon) is $38750/3$.

Example 2 $\begin{cases} x + y + z \leq 255 \\ -x \leq 0 \\ -y \leq 0 \\ -z \leq 0 \end{cases}$. It is in fact a tetrahedron defined by the four vertices $(0, 0, 0)$, $(255, 0, 0)$, $(0, 255, 0)$ and $(0, 0, 255)$. So we can easily obtain the REAL volume $(255 * 255 * 255)/6$ by hand.

We test our tool on these instances and check the ratio of the program results to the REAL volumes. The experimental results are given in Fig. 1 and 2 respectively in detail.

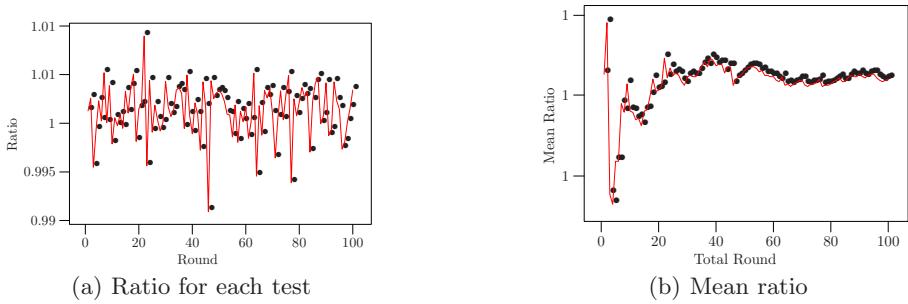


Fig. 1. Experimental results of *Example 1*

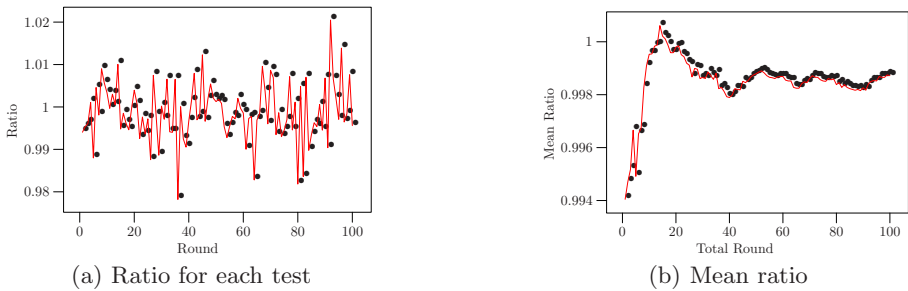


Fig. 2. Experimental results of *Example 2*

4.2 Variance Analysis

Experimental results from both Fig. 1 and Fig. 2 show that our method indeed has a good approximation. The mean values are very close to 1 and they converge well. However, there are still some small differences between Fig. 1(a) and Fig. 2(a). For example, all the ratio values in Fig. 1(a) fall within a small interval while those in Fig. 2(a) fall within a relatively large interval. As far as the variance is concerned, why does the data in Fig. 2(a) have a larger variance compared with those in Fig. 1(a)? To find out the possible reason, we experiment on *Example 2* again with fewer sampling points. Results are presented in Fig. 3. Comparing Fig. 3(a) with Fig. 2(a), we find that given a fixed volume, fewer sampling points result in relatively larger variance.

Theoretically speaking, our sampling algorithm needs sufficiently many independent uniformly distributed points, but in practice we can only generate a finite number of points. How large should this number be? It is a problem to be settled. If we use a fixed number for each dimension, then instances with smaller volume will have larger density compared with those with bigger volume. Given two polyhedra in the same dimension, the volume ratio, however, can be arbitrary large, which may result in sharp difference on density. On the other hand,

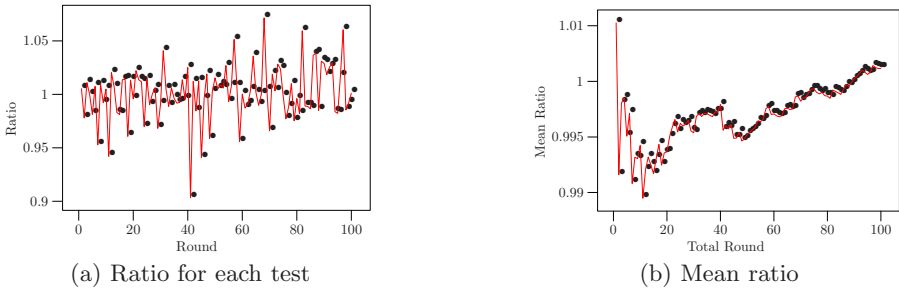


Fig. 3. Experimental results of *Example 2* (using fewer sampling points)

it is clear that this number should vary with the dimension. Maybe this number should be an exponential function on the dimension but we are not able to find a precise definition yet. In our current implementation, we use an adjustable parameter to denote the number of points before running the program. It is certainly better to find a self-adaptive heuristic, which can dynamically adjust the number during the running of the program. We leave this for our future work.

4.3 On the Mean Ratio

Although the variance varies with different sampling densities, all the mean ratios in Fig. 1(b), Fig. 2(b) and Fig. 3(b) show a very good approximation to 1. The curves reveal a nice trend of convergence to 1 as the total Round number grows from 1 to 100. This is easy to understand. Even if the ratio deviation of one particular case is 100%, it will only contribute 1% up to the total deviation of the mean value because it is divided equally among all the 100 sampling Rounds. So in general it makes sense to run more Rounds.

At last, we want to emphasize that the sampling algorithm is a general method. Although there are some negative results of it when the dimension

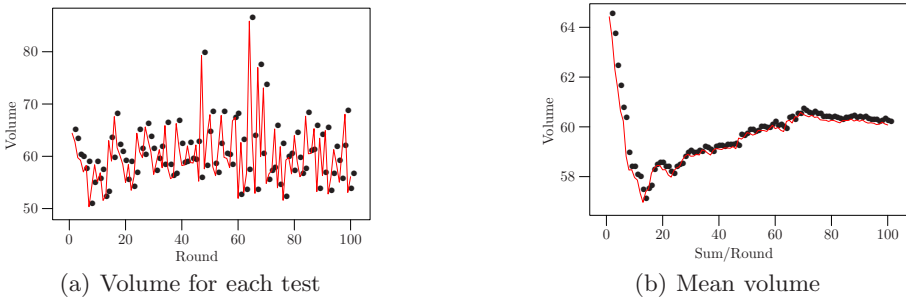


Fig. 4. Experimental results

n grows to infinity [14], our experiments show that the method is still feasible for real-life instances in high dimension. However, as for high dimensional and general non-convex instances, although the method can compute/approximate volume, we cannot always carry out comparisons as we do above, because we have no other means to obtain the REAL volume for general instances. Despite that, for some two-dimensional non-convex cases (whose REAL volume can be computed analytically) our method can obtain a mean ratio which always converges to one. For more complex non-convex cases, we believe that our algorithm also has good approximations in practice. For instance, given an in-

stance $\begin{cases} y \geq x^2 - 2 \\ y \leq \frac{1}{2}x^2 \\ -10 \leq z \leq 5 \\ xyz \leq 1 \end{cases}$, we do not know the REAL volume, but our tool can tell

us that it is about 60, as depicted in Fig. 4.5

5 Concluding Remarks

Volume computation is a very hard but widely studied problem in mathematics and computer science. Generally speaking, there are two different methods on this problem: the exact method and the randomized approximation method. The exact method itself can be classified into two classes: triangulation methods and signed decomposition methods. Benno Büeler, Andreas Enge and Komei Fukuda present some practical study on these methods [3]. But these methods are only applicable to convex bodies with linear constraints. The randomized approximation method is a more general one. It can cope with almost any constraints presented to it. Although the randomized approximation method is relatively new, a lot of progress has been made since its birth. Our work is also based on the randomized method.

However, most of these efforts on randomized algorithms are on the complexity aspect and little practical studies are given. In this paper some implementation issues of volume computation are studied. Some techniques in randomized volume computation algorithms are quantitatively evaluated. For example, our algorithm is also based on Monte Carlo sampling, but we do not use the Multi-phase method as described above. Instead, we use only one phase Monte Carlo method but we run the probing process many times to obtain a more accurate average result. On generating uniformly distributed points, we do not use the Markov Chain method although it does well in simulating the uniformly distribution in theory. Instead, we generate random points directly within the polyhedron and view them as uniformly distributed ones in our algorithm. Techniques and problems on efficient and effective implementation to achieve good performance are also discussed. Preliminary empirical results show that the tool developed by utilizing these results works very well. Of course, there are still some unsolved problems related to some of the manually adjustable parameters. We leave them for future research.

⁵ See Appendix A for more examples.

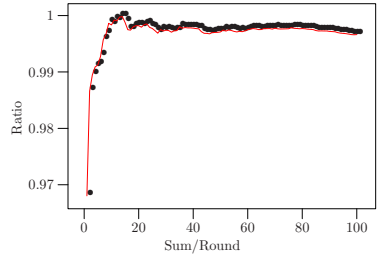
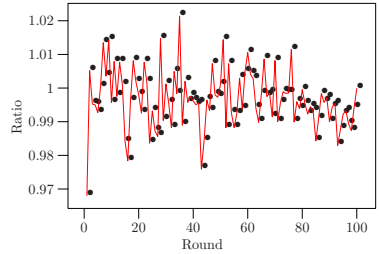
References

1. David Applegate and Ravi Kannan. Sampling and integration of near log-concave functions. In: Proc. 23rd annual ACM symp. on Theory of Computing (STOC), pp. 156–163 (1991)
2. Bollobás, B.: Volume estimates and rapid mixing. *Flavors of geometry*. Math. Sci. Res. Inst. Publ. 31, 151–182 (1997)
3. Büeler, B., Enge, A., Fukuda, K.: Exact volume computation for polytopes: a practical study. *Polytopes–combinatorics and computation* (1998)
4. Dyer, M., Frieze, A.: On the complexity of computing the volume of a polyhedron. *SIAM J. Comput.* 17(5), 967–974 (1988)
5. Martin Dyer and Alan Frieze. Computing the volume of convex bodies: A case where randomness provably helps. In: Proc. 44th Symp. in Applied Mathematics (PSAM) (1991)
6. Dyer, M., Frieze, A., Kannan, R.: A random polynomial-time algorithm for approximating the volume of convex bodies. *J. ACM* 38(1), 1–17 (1991)
7. Gritzmann, P., Klee, V.: On the complexity of some basic problems in computational convexity: II. volume and mixed volumes. *Polytopes: abstract, convex and computational* (Scarborough, ON, 1993), NATO Adv. Sci. Inst. Ser. C Math. Phys. Sci., pp. 373–466 (1994)
8. Kannan, R., Lovász, L., Simonovits, M.: Random walks and an $O^*(n^5)$ volume algorithm for convex bodies. *Random Struct. Algorithms* 11(1), 1–50 (1997)
9. Lovász, L.: How to compute the volume? *Jber. d. Dt. Math.-Verein, Jubiläumstagung*, B. G. Teubner, Stuttgart, pp. 138–151 (1990)
10. Lovász, L., Simonovits, M.: The mixing rate of markov chains, an isoperimetric inequality, and computing the volume. In: Proc. 31th IEEE Annual Symp. on Found. of Comp. Sci (FOCS), pp. 482–491 (1990)
11. Lovász, L., Simonovits, M.: Random walks in a convex body and an improved volume algorithm. *Random Struct. Algorithms* 4(4), 359–412 (1993)
12. Lovász, L., Vempala, S.: Simulated annealing in convex bodies and an $O^*(n^4)$ volume algorithm. In: Lovász, L. (ed.) Proc. 44th IEEE Annual Symp. on Found. of Comp. Sci (FOCS), pp. 650–659 (2003)
13. Rademacher, L., Vempala, S.: Dispersion of mass and the complexity of randomized geometric algorithms. In: Proc. 47th IEEE Annual Symp. on Found. of Comp. Sci (FOCS), pp. 729–738 (2006)
14. Simonovits, M.: How to compute the volume in high dimension? *Mathematical Programming* 97, 337–374 (2003)
15. Weisstein, E.: Ball. From MathWorld – A Wolfram Web Resource (2003), available at <http://mathworld.wolfram.com/Ball.html>

A More Examples

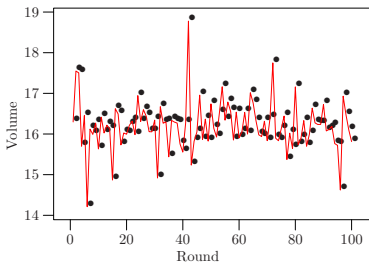
Example (a) Given a 4-dimensional polyhedron below, its REAL volume is about $2/3$. Our tool can also give the results depicted on the right of the below in-

$$\text{stance.} \left\{ \begin{array}{l} -x + y + z - u \leq 1 \\ -x + y + z + u \leq 1 \\ -x + y - z - u \leq 1 \\ -x + y - z + u \leq 1 \\ -x - y - z - u \leq 1 \\ -x - y - z + u \leq 1 \\ -x - y + z + u \leq 1 \\ -x - y + z - u \leq 1 \\ x - y - z + u \leq 1 \\ x - y - z - u \leq 1 \\ x - y + z + u \leq 1 \\ x - y + z - u \leq 1 \\ x + y - z + u \leq 1 \\ x + y - z - u \leq 1 \\ x + y + z + u \leq 1 \\ x + y + z - u \leq 1 \end{array} \right.$$

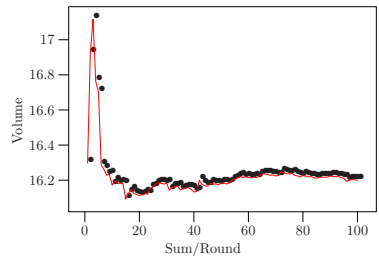


Example (b) Given a 3-dimensional body $\left\{ \begin{array}{l} 4x^2 + 2y^2 + z^2 \leq 8 \\ x^2 + 4y^2 + 2z^2 \leq 8 \\ 2x^2 + y^2 + 4z^2 \leq 8 \\ xy \leq 1 \end{array} \right.$, it is in fact

the intersection of three ellipsoids and another instance denoted by $\{xy \leq 1\}$. We do not know its REAL volume, but our tool can tell us the approximate volume (about 16.2) as depicted below.



(a) Volume for each test



(b) Mean volume

Fig. 5. Experimental results

Improved Throughput Bounds for Interference-Aware Routing in Wireless Networks*

Chiranjeeb Buragohain¹, Subhash Suri², Csaba D. Tóth³, and Yunhong Zhou⁴

¹ Amazon.com Inc., Seattle, WA 98104
chiran@amazon.com

² Department of Comp. Sci., UCSB, Santa Barbara, CA 93106
suri@cs.ucsb.edu

³ Department of Mathematics, MIT, Cambridge, MA 02139
toth@math.mit.edu

⁴ HP Labs, 1501 Page Mill Road, Palo Alto, CA 94304
yunhong.zhou@hp.com

1 Introduction

Interference is a fundamental limiting factor in wireless networks. Due to interaction among transmissions of neighboring nodes and need for multi-hop routing in large networks, it is a non-trivial problem to estimate how much throughput a network can deliver. In an important piece of work, Gupta and Kumar [3] showed that in a random model, where n identical nodes are distributed uniformly in a unit square and each node is communicating with a random destination, the capacity of the network as measured in bit-meters/sec is $O(\sqrt{n})$. This result articulates the packing constraint of the n paths: on average each path is $\Theta(\sqrt{n})$ hops long, and thus in the space of size $O(n)$, only $O(\sqrt{n})$ paths can be accommodated.

The Gupta-Kumar result is quite elegant, but its relevance to practical networks can be questioned because of the *random* source-destination ($s-t$) pairs assumption. As Li et al. [10] point out, such an assumption may hold in small networks, but as the network scales, it is unlikely that communications patterns will exhibit uniform randomness. Instead, the more relevant question is: given a particular network instance and a set of $s-t$ pairs, what is the maximum throughput that this network can deliver? Motivated by this question, Jain et al. [4], Alicherry et al. [1], and Kumar et al. [8] have investigated the capacity of wireless networks for arbitrary source-destination pairs, and arbitrary networks. All these papers model the problem as a linear program (LP), and provide a computational scheme for estimating the throughput. This is indeed an important direction and, as one of our main results, we show that a novel *node-based* LP formulation combined with a *node ordering* technique yields a $1/3$ approximation of the optimal throughput, which improves the previous best lower bound of $1/8$. But we first begin with a natural fundamental question.

Is there a generalization of the Gupta-Kumar result for arbitrary networks and arbitrary sets of $s-t$ pairs? In other words, can one estimate the network capacity in broad

* The research of Chiranjeeb Buragohain and Subhash Suri was supported in part by the National Science Foundation under grants CNS-0626954 and CCF-0514738.

terms, without resorting to computational techniques? And how widely does the capacity vary for different choices of $s-t$ pairs in the network? Recall that in the random model of Gupta and Kumar, the gap between the best-case and worst-case bounds is only a constant factor.

Of course, it is easy to observe that without some additional parameters this question is not particularly meaningful. Because we measure throughput in the number of bits transmitted (and not bit-meters as Gupta and Kumar), the capacity can vary widely depending on how far apart the sources and destinations are. If each source is adjacent to its destination, then we can achieve a throughput of $\Theta(n)$; if source-destination pairs are $\Theta(n)$ distance apart (as in a path graph), then the throughput drops to $O(1)$. Thus, a natural and important parameter is the *distance* between the source and destination nodes. However, even if two input instances have roughly equal average distance between $s-t$ pairs, their throughputs can vary widely (for instance, if a small cut separates all source-destination pairs). We show, however, that there is an intermediate ground of *structured network* and *arbitrary $s-t$ pairs*, where such a characterization is possible. The special structure we consider is a *grid network*, which is a rather natural topology.

Our Contributions

1. Suppose we have n *arbitrarily paired $s-t$ pairs* in an $\Theta(\sqrt{n}) \times \Theta(\sqrt{n})$ size grid network. We show that if the average (hop) distance among the $s-t$ pairs is d , then it is always possible to achieve a total throughput of $\Omega(n/d)$. There are instances where this bound is tight. The upper bound on the throughput follows easily from a packing argument; our contribution is to show that $\Omega(n/d)$ throughput is *always* achievable.
2. The $\Omega(n/d)$ throughput in a grid network can be achieved by a simple routing scheme that routes each flow along a *single* path. Both the routing and the scheduling algorithms are simple, deterministic, and distributed. Thus, for the grid topology networks, one can achieve (asymptotic) worst-case optimal throughput *without resorting to* computationally expensive LP type methods.
3. Our third result concerns an approximation bound for the throughput in a general network: arbitrary network topology and arbitrary $s-t$ pairs. In contrast to previous work [4,18], we introduce two novel ideas: improved interference constraints at the *node* level, and improving the approximation ratio by imposing an *ordering* on the nodes. As a result of these two ideas, we achieve an approximation ratio of 3 for the optimal throughput, improving all previous bounds.
4. An interesting corollary of our LP formulation is that it yields *provably* optimal throughput if the network topology has a special structure, such as a tree. Tree-like topologies may be quite natural in some wireless mesh networks, especially at the peripheries.
5. We show through experimentation that our node-based LP routing delivers excellent performance. In most cases, it achieves twice the throughput of the edge-based LP, and is typically within 10% of the optimal.
6. All LP based techniques split flows across multiple paths, and an obvious question is to bound the integrality gap between the optimal multi-path and single-path

routes. Simulations studies [8] suggest that, for random inputs, natural heuristics based on the classical shortest path schemes can give acceptable results. In the full version of this paper, we show that three straightforward routing schemes can have arbitrarily small throughput. On the other hand, in the special case of grid networks, we show that one can efficiently compute a single path route whose end to end throughput is within a constant factor of the optimal single path throughput.

2 Preliminaries and Related Work

We assume a standard graph model of wireless networks. The network connectivity is described by an undirected graph $G = (V, E)$, where V denotes the set of ad-hoc wireless nodes, and E denotes the set of node-pairs that are neighbors. The communication radius of every radio node $i \in \{1, 2, \dots, n\}$ is R ; throughout the paper, we assume that the communication occurs on a single radio channel, although the extension to multiple channels is straightforward. Each communicating node causes interferences at all other nodes within distance ϱ from it, where $\varrho \geq R$, is called the *interference radius* of the node. Note that we assume that all radios have an identical communication radius R , and an identical interference radius ϱ . In order to simplify the discussion, we assume that $\varrho = R$, but all our arguments can be easily extended to the general case of $\varrho > R$.

A problem instance is a network $G = (V, E)$, and a set of k source-target pairs (s_j, t_j) , $j = 1, 2, \dots, k$, where s_j and t_j are nodes of V . We assume that each source s_j wants to transmit to its target t_j at a normalized rate of 1. For simplicity, we also assume that the channel capacity is also 1; again, these are easily generalized to different rates. Our problem is to maximize the network *throughput*, which is the total amount of traffic that can be scheduled among all the s - t pairs subject to the capacity and interference constraints.

Models of Interference. The wireless network uses a broadcast medium, which means that when one node transmits, it causes interference at the neighboring nodes, preventing them from receiving (correct) signals from other nodes. The details of which nodes cause interference at which other nodes depend on the specifics of the MAC protocol being used. In this paper, we adopt the interference model corresponding to the IEEE 802.11-like MAC protocols, which require senders to send RTS control messages and receivers to send CTS and ACK messages. Currently, this is the most widely used MAC protocol in wireless networks. Under this protocol, two edges are said to *interfere* if either endpoint (node) of one is within the interference radius ϱ of a node of the other edge. In other words, the edges ij and kl interfere if $\max\{\text{dist}(i, k), \text{dist}(i, l), \text{dist}(j, k), \text{dist}(j, l)\} \leq \varrho$. It is clear that if a set of edges pairwise interfere with each other, then only one of those edges can be active at any point of time.

There are several other models of interference in the literature. The *protocol model* introduced by Gupta and Kumar [3] assumes that the transmission from node i is received correctly at j if no other node k is transmitting within interference range ϱ of j . This model corresponds to MAC protocols that *do not require an ACK from the receiver*. The throughput of a network can be higher under the protocol model because

it assumes a weaker interference condition than the 802.11-like protocols. The *transmitter model* introduced in Kumar et al. [8] assumes that two transmitting nodes are in conflict unless they are separated by *twice the interference range* (2ϱ). The interference condition assumed here is unnecessarily stronger than 802.11 MACs and leads to a lower estimate of throughput of the network. While we have chosen to work with the 802.11 model of interference, our methodology is quite general, and can be applied to these other models as well.

Related Work. Gupta and Kumar [3] provide (near) tight bounds on the throughput capacity of a *random network*, where the nodes are placed randomly in a square and sources and destinations are randomly paired. They show that the expected throughput available to each node in the network is $\Theta(1/\sqrt{n})$. Their result essentially articulates that interference leads to *geometric packing* constraint in the medium. In a follow up work, Li et al. [10] did simulations and experiments to measure the impact of interference in some realistic networks. They made the case that it might not be realistic to assume random s - t pairs. They argue that if s - t pairs are not too far from each other then the throughput improves; in fact, they observe that the throughput is bounded by $O(n/d)$ if the average s - t separation is d . They cannot tell, however, if this throughput bound can always be achieved. Kyasanur et al. [9] have recently extended the work of Gupta and Kumar [3] to study the dependence of total throughput on the number radio channels and interfaces on each network node.

While the results of Gupta-Kumar and Li et al. focused on random or grid-like networks, they did not address a very practical question: given a particular instance of a network and a set of s - t pairs, how much throughput is achievable? Jain et al. [4] formalized this problem, proved that it is NP-hard, and gave upper and lower bounds to estimate the optimal throughput. Their methods, however, do not translate to polynomial time approximation algorithms with any provable guarantees. Kodialam et al. [5] studied a variant of the throughput maximization problem for arbitrary networks, but they do not consider the effect of interference in detail. Recently Padhye et al. [11] have taken significant steps to measure interference between real radio links. Raniwala et al. [12] have designed and implemented a multichannel wireless network. Draves et al. [2] have proposed routing metrics to efficiently route in such networks. On the theoretical side, the problem of maximizing throughput in a network using multiple channels and interfaces have been studied by Alicherry et al. [1] and Kodialam et al. [6].

Kumar et al. [8,7] were the first to give a constant factor approximation algorithm for the throughput maximization problem in a network with a single radio channel. In particular, they give a 5-approximation algorithm for throughput, their algorithm assumes the *transmitter model*. As we mentioned earlier, the transmitter model is unduly restrictive compared to the 802.11-like models, and their algorithm does not give any explicit approximation bound for the 802.11 model. As mentioned above, Alicherry et al. [1] considered the problem of routing in the presence of interference with multiple radio channels and interfaces. As part of that work, they give an approximation algorithm for the throughput maximization problem with a constant factor guarantee under the 802.11-like model using interference constraints between edges. Their approximation factor is $1/8$ for the case of $\varrho = R$, and it becomes progressively worse as ϱ becomes larger compared to R . By contrast, our approximation factor is $1/3$, and does not depend on the ratio ϱ/R .

3 Maximum Throughput for Grid Topologies

Before we discuss our linear programming approach for computing interference-aware routes in arbitrary networks, it is worth asking to what extent one can estimate the throughput using *structural* facts, in the style of Gupta and Kumar [3]. In other words, are there simple characterizations of the network and the s - t distributions that allow us to derive good estimates of the achievable throughput *without* resorting to computationally expensive methods such as linear programming. We do not know of any result of this type for completely general setting (nor is one likely to exist), but we show below that for special network topologies, such as grids, one can obtain a bound on achievable throughput based on average separation among source-destination pairs. Furthermore, our investigation also leads to a simple and distributed routing scheme that achieves the optimal throughput using *single* paths.

Consider a grid network of size $\Theta(\sqrt{n}) \times \Theta(\sqrt{n})$, which can be thought of as a square lattice in the plane. We assume there are n source-destinations pairs, arbitrarily chosen by the user (or adversary). We assume that all sources and all destinations are distinct. We assume that $R = \varrho = 1$, each edge in the network has capacity 1, and each source wants to communicate with its destination at the rate of 1. We assume that these demands are persistent, i.e. the flow demands are constant over time and we are interested in the steady state flow. We wish to maximize the total throughput among all the s - t pairs. (For the moment, we do not worry about fairness among different pairs, but will briefly discuss that issue in Section 5.)

Manhattan Routing. We first consider the case when each s - t pair has (lattice) distance d . In the following subsection, we will generalize the result to average distances. A simple packing argument shows that the maximum possible throughput is at most $O(n/d)$; a similar observation was also made in Li et al. [10]. But it is far from obvious that $\Omega(n/d)$ throughput can *always* be realized (for adversarially chosen s - t pairs). By clustering sources on one side, and destinations on the other, it may be possible to create significant bottlenecks in routing.

In fact, one can see that a simple-minded routing scheme can lead to very low throughput. Consider, for instance, the particular choice of s - t pairs shown in Fig. 1. There are 4 source-destination pairs $\{(A, B), (C, D), (E, F), (G, H)\}$. Suppose we route each flow using the shortest paths, staying as close as possible to the straight line joining the s - t pair. These routes are shown using the dotted lines in the figure. Observe that all these paths go through a common node N , which becomes the bottleneck, and limits the total throughput to 1. Nevertheless, the following result shows that for any configuration of n source-destination pairs, one can achieve $\Theta(n/d)$ throughput.

Theorem 1. *Consider n source-destination pairs in an $\Theta(\sqrt{n}) \times \Theta(\sqrt{n})$ size grid, with all sources and all destinations distinct. Suppose that each s - t pair has (lattice) distance d . Then, one can always achieve a throughput of $\Omega(n/d)$, and this is also the best possible bound.*

Due to space limitations, the proof is given in the full version of this paper.

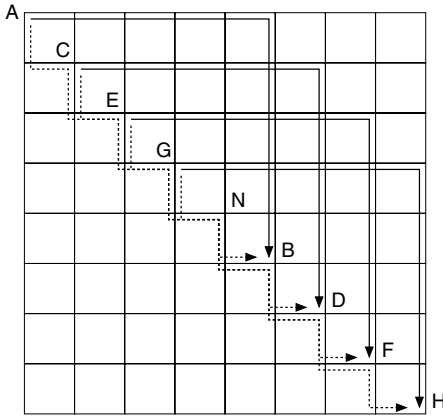


Fig. 1. Illustration of the Manhattan routing. The source destination pairs are (A,B), (C,D), (E,F) and (G,H).

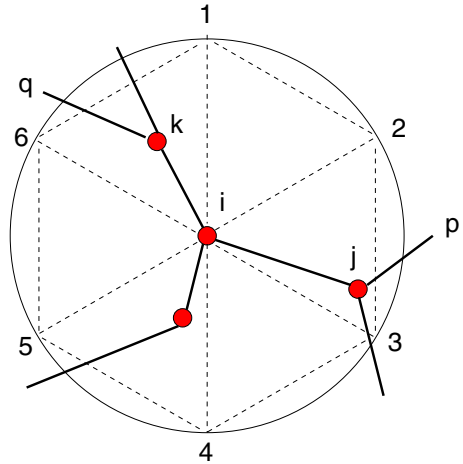


Fig. 2. Interference zone for a single node i

Extension to Average or Median Distances. The strict distance requirement for all $s-t$ pairs is clearly too restrictive. We now show that the result actually holds more broadly, for the case when d is either the *average* or the *median* distance among all pairs.

Theorem 2. Consider n source-destination pairs in an $\Theta(\sqrt{n}) \times \Theta(\sqrt{n})$ size grid, with all sources and destinations distinct. Suppose that the average (lattice) distance between the $s-t$ pairs is d . Then, one can always achieve a throughput of $\Omega(n/d)$.

Proof. We simply observe that if the n pairs have average distance d , then at least half the pairs must be at distance less than $2d$. We set the rate for all the pairs whose separation is larger than $2d$ to zero, and route the remaining pairs using Manhattan routing. By Theorem 1, the throughput of these routes is $\Omega(n/d)$.

A very similar argument shows that a throughput of $\Omega(n/d)$ is also achievable when the *median* $s-t$ distance is d .

These bounds characterize the throughput of an instance based on just one key parameter: the separation among the source and destination pairs. Given an instance of the problem, a network manager can now deduce the asymptotic worst-case optimal throughput of the network simply from the distances among the source-destination pairs. From a network manager’s perspective, this result is an encouraging one: while the traffic matrix of a network is beyond control, the network topology is something she can control. Thus, our result suggests that in sufficiently regular network topologies, one can consistently achieve high throughput *and* do so through single path routing.

4 Throughput in Arbitrary Topologies

In this section, we consider the general problem of estimating the throughput for a given (arbitrary) network with arbitrary s - t pairs (namely, the problem defined in Section 2).

A Linear Programming Approach. The throughput maximization problem is a joint routing and scheduling problem: we need to route each flow and schedule the links so that flows can be feasibly accommodated subject to the interference constraints. In an actual wireless network, the scheduling is taken care of by the MAC layer —thus for this discussion we shall assume that there is a perfect underlying MAC layer which can schedule a solution as long as the solution respects all the flow and interference constraints. In a real network, MAC layers are never perfect and hence our solution provides an upper bound on feasible flows.

We formulate the flow problem as a linear program and then add constraints to model the interference restrictions. The throughput maximization problem with only flow constraints is just the classical max-flow problem:

$$\begin{aligned} \text{Maximize} \quad & \sum_{i \in N(s)} f_{si} \quad \text{subject to} \\ & \sum_{j \in N(i)} f_{ij} = \sum_{j \in N(i)} f_{ji}, \quad \forall i \neq s, t \\ & 0 \leq f_{ij} \leq 1, \quad \forall ij \in E \end{aligned} \tag{1}$$

where $N(i)$ denotes the set of nodes adjacent to i , and f_{ij} denotes the amount of flow in edge ij from node i to node j , for each edge $ij \in E$. The objective function maximizes the total flow out of s subject to the capacity constraint on each edge; the other constraint imposes the flow conservation condition at each intermediate node. In order to simplify the discussion, we have assumed that there is only one source-destination pair (s, t) . The extension to multiple pairs is straightforward: in each term, we sum over all flows instead of just one.

We now describe how to supplement this standard multicommodity flow problem with interference constraints. The key difficulty in designing an approximation algorithm for the throughput maximization problem lies in resolving conflicts between neighbors who interfere with each other. Jain et al. [4] model this constraint using an *independent set* framework, which attempts to resolve the conflicts globally. Unfortunately, finding an independent set is NP-complete, and so it does not lead to a polynomial time approximation.

Our approach is to resolve the conflicts locally, and model the problem as a *geometric packing* problem. For example, consider an edge ij and all the edges that interfere with it. When the edge ij is active, none of the edges with which it interferes can be active. Because each edge carves out a portion of the space (its interference zone) while it is active, one can use packing arguments to derive upper and lower bounds on the throughput. Indeed, similar ideas are used in [1], where the packing constraints are formulated in the space around each edge. Unfortunately, the constant factor in their

approximation is rather large, and it also depends on the ratio between the interference and the radio ranges ϱ/R .¹

Instead of modeling the interference around edges, as has been done by others, we introduce two new ideas that lead to improved algorithms and approximation bounds. We model the interference around nodes, and introduce an ordering over nodes. These two ideas allow us to guarantee an approximation ratio of 3, which is independent of ϱ .

Modeling Interference Constraints at Nodes. Let us assume that the flow of data through the network is like fluid which is infinitely divisible. Then in a steady state, suppose an edge ij supports the flow $f_{ij} \leq 1$ (recall that each edge has unit capacity). This means that given a *unit time interval*, the edge ij is required to be active for a fraction of time f_{ij} and remains inactive for the rest of the time. Towards this end, we introduce two sets of variables τ_i and τ_{ij} as follows.

$$\begin{aligned}\tau_{ij} &= f_{ij} + f_{ji} \leq 1, \\ \tau_i &= \sum_{j \in N(i)} \tau_{ij} \leq 1, \quad \forall i \in V.\end{aligned}\tag{2}$$

Here τ_{ij} represents the *total* fraction of the unit time interval that an edge ij is active and similarly τ_i is the fraction of time for the node i . Using these variables, we now introduce the *node interference* constraint which enforces the interference restrictions. Consider the node i shown in Fig. 2 and the set of its neighbors (within interference range) denoted by $N(i)$. It is clear that while any node j in the set $N(i) \cup \{i\}$ is transmitting, all other nodes in this set must be inactive unless there is a single node that is communicating with j . This leads us to the following constraint:

$$\sum_{j \in N(i) \cup \{i\}} \tau_j - \sum_{j, k \in N(i) \cup \{i\}, jk \in E} \tau_{jk} \leq 1, \quad \forall i \in V,\tag{3}$$

where E denotes the edges of the interference graph. To understand this inequality, let us consider the unit time interval and in that time interval, which nodes can be active for how long. The first term in LHS, counts the total amount of time (out of the unit time interval) that nodes are active in the neighborhood of i . The second term accounts for the fact that if two nodes j and k in the neighborhood of i are communicating with each other, the time they spend communicating to each other is counted only once.

By construction, if the nodes satisfy condition (3), then the flow is definitely free of interference. But condition (3) is actually more restrictive than necessary. For instance, consider the nodes j and k in Fig. 2 which are separated by a distance larger than the radio range. Constraint (3) implies that the edges jp and kq cannot be active at the same time, while in reality they can. Eliminating such unnecessary constraints is key to our improved analysis, and so we next introduce the idea of node ordering.

¹ Lemma 1 of [11] proves an approximation bound of 8 for $\varrho = 2$. They also claim an approximation bound of 4 for $\varrho = 1$, which appears to be wrong, and should be 8. Also, the approximation factor grows as the ratio ϱ/R grows. For instance, the factor is 12 for $\varrho/R = 2.5$.

Node Ordering. Consider a total order on the nodes. (We will prescribe a specific order shortly.) Observe that the interference relation is symmetric. If nodes i and j interfere with each other, then constraint (3) imposes the interference condition twice: once when we consider the neighborhood of i and once for j . Therefore, if i precedes j in the ordering, then it is enough to only consider the constraint introduced by i on j . Specifically, let $N_L(i)$ denote the set of interfering nodes *preceding* node i in the ordering, then the following relaxed constraint still ensures an interference-free schedule.

$$\sum_{j \in N_L(i) \cup \{i\}} \tau_j - \sum_{j, k \in N_L(i) \cup \{i\}, jk \in E} \tau_{jk} \leq 1, \quad \forall i \in V. \tag{4}$$

In order to define $N_L(i)$, any arbitrary ordering over the nodes will work. To get a good approximation factor, we specify the following *lexicographical* order on the nodes: i *precedes* j if and only if, denoting the coordinates of the points by $i = (x_i, y_i)$ and $j = (x_j, y_j)$, we have either $x_i < x_j$ or $x_i = x_j$ and $y_i < y_j$.

LP-NODE. We are now ready to describe the complete linear program, LP-NODE.

$$\begin{aligned} \text{Maximize} \quad & \sum_{i \in N(s)} f_{si} \text{ subject to} \\ & \sum_{j \in N(i)} f_{ij} = \sum_{j \in N(i)} f_{ji}, \quad \forall i \neq s, t \\ & 0 \leq f_{ij} \leq 1, \quad \forall ij \in E \\ & \tau_{ij} = f_{ij} + f_{ji} \leq 1, \\ & \tau_i = \sum_{j \in N(i)} \tau_{ij} \leq 1, \quad \forall i \in V, \\ & \sum_{j \in N_L(i) \cup \{i\}} \tau_j - \sum_{j, k \in N_L(i) \cup \{i\}, jk \in E} \tau_{jk} \leq 1, \quad \forall i \in V. \end{aligned} \tag{5}$$

By construction, the solution to LP-NODE leads to a feasible flow. This flow can be scheduled. One can show (the proof is available in the full paper) that f_{NODE} gives a factor 3 approximation to f_{OPT} .

Theorem 3. *The flow produced by the solution of LP-NODE satisfies*

$$f_{\text{NODE}} \leq f_{\text{OPT}} \leq 3f_{\text{NODE}}.$$

Our technique can easily be extended to the case that the interference range ρ is larger than radio range $R = 1$. Consider any $\rho > 1$. The last constraint in LP-NODE will now include all nodes which are within interference range of i . We can see from Fig. 2 that within a semicircle of radius ρ , we can still pack at most 3 nodes which do not interfere with each other and hence the approximation bound given above, holds for *any* $\rho > R$. By contrast, the approximation ratio given by Alicherry et al. [1] grows monotonically with increasing ρ ; it is 8 when $\rho = 2R$, 12 when $\rho = 2.5R$, and so on.

Optimal Throughput for Tree-Structured Networks. If the underlying network is a tree, then we can show (the proof is available in the full paper) that a variation of our LP-NODE can solve the throughput maximization problem optimally.

Theorem 4. *If the network connectivity graph is a tree, then we can solve the throughput maximization problem optimally using a variant of LP-NODE.*

5 Experimental Results

We ran experiments on both the regular as well as random networks. The random networks consist of n nodes spread over a square $\sqrt{n} \times \sqrt{n}$ area with radio range 3.0. Any two nodes which are within radio range can communicate. This radio range was chosen so that the network is almost always connected. We assume that we are using a bidirectional MAC protocol like 802.11 and the radio range as well as interference range are the same. We assume that each link can support 1 unit of throughput. In our evaluation, we used three algorithms:

- LP-NODE: This is our main linear program described in Section 4. This algorithm has provable worst-case approximation ratio of 3.
- LP-EDGE: This is the best previously known linear programming based scheme, as described in Alicherry et al. [1]. This algorithm has an approximation ratio of 8, under the condition that $\rho = R$.
- OPTIMAL: Since the throughput maximization problem is NP-Complete, there is no polynomial time scheme to compute the maximum throughput. We therefore use the independent set enumeration method as described by Jain et al. [4]. We enumerate larger and larger number of independent sets and estimate the throughput until adding more independent sets do not improve the throughput any more. At this point we declare convergence and use the final throughput as optimal.

Throughput Scaling With Network Size. In this experiment, we wanted to see how well LP-NODE’s performance scales with the network size. We used a random network topology where the nodes were distributed uniformly at random throughout a square area. The source and destination are located at diagonally opposite corners. We then increased the number of nodes in the network from 32 to 64 to 96. In each case, we also computed the optimal throughput f_{OPT} by running the OPTIMAL algorithm.

In Fig. 3, left, we plot the throughput of the OPTIMAL, LP-NODE and LP-EDGE algorithms. Our LP-NODE algorithm shows excellent performance and yields close to 90% of the optimal throughput. By contrast, LP-EDGE performs much worse and achieves only 50%-60% of the OPTIMAL. In fact, even with a single source-destination pair, LP-EDGE at times failed to achieve 1/3 of the optimal throughput, which one could have achieved by routing along a single path [10]! With a single $s-t$ pair, the maximum possible throughput using multipath routing is 5/6; by contrast, the maximum throughput using a single path is 1/3. In these cases, the constant factors in the approximation algorithms become crucially important, and the LP-NODE algorithm does well.

Throughput Scaling with Source-Destination Pairs. In this experiment, we fixed the network and increased the number of $s-t$ pairs in the network to evaluate the throughput that the various routing schemes achieve. We used a random network topology with 64 nodes and up to 16 source destination pairs organized in a crosshatch pattern. In Fig. 3, middle, we plot the total throughput using LP-EDGE, LP-NODE and OPTIMAL

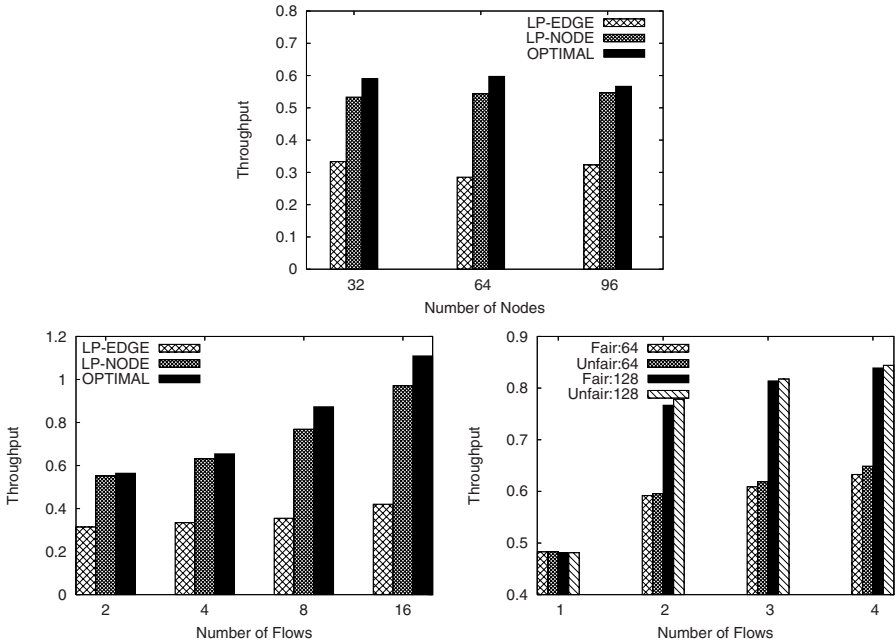


Fig. 3. Upper left: Performance of the LP-NODE, LP-EDGE and OPTIMAL algorithms compared for 32, 64 and 96 node networks. Upper right: The total throughput for different numbers of flows for a 64 node network. Down: Effect of fairness on total flow for 64 and 128 node networks. Note that the fairness constraint lowers total throughput by only a small amount.

algorithms for different number of source destination pairs. As expected we see that the throughput increases as the number of flows increases, but the dependence is not linear because interference from one set of paths reduces throughput for other pairs. Again, LP-NODE shows excellent performance, reaching near-optimal throughput in most cases, while LP-EDGE achieves less than half the throughput of LP-NODE.

Impact of Fairness on Flows. When multiple flows compete for bandwidth, the optimal flow is not necessarily fair. In practice though, fairness is an important criterion in any network protocol. To investigate the effect of fairness we again used the uniform random topology in a square area with four source destination pairs which intersect at the center of the square. For multiple flows we enforced the simplest fairness condition that each flow gets an equal amount of the total flow. We computed the total throughput using the LP-NODE algorithm and the results are shown in Fig. 3 right. As expected we see that enforcing fairness reduces total throughput, but surprisingly, *the effect is very mild*. In fact for the larger 128 node networks, the throughput for fair and unfair flows is almost identical. This is due to the fact that in larger networks the nodes have a lot of freedom in routing the flows and hence overall interference in any single node is low. Thus every pair can route equal amounts of flow without congestion.

References

1. Alicherry, M., Bhatia, R., Li, L.: Joint channel assignment and routing for throughput optimization in multiradio wireless mesh networks. In: Proc. Mobicom, ACM Press, New York (2005)
2. Draves, R.P., Padhye, J., Zill, B.: Routing in multi-radio, multi-hop wireless mesh network. In: Proc. MobiCom, ACM Press, New York (2004)
3. Gupta, P., Kumar, P.R.: The capacity of wireless networks. *IEEE Trans. Inf. Theory* 46, 388–404 (2000)
4. Jain, K., Padhye, J., Padmanabhan, V., Qiu, L.: Impact of interference on multi-hop wireless network performance. In: Proc. Mobicom, ACM Press, New York (2003)
5. Kodialam, M., Nandagopal, T.: Characterizing the achievable rates in multihop wireless networks. In: Proc. MobiCom, ACM Press, New York (2003)
6. Kodialam, M., Nandagopal, T.: Characterizing the capacity region in multi-radio multi-channel wireless mesh networks. In: Proc. MobiCom, ACM Press, New York (2005)
7. Kumar, V.S.A., Marathe, M.V., Parthasarathy, S., Srinivasan, A.: End-to-end packet-scheduling in wireless ad-hoc networks. In: Proc. SODA, pp. 1021–1030. ACM Press, New York (2004)
8. Kumar, V.S.A., Marathe, M.V., Parthasarathy, S., Srinivasan, A.: Algorithmic aspects of capacity in wireless networks. In: Proc. SIGMETRICS, ACM Press, New York (2005)
9. Kyasanur, P., Vaidya, N.H.: Capacity of multi-channel wireless networks: Impact of number of channels and interfaces. In: Proc. MobiCom, ACM Press, New York (2005)
10. Li, J., Blake, C., Couto, D.S.J.D., Lee, H.I., Morris, R.: Capacity of ad hoc wireless networks. In: Proc. MobiCom, ACM Press, New York (2001)
11. Padhye, J., Agarwal, S., Padmanabhan, V., Qiu, L., Rao, A., Zill, B.: Estimation of link interference in static multi-hop wireless networks. In: Proc. Internet Measurement Conf. ACM Press, New York (2005)
12. Raniwala, A., Chiueh, T.-C.: Architecture and algorithms for an IEEE 802.11-based multi-channel wireless mesh network. In: Proc. INFOCOM, IEEE (2005)

Generating Minimal k -Vertex Connected Spanning Subgraphs

Endre Boros¹, Konrad Borys¹, Khaled Elbassioni², Vladimir Gurvich¹,
Kazuhiro Makino³, and Gabor Rudolf¹

¹ RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway NJ 08854-8003
{boros,kborys,gurvich,grudolf}@rutcor.rutgers.edu

² Max-Planck-Institut für Informatik, Saarbrücken, Germany
elbassio@mpi-sb.mpg.de

³ Division of Mathematical Science for Social Systems, Graduate School of
Engineering Science, Osaka University, Toyonaka, Osaka, 560-8531, Japan
makino@sys.es.osaka-u.ac.jp

Abstract. We show that minimal k -vertex connected spanning subgraphs of a given graph can be generated in incremental polynomial time for any fixed k .

1 Introduction

Vertex and edge connectivity are two of the most fundamental concepts in network reliability theory. While in the simplest case only the connectedness of an undirected graph, that is, the presence of a spanning tree, is required, in practical applications higher levels of connectivity are often desirable. Given the possibility that the edges of the network can randomly fail the reliability of the network is defined as the probability that the operating edges provide a certain level of connectivity. Most methods computing network reliability depend on the efficient generation of all (or many) minimal subsets of network edges which guarantee the required connectivity [5,14].

In this paper we consider the problems of generating minimal k -vertex connected spanning subgraphs. An undirected graph G on at least $k + 1$ vertices is k -vertex connected if every subgraph of G obtained by removing at most $k - 1$ vertices is connected. A subgraph of a graph G is *spanning* if it has the same vertex set as G .

For a fixed integer k we define the problem of generating minimal k -vertex connected spanning subgraphs as follows:

Input: A k -vertex connected graph G

Output: The list of all minimal k -vertex connected spanning subgraphs of G

Note that the output of the above problem may consist of exponentially many subgraphs in terms of the input size. Thus, the efficiency of generation algorithms is measured customarily in both the input and output size (see e.g., [14,10,7]). An algorithm generating all elements of a family \mathcal{F} is said to run in *incremental polynomial time* if generating K elements of \mathcal{F} (or all if \mathcal{F} has less than K

elements) can be done in time polynomial in K and the size of the input, for an arbitrary integer K .

Our problems include as a special case the problem of generating spanning trees ($k = 1$), which can be solved efficiently [12,6,11,1]. The problem of generating 2-vertex connected subgraphs and its generalization for matroids has been considered in [8].

1.1 Main Results

We show that this generation problem can be solved in incremental polynomial time.

Theorem 1. *For every K we can generate K minimal k -vertex connected spanning subgraphs of a given graph in $O(K^3m^3n + K^2m^5n^4 + Kn^km^2)$ time, where $n = |V|$, $m = |E|$.*

We remark that the running time of our algorithm depends exponentially on k . The complexity of the above problem when k is also part of the input remains an open question.

1.2 The $X - e + Y$ Method

In this section we recall a technique from [9], which is a variant of the supergraph approach introduced by [13]. Let \mathcal{C} be a class of finite sets and for every $E \in \mathcal{C}$ let $\pi : 2^E \rightarrow \{0, 1\}$ be a monotone Boolean function, i.e., one for which $X \subseteq Y$ implies $\pi(X) \leq \pi(Y)$. We assume that $\pi(\emptyset) = 0$ and $\pi(E) = 1$. Let

$$\mathcal{F} = \{X \mid X \subseteq E \text{ is a minimal set satisfying } \pi(X) = 1\}.$$

Our goal is to generate all sets belonging to \mathcal{F} .

We remark that for every $X \subseteq E$ for which $\pi(X) = 1$ we can derive a subset $Y \subseteq X$ such that $Y \in \mathcal{F}$, by evaluating π exactly $|X|$ times. This can be accomplished by deleting one-by-one elements of X whose removal does not change the value of π . To formalize this, we can fix an arbitrary linear order \prec on elements of E , without any loss of generality, and define a mapping $Project : \{X \subseteq E \mid \pi(X) = 1\} \rightarrow \mathcal{F}$ by $Project(X) = X \setminus Z$, where Z is the lexicographically first subset of X , with respect to \prec , such that $\pi(X \setminus Z) = 1$ and $\pi(X \setminus (Z \cup e)) = 0$ for every $e \in X \setminus Z$. Clearly, by trying to delete elements of X in their \prec -order, we can compute $Project(X)$, as we remarked above, by evaluating π exactly $|X|$ times.

We next introduce a directed graph $\mathcal{G} = (\mathcal{F}, \mathcal{E})$ on vertex set \mathcal{F} . We define the neighborhood $N(X)$ of a vertex $X \in \mathcal{F}$ as follows $N(X) = \{Project((X \setminus e) \cup Y) \mid e \in X, Y \in \mathcal{Y}_{X,e}\}$, where $\mathcal{Y}_{X,e}$ is defined by $\mathcal{Y}_{X,e} = \{Y \mid Y \text{ is a minimal subset of } E \setminus X \text{ satisfying } \pi((X \setminus e) \cup Y) = 1\}$.

In other words, for every set $X \in \mathcal{F}$ and for every element $e \in X$ we extend $X \setminus e$ in all possible minimal ways to a set $X' = (X \setminus e) \cup Y$ for which $\pi(X') = 1$ (since $X \in \mathcal{F}$, we have $\pi(X \setminus e) = 0$), and introduce each time a directed arc from X to $Project(X')$. We call the obtained directed graph \mathcal{G} the *supergraph* of our generation problem.

Proposition 1 ([9]). *The supergraph $\mathcal{G} = (\mathcal{F}, \mathcal{E})$ is strongly connected.* □

Since \mathcal{G} is strongly connected by performing a breadth-first search in \mathcal{G} we can generate all elements of \mathcal{F} . Thus, given two procedures:

- $First(X, e)$, which for every $X \in \mathcal{F}$ and $e \in X$ returns an element of $\mathcal{Y}_{X,e}$ if $\mathcal{Y}_{X,e} \neq \emptyset$ and \emptyset otherwise,
- $Next(\mathcal{Y}, X, e)$, which return an element of $\mathcal{Y}_{X,e} \setminus \mathcal{Y}$ if $\mathcal{Y}_{X,e} \neq \mathcal{Y}$ and \emptyset otherwise,

the procedure $Transversal(\mathcal{G})$, defined below, generates all elements of \mathcal{F} .

```

Traversal( $\mathcal{G}$ )
Find an initial vertex  $X^0 \leftarrow Project(E)$ , initialize a queue  $\mathcal{Q} = \emptyset$  and a
dictionary of output vertices  $\mathcal{D} = \emptyset$ .
Perform a breadth-first search of  $\mathcal{G}$  starting from  $X^0$ :
1 output  $X^0$  and insert it to  $\mathcal{Q}$  and to  $\mathcal{D}$ 
2 while  $\mathcal{Q} \neq \emptyset$  do
3   take the first vertex  $X$  out of the queue  $\mathcal{Q}$ 
4   for every  $e \in X$  do
5      $\mathcal{Y} \leftarrow \emptyset, Y \leftarrow First(X, e)$ 
6     while  $Y \neq \emptyset$  do
7       compute the neighbor  $X' \leftarrow Project((X \setminus e) \cup Y)$ 
8       if  $X' \notin \mathcal{D}$  then output  $X'$  and insert it to  $\mathcal{Q}$  and to  $\mathcal{D}$ 
9       add  $Y$  to  $\mathcal{Y}, Y \leftarrow Next(\mathcal{Y}, X, e)$ 

```

Proposition 2. *Assume that the procedure $First(X, e)$ works in time $O(\phi_1(E))$, the for every K procedure $Next(\mathcal{Y}, X, e)$ outputs K elements of $\mathcal{Y}_{X,e}$ in time $\phi_2(K, E)$ and there is an algorithm evaluating π in time $O(\gamma(E))$. Then $Traversal(\mathcal{G})$ outputs K elements of \mathcal{F} in time $O(K^2|E|^2\gamma(E)+K^2\log(K)|E|^2+K|E|\phi_2(K, E) + K|E|\phi_1(E))$.*

2 Proof of Theorem 1

In this section we apply the $X - e + Y$ method to the generation of all minimal k -vertex connected spanning subgraphs.

For a given k -vertex connected graph (V, E) we define a Boolean function π as follows: for a subset $X \subseteq E$ let

$$\pi(X) = \begin{cases} 1, & \text{if } (V, X) \text{ is } k\text{-vertex connected;} \\ 0, & \text{otherwise.} \end{cases}$$

Clearly π is monotone, $\pi(\emptyset) = 0, \pi(E) = 1$. Then $\mathcal{F} = \{X \mid X \subseteq E \text{ is a minimal set satisfying } \pi(X) = 1\}$ is the family of edge sets of all minimal k -vertex connected spanning subgraphs of (V, E) .

2.1 $(k - 1)$ -Separators of $(V, X \setminus e)$

Before describing procedures $First(X, e)$ and $Next(\mathcal{Y}, X, e)$ we need the additional notions and elementary results.

A k -separator of a graph is a set of k vertices whose removal (simultaneously removing all edges adjacent to those vertices) makes the graph no longer connected. Note that a k -vertex connected graph has no k' -separators for $k' < k$.

Let $G = (V, X)$ be a minimal k -vertex connected spanning subgraph of a k -vertex connected graph (V, E) (see Figure 1).

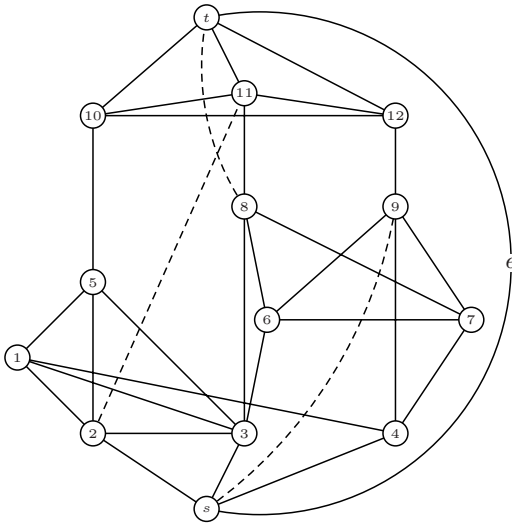


Fig. 1. 4-vertex connected graph (V, E) and its minimal 4-vertex connected subgraph $G = (V, X)$. Solid lines are edges in X .

Let $e = st$ be an arbitrary edge of G and let W be a $(k - 1)$ -separator of $G_e = (V, X \setminus e)$. Note that W contains neither s nor t , since otherwise W would also be a $(k - 1)$ -separator of G . We denote by S_W and T_W the vertex sets of the components (i.e., maximal connected subgraphs) of $G_e[V \setminus W]$ containing s and t , respectively.

Claim. $G_e[V \setminus W]$ consists of two components, $G_e[S_W]$ and $G_e[T_W]$ (see Figure 2).

We denote by $N(\cdot)$ a neighborhood in the graph G_e . Let \mathcal{W} be the set of all $(k - 1)$ -separators of $G_e = (V, X \setminus e)$ and let $\mathcal{S} = \{S \subseteq V \mid |N(S)| = k - 1, s \in S, t \notin S \cup N(S)\}$. We call an element of \mathcal{S} a $(k - 1)$ -source. Note that the mapping $W \mapsto S_W$ is a bijection between \mathcal{W} and \mathcal{S} whose inverse is $S \mapsto N(S)$.

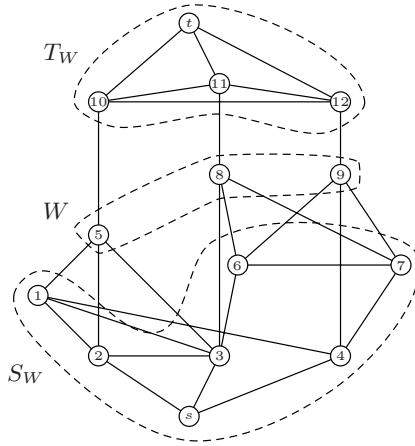


Fig. 2. 3-separator $W = \{5, 8, 9\}$ and the corresponding 3-source $S_W = \{s, 1, 2, 3, 4, 6, 7\}$

For two vertices $u, v \in V$ let $D_{u,v} = \{S_W \in \mathcal{S} \mid u \in S_W, v \in T_W\}$. For an edge $f = uv$ let $D_f = D_{u,v} \cup D_{v,u}$

We call a set of hyperedges whose union contains every vertex a *hyperedge cover*. We show that elements of $\mathcal{Y}_{X,e}$ are in one to one correspondence with the minimal hyperedge covers of $\mathcal{H}_{X,e}$.

Claim. Let $Y \subseteq E \setminus X$. The graph $(V, X \setminus e \cup Y)$ is k -vertex connected if and only if $\bigcup_{f \in Y} D_f = \mathcal{S}$.

2.2 Procedures $First(X, e)$ and $Next(\mathcal{Y}, X, e)$

We describe $First(X, e)$ and $Next(\mathcal{Y}, X, e)$, procedures generating all elements of $\mathcal{Y}_{X,e}$.

First(X, e)

- 1 construct a hypergraph $\mathcal{H}_{X,e}$ on vertex set \mathcal{S} with edge set $\mathcal{E} = \{D_f \mid f \in E \setminus X\}$
- 2 find a minimal hyperedge cover \mathcal{C} of $\mathcal{H}_{X,e}$
- 3 return a set $\{f \mid D_f \in \mathcal{C}\}$

Next(\mathcal{Y}, X, e)

- 1 find a a minimal hyperedge cover \mathcal{C} of $\mathcal{H}_{X,e}$ not in $\{D_f \mid f \in Y, Y \in \mathcal{Y}\}$
- 2 return a set $\{f \mid D_f \in \mathcal{C}\}$

In the remainder of this section we show that we can generate minimal hyperedge covers of $\mathcal{H}_{X,e}$ efficiently.

2.3 Structure of $(k - 1)$ -Separators

Consider the poset $L = (\mathcal{S}, \subseteq)$ of the $(k - 1)$ -sources ordered by inclusion.

Proposition 3. *The poset L with operations \cap and \cup is a lattice.*

We show that the ordering of $(k - 1)$ -sources in L has a natural interpretation for the corresponding $(k - 1)$ -separators.

Since the graph G_e is $(k - 1)$ -vertex connected, by Menger’s Theorem it contains $k - 1$ internally vertex disjoint s - t paths. Let $P_1 = sv_1^1 \dots v_{l_1}^1 t$, $P_2 = sv_1^2 \dots v_{l_2}^2 t$, \dots , $P_{k-1} = sv_1^{k-1} \dots v_{l_{k-1}}^{k-1} t$ denote such a collection of paths (see Figure 3). We denote by V_P the set of all vertices belonging to the paths P_1, \dots, P_{k-1} . Note that not all vertices in V necessarily belong to V_P .

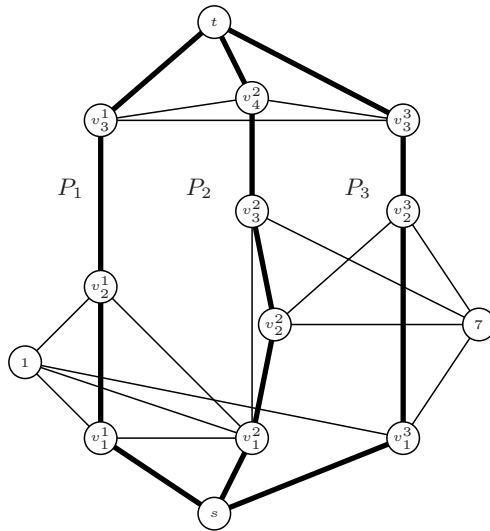


Fig. 3. Internally vertex disjoint paths P_1, P_2, P_3 of G_e represented by thick edges

Consider a $(k - 1)$ -separator W . Since the removal of W disconnects G_e , W contains at least one internal vertex from each path P_i , $i = 1, \dots, k - 1$. As W has $k - 1$ vertices, $W = \{v_{\alpha(W,1)}^1, \dots, v_{\alpha(W,k-1)}^{k-1}\}$, where $\alpha(W, i)$ is the index of the vertex of P_i belonging to W .

Claim. Let $W, U \in \mathcal{W}$. $S_W \subseteq S_U$ if and only if $\alpha(W, i) \leq \alpha(U, i)$ for all $i = 1, \dots, k - 1$.

Lemma 1. *Let S_W, S_U be $(k - 1)$ -sources of G_e . Either $S_W \cap T_U = \emptyset$ or $T_W \cap S_U = \emptyset$.*

Proof. We partition $\{1, \dots, k - 1\}$ into sets I, J and K as follows: $I = \{i \mid \alpha(W, i) > \alpha(U, i)\}$, $J = \{i \mid \alpha(W, i) = \alpha(U, i)\}$, $K = \{i \mid \alpha(W, i) < \alpha(U, i)\}$.

Let $C = \{v_{\alpha(U,i)}^i \mid i \in I\} \cup \{v_{\alpha(W,i)}^i \mid i \in I\} \cup \{v_{\alpha(W,i)}^i \mid i \in J\}$. Observe that $|C| = 2|I| + |J|$.

We show that $N(S_W \cap T_U) \subseteq C$. Note that $V \setminus ((S_W \cap T_U) \cup C) = T_W \cup S_U$ (see Figure 4). Since W and U are $(k - 1)$ -separators of G_e , there is no edge between $S_W \cap T_U$ and $T_W \cup S_U$, thus $N(S_W \cap T_U) \subseteq C$.

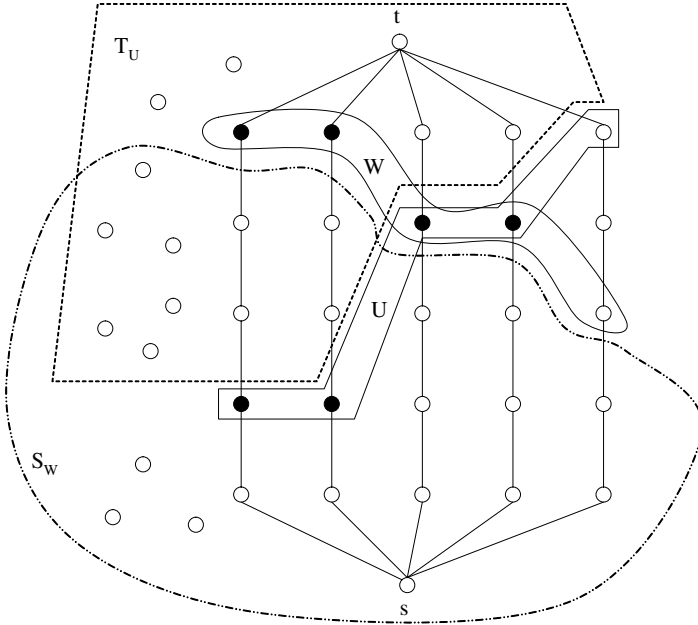


Fig. 4. $(k - 1)$ -separators W and U . Black nodes are vertices of C .

Let $D = \{v_{\alpha(U,i)}^i \mid i \in K\} \cup \{v_{\alpha(W,i)}^i \mid i \in K\} \cup \{v_{\alpha(W,i)}^i \mid i \in J\}$. Similarly, we obtain that $N(T_W \cap S_U) \subseteq D$.

Suppose for contradiction that $S_W \cap T_U \neq \emptyset$ and $T_W \cap S_U \neq \emptyset$. Since $S_W \cap T_U$ contains neither s nor t , the removal of $N(S_W \cap T_U)$ disconnects G . As G is k -vertex connected, we obtain $k \leq |N(S_W \cap T_U)| \leq |C|$, thus $2|I| + |J| \geq k$. Similarly, we have $2|K| + |J| \geq k$. Recall that I, J and K partition $\{1, \dots, k-1\}$, thus $k - 1 = |I| + |J| + |K|$.

Combining this with the above inequalities we obtain $2((k - 1) + |I| + |J| + |K|) \geq 2(k + |I| + |J| + |K|)$, a contradiction. \square

2.4 Bounding the Number of $(k - 1)$ -Sources

It is easy to see that the numebr of $(k - 1)$ -sources is at most $\binom{|V|}{k-1}$, since each one corresponds to different $(k - 1)$ -separators. In this section we provide a better bound on this number.

Corollary 1. *If S_W and S_U are incomparable in L then there exists some $i \in \{1, \dots, k - 1\}$ such that $|\alpha(W, i) - \alpha(U, i)| = 1$, i.e., the vertices $v_{\alpha(W, i)}^i$ and $v_{\alpha(U, i)}^i$ are adjacent on the path P_i .*

Proof. Suppose on the contrary that $|\alpha(W, i) - \alpha(U, i)| > 1$ for all $i = 1, \dots, k - 1$. Then since S_W and S_U are incomparable, by Claim 2.3 there exist $j, l \in \{1, \dots, k - 1\}$ such that $\alpha(U, j) + 1 < \alpha(W, j)$ and $\alpha(W, l) + 1 < \alpha(U, l)$. Then $v_{\alpha(U, j)+1}^j \in S_W \cap T_U$, $v_{\alpha(W, l)+1}^l \in T_W \cap S_U$ contradicting Lemma 1. □

The *width* of a poset is the size of its largest antichain. We show that the width of L is bounded.

Proposition 4. *The width of L is at most 2^{k-1} .*

Proof. We associate to every $(k - 1)$ -separator W a 0-1 vector $\pi(W) = (\alpha(W, 1) \bmod 2, \dots, \alpha(W, k - 1) \bmod 2)$. By Corollary 1, if two $(k - 1)$ -separators W, U are incomparable, there exists some $i \in \{1, \dots, k - 1\}$ such that $|\alpha(W, i) - \alpha(U, i)| = 1$, implying $\pi(W) \neq \pi(U)$.

Since the number of different 0-1 vectors of length $k - 1$ is 2^{k-1} , every antichain in P has size at most 2^{k-1} . □

Corollary 2. *For every fixed k the number of $(k - 1)$ -sources is $O(|V|)$.*

2.5 Generating Minimal Hyperedge Covers of $\mathcal{H}_{X,e}$

In this section we reduce the problem of generating minimal hyperedge covers of $\mathcal{H}_{X,e}$ to the problem of generating minimal transversals of 2-conformal hypergraphs. For the latter problem the algorithm is provided in 3.

A *transversal* is a set of vertices intersecting every hyperedge. A hypergraph is δ -conformal if its transpose is δ -Helly (see 2 for other equivalent definitions).

First we show that the hypergraphs $\mathcal{H}_{X,e}$ are 2-Helly.

Claim. Either $D_{u,v} = \emptyset$ or $D_{v,u} = \emptyset$ for all $u, v \in V$.

Proof. Suppose on the contrary that we have $S_W \in D_{u,v}$ and $S_U \in D_{v,u}$. Then $u \in S_W \cap T_U$ and $v \in T_W \cap S_U$, contradicting Lemma 1. □

Claim. D_f is a sublattice of L (see Figure 5).

Proof. Let $f = uv$. Without loss of generality we can assume that $D_f = D_{u,v}$. Let $S, S' \in D_{u,v}$. Then $u \in S \cap S'$ and $v \notin (S \cap S') \cup N(S \cap S')$. Similarly, $u \in S \cup S'$ and $v \notin S \cup S' \cup N(S \cup S')$. Thus $S \cap S', S \cup S' \in D_f$. □

Since the edges of $\mathcal{H}_{X,e}$ are sublattices of L , the hypergraphs $\mathcal{H}_{X,e}$ are 2-Helly (2, Example 2 on page 21]). Thus the hypergraphs $\mathcal{H}_{X,e}^T$ are 2-conformal.

Note that minimal hyperedge covers of $\mathcal{H}_{X,e}$ are minimal transversals of $\mathcal{H}_{X,e}^T$. An algorithm from 3 generates K minimal transversals of δ -conformal

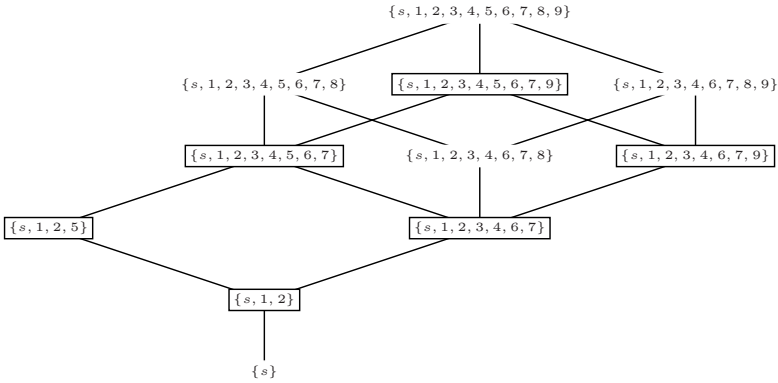


Fig. 5. Elements of $D_{2,11}$ are in black rectangles. Note that $D_{11,2} = \emptyset$.

hypergraph in $O(K^2i^2j + Ki^{\delta+2}j^{\delta+2})$, where i and j are the number of vertices and number of hyperedges, respectively.

2.6 Complexity

In this section we analyze the complexity of $Traversal(\mathcal{G})$. Let $n = |V|$, $m = |E|$. Since G is k -vertex connected we have $m \geq n$.

Note that $\pi(X)$ can be evaluated in $O(k^3|V|^2)$ time [4], thus $\gamma(E) = n^2$.

Claim. For every $X \in \mathcal{F}$ and $e \in X$ the hypergraph $\mathcal{H}_{X,e} = (\mathcal{S}, \mathcal{E})$ has $O(n)$ vertices and $O(m)$ edges and it can be constructed in $O(n^k m)$ time.

Proof of Claim 6: By Corollary 2 the number of vertices of $\mathcal{H}_{X,e} = (\mathcal{S}, \mathcal{E})$ is at most $O(n)$. The number of edges of $\mathcal{H}_{X,e} = (\mathcal{S}, \mathcal{E})$ is exactly $|E \setminus X| \leq m$ since we add an edge to $\mathcal{H}_{X,e} = (\mathcal{S}, \mathcal{E})$ for every edge of $E \setminus X$.

To construct $\mathcal{H}_{X,e} = (\mathcal{S}, \mathcal{E})$ we first need to find all $(k-1)$ -sources and $(k-1)$ -separators. We can check if after removing a given set of $k-1$ vertices the graph G is still connected in $O(n+m)$ time using, e.g., depth first search. Thus we can find all $(k-1)$ -separators by repeating the above procedure for every $(k-1)$ -element subset of V . The number of such subsets is $\binom{n}{k-1} \leq n^{k-1}$. Thus we can compute all $(k-1)$ -sources and $(k-1)$ -separators in $O(n^{k-1}m)$ time.

To add edges we need to check for every $f \in E \setminus X$ and every $(k-1)$ -separator W if S_W belongs to D_f , which can be done in $O(n)$ time for each pair f and W . Thus the complexity of constructing edges of \mathcal{H} is $O(n^2m)$. \square

Since we can find a minimal transversal of $\mathcal{H}_{X,e}^T$ in $O(|\mathcal{E}|)$ time and by Claim 2.6 we have $\phi_1(E) = n^k m$. Recall that $\phi_2(K, E) = K^2 m^2 n + K m^4 n^4$ (see Section 2.5). Thus by Proposition 2 the complexity of $Traversal(\mathcal{G})$ is $O(K^3 m^3 n + K^2 m^5 n^4 + K n^k m^2)$.

References

1. Tamura, A., Shioura, A., Uno, T.: An optimal algorithm for scanning all spanning trees of undirected graphs. *SIAM Journal on Computing* 26(3), 678–692 (1997)
2. Berge, C.: *Hypergraphs*. Elsevier-North Holland, Amsterdam (1989)
3. Boros, E., Elbassioni, K., Gurvich, V., Khachiyan, L.: Generating maximal independent sets for hypergraphs with bounded edge-intersections. In: Farach-Colton, M. (ed.) *LATIN 2004*. LNCS, vol. 2976, pp. 488–498. Springer, Heidelberg (2004)
4. Cheriyan, J., Kao, M.-Y., Thurimella, R.: Algorithms for parallel k -vertex connectivity and sparse certificates. 22, 157–174 (1993)
5. Coulbourn, C.J.: *The Combinatorics of Network Reliability*. Oxford University Press, Oxford (1987)
6. Gabow, H.N., Myers, E.W.: Finding all spanning trees of directed and undirected trees. *SIAM Journal on Computing* 117, 280–287 (1978)
7. Johnson, D.S., Papadimitriou, Ch.H.: On generating all maximal independent sets. *Information Processing Letters* 27, 119–123 (1988)
8. Khachiyan, L., Boros, E., Borys, K., Elbassioni, K., Gurvich, V., Makino, K.: Enumerating spanning and connected subsets in graphs and matroids. Manuscript.
9. Khachiyan, L., Boros, E., Borys, K., Elbassioni, K., Gurvich, V., Makino, K.: Generating cut conjunctions and bridge avoiding extensions in graphs. In: Deng, X., Du, D.-Z. (eds.) *ISAAC 2005*. LNCS, vol. 3827, pp. 156–165. Springer, Heidelberg (2005)
10. Lawler, E., Lenstra, J.K., Kan, A.H.G.R.: Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing* 9, 558–565 (1980)
11. Matsui, T.: Algorithms for finding all the spanning trees in undirected graphs. Technical report, Department of Mathematical Engineering and Information Physics, Faculty of Engineering, University of Tokyo, 1993. Report METR93-08
12. Read, R.C., Tarjan, R.E.: Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks* 5, 237–252 (1975)
13. Schwikowski, B., Speckenmeyer, E.: On enumerating all minimal solutions of feedback problems. *Discrete Applied Mathematics* 117, 253–265 (2002)
14. Valiant, L.: The complexity of enumeration and reliability problems. *SIAM Journal on Computing* 8, 410–421 (1979)

Finding Many Optimal Paths Without Growing Any Optimal Path Trees

Danny Z. Chen^{1,*} and Ewa Misiołek²

¹ Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA
chen@cse.nd.edu.

² Mathematics Department, Saint Mary's College, Notre Dame, IN 46556, USA
misiolek@saintmarys.edu.

Abstract. Many algorithms seek to compute actual optimal paths in weighted directed graphs. The standard approach for reporting an actual optimal path is based on building a single-source optimal path tree. A technique was given in [1] for a class of problems such that a single actual optimal path can be reported without maintaining any single-source optimal path tree, thus significantly reducing the space bound of those problems with no or little increase in their running time. In this paper, we extend the technique in [1] to the generalized problem of reporting many actual optimal paths with different starting and ending vertices in certain directed graphs. We show how this new technique yields improved results on several application problems, such as reconstructing a 3-D surface band bounded by two simple closed curves, finding various constrained segmentation of 2-D medical images, and circular string-to-string correction. Although the generalized many-path problem seems more difficult, our algorithms have nearly the same space and time bounds as those of the single-path cases. Our technique is likely to help improve other optimal paths or dynamic programming algorithms. We also correct an error in the time/space complexity for the circular string-to-string correction algorithm in [7] and give improved results for it.

1 Introduction

Many algorithms seek to compute actual optimal paths in a weighted directed graph $G = (V, E)$. The standard approach for finding an actual optimal path [3] builds a single-source optimal path tree using $O(|V|)$ space. Chen *et al.* [1] developed a technique for a class of problems that reports a single actual optimal path without maintaining any single-source optimal path tree, thus significantly reducing the space bound of those problems with no or little increase in their running time. Their technique is a combination of dynamic programming and divide-and-conquer methods [1]. However, some applications need to find many optimal paths with different starting and ending vertices, for which the technique in [1] does not seem immediately applicable. In this paper, we extend the

* This research was supported in part by the National Science Foundation under Grant CCF-0515203.

technique in [1] to the generalized problem of finding $k > 1$ actual optimal paths in certain directed graphs. This new technique yields improved space bounds of several problems, such as reconstructing a 3-D surface band defined by two simple closed curves [4,10], finding various constrained segmentation of 2-D medical images [2,8,11], and circular string-to-string correction [5,6,7,9]. Although this generalized many-path problem appears more difficult, our algorithms have nearly the same space and time bounds as those of the single-path cases. This new technique is likely to help improve other optimal paths or dynamic programming algorithms.

In this paper, we consider several problems where a “best” actual optimal path is sought from among $k = O(|V|)$ optimal paths in some special weighted directed graphs, namely, directed regular grid graphs. Our new space-efficient technique for these problems is based on the following observations. (1) The computation of the k optimal paths actually organizes these paths in a manner of a complete binary tree T of $O(k)$ nodes (i.e., each tree node is for one of the paths). (2) The computational process of these paths can be viewed as following a path $p(v)$ in T from the root to a node v , in the depth-first search fashion. (3) Although there are possibly many nodes on the root-to- v path $p(v)$ in T and each such node is associated with an already computed optimal path, we only need to store the two actual paths that bound the subgraph associated with the node v ; the actual paths for other nodes on the path $p(v)$ can be represented in an *encoded form* that uses much less space. (4) The graph G can be represented implicitly. Consequently, the space bounds of our algorithms are basically the same as those for computing a *single* actual path, yet our time bounds are only a small increase on those of the corresponding k -path algorithms [2,4,6,7,11]. In fact, our technique can report all k actual paths with no further increase in the time and space bounds.

We illustrate our technique by applying it to the problem of reconstructing a 3-D surface band defined by two simple closed curves. The reconstructed surface consists of a sequence of triangles in 3-D with the minimum total area. This surface reconstruction problem arises in computer-aided design and manufacturing (CAD/CAM), computer graphics, biological research, and medical diagnosis and imaging. Fuchs *et al.* [4] modeled this problem as finding an actual shortest path among k shortest paths in a weighted acyclic grid graph $G = (V, E)$, where $|V| = O(|E|) = O(kl)$ and k and l are positive integer input parameters (with $k \leq l$). They gave an $O(kl \log k)$ time algorithm. If a standard shortest path algorithm [3] is used, then the space bound in [4] is $O(kl)$. Our technique yields an $O(kl(\log k)^2)$ time and $O(k + l)$ space algorithm. In fact, our technique gives a general trade-off relation between the space and time bounds (see Theorem [3]).

Our technique can also solve several variations of the 3-D surface reconstruction problem in [10] and can significantly reduce the space bounds for four of the several optimization criteria considered in [10].

In the full version of the paper, we also show how to apply our technique to reduce the space needed for computing various constrained segmentation of medical images. Image segmentation is important to medical image analysis for

medical diagnosis and treatment, computer vision, pattern recognition, mechanical and material study, and data mining. Its goal is to define the boundary for an object of interest in the image, separating it from its surrounding. The best known algorithms for this problem [2,11] take $O(kl \log k)$ time and $O(kl)$ space. Using our technique, the time bound is increased from that of [2,11] by only a factor of less than or equal to $\log l$, but the working space of these algorithms is decreased significantly to almost linear.

Another key application is the circular string-to-string correction problem [5,6,7,9], which seeks a minimum-cost sequence of editing operations for changing a circular string $A = (a_1, a_2, \dots, a_k)$ to another circular string $B = (b_1, b_2, \dots, b_l)$ (assume $k \leq l$). This problem arises in many areas such as shape recognition, pattern recognition, speech recognition, and circular DNA and protein sequence alignment, and is well studied. To our knowledge, the best known algorithm for this problem is due to Maes [7]. It is based on a divide-and-conquer scheme for computing k shortest paths in a rectangular grid graph G , and the author claimed it takes $O(kl \log k)$ time and $O((k+l) \log k)$ space. However, these time and space bounds do not appear correct. As stated in [7], it just applies Hirschberg's algorithm [5] to find each of the k paths. However, when computing a single shortest path in a subgraph G_i of the original graph G using the method in [5], unlike in the situation of finding only *one* actual path in the original grid graph G , there is no guarantee that as the recursion in [5] proceeds, the total size of the subgraphs always decreases significantly (say, by a constant fraction) from one recursion level to the next. This implies that when computing k paths in G , finding one actual path in a subgraph G_i of G using the method in [5] actually takes $O(|G_i| \log(k+l))$ time, instead of $O(|G_i|)$. Thus, to obtain an $O((k+l) \log k)$ space bound, the algorithm as stated in [7] actually takes $O(kl \log k \log(l+k))$ time (see our full paper for more detailed discussion and analysis). Applying our technique, we improve the best known circular string-to-string correction algorithm in [7], e.g., to $O(l + k(\log k)^\epsilon)$ space and $O(kl \frac{(\log k)^2}{\log \log k})$ time, for any constant ϵ with $0 < \epsilon < 1$ (the exact results are as those stated in Theorem 3).

2 Preliminaries

In this section, we sketch the approaches in [1] and [6,2,4,7,11], which will be referred to and build upon in the later sections.

Let $G = (V, E)$ be a directed graph with nonnegative edge weights. For two vertices $s, t \in V$, the standard method [3] for finding an actual optimal s -to- t path in G is to build and store a *single-source optimal path tree* T rooted at s . Then for any vertex v , an actual optimal s -to- v path in G is obtained by traversing in T the v -to- s path. It is well known that no asymptotically faster algorithm is known for reporting a single actual optimal path than for building a single-source optimal path tree. However, for a set of optimal path problems on regular grid graphs, Chen *et al.* [1] gave a technique with space bound for computing a single actual optimal s -to- t path asymptotically better than that for building a single-source tree. Their method is hinged on the *clipped tree*

data structure and a variation of the divide-and-conquer called *marriage-before-conquer*.

A regular grid graph $G = (V, E)$ has vertices lying on a rectangular $k \times l$ grid and the vertices are connected by edges only to the adjacent vertices from the same row or to the vertices from the row below, thus $|V| = kl$ and $|E| = O(|V|)$. Figures 1(a),(b) show examples of such graphs. A clipped tree T_{clp} is a compressed version of a single-source tree T with the following characteristics: (1) T_{clp} contains a subset of the nodes in T , including s, t , and vertices from only τ rows of G ; (2) two nodes v and w in T_{clp} form an ancestor-descendant relation in T_{clp} if and only if they form the same relation in T .

The following result in 1 is useful to us.

Lemma 1. 1 *Given a regular grid graph G of size $k \times l$, two vertices s and t in G , and a parameter τ with $1 \leq \tau \leq k$, it is possible to use $O(|T_{clp}|) = O(l\tau)$ space to report an optimal s -to- t path in G in time $O((T(\cdot)(\log k))/\log(\tau - 1))$, where $T(\cdot)$ is the running time of a standard optimal path algorithm on G .*

In some special cases the extra $(\log n)/(\log(\tau - 1))$ time factor can be avoided. For example, for the special graph in Fig. 1(a), to find a shortest path from s to t , one can first locate a vertex v_2 in the middle row of the graph, and then recursively find the shortest s -to- v_2 and v_2 -to- t paths in two rectangular subgraphs of G . Observe, that the sum of the sizes of the two subgraphs in the recursive calls is half the size of the original graph, resulting in the same time bound as that of a standard shortest path algorithm.

It should be pointed out that the method in 1 may be applied to graphs even if their structures are not grid-like. In fact, the paradigm can be applied to some problems that are solved using dynamic programming, by “divide-and-conquering” the dynamic programming table instead of the graph itself.

We extend the technique in 1 to the type of problems where an optimal path is to be selected from a set of k optimal paths with k pairs of different source-destination vertices in a directed grid graph G , with $k = O(n)$.

A straightforward approach for solving such a k -path problem is to apply an optimal path algorithm to compute each of the k paths and then determine the best one. However, more efficient algorithms are possible for regular, rectangular grid graphs. A key idea, as shown in [6,2,4,7,11], is to first find one of the k paths and then recursively search for the remaining paths in increasingly smaller subgraphs of G . The description below summarizes this approach and introduces some of the notation used in the later sections.

Suppose we are given a directed acyclic graph G and a sequence of k pairs of vertices $(v_i, w_i), i = 1, 2, \dots, k$, between which the shortest paths in G are sought. Let SP_i denote a shortest path between the pair (v_i, w_i) in G . We assume that the graph G has the following important property.

Property 2. For any i and $j, 1 \leq i \leq j \leq k$, two shortest paths SP_i and SP_j can be computed in G , such that SP_i and SP_j bound a subgraph $G_{i,j}$ of G and $G_{i,j}$ contains the shortest paths SP_g in G , for all $g = i, i + 1, \dots, j$.

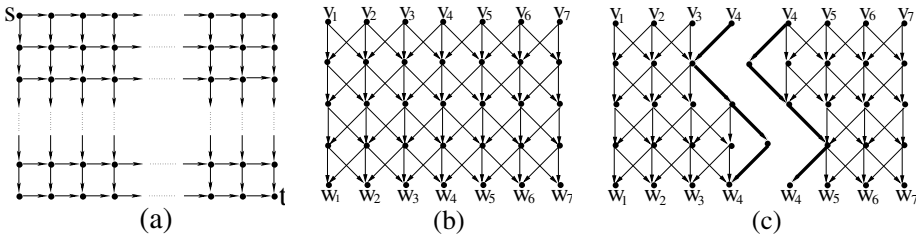


Fig. 1. (a),(b) An example of a regular rectangular grid graph G . (c) G is split along the “middle” shortest path SP_4 into two subgraphs for the recursive subproblems.

For example, the graphs in [2,4,6,7,11] all have this property since they are all regular, rectangular grid graphs embedded in the plane, the sequence of vertex pairs (v_i, w_i) are all on the outer face of the embedding, and subgraphs $G_{i,j}$ can be obtained by finding the shortest paths SP_i and SP_j . Let $SP(G_{i,j}, i, j)$ denote a procedure for computing the “middle” shortest path SP_t in the subgraph $G_{i,j}$, $t = \lfloor \frac{i+j}{2} \rfloor$. Then the k -path algorithm in [6,2,4,7,11] proceeds as follows.

1. Find SP_1 and SP_k . Let $i = 1$ and $j = k$.
2. Run $SP(G_{i,j}, i, j)$ to obtain SP_t (for the pair (v_t, w_t) , with $t = \lfloor \frac{i+j}{2} \rfloor$).
3. Recursively solve the subproblems on subgraphs $G_{i,t}$ and $G_{t,j}$, respectively.

At the end of the algorithm, the “best” of these k shortest paths (i.e., the one with the overall shortest path length) is selected and the actual path is reported. For example, to find the shortest of the $k = 7$ paths between the pairs of vertices (v_i, w_i) , $i = 1, \dots, 7$, in the graph in Fig. 1(b), we first find a shortest path between the “middle” pair (v_4, w_4) , then split the graph along that path to recursively compute the remaining paths in the resulting subgraphs (Fig. 1(c)).

A straightforward implementation of this paradigm needs to construct and store both the grid graph G (say, of size $k \cdot l$) and single-source shortest path trees for each source vertex, thus using $O(|V|) = O(|E|) = O(kl)$ space. The running time of this approach is $O(T(\cdot) \log k)$, where $T(\cdot)$ is the standard time bound for computing one shortest path in G .

It should be pointed out that in the above paradigm, since the already computed shortest paths are used to define the boundaries of the subgraphs in which the remaining paths are to be searched, the computed actual paths need to be stored. A straightforward method could store $O(k)$ actual paths at the same time, thus using a lot of space. To achieve a space-efficient algorithm for the k -path problem, we must avoid storing too many actual paths simultaneously. This is made possible by exploiting the special structures of the graphs we use. We will demonstrate our algorithm by applying it to the problem of reconstructing a 3-D surface band bounded by two simple closed curves.

3 Optimally Triangulating a 3-D Surface Band

3.1 The Surface Band Triangulation and a Previous Algorithm

The desired output of the surface reconstruction problem, also called a *surface triangulation problem*, is a sequence of triangular tiles that best approximates the surface band based on a given optimization criteria. Below we define the problem and sketch its time-efficient algorithm as given in [4].

Suppose a 3-D surface \mathcal{S} , which is to be reconstructed, is divided into a sequence of surface “bands” by a set of mutually non-crossing simple closed curves (also called *contours*) lying on \mathcal{S} , such that each surface band is bounded by two of the curves. To reconstruct \mathcal{S} , it is sufficient to reconstruct each of the surface bands. Thus, we focus on the problem of reconstructing a single surface band bounded by two simple closed curves C_P and C_Q in 3-D [4][10].

From the computational view point, each of the two curves, is represented by a sequence of points along the curve. That is, a curve C_P is approximated by a closed polygonal curve $P = \{p_1, p_2, \dots, p_k\}$ in such a way, that the portion of C_P between any two consecutive points p_i and p_{i+1} is approximated by the line segment $\overline{p_i p_{i+1}}$. The triangulation of the surface band bounded by $P = \{p_1, p_2, \dots, p_k\}$ and $Q = \{q_1, q_2, \dots, q_l\}$ gives a sequence of non-overlapping triangles, whose edges are the edges of P or Q , and line segments connecting a vertex of P with a vertex of Q . More precisely, each triangle has one edge that is an edge of one curve (say $\overline{p_i p_{i+1}}$ of P), called the “base” edge, and two other edges that share a vertex of the other curve (say q_j of Q), called the “bridges”. Thus a triangle $\Delta p_i q_j p_{i+1}$ with base on P , has the edges $\overline{p_i p_{i+1}}$, $\overline{p_i q_j}$, and $\overline{q_j p_{i+1}}$, similarly, a triangle $\Delta q_j q_i q_{j+1}$ with base on Q , has the edges $\overline{q_j q_{j+1}}$, $\overline{q_j p_i}$, and $\overline{p_i q_{j+1}}$. The set of triangles of a feasible triangulation forms a partition of the surface band into a set of triangular faces such that the set of vertices of the faces is exactly the set of the vertices of P and Q . Also, all edges of P and Q belong to the set of the triangulation edges. See Fig. 2(a) for an example of a feasible triangulation.

An optimal triangulation is a feasible triangulation that optimizes the given criteria. Possible optimization criteria for triangulating a surface band include minimizing the total area, bending energy, and mean curvature variation, or maximizing the total twist, or optimizing a certain combination of these or other optimization criteria (see [10] for detailed descriptions of such criteria). For our discussion in this paper, we assume that the optimization criterion is to minimize the total triangulated area. However, three other optimization criteria in [10] (minimum total twist, maximum convexity, and minimum normal variation) can also be used by our technique.

Note that the number of feasible triangulations of the surface band between the two curves P and Q is prohibitively large. An efficient approach [4] models this triangulation problem as one of searching for a best path among the shortest paths between k pairs of vertices in a directed graph $H = (V_H, E_H)$, as follows. Without loss of generality, we assume $k \leq l$ throughout this section.

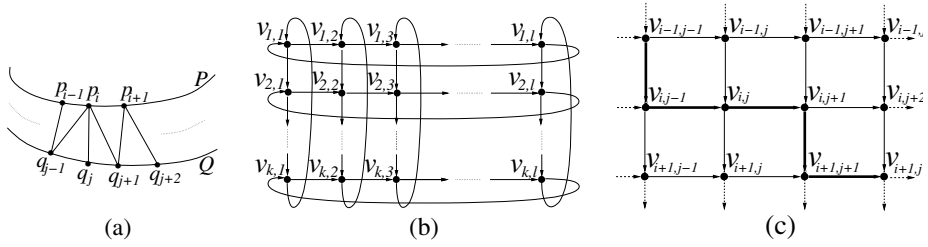


Fig. 2. (a) A feasible triangulation for a portion of the surface band between the polygonal curves P and Q . (b) A toroidal graph H defined by P and Q . (c) A path in H corresponds to a triangulation in (a).

- Every possible “bridge” segment $\overline{p_i q_j}$ corresponds to a vertex $v_{i,j}$ in V_H .
- For every $j = 1, 2, \dots, l$, there is a directed edge $(v_{i,j}, v_{i,j+1})$ in E_H corresponding to a possible triangle $\Delta q_j p_i q_{j+1}$ with the “base” edge $\overline{q_j q_{j+1}}$, where the index $l + 1$ means 1. For every $i = 1, 2, \dots, k$, there is a directed edge $(v_{i,j}, v_{i+1,j})$ in E_H corresponding to a possible triangle $\Delta p_i q_j p_{i+1}$ with the “base” edge $\overline{p_i p_{i+1}}$, where the index $k + 1$ means 1 (see Fig. 2(c)).
- The weight of every edge is the surface area of the triangle represented by that edge.

Note that the graph H is a toroidal graph of size $O(kl)$. See Fig. 2(b)–2(c) for an example of H ; the thick path in Fig. 2(c) represents the given portion of a triangulation of the band in Fig. 2(a). Any triangulation of the surface band corresponds to a simple closed path, called a *triangulating path*, in H that visits every row and every column of the “flattened” version of H 4.

To simplify the shortest path search, a new (“flattened”) graph G is created from H by appending a copy of H after the last row of H (without the wrap-around edges), and duplicating the first column (i.e., the vertices $v_{i,1}$) of the newly created graph and appending it after the last column (the appended vertices are denoted by w_i ’s). Fig. 3(b) shows the graph G created from the graph H in Fig. 3(a); the thick path in G corresponds to a triangulating path in H .

Now, the problem becomes that of finding the k shortest paths between the k pairs of vertices $(v_{i,1}, w_{k+i})$ in G for $i = 1, 2, \dots, k$ ($v_{i,1}$ ’s are denoted as v_i ’s in Fig. 3), and selecting the shortest of those as the solution. Note that G thus created is a directed planar acyclic grid graph, and hence it satisfies Property 2. Therefore, this k -path problem on G can be solved in $O(kl \log k)$ time and $O(kl)$ space using the divide-and-conquer method in 4.6 and described in Sect. 2.

3.2 Our Basic Algorithm

We now present a method that significantly reduces the space bound of the solution in 4.6. For clarity of discussion, we first present a less efficient algorithm, followed by further improvements.

Recall from Sect. 2 that in order to solve the k -path problem in a time-efficient manner, we need to recursively compute shortest paths on increasingly

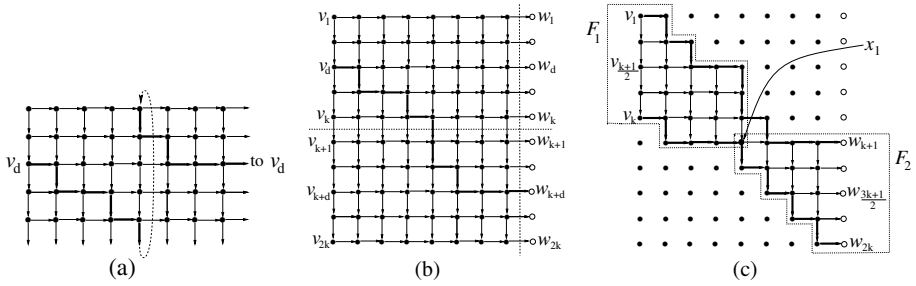


Fig. 3. (a) A “flat” version of the toroidal graph H . (b) The planar grid graph G created from H . (c) The two subgraphs F_1 and F_2 of $G_{1,k}$ for the two subsequent subproblems in the recursion. The graph $F_1 \cup F_2$ has only a single vertex and two edges (those outside the dashed boundaries) fewer than $G_{1,k}$.

smaller subgraphs bounded by two already computed paths. Therefore, since paths are needed to define subgraphs for future recursive calls, in the worst case, a straightforward algorithm may require storage of all k paths. In this section we present a method that requires storage of only a constant number of shortest paths by organizing the k shortest paths using the recursion tree and exploiting the structure of such organization. We combine our method with the technique in [1] to significantly decrease the space complexity of algorithms in [4,6] with only a small increase in time. In fact, depending on the application, one can select a trade-off between the time and space saving produced by our method. We now proceed to explain the method.

First notice, that we need not explicitly construct and store G since for any vertex of G , its two incoming and two outgoing edges as well as their weights can be determined in $O(1)$ time. Since the input consists of $k + l$ points representing the curves P and Q , the implicit representation of G uses only $O(k + l)$ space, a significant saving since G has $n = 2kl$ vertices and $m = 4kl$ edges.

Further, whenever an actual shortest path SP_i in a subgraph G_i of G from $v_{i,1}$ to w_{k+i} is computed in the k -path scheme of [4,6], we apply the space-efficient algorithm of Chen *et al.* [1]. This way we avoid storing a single-source tree (of size $O(kl)$) as in the standard single-source shortest path algorithm. This, however, causes a $\log k$ factor increase in the time complexity of the algorithm. To find an actual path SP_i in G_i , the method in [1] divides G_i into subgraphs and recursively reports vertices of SP_i in these subgraphs. However, G_i , although is a subgraph of the original rectangular grid graph G , has an irregular shape, see Fig. 3(c). Thus, when recursively computing SP_i on G_i , there is no guarantee that the total size of the subgraphs containing the pieces of SP_i at one recursion level decreases significantly (say, by a constant fraction) from the total size of the subgraphs in G_i at the “parent” level; i.e., from one recursion level to the next, it is not guaranteed that a constant fraction of the subgraphs is always “pruned” away. For example, in Fig. 3(c), the two subgraphs F_1 and F_2 of $G_{1,k}$ for the two subsequent subproblems are such that the graph $F_1 \cup F_2$ has only a

single vertex and two edges fewer than $G_{1,k}$. While the $O(\log k)$ time increase can be avoided for computing *one* or even $O(1)$ actual paths in the original grid graph G by using the method in [1] (as in Fig. 1(a)), the same is not true for computing *each of many* actual paths when using the k -path scheme in [4,6]. Thus, in the setting of computing many shortest paths, and in comparison with the common tree-growing approach, the recursive process for reporting an actual path in G_i incurs a $\log k$ factor increase in the time bound, but also results in a significant reduction of needed space.

Now, to resolve the problem of storing k shortest paths, first observe that the k shortest paths SP_i in G for the vertex pairs $(v_{i,1}, w_{k+i})$, $i = 1, \dots, k$, can be organized based on the recursion tree used by the scheme in [4,6]. This recursion tree is a complete binary tree of $O(k)$ nodes, called the k -path tree T_k . The root of T_k represents the “middle” path SP_t in the subgraph $G_{1,k} \subseteq G$ ($t = \lfloor \frac{1+k}{2} \rfloor$); the children of each internal node in T_k are for the subproblems on their associated subgraphs. For example, the children of the root are associated with the subgraphs $G_{1,t}$ and $G_{t,k}$, and represent their “middle” paths computed by the procedures $SP(G_{i,t}, i, t)$ and $SP(G_{t,j}, t, j)$, respectively.

If the k actual shortest paths represented by the k -path tree T_k were to be all stored at the same time, then the space bound would be $O(k(k+l))$ (since each path SP_i in G has $O(k+l)$ vertices). Fortunately, only a small subset of the paths associated with the nodes in T_k is needed at any time of the computation. We call such nodes in T_k the *active nodes*. It is sufficient to consider only the space used by the active nodes.

To compute the middle shortest path SP_t in a subgraph $G_{i,j}$, we need to store two actual shortest paths SP_i and SP_j that bound the subgraph $G_{i,j}$. Let u_t be the node in the k -path tree T_k corresponding to SP_t and $G_{i,j}$. Then an important observation is that the two paths SP_i and SP_j are always at two nodes on the root-to- u_t path in T_k . This implies that when the computation is performed at a node u of T_k , only the ancestor nodes of u (including u itself) are active. Clearly, at any time of the computation, there are only $O(\log k)$ active nodes in T_k .

In fact, the computation of the k paths in G proceeds in the order as if a depth-first search is done on the tree T_k . During the computation, we store all actual paths represented by the active nodes of T_k . In the worst case, $2 + \log k$ actual paths need to be stored (the paths for $\log k$ active nodes plus SP_1 and SP_k). In addition, we keep track of the vertex pair $(v_{i,1}, w_{k+i})$ such that the length value of SP_i is the smallest among all paths computed thus far. After all k shortest paths (and their lengths) are computed, we use the algorithm in [1] to report the actual shortest path in G whose length value is the smallest among all k paths. Given the “best” actual shortest path in G , the corresponding optimal triangulation of the 3-D surface band can be easily obtained.

Since we store $O(\log k)$ actual shortest paths, each of which has $O(k+l)$ vertices, our basic algorithm uses $O((k+l)\log k)$ space, improving the straightforward $O(kl)$ space bound. Because the k -path scheme in [4,6] takes $O(kl \log k)$ time and our space-efficient algorithm incurs a factor of $O(\log k)$ time increase, the total time of our basic algorithm is $O(kl(\log k)^2)$.

3.3 A Space Improvement Scheme

It is possible to reduce the space bound of the above basic algorithm from $O((k+l)\log k)$ to $O(k+l)$, without affecting its running time. Note that this $O(k+l)$ space bound matches the one for finding *one* actual shortest path [1]. The idea is to use a bit encoding scheme to represent most of the actual paths for the active nodes of T_k during the computation.

When computing a shortest path SP_t in a subgraph $G_{i,j}$ ($t = \lfloor \frac{i+j}{2} \rfloor$), we need to store the two actual paths SP_i and SP_j that bound $G_{i,j}$. Actually, SP_i and SP_j are the only actual paths needed for computing SP_t in $G_{i,j}$. That is, the other paths represented by the active nodes in T_k are not really needed for computing SP_t . Therefore, it is sufficient to explicitly store only SP_i and SP_j when computing SP_t . All other shortest paths of the active nodes in T_k can be temporarily stored in a “compressed” form. This “compressed” form should use less space, yet can be easily converted back and forth between the explicit form (so that these paths can be used for future computation).

Our choice of this “compressed” representation is a bit encoding scheme for the actual paths in G . Our observation is that each shortest path in the grid graph G consists of a sequence of “rightward” edges and “downward” edges, which can be represented by 0’s and 1’s, respectively. For example, the thick path in Fig. 3(b) can be represented by the sequence (0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0).

Given the starting vertex of a path, it is easy to convert a bit-encoded path representation to or from an explicit actual path in $O(k+l)$ time. Such conversions need to be performed $O(k)$ times by our algorithm, which add only $O(k(k+l)) = O(kl)$ time to the algorithm (since $k \leq l$). Hence, the total running time of our algorithm is $O(kl(\log k)^2)$ as the basic algorithm.

Thus, at any time of the computation, our algorithm stores only $O(1)$ actual paths explicitly, and the other $O(\log k)$ paths using bit encoding. To analyze the space bound of our algorithm, note that storing one path using bit encoding needs $O(k+l)$ bits. Thus, storing $O(\log k)$ paths in this scheme uses $O((k+l)\log k)$ bits, which are equal to $O(k+l)$ space. Also, to explicitly store $O(1)$ actual paths with $k+l$ vertices each, $O(k+l)$ space is needed. Hence, the overall space bound of our algorithm is $O(k+l)$, and its time bound is still $O(kl(\log k)^2)$.

3.4 Our Final Algorithm

We now show how to improve both the space and time bounds of our basic algorithm. In fact, we give a trade-off relation between the space and time bounds. This is achieved by combining the space improving idea in Subsection 3.3 with a more careful application of the algorithm in [1].

Based on the idea and analysis in Subsection 3.3, it is clear that our algorithm can store all actual shortest paths needed by the computation using only $O(k+l)$ space.

Another important fact is that Chen *et al.*’s algorithm [1] can use the clipped tree T_{clip} to store $\tau \geq 3$ vertices on any s -to- t path. Our basic algorithm stores only $\tau = 3$ vertices for any sought path in the tree T_{clip} (by recording mainly

the vertices in the middle column of the grid subgraph); this divides the sought path into only two subpaths for the recursive reporting. Our idea here is that a larger value of τ can be used. That is, we can store τ columns of vertices of the subgraph in T_{clp} . This means that τ vertices on the sought path are stored in T_{clp} , which cut the sought path into $\tau - 1$ subpaths for the recursive reporting. This implies that the factor of time increase incurred by our algorithm over that of the method in [46] becomes $\frac{\log k}{\log \tau}$ (instead of $\log k$ as for the basic algorithm). Clearly, storing τ columns of vertices of a subgraph in T_{clp} uses $O(k\tau)$ space.

We can choose τ in the following manner. Suppose we consider a function $f(k)$. If $f(k) = O(1)$, then let $\tau = O(1) \geq 3$; if $f(k) > O(1)$, then let $\tau = (f(k))^\epsilon$ for any constant ϵ with $0 < \epsilon < 1$ (e.g., τ can be k^ϵ , $(\log k)^\epsilon$, $(\log \log k)^\epsilon$, etc). For example, by letting $\tau = (\log k)^\epsilon$, our algorithm takes $O(l + k(\log k)^\epsilon)$ space and $O(kl \frac{(\log k)^2}{\log \log k})$ time.

Theorem 3. *The problem of optimally triangulating a 3-D surface band bounded by two closed polygonal curves P and Q can be solved in $O(l + k\tau)$ space and $O(kl \frac{(\log k)^2}{\log \tau})$ time, where $k = |P|$, $l = |Q|$, $k \leq l$, and unless $\tau = O(1)$, τ is of the form $(f(k))^\epsilon$ for any chosen function $f(k) > O(1)$ and any constant ϵ with $0 < \epsilon < 1$.*

References

1. Chen, D.Z., Daescu, O., Hu, X.S., Xu, J.: Finding an optimal path without growing the tree. *Journal of Algorithms* 49(1), 13–41 (2003)
2. Chen, D.Z., Wang, J., Wu, X.: Image segmentation with asteroidality/tubularity and smoothness constraints. *International Journal of Computational Geometry and Applications* 12(5), 413–428 (2002)
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: *Introduction to Algorithms*, 2nd edn. McGraw-Hill, New York (2001)
4. Fuchs, H., Kedem, Z.M., Uselton, S.P.: Optimal surface reconstruction from planar contours. *Communications of the ACM* 20(10), 693–702 (1977)
5. Hirschberg, D.S.: A linear space algorithm for computing maximal common subsequences. *Communications of the ACM* 18(6), 341–343 (1975)
6. Kedem, Z.M., Fuchs, H.: A fast method for finding several shortest paths in certain graphs. *Proc. 18th Allerton Conf.*, 677–686 (1980)
7. Maes, M.: On a cyclic string-to-string correction problem. *Information Processing Letters* 35(2), 73–78 (1990)
8. Thedens, D.R., Skorton, D.J., Fleagle, S.R.: Methods of graph searching for border detection in image sequences with applications to cardiac magnetic resonance imaging. *IEEE Trans. on Medical Imaging* 14(1), 42–55 (1995)
9. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. *Journal of the ACM* 21(1), 168–173 (1974)
10. Wang, C., Tang, K.: Optimal boundary triangulations of an interpolating ruled surface. *ASME Journal of Computing and Information Science in Engineering* 5(4), 291–301 (2005)
11. Wu, X.: Segmenting doughnut-shaped objects in medical images. In: Ibaraki, T., Katoh, N., Ono, H. (eds.) *ISAAC 2003*. LNCS, vol. 2906, pp. 375–384. Springer, Heidelberg (2003)

Enumerating Constrained Non-crossing Geometric Spanning Trees

Naoki Katoh and Shin-ichi Tanigawa

Department of Architecture and Architectural Engineering, Kyoto University,
Kyoto 615-8450 Japan
{naoki, is.tanigawa}@archi.kyoto-u.ac.jp

Abstract. In this paper we present an algorithm for enumerating without repetitions all non-crossing geometric spanning trees on a given set of n points in the plane under edge inclusion constraints (i.e., some edges are required to be included in spanning trees). We will first prove that a set of all edge-constrained non-crossing spanning trees is connected via remove-add flips, based on the constrained smallest indexed triangulation which is obtained by extending the lexicographically ordered triangulation introduced by Bespamyatnikh. More specifically, we prove that all edge-constrained triangulations can be transformed to the smallest indexed triangulation among them by $O(n^2)$ times of greedy flips. Our enumeration algorithm generates each output graph in $O(n^2)$ time and $O(n)$ space based on reverse search technique.

1 Introduction

Given a graph $G = (V, E)$ with n vertices and m edges where $V = \{1, \dots, n\}$, G is a *spanning tree* if and only if G is connected and does not contain any cycle. An embedding of the graph on a set of points $P = \{p_1, \dots, p_n\} \subset \mathbf{R}^2$ is a mapping of the vertices to points in the Euclidian plane $i \mapsto p_i \in P$. The edges ij of G are mapped to straight line segments $p_i p_j$. An embedding is *non-crossing* if each pair of segments $p_i p_j$ and $p_k p_l$ have no point in common without their endpoints.

If the spanning tree embedded on the plane is non-crossing, it is called *geometric non-crossing spanning tree*, which is simply called non-crossing spanning tree (NST) in this paper. We assume in this paper that spanning trees are embedded on a fixed point set P in \mathbf{R}^2 . Let F be a set of non-crossing line segments on P . A spanning tree containing F is called *F -constrained spanning tree*. Then in this paper we give an algorithm for enumerating all the *F -constrained non-crossing spanning trees* (F -CNST). We simply denote a vertex p_i by i and an edge $p_i p_j$ by ij throughout the paper.

The algorithm we propose requires $O(n^2)$ time per output and $O(n)$ space. For the unconstrained case (i.e. $F = \emptyset$), the algorithm by Avis and Fukuda [6] requires an $O(n^3)$ time per output and $O(n)$ space. Recently Aichholzer et al. [2] have developed an algorithm for enumerating $O(n)$ time per output based on

the Gray code enumeration, whose space complexity is not given. Although their algorithm is superior to ours in the unconstrained case, it seems that it cannot be extended to the edge-constrained case.

It is well known that the number of non-crossing spanning trees grows too rapidly to allow a complete enumeration for a significantly larger point set. In view of both theoretical and practical applications the number of objects to be enumerated or the computational cost must be reduced by imposing several reasonable constraints. For this purpose, the edge constraint would be naturally considered, and thus our algorithm has much advantage in practice.

For the edge-constrained case, in our recent paper [7], we proposed an algorithm for enumerating the F -constrained non-crossing minimally rigid frameworks embedded on a given point set in the plane. We remarked therein that based on a similar approach, we could develop an $O(n^3)$ algorithm for enumerating F -CNSTs. Although we have not given either any algorithm details or analysis of the running time, it seems difficult to improve this running time.

Let \mathcal{O} be a set of objects to be enumerated. Two objects are *connected* iff they can be transformed to each other by a *local operation*, which generates one object from the other by means of a small change. Especially, it is sometimes called *(1-)flip* if they have all but one edge in common. Define a graph $\mathcal{G}_{\mathcal{O}}$ on \mathcal{O} with a set of edges connecting between objects that can be transformed to each other by one local operation. Then the natural question is how we can design local operation so that $\mathcal{G}_{\mathcal{O}}$ is connected, or how we can design $\mathcal{G}_{\mathcal{O}}$ with small diameter. There are several known results for these questions for triangulations (e.g. [11]), pseudo-triangulations [1], geometric matchings [10], some classes of simple polygons [9] and also for NSTs [2,1,3,4,6]. Especially relevant to the historical context of our work are the results for NSTs [6,1,3,4,2]. Let \mathcal{ST} be a set of all NSTs on a given set of n points. Avis and Fukuda [6] have developed 1-flip such that $\mathcal{G}_{\mathcal{ST}}$ is connected with diameter $2n - 4$. For the case of a local operation other than 1-flip, the operations with diameters of $O(\log n)$ [3] and the improved result [1] are known. Aichholzer et al. in [3,4] also tried to design 1-flip with the additional requirement, called *edge slide*, such that removed edge moves to the other one along an adjacent edge keeping one endpoint of the removed edge fixed. Aichholzer et al. in [2] showed that the graph $\mathcal{G}_{\mathcal{ST}}$ contains a Hamiltonian path by developing Gray code enumeration schemes. In this paper, we will propose 1-flip with the additional requirement such that removed and added edges are sharing one endpoint, and show that all F -CNSTs are connected by $O(n^2)$ such flips plus $O(n)$ base exchange operations. We notice that all 1-flips designed in the previous works seem to be difficult to extend to the edge-constrained case.

Main tools we use are reverse search and the *smallest indexed triangulation* (SIT). Reverse search developed by Avis and Fukuda [5,6] generates all the elements of \mathcal{O} by tracing the nodes in $\mathcal{G}_{\mathcal{O}}$. To trace $\mathcal{G}_{\mathcal{O}}$ efficiently, it defines a *root* on $\mathcal{G}_{\mathcal{O}}$ and a *parent* for each node except the root. Define the parent-child relation satisfying the following conditions: (1) each non-root object has a unique parent, and (2) an ancestor of an object is not itself. Then, iterating going up to

the parent leads to the root from any other node in \mathcal{G}_O if \mathcal{G}_O is *connected*. The set of such paths defines a spanning tree, known as a *search tree*, and the algorithm traces it by depth-first search manner. So, the necessary ingredients to use the method are an implicitly described connected graph \mathcal{G}_O and an implicitly defined search tree in \mathcal{G}_O . In this paper we supply these ingredients for the problem of generating all F -CNSTs.

The idea of the SIT is derived from a lexicographically ordered triangulation developed by Bespamyatnikh [8] for enumerating triangulations efficiently. We generalize it to edge-constrained case by associating an appropriate index with each triangulation. The SIT plays a crucial role in the development of our algorithm. We conjecture that the general idea to use triangulations proposed in this paper can be extended to an efficient algorithm for enumerating non-crossing graphs other than NSTs and minimally rigid frameworks because any non-crossing graph can be augmented to a triangulation.

2 Smallest Indexed Triangulation

In this section, we define an F -constrained smallest indexed triangulation (F -CSIT). Then we show that any edge-constrained triangulation can be transformed into CSIT by $O(n^2)$ flips. We remark again that CSIT is derived from the lexicographically ordered triangulation developed by Bespamyatnikh [8] although he had not extended his results to the edge-constrained case.

2.1 Notations

Let us first define several notations. Let P be a set of n points on the plane, and for simplicity we assume that the vertices $P = \{1, \dots, n\}$ are labeled in the increasing order of x -coordinates. We assume that x -coordinates of all points are distinct and that no three points in P are colinear. For two vertices $i, j \in P$, we use the notation, $i < j$, if $x(i) < x(j)$ holds, where $x(\cdot)$ represents a x -coordinate of a point. Considering $i \in P$, we often pay our attention only to its right point set, $\{i + 1, \dots, n\} \subseteq P$. So, let us denote $\{i + 1, \dots, n\}$ by P_i .

We usually denote an edge between i and j with $i < j$ by ij . For three points i, j, k , the signed area $\Delta(i, j, k)$ of a triangle Δijk tells us that k is on the left or right side of a line passing through i and j when moving along the line from i to j by $\Delta(i, j, k) > 0$ or $\Delta(i, j, k) < 0$, respectively. Then the lexicographical ordering on a set of edges is defined as follows: for $e = ij$ and $e' = kl$, e is lexicographically smaller than e' (denoted by $e \prec e'$ or $e' \succ e$) iff $i < k$, or $i = k$ and $\Delta(i, j, l) < 0$ ¹, and denote by $e = e'$ when they coincide. Notice that, when $i = k$, the lexicographical ordering corresponds to the clockwise ordering around i in our definition.

¹ In general the lexicographical ordering for edge set is defined in such a way that $e = ij$ is smaller than $e' = kl$ iff either $i < k$ or $i = k$ and $j < l$ holds. But in this paper we adopt our lexicographical ordering for efficient enumeration described in Section 4.

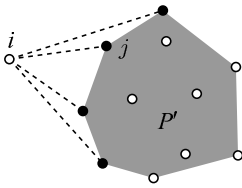


Fig. 1. Black vertices represent a set of vertices visible from i with respect to $\text{conv}(P')$

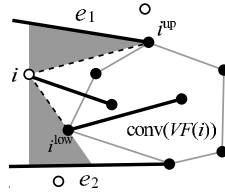


Fig. 2. Example of the upper and lower tangents. Bold edges represent F . Empty regions of ii^{up} and ii^{low} are shaded.

For two vertices $i, j \in P$, j is *visible* from i with respect to a constrained edge set F when an edge ij and any edge in F do not have a point in common except their endpoints. We assume that j is visible from i if $ij \in F$. And we denote a set of vertices of P_i visible from i with respect to F by $P_i(F)$.

Let $\text{conv}(P')$ be a convex hull of a point set P' . For a vertex i with $i \notin \text{conv}(P')$, $j \in P'$ is visible from i with respect to (the boundary of) $\text{conv}(P')$ when $j = ij \cap \text{conv}(P')$ holds (see Fig. 1).

For an edge $e = ij$, let $l(e)$ and $r(e)$ denote the left and right endpoints of e , i.e. $l(e) = i$ and $r(e) = j$, respectively. A straight line passing through i and j splits \mathbf{R}^2 into two regions that are open regions of left and right sides of ij , i.e. $R^+(ij) = \{p \in \mathbf{R}^2 \mid \Delta(i, j, p) > 0\}$ and $R^-(ij) = \{p \in \mathbf{R}^2 \mid \Delta(i, j, p) < 0\}$. Similarly, the closed regions $\bar{R}^+(ij)$ and $\bar{R}^-(ij)$ are defined.

For $i \in P$, let $F(i) \subseteq F$ denote a set of constrained edges whose left endpoints coincide with i . *Upper* and *lower tangents*, ii^{up} and ii^{low} , of i with respect to (the constrained edge set) F are defined as those from i to the convex hull of $P_i(F)$, (see Fig. 2). Notice that $P_i(F) \subset \bar{R}^-(ii^{\text{up}})$ and $P_i(F) \subset \bar{R}^+(ii^{\text{low}})$ hold. Then they define *empty region* in which no point of P exists as we describe below. Let l be a line perpendicular to x -axis passing through i , and let e_1 and e_2 be the closest edges from i among F intersecting with l in the upper and lower side of i (if such edge exists). Then there exists no point of P inside the region bounded by l , e_1 (resp. e_2), and the line through i and ii^{up} (resp. ii^{low}). When e_1 (resp. e_2) does not exist, the empty region is defined by the one bounded by l and the line through i and ii^{up} (resp. ii^{low}). We call this fact *empty region property* of the upper and lower tangents.

2.2 Constrained Smallest Indexed Triangulation

Although we have not explain our *index* for an triangulation yet, let us first define the triangulation called F -constrained smallest indexed triangulation in this paper.

Definition 1. Let ii^{up} and ii^{low} be the upper and lower tangents of $i \in P$ with respect to F , and denote a set of edges of $F(i) \cup \{ii^{\text{up}}, ii^{\text{low}}\}$ by ii_0, ii_1, \dots, ii_k arranged in clockwise order around i , (where $i_0 = ii^{\text{up}}$ and $i_k = ii^{\text{low}}$ hold). Let

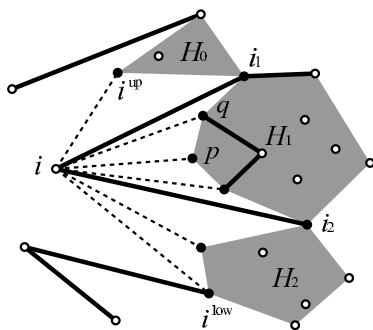


Fig. 3. The part of the CSIT around i , where bold edges represent F . The CSIT has edges between i and black vertices.

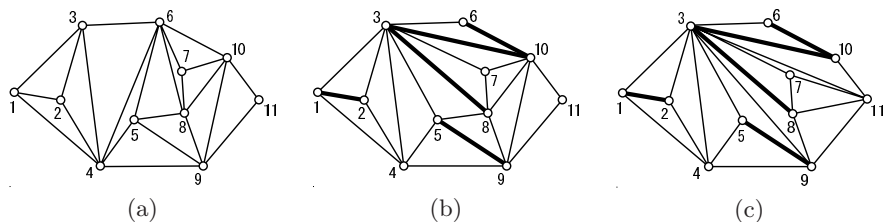


Fig. 4. (a)SIT. (b)CSIT. (c)non-CSIT.

$H_l = \text{conv}(P_i(F) \cap \bar{R}^-(ii_l) \cap \bar{R}^+(ii_{l+1}))$ for l with $0 \leq l \leq k - 1$. Then for every l with $0 \leq l \leq k - 1$ the F -constrained smallest indexed triangulation (F -CSIT) has an edge ij with $j \in H_l$ if and only if j is visible from i with respect to the convex hull H_l (see Fig. 3).

We call H_l of Definition 1 the convex hull of $P_i(F)$ bounded by ii_l and ii_{l+1} . We give an example of CSIT in Fig 4, and also give an example when $F = \emptyset$. Notice that CSIT always has the edges of $F(i) \cup \{ii^{\text{up}}, ii^{\text{low}}\}$ for all $i \in P$.

Lemma 1. *The CSIT is a triangulation of the point set P .*

2.3 Greedy Flipping in Constrained Triangulations

Let T^* denote CSIT on a given point set. For any F -constrained triangulation, an *index* of T is defined as a pair of integers $n - c$ and d , and denoted by $\text{index}(T) = (n - c, d)$, where $c \in \{1, \dots, n - 1\}$ and $d \in \{1, \dots, n - 3\}$ are a label of a *critical vertex* of T and a *critical degree* of the critical vertex, respectively. The *critical vertex* is the smallest label of a vertex whose incident edges differ from the corresponding set of incident edges in T^* . The *critical degree* is the number of edges incident to the critical vertex not contained in T^* . The index of T^* is defined to be $(0, 0)$. Then, for two triangulations T and T' with $\text{index}(T) =$

$(n - c, d)$ and $\text{index}(T') = (n - c', d')$, T has smaller index than that of T' when $c > c'$, or $c = c'$ and $d < d'$ holds. Note that the index decreases as the label of the critical vertex increases. For example, a triangulation in Fig. 4(c) has an index $(8, 2)$.

For an edge e in a triangulation T , e is *flippable* when two triangles incident to e in T form a *convex* quadrilateral Q . *Flipping* e in T generates a new triangulation by replacing e of T with the other diagonal of Q . Such operation is called *improving flip* if the triangulation obtained by flipping e has a smaller index than the previous one, and e is called *improving flippable*. Now let us show the greedy flipping property of the constrained triangulations.

Lemma 2. *Let T be the F -constrained triangulation with $T \neq T^*$ and c be the critical vertex of T . Then there exists at least one improving flippable edge incident to c in $T \setminus T^*$.*

Proof. Let $F(c)$ and $T^*(c)$ be sets of edges of F and T^* whose left endpoints coincide with c , respectively. And let cc^{up} and cc^{low} be the upper and lower tangents of c with respect to F . Now we show that T contains all edges of $T^*(c)$. First let us show that T contains $cc^{\text{up}} (\in T^*(c))$. Suppose that cc^{up} is missing in T . Since T is a triangulation, T has some edge $e \in T \setminus T^*$ intersecting cc^{up} . Then, from the empty region property of cc^{up} discussed in Section 2.1, $l(e) < c$ holds, which implies that the vertex $l(e)$ is incident to $e (\notin T^*)$ and contradicts that c is the critical vertex. (The fact that cc^{low} is contained in T can be similarly proved.)

Next let us show that each edge cv of $T^*(c)$ other than $F(c) \cup \{cc^{\text{up}}, cc^{\text{low}}\}$ is contained in T . Suppose that cv is missing in T . Then there exists some edge $e \in T \setminus T^*$ intersecting cv . Let cc_0, \dots, cc_k be the edges of $F(c) \cup \{cc^{\text{up}}, cc^{\text{low}}\}$ arranged in clockwise ordering around c . Since $cv \in T^*(c)$, there exists a unique l with $0 \leq l \leq k - 1$ of the convex hull of $P_c(F)$ bounded by cc_l and cc_{l+1} such that v is on the boundary of the convex hull. Then the edges cc_l, cc_{l+1} and the part of the boundary edges of such convex hull (convex chain) from c_l to c_{l+1} forms a pseudo-triangle with three corners c, c_l and c_{l+1} . Since there exists no point of P inside of such pseudo-triangle, the fact that e intersects cv implies that e also intersects at least one of cc_l and cc_{l+1} , which contradicts that T contains all edges of $F(c) \cup \{cc^{\text{up}}, cc^{\text{low}}\}$. Hence T contains $T^*(c)$.

Now let us show that there exists at least one improving flippable edge $e^* \notin T^*$ incident to c . Since c is a critical vertex, T has at least one edge $e' \notin T^*$. Let cp_1 and cp_2 be a pair of edges of $T^*(c) (\subset T)$ such that e' exists between cp_1 and cp_2 and an angle $\angle p_1cp_2$ is minimum for all pairs of $T^*(c)$ (see Fig. 5). Consider a set of edges in T incident to c between cp_1 and cp_2 , and denote them by $cq_1, cq_2, \dots, cq_{\bar{j}}$ in clockwise order around c . Note that $cq_j \in T \setminus T^*$ holds for all $j = 1, \dots, \bar{j}$, and then no vertex of q_j is inside of the triangle Δcp_1p_2 , since T^* contains empty triangle face Δcp_1p_2 . Therefore, all edges $cq_1, \dots, cq_{\bar{j}}$ intersect p_1p_2 . Let q_{j^*} be a vertex furthest from the line through p_1 and p_2 among q_j . Then a quadrilateral $cq_{j^*-1}q_{j^*}q_{j^*+1}$ is convex because q_{j^*-1}, q_{j^*} and q_{j^*+1} are not colinear, and flipping cq_{j^*} produces a triangulation with a smaller index than the previous one because $c < q_{j^*-1}$ and $c < q_{j^*+1}$ hold now. \square

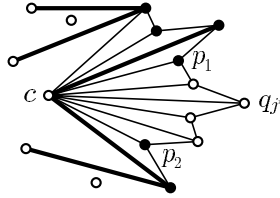


Fig. 5. Existence of an improving flippable edge cq_{j^*} . Bold edges represent the edges of F , and black vertices represent the vertices incident to c in T^* .

Theorem 1. Every F -constrained triangulation T can be transformed into F -CSIT by $O(n^2)$ flips.

Proof. From Lemma 2, $T (\neq T^*)$ always has an improving flippable edge, and flipping such edge reduces $\text{index}(T)$. Since the number of distinct indices is $O(n^2)$, T can be transformed into F -CSIT by $O(n^2)$ improving flips. \square

3 Constrained Non-crossing Spanning Trees

Let F be a non-crossing edge set on P , and we assume that F is a forest. In this section we show that a set of F -constrained non-crossing spanning trees on P , denoted by \mathcal{ST} , is connected by $O(n^2)$ flips.

Let $E = \{e_1 \prec \dots \prec e_m\}$ and $E' = \{e'_1 \prec \dots \prec e'_m\}$ be lexicographically ordered edge lists. Then E is lexicographically smaller than E' if $e_i \prec e'_i$ for the smallest i such that $e_i \neq e'_i$. Consider the F -CSIT, which is denoted by $T^*(F)$ in what follows. F -constrained smallest indexed spanning tree (F -CSIST) is a F -constrained non-crossing spanning tree that is a subset of $T^*(F)$, and we denote a set of all F -CSISTs by \mathcal{CSIST} . Define ST^* as a spanning tree consisting of the lexicographically smallest edge list among \mathcal{CSIST} . The following lemma holds from the known fact about matroid (see e.g. [12]). Namely each $ST \in \mathcal{CSIST}$ is a base of graphic matroid restricted to the edge set of $T^*(F)$ (see the proof of Theorem 3 of [7] for more details).

Lemma 3. Every non-crossing spanning tree of \mathcal{CSIST} can be transformed into ST^* by at most $n - 1$ flips.

Now we will define an index for each spanning tree $ST \notin \mathcal{CSIST}$ to represent how far it is from one of \mathcal{CSIST} . For each F -constrained triangulation T we have defined its index with respect to F by $\text{index}_F(T) = (n - c, d)$, which represents how far T is from $T^*(F)$ by means of the critical vertex c and the critical degree d , (see the definitions in Section 2.3). We associate ST -constrained smallest indexed triangulation $T^*(ST)$ with each spanning tree ST , and define an index of ST denoted by $\text{index}(ST) = (n - c_{ST}, d_{ST})$ as $\text{index}_F(T^*(ST))$. We also call c_{ST} the critical vertex of ST . Fig. 6 shows an example of ST whose critical vertex is 1 and $\text{index}(ST)$ is $(7, 2)$.

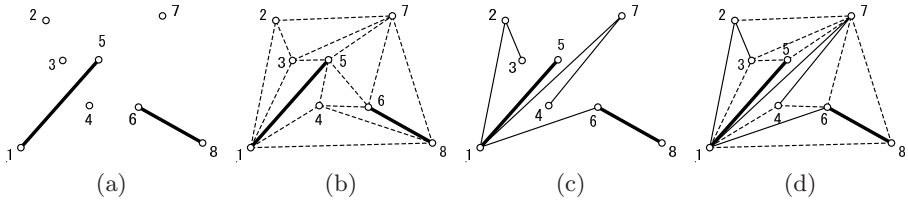


Fig. 6. (a) F , (b) $T^*(F)$, (c) ST , and (d) $T^*(ST)$, where bold edges represent F and dotted edges represent added edges for triangulations

For $i \in P$, let $ST(i)$ and $T^*(ST; i)$ be the edge sets of ST and $T^*(ST)$ whose left endpoints coincide with i , respectively. It is clear from the definition of CSIT (Definition 1) that the newly added edges for obtaining $T^*(ST)$ from ST are not flippable in $T^*(ST)$ except for the upper and lower tangents. Thus the next observation follows:

Observation 1. For $i \in P$ any edge of $T^*(ST; i) \setminus (ST(i) \cup \{ii^{up}, ii^{low}\})$ is not flippable in $T^*(ST)$, where ii^{up} and ii^{low} are upper and lower tangents of i with respect to ST .

Then we derive the followings from Lemma 2 and Observation 1

Lemma 4. Let $ST \notin CSIST$ and c be the critical vertex of ST . Then (i) there exists at least one improving flippable edge in $T^*(ST) \setminus T^*(F)$, and (ii) an edge $e \in T^*(ST)$ is improving flippable if and only if e is flippable and $e \in ST(c) \setminus (F(c) \cup \{cc^{up}, cc^{low}\})$, where cc^{up} and cc^{low} are upper and lower tangents of c with respect to ST .

Proof. Since $T^*(ST) \neq T^*(F)$ holds, (i) immediately follows from Lemma 2. Let us show (ii). From the definition of the index of $T^*(ST)$, we notice that flipping an improving flippable edge decreases the number of edges incident to c . Hence all improving flippable edges must be incident to c with their left endpoints. From Observation 1 any edge of $T^*(ST; c) \setminus (ST(c) \cup \{cc^{up}, cc^{low}\})$ is not flippable. Then the proof is completed by showing that neither cc^{up} nor cc^{low} are improving flippable edges. Suppose that cc^{up} is improving flippable. (The other case is similarly proved.) Then there exists a triangle face $\Delta cc^{up}v$ incident to cc^{up} in $T^*(ST)$ with $v \in R^+(cc^{up})$. Since cc^{up} is improving flippable edge, $c < v$ holds, and then $v \in P_c(ST)$. This contradicts the empty region property of cc^{up} with respect to ST . \square

Lemma 5. Every F -constrained non-crossing spanning tree $ST \notin CSIST$ can be transformed into a spanning tree in $CSIST$ by at most $O(n^2)$ flips.

Proof. Let c be a critical vertex of ST . From Lemma 4 there exists an edge $cc^* \in ST \setminus F$ that is improving flippable in $T^*(ST)$. There exist two vertices c_1^* and c_2^* incident to both c^* and c in $T^*(ST)$ such that $cc^*c_1^*$ and $cc^*c_2^*$ are triangle faces of $T^*(ST)$. When removing cc^* from ST , the set of vertices of $ST - cc^*$ is

partitioned into two components, where c^* and c belong to different components, and c_1^* can belong to only one of them. Therefore adding one of cc_1^* or $c_1^*c^*$ to $ST - cc^*$, we obtain a new non-crossing spanning tree ST' . Note that index of $T^*(ST')$ is smaller than that of $T^*(ST)$ because $T^*(ST')$ does not have cc^* but has $c_1^*c_2^*$ instead and the critical degree decreases, i.e. the edge-constraint of cc^* is released and improving flip occurs in the underlying triangulation. Repeating this procedure, the underlying triangulation eventually reaches the F -CSIT. \square

From Lemmas 3 and 5 we have the following theorem:

Theorem 2. *Every F -constrained non-crossing spanning tree is connected by $O(n^2)$ flips.*

4 Enumerating Constrained Non-crossing Spanning Trees

Let ST^* be an F -CSIST with the lexicographically smallest edge list as defined in Section 3. Let I_{ST} be a set of improving flippable edges in ST . We define the following *parent function* $f : ST \setminus \{ST^*\} \rightarrow ST$ based on the results of the previous section:

Definition 2. (*Parent function*) *Let $ST \in \mathcal{ST}$ with $ST \neq ST^*$, and c be a critical vertex of ST . $ST' = ST - e_1 + e_2$ is the parent of ST , where*

- Case 1: $ST \in \mathcal{CSIST}$,*
- $\bullet e_1 = \max\{e \mid e \in ST \setminus ST^*\}$, *and* $e_2 = \min\{e \in ST^* \setminus ST \mid ST - e_1 + e \in \mathcal{ST}\}$,
- Case 2: $ST \notin \mathcal{CSIST}$,*
- $\bullet e_1 = cc^* = \min\{e \in I_{ST}\}$, *and* e_2 *is either* cc_1^* *or* $c_1^*c^*$ *such that* $ST - e_1 + e_2 \in \mathcal{ST}$, *where* c_1^* *is a vertex of triangle face* $\Delta cc^*c_1^*$ *in* $T^*(ST)$ *with* $\Delta(c, c^*, c_1^*) > 0$.

Note that $I_{ST} \neq \emptyset$ and I_{ST} is the subset of $ST(c) \setminus F$ from Lemma 4. Therefore e_1 of Case 2 always exists. There exist two triangle faces, $\Delta cc^*c_1^*$ and $\Delta cc^*c_2^*$, incident to both cc^* in $T^*(ST)$. Here, we adopt c_1^* in order to define the unique parent. In Fig. 7 we show how the parent function works in Case 2. From Lemmas 3 and 5 these parent-child relationships form the search tree of \mathcal{ST} explained in Section 1. To simplify the notations, we denote the parent function depending on Cases 1 and 2 by $f_1 : \mathcal{CSIST} \setminus \{ST^*\} \rightarrow \mathcal{CSIST}$ and $f_2 : ST \setminus \mathcal{CSIST} \rightarrow ST$, respectively.

Let $\text{elist}_{ST'}$ and elist_{K_n} be the edge lists of ST' and K_n ordered lexicographically, and let $\text{elist}_{ST'}(i)$ and $\text{elist}_{K_n}(i)$ be their i -th element. Then, based on the algorithm in [5,6], we describe our algorithm in Fig. 8. The parent function needs $O(n + T_{\text{CSIT}})$ time for each execution, where T_{CSIT} denotes the time to calculate $T^*(ST)$. Then, the while-loop from lines 4 to 15 has $|ST'| \cdot |K_n|$ iterations which require $O(n^3(n + T_{\text{CSIT}}))$ time if simply checking lines 8 and 9. We can improve it to $O(n^2)$ time. Because of the lack of space, let us explain the outline and its rigorous description is omitted in this proceeding version.

Remember that the removing edge e_1 and the adding edge e_2 must share one endpoint in Case 2 of the parent function, so the inner while-loop from lines 6 to 14 can be reduced to $O(n)$ iterations. Then, to achieve $O(n^2)$ time algorithm,

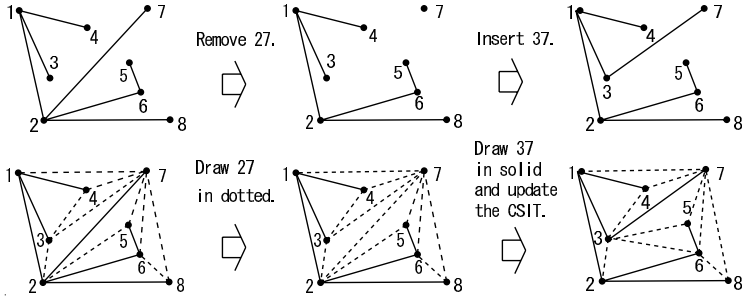


Fig. 7. An example of the parent function for $ST \notin CSIST$, where $index(ST) = (6, 2)$ and $F = \emptyset$. Removing 27 and adding 37 we obtain a new spanning tree with index $(6, 1)$.

Algorithm Enumerating F -constrained non-crossing spanning trees.

```

1:  $ST^* := F$ -CSIST with lexicographically smallest edge list;
2:  $ST' := ST^*$ ;  $i, j := 0$ ; Output( $ST'$ );
3: repeat
4:   while  $i \leq |ST'|$  do
5:     do  $\{i := i + 1; e_{rem} := elist_{ST'}(i); \}$  while ( $e_{rem} \in F$ );
6:     while  $j \leq |K_n|$  do
7:       do  $\{j := j + 1; e_{add} := elist_{K_n}(j); \}$  while ( $e_{add} \in ST'$ );
8:       if  $ST' - e_{rem} + e_{add} \in ST$  then
9:         if  $f_1(ST' - e_{rem} + e_{add}) = ST'$  or  $f_2(ST' - e_{rem} + e_{add}) = ST'$  then
10:           $ST' := ST' - e_{rem} + e_{add}$ ;  $i, j := 0$ ; Output( $ST'$ );
11:          go to line 4;
12:        end if
13:      end if
14:    end while
15:  end while
16:  if  $ST' \neq ST^*$  then
17:     $ST := ST'$ ;
18:    if  $ST \in CSIST$  then  $ST' := f_1(ST)$ ; else  $ST' := f_2(ST)$ ;
19:    determine integers pair  $(i, j)$  such that  $ST' - elist_{ST'}(i) + elist_{K_n}(j) = ST$ ;
20:     $i := i - 1$ ;
21:  end if
22: until  $ST' = ST^*$  and  $i = |ST'|$  and  $j = |K_n|$ ;

```

Fig. 8. Algorithm for enumerating F -constrained non-crossing spanning trees

we need to show that the inner-while loop can be implemented in $O(n)$ time. Let ST and ST' be two distinct F -CNSTs for which $ST = ST' - e_{rem} + e_{add}$ for $e_{rem} \in ST' \setminus F$ and $e_{add} \in K_n \setminus ST'$. Our goal is to enumerate all the edge pairs of (e_{rem}, e_{add}) such that ST is a child of ST' in linear time for each e_{rem} . More precisely we will describe necessary and sufficient conditions for which e_{rem} and e_{add} satisfy either $f_1(ST) = ST'$ or $f_2(ST) = ST'$. And then for each

$e_{\text{rem}} \in ST' \setminus F$ we can enumerate, in $O(n)$ time with $O(n)$ space, a set of edge pairs satisfying these conditions. Since the number of candidates for e_{rem} is $O(n)$, we have the following theorem:

Theorem 3. *The set of all F -constrained non-crossing spanning trees on a given point set can be reported in $O(n^2)$ time per output using $O(n)$ space.*

Acknowledgment

We would like to thank Professor David Avis and Professor Ileana Streinu for their contributions of crucial ideas and many research discussions. This work is supported by the project *New Horizons in Computing*, Grant-in-Aid for Scientific Research on Priority Areas, NEXT Japan.

References

1. Aichholzer, O., Aurenhammer, F., Huemer, C., Krasser, H.: Transforming spanning trees and pseudo-triangulations. *Inf. Process. Lett.* 97(1), 19–22 (2006)
2. Aichholzer, O., Aurenhammer, F., Huemer, C., Vogtenhuber, B.: Gray code enumeration of plane straight-line graphs. In: *Proc. 22th European Workshop on Computational Geometry (EuroCG '06)*, pp. 71–74. Greece (2006)
3. Aichholzer, O., Aurenhammer, F., Hurtado, F.: Sequences of spanning trees and a fixed tree theorem. *Comput. Geom.* 21(1-2), 3–20 (2002)
4. Aichholzer, O., Reinhardt, K.: A quadratic distance bound on sliding between crossing-free spanning trees. In: *Proc. 20th European Workshop on Computational Geometry (EWCG04)*, pp. 13–16 (2004)
5. Avis, D., Fukuda, K.: A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete and Computational Geometry* 8, 295–313 (1992)
6. Avis, D., Fukuda, K.: Reverse search for enumeration. *Discrete Applied Mathematics* 65(1-3), 21–46 (1996)
7. Avis, D., Katoh, N., Ohsaki, M., Streinu, I., Tanigawa, S.: Enumerating constrained non-crossing minimally rigid frameworks, http://arxiv.org/PS_cache/math/pdf/0608/0608102.pdf
8. Bespamyatnikh, S.: An efficient algorithm for enumeration of triangulations. *Comput. Geom. Theory Appl.* 23(3), 271–279 (2002)
9. Hernando, M.C., Houle, M.E., Hurtado, F.: On local transformation of polygons with visibility properties. In: Du, D.-Z., Eades, P., Sharma, A.K., Lin, X., Estivill-Castro, V. (eds.) *COCOON 2000*. LNCS, vol. 1858, pp. 54–63. Springer, Heidelberg (2000)
10. Hernando, C., Hurtado, F., Noy, M.: Graphs of non-crossing perfect matchings. *Graphs and Combinatorics* 18(3), 517–532 (2002)
11. Hurtado, F., Noy, M., Urrutia, J.: Flipping edges in triangulations. *Discrete & Computational Geometry* 22(3), 333–346 (1999)
12. Welsh, D.J.A.: Matroids: fundamental concepts. In: Graham, R.L., Grötschel, M., Lovász, L. (eds.) *Handbook of Combinatorics*, vol. I, pp. 481–526. North-Holland, Amsterdam (1995)

Colored Simultaneous Geometric Embeddings^{*}

U. Brandes¹, C. Erten², J. Fowler³, F. Frati⁴, M. Geyer⁵, C. Gutwenger⁶,
S. Hong⁷, M. Kaufmann⁵, S.G. Kobourov³, G. Liotta⁸,
P. Mutzel⁶, and A. Symvonis⁹

¹ Department of Computer & Information Science, University of Konstanz
`ulrik.brandes@uni-konstanz.de`

² Department of Computer Science, Isik University
`cesim@isikun.edu.tr`

³ Department of Computer Science, University of Arizona
`{jfowler,kobourov}@cs.arizona.edu`

⁴ Department of Computer Science, University of Roma Tre
`frati@dia.uniroma3.it`

⁵ Wilhelm-Schickard-Institute of Computer Science, University of Tübingen
`{geyer,mk}@informatik.uni-tuebingen.de`

⁶ Department of Computer Science, University of Dortmund
`{petra.mutzel,carsten.gutwenger}@cs.uni-dortmund.de`

⁷ NICTA Ltd. and School of Information Technologies, University of Sydney
`seokhee.hong@nicta.com.au`

⁸ School of Computing, University of Perugia
`liotta@diei.unipg.it`

⁹ School of Applied Math. & Phys. Sciences, National Technical University of Athens
`symvonis@math.ntua.gr`

Abstract. We introduce the concept of *colored simultaneous geometric embeddings* as a generalization of simultaneous graph embeddings with and without mapping. We show that there exists a universal pointset of size n for paths colored with two or three colors. We use these results to show that colored simultaneous geometric embeddings exist for: (1) a 2-colored tree together with any number of 2-colored paths and (2) a 2-colored outerplanar graph together with any number of 2-colored paths. We also show that there does not exist a universal pointset of size n for paths colored with five colors. We finally show that the following simultaneous embeddings are not possible: (1) three 6-colored cycles, (2) four 6-colored paths, and (3) three 9-colored paths.

1 Introduction

Visualizing multiple related graphs is useful in many applications, such as software engineering, telecommunications, and computational biology. Consider the case where a pair of related graphs is given and the goal is to visualize them so as to compare the two, e.g., evolutionary trees obtained by different algorithms.

^{*} Work on this paper began at the BICI Workshop on Graph Drawing, held in Bertinoro, Italy in March 2006.

When visually examining relational information, such as a graph structure, viewers construct an internal model called the mental map, for example, using the positions of the vertices relative to each other. When viewing multiple graphs the viewer has to reconstruct this mental map after examining each graph and a common goal is to aid the viewer in this reconstruction while providing a readable drawing for each graph individually. Simultaneous embeddings [4] aid in visualizing multiple relationships between the same set of objects by keeping common vertices and edges of these graphs in the same positions.

A simultaneous geometric embedding is a generalization of the traditional planar graph embedding problem, where we look for a common embedding of multiple graphs defined on the same vertex set. We omit the “geometric” clarification in the rest of the paper as we only consider straight-line drawings. There are two main variations of the problem. In *simultaneous embedding with mapping* the embedding consists of plane drawings for each of the given graphs on the same set of points, with corresponding vertices in the different graphs placed at the same point. In *simultaneous embedding without mapping* the embedding consists of plane drawings for each of the given graphs on the same set of points, where any vertex can be placed at any of the points in the point set.

Restricted subclasses of planar graphs, such as pairs of paths, pairs of cycles, and pairs of caterpillars, admit a simultaneous embedding with mapping, while there exist pairs of outerplanar graphs and triples of paths that do not [4]. Recently, it was shown that pairs of trees do not always have such embeddings [9]. Fewer results are known for the less restricted version of the problem where the mapping is not predefined. While it is possible to simultaneously embed without mapping any planar graph with any number of outerplanar graphs, it is not known whether any pair of planar graphs can be simultaneously embedded without mapping [4].

Simultaneous embedding is related to universal pointsets, graph thickness, and geometric thickness. While de Fraysseix *et al.* [6] showed that there does not exist a universal pointset of size n in the plane for n -vertex planar graphs, Bose [3] showed that a set of n points in general position is a universal pointset for trees and outerplanar graphs. Using simultaneous embedding techniques, Duncan *et al.* [8] showed that degree-four graphs have geometric thickness two.

As we show, colored simultaneous embeddings allow us to generalize the problems above so that the versions with and without mappings become special cases. Formally, the problem of *colored simultaneous embedding* is defined as follows. The input is a set of planar graphs $G_1 = (V, E_1), G_2 = (V, E_2), \dots, G_r = (V, E_r)$ on the same vertex set V and a partition of V into k classes, which we refer to as *colors*. The goal is to find plane straight-line drawings D_i of G_i using the same $|V|$ points in the plane for all $i = 1, \dots, r$, where vertices mapped to the same point are required to be of the same color.

We call such graphs k -colored graphs. Given the above definition, simultaneous embeddings with and without mapping correspond to colored simultaneous embeddings with $k = |V|$ and $k = 1$, respectively. Thus, when a set of input graphs allows for a simultaneous embedding without mapping but does not

allow for a simultaneous embedding with mapping, there must be a threshold for the number of colors beyond which the graphs can no longer be embedded simultaneously.

In this paper we present the first results about colored simultaneous embeddings. We study different values of k and show that any line-separated set of points of size n is a universal pointset for n -vertex 2-colored paths. Moreover, there exists a universal pointset of size n for n -vertex 3-colored paths while there is no such universal pointset n -vertex 5-colored paths. We also show how to simultaneously embed a 2-colored outerplanar graph and any number of 2-colored paths. Finally we show the existence of three 6-colored cycles (or four 6-colored paths, or three 9-colored paths) that cannot be simultaneously embedded.

2 Two-Colored Simultaneous Embeddings

We begin by showing the existence of a universal pointset for 2-colored paths. The following lemma extends a result of Abellanas *et al.* [1] on proper 2-colorings of paths.

Lemma 1. *Given a 2-colored path P of r red and b blue vertices and a set S of r red and b blue points separated by a line and in general position, there exists a planar straight-line embedding of P into S .*

Proof. Without loss of generality we can assume that S is separated by a vertical line, and that the red points are on the left of that line. Let $P = v_0, v_1, \dots, v_n$ and let P_i be the drawing of the path after the first i vertices of P have been embedded. Let H_i be the lower convex envelope of the points of S not used by P_i . We maintain the following invariants for all $i = 0, \dots, n - 1$ for which the colors of v_i and v_{i+1} are different:

1. The drawing of P_i does not intersect H_i .
2. The point p_i into which the most recent vertex v_i has been embedded can see a point of H_i of the other color and P_i does not intersect the area bounded by this line of sight and the vertical line from p_i upward.

Assume that v_i is of different color than v_{i+1} and let h , $1 \leq h \leq n - i$, be maximal such that $v_{i+1}, v_{i+2}, \dots, v_{i+h}$ all have the same color. To maintain the above invariants, we find a line that cuts off the required number h of points of color different from v_i from H_i (identified with the area on and above it). Assume v_i is red (which implies that it has been placed at a point p_i in the left half-plane) and v_{i+1} is blue; see Fig. 1.

Consider now the red end-point r_i of the unique edge of H_i that crosses the vertical separation line. We rotate a ray emanating from r_i counterclockwise until either h unused blue points are encountered, or a red point r'_i lies on the ray. In the latter case, we continue by rotating counterclockwise the ray around r'_i . We repeat this process until h blue points are found, and let B_i be the set of identified blue points. Let C_{B_i} be the convex hull of B_i . These points can be added to the path, as follows: Let a be the first blue point of H_i that is hit by a ray emanating

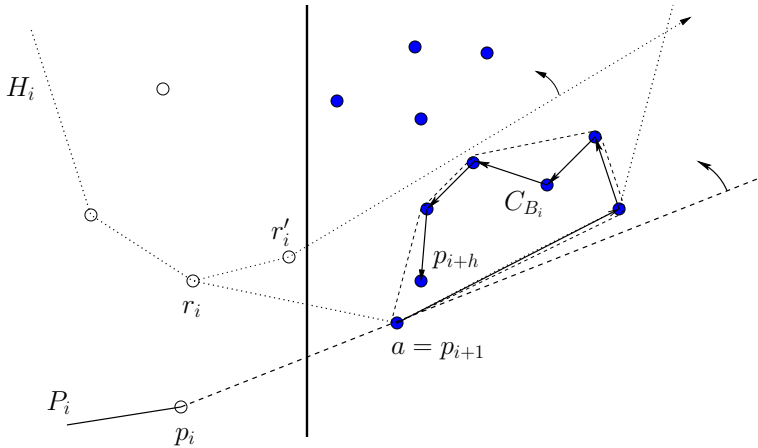


Fig. 1. Embedding a 2-colored path

from p_i and rotated counterclockwise. Point a also belongs to C_{B_i} . We can then connect p_i to point a . From point a we move counterclockwise along C_{B_i} until the right-most point of C_{B_i} is reached, while adding each encountered point to the drawing of the path. The remaining points of B_i are taken in decreasing value of their x -coordinates until the final point, p_{i+h} .

The resulting path ending at p_{i+h} satisfies the invariants: P_{i+h} does not intersect H_{i+h} and since p_{i+h} is the leftmost point of B_i the second invariant is also satisfied. \square

Using Lemma \square we can embed k 2-colored paths for any $k > 0$ on a set of 2-colored points in general position in the plane that are separated by a straight-line, provided we have sufficient number of points of each color. The resulting set of points is a universal one for these k 2-colored paths, which yields the following theorem:

Theorem 1. *Any number of 2-colored paths can be simultaneously embedded.*

2.1 A Tree and Paths on Two Colors

We first show that it is always possible to draw a 2-colored tree in such a way that the two colors are separated by a line.

Lemma 2. *Any 2-colored tree can be embedded so that the colors are separated by a straight line.*

Proof. We use a divide-and-conquer approach and recursively process the tree from an arbitrary root node. We begin by drawing a vertical line l and assigning the left side to color 1 and the right side to color 2. Next we sort the children of the root by their colors. Let j of the children have color 1 and k children have color 2.

We can assume without loss of generality that the root is of color 1 and can place it on the left side of line l . The j children of color 1 are placed consecutively, such that the first is strictly beneath and to the left of the root, the second is strictly beneath and to the left of the first, and so on. We place the k children of color 2 to the right of line l in a similar fashion. We place the first child strictly beneath and to the right of the root, the second strictly beneath and to the right of the first, and so on. Note that every child has unobstructed line of sight to an horizontal sliver of the plane on both sides of line l . Thus, we can recursively place the children of the $j + k$ vertices until the entire tree has been processed. \square

Now using the result from Lemma 2 we can embed a 2-colored tree on a set of 2-colored points in the plane that are separated by a straight-line. Then we can perturb the positions of the vertices until they are in general position. This can be done without introducing crossings as shown in 4. From Lemma 1, the resulting set of points is a universal one for 2-colored paths. Together these two results yield the next theorem:

Theorem 2. *A 2-colored tree and any number of 2-colored paths can be simultaneously embedded.*

2.2 Planar Graph and Paths on Two Colors

We have seen that in order to simultaneously embed a 2-colored planar graph G with any number of 2-colored paths it suffices to find a plane drawing of G in which the vertex sets of the same color, V_1 and V_2 , can be separated by a line. Let G_1 and G_2 be the two subgraphs induced by the vertex sets V_1 and V_2 respectively. We call such a partition a *bipartition*, and the edges with vertices from both graphs are called *bipartition edges*.

Next we present a characterization of the class of 2-colored planar graphs that can be separated by a line. We make extensive use of the characterization and the embedding algorithm for HH layouts by Biedl *et al.* 2. An HH layout is a drawing of a planar bipartition without crossings (but not necessarily using straight-line edges), in which the two vertex sets are separated by a horizontal line. We begin with the characterization of planar bipartitions that can be drawn as HH layouts.

Lemma 3. 2] *Planar bipartitions can be realized as HH layouts only if the subgraph D of the dual graph induced by the dual edges of the bipartition edges is connected.*

Moreover, it is shown in 2] that D is Eulerian and that it is possible to construct y -monotone HH layouts with few bends in linear time. The construction is roughly as follows. Find an Eulerian circuit of D that separates the sets V_1 and V_2 . Then dummy vertices, that will become bends later, are introduced along the bipartition edges. Next the chain of dummy vertices is processed in the order of the Eulerian circuit and the straight-line drawing algorithm of Chrobak and Kant 5] is applied to the two subgraphs separately by placing one of them

below (without loss of generality, say, G_1) and the other above the chain. The final result is straight-line planar drawing with the exception of the bipartition edges which have exactly one bend each; see Fig. 2(a).

This approach does not produce exactly the result that we need. We now show how to obtain a drawing with no bends, while not introducing any crossings, after applying the above technique to the planar bipartition and obtaining the HH layout (which may have some bends).

Lemma 4. *From each HH layout with some bends on the separation line, we can derive a straight-line drawing, while keeping the two partitions separated by a line.*

Proof. We begin by directing all the edges upward with respect to the basic HH layout L in order to obtain an upward planar embedding E of G . A theorem of Di Battista and Tamassia [7] states that the upward planar embedding E can be realized as a straight-line upward drawing. The resulting drawing, however, may not separate the two sets by a straight horizontal line. Below we show how to obtain the needed straight-line drawing in which the two sets are indeed separable by a line.

Let Γ_1 be the upward embedding of the graph G_1 with an upper boundary B_1 made of vertices adjacent to the bipartition edges. We extend Γ_1 by adding a top vertex t which we connect to all the boundary vertices by edges (v, t) , where $v \in B_1$. Now we can apply the straight line drawing algorithm of Di Battista and Tamassia to the extended embedding and obtain an upward straight-line drawing, with the vertices on the boundary B_1 drawn with increasing x -coordinates; see Fig 2(b). After removing vertex t , B_1 is once again the upper boundary. Similarly, we can extend the embedding Γ_2 of G_2 in order to obtain a drawing with x -monotone lower boundary B_2 .

Next we stretch the two layouts in the x -direction so that the slopes of the boundary edges become smaller. In particular, we stretch the layouts until all slopes are less than 40° . Note that stretching preserves both planarity and upwardness of the layouts.

Finally we place the two layouts of Γ_1 and Γ_2 above each other and at vertical distance twice the larger of their widths. Now we can safely insert the bipartition edges which connect the two boundaries B_1 and B_2 . By the choice of separation distance, the slopes of the bipartition edges are larger than 60° . Thus the bipartition edges cannot introduce any crossings and now the two parts can be separated by an horizontal line as desired; see Fig. 2(c). □

Lemma 4 and the algorithm above yield the following lemma:

Lemma 5. *Let G be a planar bipartition graph in which the dual graph of the subgraph induced by the bipartition edges is connected. (a) Then a straight-line drawing for G can be constructed where the two parts are separated by a horizontal line. (b) Since the bipartition includes a 2-coloring, G plus any number of 2-colored paths can be simultaneously embedded.*

As 2-colored outerplanar graphs fulfill the conditions of Lemma 5, we have the following theorem:

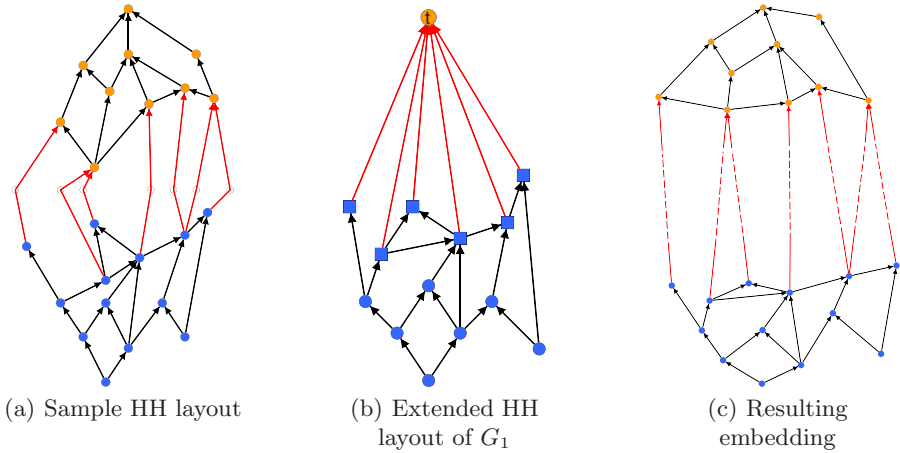


Fig. 2. HH Layouts

Theorem 3. *A 2-colored outerplanar graph and any number of 2-colored paths can be simultaneously embedded.*

3 k -Colored Simultaneous Embeddings

In this section we extend the investigation to more than two colors. We recall that there exist three paths which do not admit a simultaneous embedding with mapping [4], whereas it is easy to see that any number of paths have a simultaneous embedding without mapping. Now we consider k -colored paths and/or k -colored k -cycles for $3 \leq k \leq 9$.

3.1 Three Colors

As in the case of 2-colored embeddings we are looking for a universal pointset for paths. A slight modifications of the original universal pointset for 2-colored paths allows us to extend its utility to the 3-colored case.

Theorem 4. *Any number of 3-colored paths can be simultaneously embedded.*

Proof. Let P be any 3-colored path with c_1 vertices of color 1, c_2 vertices of color 2 and c_3 vertices of color 3, where $c_1 + c_2 + c_3 = n$. Let l_1, l_2 and l_3 be three line-segments with a common endpoint O and meeting at 120° angle. Place c_1 points along l_1 , c_2 points along l_2 , and c_3 points along l_3 , ensuring that the origin O is not used.

Next map every vertex of the path, in order, to the point of the corresponding color that is closest to the origin and is not already taken. Since every point has line of sight to any other point and for a given p_i of P the previous path only blocks line of sight to the points already taken, the result is a plane drawing. \square

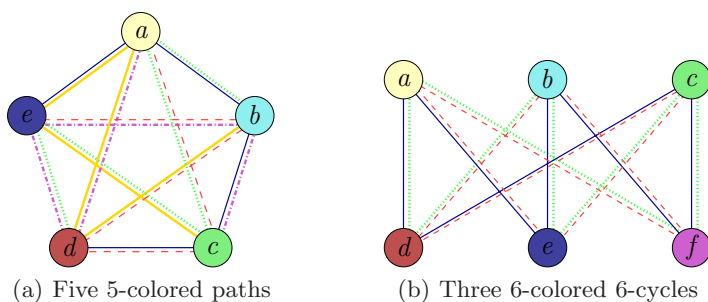


Fig. 3. Sets of k -colored graphs for $k \in \{5, 6\}$ on distinctly colored points whose unions form a K_5 and a $K_{3,3}$

3.2 Four and Five Colors

While universal pointsets exist for 1-colored paths, 2-colored paths and 3-colored paths, we have not been able to find one for 4-colored paths. However, we can show that for $k > 4$ universal pointsets for k -colored paths do not exist.

Theorem 5. *There does not exist a universal pointset for 5-colored paths.*

Proof. Consider the following five 5-colored paths on 5 points given in Fig. 3(a) whose union is K_5 where each edge in the K_5 belongs to exactly two paths:

1. $a-c-d-b-e$ (thin red dashed edges),
2. $a-d-e-b-c$ (thick light purple alternating dash and dot edges),
3. $b-a-c-e-d$ (thick green dotted edges),
4. $b-d-a-e-c$ (thick yellow solid edges), and
5. $e-a-b-c-d$ (thin blue solid edges).

In any drawing of K_5 there must be at least one crossing. If this crossing is formed by a pair of edges from different paths then a simultaneous embedding might be possible. However, the paths above were chosen in such a way that every pair of edges either belongs to the same path or is incident. As straight-line incident edges cannot form the crossing pair it suffices to examine all pairs of non-adjacent edges in order to verify that they occur in at least one of the paths.

3.3 Six and Nine Colors

Here we consider sets of graphs on pointsets of six or more colors, in which the sets of graphs to simultaneously embed have cardinality less than five.

Lemma 6. *There exist three 6-colored cycles that cannot be simultaneously embedded.*

Proof. Consider the following three cycles, also shown in Fig. 3(b):

1. $e-a-d-c-f-b-e$ (thin blue solid edges),
2. $e-a-f-b-d-c-e$ (thin red dashed edges), and
3. $a-f-c-e-b-d-a$ (thick green dotted edges).

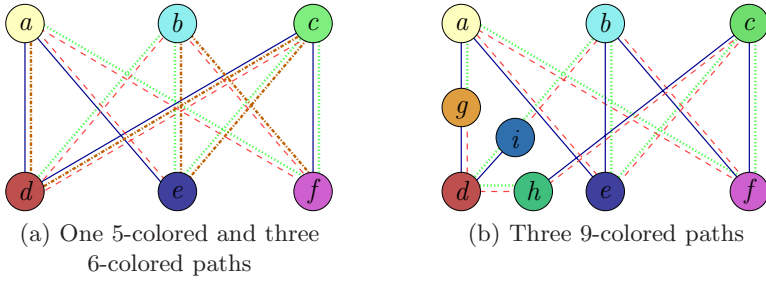


Fig. 4. Sets of k -colored graphs for $k \in \{6, 9\}$ on distinctly colored points whose unions form a $K_{3,3}$ or a subdivision thereof

A visual examination of Fig. 3(b) shows that the union of these cycles forms a $K_{3,3}$. Moreover, every edge in the $K_{3,3}$ belongs to two of the three cycles. In any drawing of $K_{3,3}$ there must be at least one crossing. Since there are only three paths altogether, every pair of edges in the $K_{3,3}$ must share a common 6-cycle, which forces a self-intersecting cycle. \square

Lemma 7. *There exist four 6-colored paths that cannot be simultaneously embedded.*

Proof. Fig. 4(a) depicts the following set of one 5-colored path and three 6-colored paths whose union forms $K_{3,3}$:

1. $e-a-d-c-f$ (thin blue solid edges),
2. $e-a-f-b-d-c$ (thin red dashed edges),
3. $a-f-c-e-b-d$ (thick green dotted edges), and
4. $a-d-c-e-b-f$ (thick brown dash-and-dots edges).

Every edge in $K_{3,3}$ belongs to at least two of the four paths. As a result, since there are more than three paths, it is easy to manually inspect all 18 pairs of non-adjacent edges to verify that each pair shares a common path. Thus at least one of the paths must be self-intersecting. \square

Lemma 8. *There exist three 9-colored paths that cannot be simultaneously embedded.*

Proof. Fig. 4(b) shows that every edge in the subdivided $K_{3,3}$ union belongs to exactly two of the following three paths:

1. $h-c-f-b-e-a-g-d-i$ (thin blue solid edges),
2. $g-d-h-c-e-a-f-b-i$ (thin red dashed edges), and
3. $g-a-f-c-e-b-i-d-h$ (thick green dotted edges).

Since there are only three 9-colored paths altogether, every pair of edges in the subdivided $K_{3,3}$ must share a common path forcing a self-intersecting path. Note that this result is a simplified version of Theorem 2 of Brass *et al.* 4. \square

4 Conclusions and Open Problems

Table 1 summarizes the current status of the newly formulated problem of colored simultaneous embedding. A “✓” indicates that it is always possible to simultaneously embed the type of graphs, a “✗” indicates that it is not always possible, and a “?” indicates an open problem.

Table 1. k -colored simultaneous embeddings: results and open problems

	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 9$	$k = n$
Paths $P_1 \dots P_3$	✓	✓	✓	?	?	?	✗	✗
Paths $P_1 \dots P_4$	✓	✓	✓	?	?	✗	✗	✗
Any number of paths	✓	✓	✓	?	✗	✗	✗	✗
Planar Graph G and Path P	✓	✓	?	?	?	?	✗	✗
Outerplanar Graph G and Path P	✓	✓	?	?	?	?	?	?
Tree T and Path P	✓	✓	?	?	?	?	?	?
Two trees T_1, T_2	✓	?	?	?	?	?	?	✗
Two planar graphs G_1, G_2	?	?	?	?	?	?	✗	✗

References

1. Welsh, D.J.A.: Matroids: fundamental concepts. In: Graham, R.L., Grötschel, M., Lovász, L. (eds.) Handbook of Combinatorics, vol. I, pp. 481–526. North-Holland, Amsterdam (1999)
2. Biedl, T., Kaufmann, M., Mutzel, P.: Drawing planar partitions: HH-drawings. In: 24th Workshop on Graph-Theoretic Concepts in Computer Science (WG), pp. 124–136 (1998)
3. Bose, P.: On embedding an outer-planar graph in a point set. Computational Geometry: Theory and Applications 23(3), 303–312 (2002)
4. Brass, P., Cenek, E., Duncan, C.A., Efrat, A., Erten, C., Ismailescu, D., Kobourov, S.G., Lubiw, A., Mitchell, J.S.B.: On simultaneous graph embedding. In: 8th Workshop on Algorithms and Data Structures (WADS), pp. 243–255 (2003)
5. Chrobak, M., Kant, G.: Convex grid drawings of 3-connected planar graphs. Intl. Journal of Computational Geometry and Applications 7(3), 211–223 (1997)
6. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica 10(1), 41–51 (1990)
7. Di Battista, G., Tamassia, R.: Algorithms for plane representation of acyclic digraphs. Theoretical Computer Science 61(2-3), 175–198 (1988)
8. Duncan, C.A., Eppstein, D., Kobourov, S.G.: The geometric thickness of low degree graphs. In: 20th Annual ACM-SIAM Symposium on Computational Geometry (SCG), pp. 340–346 (2004)
9. Kaufmann, M., Vrto, I., Geyer, M.: Two trees which are self-intersecting when drawn simultaneously. In: 13th Symposium on Graph Drawing (GD), pp. 201–210 (2005)

Properties of Symmetric Incentive Compatible Auctions*

Xiaotie Deng¹, Kazuo Iwama², Qi Qi¹, Aries Wei Sun¹, and Toyotaka Tasaka²

¹ Department of Computer Science, City University of Hong Kong
csdeng@cityu.edu.hk, qi.qi@student.cityu.edu.hk, sunwei@cs.cityu.edu.hk

² School of Informatics, Kyoto University, Japan
iwama@kuis.kyoto-u.ac.jp, TasakaToyotaka@t13.mbox.media.kyoto-u.ac.jp

Abstract. We formalize the definition of symmetric auctions to study fundamental properties of incentive compatible auction protocols. We characterize such auction protocols for those with a fixed number of items to sell and study some important properties of those with an indefinite number of sales.

1 Introduction

In recent years, auction based protocols have become a popular solution to E-commerce trading systems. Their adoption in real commercial processes have been at a scale never seen previously. Originally, auctions are used to sell normal items that are in limited supply [8,47]. Largely because of digital goods of zero marginal cost, auction models with unlimited supply [6,3,2,15] have become an important research topic in recent year. The study of digital goods has also invented new variations of auctions and provoked new thinkings in the principles governing auction protocol designs.

In particular, a new concept of “competitive auction” introduced in [6] made it the first priority that the total sales of the auctioneer be maximized, approximately within a constant factor of a benchmark optimum. To achieve that, several important properties are examined in details at a technical depth that has not been necessary for many previously well known auction protocols. At the same time, a nature question has arisen to demand an extensive careful study is that what are the basic principles we have to stick to in economic transactions at this micro level. Our study is motivated to study this fundamental question to understand the limitation and the possibility in auction protocols.

In designing auction protocols, a number of common properties are usually desired, though sometimes not explicitly stated. We should focus on incentive compatible protocols, for which bidding the trues value would be an optimal strategy of every participating agent. In addition, we will restrict our discussion to those with the *symmetric* property. We should use a specific definition of symmetric auction protocols. We comment that there may be other formalization

* This research is supported by SRG grant (7001989) of City University of Hong Kong.

of the concept symmetry for auction designs. Our choice is one of the simplest forms.

We study several other interesting properties to analyse their effects on incentive compatible auction protocol designs as well as their relationships with each other. We introduce those properties and present some preliminary results in §2.

As we shall expect, the *symmetric* property will *not* prevent lower bidders to win over higher bidders. Even though it might be possible for some special types of discriminative auctions, it is not “reasonable” and unfair in most situations. In §3 we define the property a bidder wins if any lower one does as “Winner Monotone”. We also define two other equally plausible monotone properties, “Price Function Monotone” and “Revenue Monotone”. Our investigations into the relationships of the three properties give very interesting results and show that we should not take for granted that an arbitrarily designed auction is “reasonable”.

Then an interesting question arises, when will an auction be “reasonable”? A most common benchmark auction is the celebrated Vickrey auction [8]. We study Vickrey auction in depth in §4. Our analysis show two *sufficient* and *necessary* conditions for a symmetric incentive compatible auction to be equivalent to Vickrey auction. One is that the number of winners is always fixed (See §4.1). Another is that the auction is homogeneous, revenue monotone, price function monotone, and single winning price (See §4.2).

We conclude our work in §5 with a discussion on future works.

2 Preliminaries

We present the models of auction protocols, the mathematical notations for agents’ parameters, as well as acronyms for important properties to be discussed.

2.1 Basic Model and Notations

We restrict our attentions to auctions satisfying the following properties:

1. Every bidder wants at most one item.
2. The items sold by the auction are the same.
3. The auction is deterministic.
4. The auction is carried out in a one-round sealed-bid manner.
5. The bidders know the auction protocol.

The auctions we study in this paper can be viewed as algorithms that take the bidders’ bids $(b^{(1)}, b^{(2)}, \dots, b^{(n)})$ as input, and, upon termination, give the result as output. The output consists of two parts. The first part is allocation $w^{(i)}$, $i = 1, 2, \dots, n$. $w^{(i)}$ indicates the number of items agent $a^{(i)}$ has won. In our model $w^{(i)}$ is either 0 or 1. We call an agent a **winner** if it has won at least one item, or a **loser** otherwise. The second part is price $p^{(i)}$, $i = 1, 2, \dots, n$. $p^{(i)}$ indicates the price that agent $a^{(i)}$ needs to pay for each unit. Notice that if agent $a^{(i)}$ is a loser that has won no item, it pays a price 0, while $p^{(i)}$ may not be equal to 0. The auctioneer’s revenue is defined as $\sum_{i=1}^n w^{(i)} \times p^{(i)}$.

For convenience, we use the following notations and terminologies throughout the paper:

- n the total number of bidders
- k the total number of winners
- $a^{(i)}$ the i -th agent (or bidder) in the auction
- $b^{(i)}$ the bid submitted by $a^{(i)}$. $b^{(i)} \geq 0$.
- $v^{(i)}$ the private valuation on the product of $a^{(i)}$. We restrict our discussion on normal goods characterized by $v^{(i)} \geq 0$.
- $\mathbf{b}^{(\sim i)} = (b^{(1)}, b^{(2)}, \dots, b^{(i-1)}, b^{(i+1)}, \dots, b^{(n)})$
- $w^{(i)}$ The number of items agent $a^{(i)}$ has won from the auction.
- $p^{(i)}$ The price agent $a^{(i)}$ should pay for *each unit*. $p^{(i)} \geq 0$.
- $u^{(i)}$ The utility of agent $a^{(i)}$. $u^{(i)} = (v^{(i)} - p^{(i)}) \times w^{(i)}$.

The one shot auction protocol takes the sealed-bids of the participants, determines an allocation policy of the selling item, as well as the prices charged to all the winners. For the incentive compatible property, the most fundamental property in the tradition of auction protocol design since Vickery [8], the description of allocation policies can be much simplified by a set of pricing functions, $f_i(\cdot)$, one each for the participating agents. $a^{(i)}$ wins the item if its bid $b^{(i)}$ is larger than $f_i(\cdot)$, loses otherwise. In the event bid $b^{(i)}$ is equal to $f_i(\cdot)$, the agent is called a zero winner or a zero loser depending on whether it is allocated with the selling item by the auction protocol.

2.2 Basic Properties and Their Notations

We introduce important properties and their acronyms so that they can be referred to conveniently.

Definition 1 (Symmetric Auction). *An auction is symmetric if and only if for any input:*

1. $b^{(i)} = b^{(j)} \Rightarrow p^{(i)} = p^{(j)}$.
2. $p^{(i)}$ remains the same if two other agents exchange their bids.

Definition 2 (Homogeneous Auction). *An auction is homogeneous if after multiply every bidder's bid by a factor $m > 0$, the result of the auction satisfies:*

1. A bidder wins if and only if it previously wins.
2. The return value of the price function of every bidder is equal to m times its previous value.

IR Individual rational. An auction is said to be individual rationality if $b^{(i)} < p^{(i)} \rightarrow w^{(i)} = 0, \forall i = 1, 2, \dots, n$.

IC Incentive compatible. Also called *truthful*. An auction is incentive compatible if and only if reporting truth is each agent (or bidder)'s dominant strategy.

SYM Symmetric. An auction is said to be symmetric if it satisfies Definition 1.

SEL(n,k) The auction is participated by n bidders and selects exactly k winners. $0 < k < n$ unless otherwise stated.

SWP Single Winning Price. An auction is said to be single winning price if and only if all winners' prices are the same, i.e. $w^{(i)} > 0$ and $w^{(j)} > 0 \Rightarrow p^{(i)} = p^{(j)}, \forall 1 \leq i < j \leq n$.

HOMO Homogeneous. An auction is homogeneous if it satisfies Definition 2.

2.3 Useful Results

Lemma 1. *In an auction satisfying **SEL**(n,k), there are at most k bidders with positive utilities.*

Lemma 2. *In an auction satisfying **IR** and **IC**,*

$$w^{(i)} \times (b^{(i)} - p^{(i)}) \geq 0, \forall i = 1, 2, \dots, n$$

where $w^{(i)}$ is the number of items agent $a^{(i)}$ has won, $b^{(i)}$ is the bid of agent $a^{(i)}$, $p^{(i)}$ is the price agent $a^{(i)}$ need to pay for each unit of items it has won.

We restate Theorem 2.5 on page 5 in [6] here:

Lemma 3 (Bid Independent Pricing. Folklore, see e.g. [6]). *In an auction satisfying **IR** and **IC**, the pricing function of $a^{(i)}$ does not depend on $b^{(i)}$.*

In other words, $a^{(i)}$'s price function does not take its bid $b^{(i)}$ as a variable. i.e.

$$p^{(i)} = f_i(b^{(\sim i)})$$

And for any bidder $a^{(i)}$, if $b^{(i)} > p^{(i)}$, $a^{(i)}$ is a winner.

From the above lemma and the definition of **SYM**, if $b^{(i)} = b^{(j)}$ and $a^{(i)}$ wins while $a^{(j)}$ loses, then we must have $p^{(i)} = b^{(i)}$.

Lemma 4. *In an auction satisfying **IR**, **IC** and **SYM**, every bidder $a^{(i)}$'s pricing function is independent of its index i .*

Given Lemma 4, we can remove the subscript from every bidder $a^{(i)}$'s price function $f_i(\cdot)$, and denote it by $f(\cdot)$. Thus, the output of the auction is independent of the order of the bidders. In the following parts, we always assume the bid vector is sorted in decreasing order.

3 Monotone Properties

In general, the symmetric property would not prevent lower bidders to win over higher bidders. Even though this may sometimes be acceptable for special types of discriminative auctions, it is not reasonable for symmetric auction where all bidders are regarded equal. To handle this abnormality, auction protocols are often required to be monotone.

However, careful examination into monotone properties shows that there may be different properties of monotonicity. We should distinguish three types of monotone properties:

1. **Winner Monotone (WM).** High bidder wins if any lower one does.
2. **Price Function Monotone (PFM).** The price function f is a non-decreasing function in each of its variables.
3. **Revenue Monotone (RM).** The total value of the sales is a non-decreasing function in each of its variables.

All the three monotone properties cover some aspects of the generic concept of monotonicity. However, they are not the same and yet they are not all independent. In this section we examine the relationships among them.

3.1 Some Exotic Auctions

We introduce some exotic auctions to help disprove some of the implication relations. Each auction is carefully selected to perform as much disproof as possible. It is straight forward to prove that all the auctions presented in this subsection satisfies **IR**, **IC** and **SYM**. The analysis of monotonicities of these auctions can be found in §??.

Auction 1 (k-Winner Vickrey Auction). $p^{(i)}$ is set to the k -th highest bid of all other bidders. A bidder who bids higher than its price will definitely win. A bidder who bids lower than its price will definitely lose. A bidder who bids equal to its price may win or lose. The auction always selects k winners.

The auction is **WM**, **PFM** and **RM**.

Auction 2. $p^{(i)}$ is set as the minimum bid times the number of minimum bids in the bid vector containing all other bidders' bids. Anyone who bids higher than its price is a winner.

The auction is **WM** but it is not **PFM**.

Auction 3. $p^{(i)}$ is set as the minimum bid of all other bidders. Any bidder bid higher than its price is winner.

The auction is **PFM** and **WM**. But it is not **RM**.

Auction 4. $p^{(i)}$ is set as follows: 1) 98, If at least one other bidder bids no less than 100; 2) 101, otherwise. Any $a^{(i)}$ bidding higher than its price is a winner.

The auction is **RM**. But it is neither **PFM** nor **WM**.

Auction 5. $p^{(i)}$ is set as follows: 1) 0, if no other bidder bids no less than 100; 2) 101, if one and only one other bidder bids no less than 100; 3) 100, if more than one bidders bid no less than 100.

Everyone who bids no less than its price is a winner.

The auction is **WM** and **RM**, but not **PFM**.

3.2 Implication Relationships

In this subsection, we derive the implication relationships among the three monotone properties, with the help of the examples introduced in the previous subsection.

Lemma 5. *In an auction satisfying IR, IC and SYM, $PFM \Rightarrow WM$.*

However, this is the only implication we know of the monotone properties. As we should show next, none of the other possible implications holds.

Theorem 1. *In an auction satisfying IR, IC and SYM, the implication relationships between any two of the three monotone properties are as follows:*

$PFM \Rightarrow WM$, $WM \not\Rightarrow PFM$; $RM \not\Rightarrow WM$, $WM \not\Rightarrow RM$; $PFM \not\Rightarrow RM$, $RM \not\Rightarrow PFM$.

Moreover, two of the monotone properties cannot strengthen the implication relationships either, as we should see in the following theorem.

Theorem 2. *In an auction satisfying IR, IC and SYM, the implication relationships from any two to the other one of the three monotone properties are as follows:*

PFM and $RM \Rightarrow WM$; WM and $RM \not\Rightarrow PFM$; WM and $PFM \not\Rightarrow RM$.

Theorem 3. *It is possible that the three monotone properties, WM , PFM and RM , simultaneously exist in an auction satisfying IR, IC and SYM.*

In other words, the existence of any one or two of the three monotone properties does not imply the non-existence of the other monotone properties (or property).

4 Vickrey Auction in Depth

In this section, we study Vickrey auction in depth. Our analysis show two sufficient and necessary conditions for a symmetric incentive compatible auction to be equivalent to Vickrey auction. One is that the number of winners is always fixed. Amazingly, symmetry is all we need for this class to be a Vickrey auction. When the number of winners is determined by the auction protocol, we consider a variation of Vickery auction protocols. We show its necessary and sufficient condition is that the auction is homogeneous, revenue monotone, price function monotone, and single winning price.

4.1 Fixed Number of Winners

In this subsection, we prove that for auctions with a fixed number of winners, a symmetric incentive compatible auction is all we need for it to be the Vickrey auction. Though it seems to be a classical type result which may have already known, we have not been able to locate a statement of such a theorem in the literatures. To be on the safe side, we would not claim the result as completely new but to include it only for the sake of completeness.

Since it is trivial that Vickrey auction must select a fixed number of winners, we only need to prove the sufficient part.

We first prove the 1-winner case, with an extension to the k -winner case later.

1 Winner

Theorem 4. *An auction satisfying IR, IC, SYM and SEL(n,1) must be equivalent to 1-item n-bidder Vickrey auction. i.e.:*

1. The total number of winners equals to 1.
2. (One of) the highest bidder wins
3. The winner pays the price of the highest bid of all other bidders

k Winners. The result can be extended to protocols with exactly k winners with $n \geq k + 1$ bidders. However the proof is neither simple nor obvious.

We put the bids in a set \mathbb{S} without duplicate values. Then we construct an n -dimensional counter vector $c = (c_1, c_2, \dots, c_n)$, where c_i is the number of bidders whose bids equals to the i -th highest bid in set \mathbb{S} . If there is no such bidder, $c_i = 0$. Obviously, for any auction with n bidders and $k < n$ winners, the set of c 's possible values is limited.

We define the order on c 's possible value set as follows:

1. If there are more non-zero elements in c than c' , then $c > c'$; else
2. If there are more non-zero elements in c' than c , then $c < c'$; else
3. If $c_i = c'_i, \forall i$, then $c = c'$; else
4. Let $i = \min\{k | c_k \neq c'_k\}$. If $c_i > c'_i$ then $c > c'$, else $c < c'$.

Lemma 6. *An auction satisfying IR, IC, SYM and SEL(n,k) must satisfy:*

1. $b^{(i)} > b^{(k+1)} \Rightarrow b^{(i)} > p^{(i)}$
2. $b^{(i)} < b^{(k+1)} \Rightarrow b^{(i)} < p^{(i)}$

where $b^{(k+1)}$ is the $(k + 1)$ -th highest bid.

Proof. We prove the lemma by mathematical induction.

1. **(Initial Step).** The smallest possible value of counter vector c is $(n, 0, \dots, 0)$. At this time, all the bidders have the same bids, and the lemma is trivially true.
2. **(Induction Step).** In induction step, we want to show that the lemma is true for counter vector c if the induction assumption (the lemma is true for all smaller counter vectors) is true.

(a) We first prove $b^{(i)} > b^{(k+1)} \Rightarrow b^{(i)} > p^{(i)}$. The argument is by contradiction. Suppose $\exists i, b^{(i)} > b^{(k+1)}$, but $b^{(i)} \leq p^{(i)}$. Now we decrease $a^{(i)}$'s bid to $\widetilde{b}^{(i)} = b^{(k+1)}$. Let c' denote the new counter vector after this change. Obviously $c' < c$ and the $(k + 1)$ -th highest bid now is still $b^{(k+1)}$.

By Lemma 3, $a^{(i)}$'s new price $\widetilde{p}^{(i)} = p^{(i)} \geq b^{(i)} > \widetilde{b}^{(i)}$. By SYM, $\forall j, b^{(j)} = \widetilde{b}^{(i)} \Rightarrow p^{(j)} = \widetilde{p}^{(i)} > b^{(j)}$.

Thus all bidders bidding $b^{(k+1)}$ will *definitely* lose and there are at most $k - 1$ bidders bidding higher than $b^{(k+1)}$ in the current situation or we say in c' . By SEL(n,k), $\exists L, b^{(L)} < b^{(k+1)}$ and $b^{(L)} \geq p^{(L)}$ and $a^{(L)}$ wins. *This contradicts the induction assumption that the lemma stands for c .*

- (b) We then prove $b^{(i)} < b^{(k+1)} \Rightarrow b^{(i)} < p^{(i)}$. Similarly, suppose $\exists i, b^{(i)} < b^{(k+1)}$ but $b^{(i)} \geq p^{(i)}$.

Now we increase $a^{(i)}$'s bid to $\widetilde{b}^{(i)} = b^{(k+1)}$. Let c' denote the new counter vector after this first change.

By Lemma 3, $a^{(i)}$'s new price $\widetilde{p}^{(i)} = p^{(i)} \leq b^{(i)} < \widetilde{b}^{(i)}$. By **SYM**, $\forall j, b^{(j)} = \widetilde{b}^{(i)} \Rightarrow p^{(j)} = \widetilde{p}^{(i)} < b^{(j)}$.

Thus, all bidders bidding $b^{(k+1)}$ now will *definitely* win and there are at least $k + 2$ bidders bidding no lower than $b^{(k+1)}$ in the current situation or we say in c' . By **SEL(n,k)**, $\exists H, b^{(H)} > b^{(k+1)}$ and $b^{(H)} \leq \widetilde{p}^{(H)}$ and $a^{(H)}$ loses.

Now we decrease $a^{(H)}$'s bid to $\widetilde{b}^{(H)} = b^{(k+1)}$. Let c'' denote the new counter vector after this *second* change. Obviously $c'' < c$ and the $(k+1)$ -th highest bid is still $b^{(k+1)}$.

By Lemma 3, $a^{(H)}$'s new price remains $\widetilde{p}^{(H)} \geq b^{(H)} > \widetilde{b}^{(H)}$. By **SYM**, $\forall j, b^{(j)} = \widetilde{b}^{(H)} \Rightarrow p^{(j)} = \widetilde{p}^{(H)} > b^{(j)}$.

Thus all bidders bidding $b^{(k+1)}$ will *definitely* lose under current situation.

Now there are at most $k - 1$ bidders bidding higher than $b^{(k+1)}$. By **SEL(n,k)**, $\exists L, b^{(L)} < b^{(k+1)}$ and $b^{(L)} \geq p^{(L)}$ and $a^{(L)}$ wins. *This contradicts the induction assumption that the lemma stands for c'' .*

3. **(Conclusion Step)**. From the above reasonings, we conclude that the lemma is true for all possible values of the counter vector.

The above reasonings complete the proof of the lemma.

Lemma 7. *An auction satisfying **IR**, **IC**, **SYM** and **SEL(n,k)** must satisfy:*

1. $b^{(i)} = b^{(k+1)}$ and $a^{(i)}$ wins $\Rightarrow p^{(i)} = b^{(k+1)}$
2. $b^{(i)} > b^{(k+1)} \Rightarrow p^{(i)} = b^{(k+1)}$

where $b^{(k+1)}$ is the $(k + 1)$ -th highest bid.

Theorem 5. *An auction satisfying **IR**, **IC**, **SYM** and **SEL(n,k)** must be equivalent to k -item n -bidder Vickrey auction. i.e.:*

1. Each bid higher than the $(k + 1)$ -th highest bid will **definitely** win.
2. Each bid lower than the $(k + 1)$ -th highest bid will **definitely** lose.
3. Each winner's price **must** be equal to the $(k + 1)$ -th highest bid.
4. The total number of winners is equal to k .

4.2 Homogeneous Monotone Auction and Vickrey Auction

In this subsection, we prove that **SWP**, **PFM**, **RM** and **HOMO** is a sufficient and necessary condition that a symmetric incentive compatible auction is equivalent to the Vickrey auction. Again it is trivial that Vickrey auction satisfies those properties, we only need to prove the sufficient part.

Lemma 8. *In an auction satisfying IR, IC, SYM, SWP, PFM, RM and HOMO, if a bid vector \mathbf{b} of size n results in exactly k winners, then after increasing any bidder $a^{(c)}$'s bid by δ and $0 < \delta < \frac{b^{(c)}}{n}$ the auction will still result in exactly k winners.*

Proof. Let the original winning price be p , the auctioneer's revenue be \tilde{R} and the number of winners be \tilde{k} after $a^{(c)}$'s bid increasing by δ .

By HOMO, if we increase every bidder's bid to $\frac{b^{(c)} + \delta}{b^{(c)}}$ times larger, the auction will still have exactly k winners and the winning price is $\frac{b^{(c)} + \delta}{b^{(c)}} \times p$.

By RM, after only increasing $b^{(c)}$ by δ , auctioneer's revenue should be no less than before but no more than increasing every bid to $\frac{b^{(c)} + \delta}{b^{(c)}}$ times larger. Or:

$$p \times k \leq \tilde{R} \leq \frac{b^{(c)} + \delta}{b^{(c)}} \times p \times k$$

By PFM, after changing $b^{(c)}$, every bidder's price should be no less than before but no more than changing every bid $\frac{b^{(c)} + \delta}{b^{(c)}}$ times larger. Or:

$$p \times \tilde{k} \leq \tilde{R} \leq \frac{b^{(c)} + \delta}{b^{(c)}} \times p \times \tilde{k}$$

From the above two inequations we get:

$$\begin{aligned} p \times k &\leq \frac{b^{(c)} + \delta}{b^{(c)}} \times p \times \tilde{k} \\ p \times \tilde{k} &\leq \frac{b^{(c)} + \delta}{b^{(c)}} \times p \times k \end{aligned}$$

Which further gives the following inequation:

$$\frac{b^{(i)}}{b^{(i)} + \delta} \times k \leq \tilde{k} \leq \frac{b^{(i)} + \delta}{b^{(i)}} \times k$$

Since $0 < \delta < \frac{b^{(i)}}{n}$, we must have:

$$\begin{aligned} \frac{b^{(i)}}{b^{(i)} + \delta} \times k &> k - 1 \\ \frac{b^{(i)} + \delta}{b^{(i)}} \times k &< k + 1 \end{aligned}$$

Hence:

$$k - 1 < \tilde{k} < k + 1$$

And we must have $\tilde{k} = k$. This completes the proof. □

Lemma 9. *In an auction satisfying IR, IC, SYM, SWP, PFM, RM and HOMO, if a bid vector \mathbf{b} of size n results in exactly k winners, then after increasing any bidder $a^{(c)}$'s bid by δ and $\delta > 0$ the auction will still result in exactly k winners.*

Lemma 10. *In an auction satisfying IR, IC, SYM, SWP, PFM, RM and HOMO, if a bid vector \mathbf{b} of size n results in exactly k winners, then any bid vector \mathbf{b}' of size n satisfying $b'^{(i)} \geq b^{(i)}, \forall i$, will also result in k winners.*

Theorem 6. *An auction satisfying IR, IC, SYM, SWP, PFM, RM and HOMO must have a fixed number k of winners as long as the total number of bidders n is being held fixed and every bid is a positive real number.*

Theorem 7. *In an auction satisfying IR, IC, SYM, SWP, PFM, RM and HOMO, when every bid is a positive real number, there must exist a function f such that for any bidder number n , the auction must be equivalent to $(k = f(n))$ -winner Vickrey auction. i.e.:*

1. Bidding higher than the $(k + 1)$ -th highest bid will **definitely** win.
2. Bidding lower than the $(k + 1)$ -th highest bid will **definitely** lose.
3. Each winner's price **must** be equal to the $(k + 1)$ -th highest bid.
4. The total number of winners is equal to $k = f(n)$.

5 Conclusions

In this paper, we have formally defined some basic properties of symmetric incentive compatible single-item single-unit auctions. We have shown that some of the relationships among them are quite complex. In §3 we have studied the implication relationships among three monotone properties. In §4 we have studied two sufficient and necessary conditions that an auction is equivalent to the Vickrey auction. Our results are of substantial value to auction research.

In the end we provide a non-restricting list of some interesting open problems as our future works. Can we extend our results to multi-unit auctions where a bidder demands multiple units of a single item? Can we extend the implications relationships among the monotone properties to randomized auctions? What are the foundational properties of the more generalized auctions that need not to be symmetric, where the order of the bids does matter?

References

1. Aggarwal, G., Goel, A., Motwani, R.: Truthful auctions for pricing search keywords. In: ACM Conference on Electronic Commerce (EC), ACM Press, New York (2006)
2. Aggarwal, G., Hartline, J.: Knapsack auctions. In: SODA (2006)
3. Bu, T., Qi, Q., Sun, A.W.: Unconditional competitive auctions with copy and budget constraints. In: Spirakis, P.G., Mavronicolas, M., Kontogiannis, S.C. (eds.) WINE 2006. LNCS, vol. 4286, Springer, Heidelberg (2006)
4. Clarke, E.H.: Multipart pricing of public goods. Public Choice 11, 17–33 (1971)
5. Deng, X., Huang, L., Li, M.: On walrasian price of cpu time. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, Springer, Heidelberg (2005)
6. Goldberg, A., Hartline, J., Karlin, A., Saks, M., Wright, A.: Competitive auctions. Games and Economic Behavior 55(2), 242–269 (2006)
7. Groves, T.: Incentives in teams. Econometrica 41(4), 617–631 (1973)
8. Vickrey, W.: Counterspeculation, auctions, and competitive sealed tenders. Journal of Finance 16, 8–37 (1961)

Finding Equilibria in Games of No Chance

Kristoffer Arnsfelt Hansen, Peter Bro Miltersen, and Troels Bjerre Sørensen

Department of Computer Science, University of Aarhus, Denmark
{arnsfelt,bromille,trold}@daimi.au.dk

Abstract. We consider finding maximin strategies and equilibria of explicitly given extensive form games with imperfect information but with no moves of chance. We show that a maximin pure strategy for a two-player game with perfect recall and no moves of chance can be found in time linear in the size of the game tree and that all pure Nash equilibrium outcomes of a two-player general-sum game with perfect recall and no moves of chance can be enumerated in time linear in the size of the game tree. We also show that finding an optimal behavior strategy for a one-player game of no chance without perfect recall and determining whether an equilibrium in behavior strategies exists in a two-player zero-sum game of no chance without perfect recall are both **NP**-hard.

1 Introduction

In a seminal paper, Koller and Megiddo [3] considered the complexity of finding maximin strategies in *two-player zero-sum imperfect-information extensive form games*. An extensive form game is an explicitly given game tree with *information sets* modeling hidden information (for details, see [3] or any text book on game theory). A main result of Koller and Megiddo was the existence of a polynomial time algorithm for finding an equilibrium in behavior strategies (or equivalently, a pair of maximin behavior strategies) of such a game when the game has *perfect recall*. Informally speaking, a game has perfect recall when a player never forgets what he once knew (for a formal definition, see below). In contrast, for the case of imperfect recall, the problem of finding a maximin strategy was shown to be **NP**-hard.

Pure equilibria (i.e, equilibria avoiding the use of randomization) play an important role in game theory and it is of special interest to know if a game possesses such an equilibrium. For the case of a zero-sum games, one may determine if a game has a pure equilibrium by computing a maximin pure strategy for each of the two players and checking that these strategies are best responses to one another. Unfortunately, Blair *et al.* [1] established that the problems of finding a maximin pure strategy of a two-player extensive form game or determining whether a pure equilibrium exists are both **NP**-hard, even for the case of zero-sum games of perfect recall. Their proof is an elegant reduction from the EXACT PARTITION (or BINPACKING) problem and relies heavily on the fact that the extensive form game is allowed to contain *chance* nodes, i.e., random events not controlled by either of the two players.

Extensive form games *without* chance nodes is a very natural special case to consider (natural non-trivial examples include such popular parlor games as variants of *Spoof*). In this paper we consider the equilibrium computation problems considered by Koller and Megiddo and by Blair *et al.* for this special case. Our main results are the following:

First, we show that *a maximin pure strategy for a two-player extensive form game of no chance with imperfect information but perfect recall can be found in time linear in the size of the game tree*. As stated above, Blair *et al.* show that *with* chance moves, the problem is **NP**-hard. Apart from the obvious practical interest, the example is also interesting in light of the recent work of von Stengel and Forges [6]. They introduced the notion of *extensive form correlated equilibria* (EFCEs) of two-player extensive form games. They showed that finding such equilibria in games *without* chance moves can be done in polynomial time while finding them in games *with* chance moves may be **NP**-hard. They remark that EFCE seems to be the first example of a game-theoretic solution concept where the introduction of chance moves marks the transition from polynomial-time solvability to **NP**-hardness. Our result combined with the result of Blair *et al.* provides a second and much more elementary such example.

Second, we extend the above result from maximin pure strategies to pure Nash equilibria. We show that *all pure Nash equilibrium outcomes of a two-player general-sum extensive form game of no chance with imperfect information but perfect recall can be enumerated in time linear in the size of the game tree*. Here, an outcome is a leaf of the tree defining the extensive form. Also, given one such pure Nash equilibrium outcome, we can in linear time construct a pure equilibrium (in the form of a strategy profile) with that particular outcome. In contrast, the recent breakthrough result of Chen and Deng [2] implies that finding a behavior Nash equilibrium for a game of this kind is **PPAD**-hard.

The results of Blair *et al.* and those of Koller and Megiddo give a setting where finding a pure equilibrium is **NP**-hard while finding an equilibrium in behavior strategies can be done in polynomial time. Considering games without perfect recall, we give an example of the opposite. We show that *determining whether a one-player game in extensive form with imperfect information, imperfect recall and no moves of chance has a behavior strategy that yields a given expected payoff is NP-hard*. In contrast, it is easy to see that finding an optimal pure strategy for such a game can be done in linear time. Our result strengthens a result of Koller and Megiddo [3, Proposition 2.5] who showed **NP**-hardness of finding a maximin behavior strategy in a *two-player* game with imperfect recall and no moves of chance. Koller and Megiddo [3, Example 2.12] also showed that a maximin behavior strategy in such a two-player game may require irrational behavior probabilities. We give a one-player example with the same property.

Finally, we show that determining whether a Nash equilibrium in behavior strategies exists in a two-player extensive form zero-sum game with no moves of chance but without perfect recall is **NP**-hard.

The rest of the paper is organized as follows. In section 2, we formally define the objects of interest and introduce the associated terminology (for a less concise

introduction, see the paper by Koller and Megiddo, or any textbook on game theory). In sections 3,4,5 and 6, we prove each of the four results mentioned above.

2 Preliminaries

A two-player *extensive form game* is given by a finite rooted tree with pairs of payoffs (one payoff for each of the two players) at the leaves, and information sets partitioning nodes of the tree. In a *zero-sum* game, the sum of each payoff pair is zero. A *general-sum* game is a game without this requirement. In this paper, we do not consider games with nodes of chance, so every node in the tree is owned by either Player 1 or to Player 2. All nodes in an information set belong to the same player. Intuitively, the nodes in an information set are indistinguishable for the player they belong to. In a one-player game, all nodes belong to Player 1. Actions of a player are denoted by labels on edges of the tree. Given a node u and an action c that can be taken in u , we let $\text{apply}(u, c)$ be the unique successor node v of u with the edge (u, v) being labeled c . Each node in an information set has the same set of outgoing actions. The set of possible actions in information set h we denote C_h . The actions belong to the player owning the nodes of the information set. *Perfect recall* means that all nodes in an information set belonging to a player share the sequence of actions and information sets belonging to that player that are visited on the path from the root to each of the nodes.

A *pure strategy* for a player assigns to each information set belonging to that player a chosen action. A *behavior strategy* assigns to each action at each information set belonging to that player a probability. A pure strategy can also be seen as a behavior strategy that only uses the probabilities 0 and 1. Thus, concepts defined below for behavior strategies also apply to pure strategies. A (pure or behavior) *strategy profile* is a pair of (pure or behavior) strategies, one for each player. Given a pure strategy profile for a game without chance nodes, there is a unique path in the tree from the root to a leaf formed by the chosen actions of the two players. The leaf is called the *outcome* of the profile. A behavior strategy profile defines in the natural way a probability distribution on the leaves of the tree and hence a probability distribution on payoffs for each of the two players. So given a behavior strategy profile we can talk about the expected payoff for each of the two players.

A *maximin pure strategy* for a player is a pure strategy that yields the maximum possible payoff for that player assuming a worst case opponent, i.e., the maximum possible guaranteed payoff. A *maximin behavior strategy* for a player is a behavior strategy that yields the maximum possible expected payoff for that player assuming a worst case opponent, i.e., the maximum possible guaranteed expected payoff. A *Nash equilibrium* is a strategy profile (s_1, s_2) so that no strategy s'_1 yields strictly better payoff for Player 1 than s_1 when Player 2 plays s_2 and no strategy s'_2 yields strictly better payoff for Player 2 than s_2 when Player 1 plays s_1 .

Kuhn [5] showed that for an extensive form two-player zero-sum game with perfect recall, a pair of maximin behavior strategies is a Nash equilibrium. The expected payoff for Player 1 is the same in any such equilibrium and is called the *value* of the game. Any extensive form general-sum game with perfect recall in fact possesses a Nash equilibrium in behavior strategies.

3 Maximin Pure Strategies in Games with Perfect Recall

Consider a two-player extensive form game G with perfect recall and without chance nodes. We shall consider computing a maximin pure strategy for one of the players, say, Player 1. For the purpose of computing such a strategy, we can consider G to be a zero-sum game where Player 1 (henceforth the max-player) attempts to maximize his payoff and Player 2 (henceforth the min-player) attempts to minimize the payoff of Player 1. Let G' be the zero-sum game obtained from G by dissolving all information sets of the min-player into singletons.

Note that the set of strategies for the max-player is the same in G and G' . For the min-player, however, the set of strategies is larger in G' thereby making G' a better game than G for the min-player, so its *value* as a zero-sum game is at most the value of G . However we have the following key lemma. Note that the lemma fails badly for games containing chance nodes.

Lemma 1. *A pure strategy π for the max-player has the same payoff against an optimal counter strategy in G as it has against an optimal counter strategy in G' (note that the statement makes sense as the max-player has the same set of strategies in the two games).*

Proof. Let σ be a pure best counter strategy against π in G' . As there are no chance nodes, σ and π defines a single path in the tree of G' from the root to a leaf. Due to perfect recall, none of the choices made by the min-player along the path are choices of the same information set. Thus, the same sequence of choices can also be made by a strategy in G . Thus, there is a counter strategy in G that achieves the same payoff against π as σ does in G' , and since the set of possible counter strategies is bigger in G' , the best in each game each achieves exactly the same payoff.

To compute the best payoff that can be obtained by a pure strategy in G' , we define for information set h of G' a value $\text{pval}(h)$ (“pure value”) inductively in the game-tree as follows.

- If h belongs to the min-player, and therefore consists of a single node u , define

$$\text{pval}(h) = \min_{c \in C_h} \text{pval}(\text{apply}(u, c))$$

- If h belongs to max-player, define

$$\text{pval}(h) = \max_{c \in C_h} \min_{u \in h} \text{pval}(\text{apply}(u, c))$$

The induction is well-founded due to perfect recall and the fact that there are no chance nodes, see [6, Lemma 3.2].

Lemma 2. *For every pure strategy π for the max-player, there exists a pure strategy σ for the min-player with the following property. For every information set h of the max-player there is some node $u \in h$ such that play from u using the pair of strategies (π, σ) yields payoff at most $\text{pval}(h)$. Similarly, for every information set g of the min-player, play from the single node u of h using the pair of strategies (π, σ) yields payoff at most $\text{pval}(h)$.*

Proof. Given a pure strategy π for the max-player, we construct the strategy σ inductively in the game tree. Let h be a given information set of the max-player. Then, by definition of $\text{pval}(h)$ there must be a path from some node $u \in h$ using the action chosen by π out of u (say, L), then going through min-nodes to an information set g of the min-player with $\text{pval}(g) \leq \text{pval}(h)$, or to a leaf l with payoff less than or equal to $\text{pval}(h)$.

In the latter case we simply let σ take the choices defining the path to the leaf l . In the former case, by induction, we know we have constructed a pure strategy σ for min from g onwards so that for some node $v \in g$, play from g using π and σ leads to payoff at most $\text{pval}(g)$. Note that we have a path from u to some (possibly) other node $v' \in g$ using min-nodes. We claim that there is a path from some node $\bar{u} \in h$ to v using min-nodes and also choosing the action L in \bar{u} (see Fig. 1).

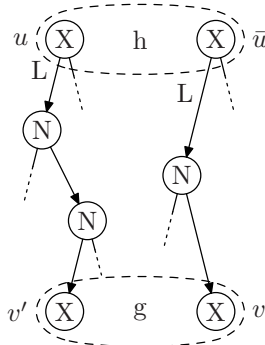


Fig. 1. Finding \bar{u}

Indeed, assume that this is not the case. Then the sequence of information sets and own actions encountered by max on the way to v differs from the corresponding sequence in some of other node (namely v') in the information set of v , contradicting perfect recall. But then, the node \bar{u} establishes the induction claim, with the desired strategy σ taking the choices defining the path from \bar{u} to v .

It remains to provide the first actions for the min-player in case the root node belongs to the min-player. In this case there is a path from the root r , going

through min-nodes to an information set h of max with $\text{pval}(h) \leq \text{pval}(r)$, or to a leaf l with payoff equal to $\text{pval}(r)$. As before we let σ take the choices defining this path.

With this we can now obtain the following result.

Theorem 1. *Given a two-player extensive form game with perfect recall G without chance nodes, we can compute a maximin pure strategy for a player in linear time in the size of the game tree.*

Proof. We describe how to compute a maximin strategy for one of the players, say Player 1. By Lemma 1 we can compute this by computing a pure maximin strategy in the game G' . We compute the pval function of the information sets in G' and let the strategy of the max-player be the choices that obtains the maximum in the definition of pval for every information set, i.e., the choice in information set h is $\text{argmax}_{c \in C_h} \min_{u \in h} \text{pval}(\text{apply}(u, c))$. We claim that the value $\text{pval}(r)$ assigned to the root is the best guaranteed payoff the max-player can get in G' using some pure strategy. Indeed the max-player is guaranteed payoff $\text{pval}(r)$, where r is the root of G' , playing this strategy, and Lemma 2 establishes this is the best he can be guaranteed.

Note also that having computed the maximin pure strategy, we can determine whether it is also maximin as a behavior strategy by computing the value v of the game in polynomial time using, e.g., the algorithm of Koller and Megiddo [3] or the more practical one by Koller, Megiddo and von Stengel [4] and checking if the computed pure value $\text{pval}(r)$ of the root equals v .

4 Enumerating All Pure Equilibria of Games with Perfect Recall

Let G be a 2-player *general sum* extensive form game with perfect recall and without chance nodes. Let (π, σ) be a pair of pure strategies. For (π, σ) to be a pure equilibrium we must have that π is a best response to σ and vice versa. Play using the pair (π, σ) will lead to a unique leaf of G , since there are no chance nodes. Consider now a leaf l of G , as a potential outcome of a pure equilibrium. Clearly the actions along the path from the root r of G to the leaf must be such that they follow the path. Hence what remains are to find the actions of the remaining information sets. Player 1 must find pure actions in his remaining information sets such that Player 2 can not obtain greater payoff than she receives at l . Similarly Player 2 must find pure actions in her information sets such that Player 1 can not obtain greater payoff than he receives at l . Given l , we can define zero-sum games G_1 and G_2 by modifying G such that such actions, if they exist, can be found in linear time using Theorem 1.

We can simply construct G_1 from G as follows (the construction of G_2 being the same with Player 1 and Player 2 exchanged). Player 1 will be the max-player of G_1 and Player 2 will be the min-player. For every information set of Player

1 along the path from the root to l we remove all choices (and the subgames below) except the ones agreeing with the path. The payoff at a leaf in G_1 is the negative of the payoff that Player 2 receives in the corresponding leaf in G . The following lemma is immediate.

Lemma 3. *There is a pure strategy for Player 1 in G leading towards l ensuring that Player 2 can obtain at most payoff p if and only if there is a pure strategy for the max-player of G_1 ensuring payoff at least $-p$.*

Using this lemma, it is easy to check in linear time if a given leaf l with payoffs (p_1, p_2) is a pure equilibrium outcome: We check that the maximin pure strategy for Player 1 in G_1 ensures payoff at least $-p_2$ and we check that the maximin pure strategy for Player 2 in G_2 ensures payoff at least $-p_1$. Also, given such an outcome, we can in linear time construct a pure strategy equilibrium with this outcome: The equilibrium is the profile consisting of transferring in the obvious way to G the maximin pure strategies for Player 1 in G_1 and for Player 2 in G_2 .

Since we can check in linear time if a given leaf is an outcome, we can enumerate the set of outcomes in quadratic time. To get a linear time algorithm, we will go one step further and work with a derived game that is independent of the leaf l .

Let G'_1 be the zero-sum game obtained from G by dissolving the information sets of Player 2 and letting payoff at a leaf in G'_1 be the negative of the payoff that Player 2 receives in the corresponding leaf in G . We define the pval function on G'_1 as in section 3.

Let T_1 be a tree on the information sets of Player 1 and the leaves together with a root, such that the parent of an information set or leaf is the first information set on the path to the root in G'_1 or the root itself.

Define a *point of deviation* with respect to a given leaf l , to be a node in T not on the path from the root to l , but sharing the sequence of actions leading to the node with a node on the path from the root to l . Thus only nodes that have their parents on the path can be points of deviation. See Fig. 2 for an example. Intuitively, a point of deviation is an information set where Player 1 first observes that Player 2 has deviated from the strategy leading to l .

The following lemma is easy to establish.

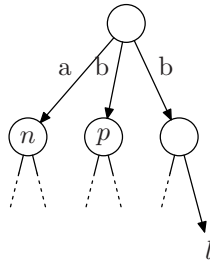


Fig. 2. Node p is a point of deviation, node n is not

Lemma 4. *There is a pure strategy for Player 1 in G leading towards l ensuring that Player 2 can obtain at most the payoff p if and only if for every point of deviation h with respect to l we have $\text{pval}(h) \geq -p$.*

Theorem 2. *Given a 2-player general-sum extensive form game with perfect recall G without chance nodes, we can in linear time in the size of the game tree enumerate the set of leaves that are outcomes of pure equilibria.*

Proof. Using Lemma 4, we compute the leaves l such that Player 1 has a pure strategy leading towards l ensuring that Player 2 can obtain at most the payoff received at l and conversely Player 2 has a pure strategy leading towards l ensuring that Player 1 can obtain at most the payoff received at l . These sets can be computed separately; we describe how to compute the former.

We construct the game G'_1 and compute the pval function on G'_1 in linear time. In linear time we then construct the tree T_1 and record the computed pval values in the nodes. Finally we traverse the tree T . During this traversal we maintain the minimum pval value that is on any sibling to the nodes on the path to the root, corresponding to the points of deviations relevant for the leaves in the subtree of the current node. Once we visit a leaf we can then directly decide the criteria of Lemma 4 by comparing with the payoff of the leaf.

5 Optimal Behavior Strategies in One-Player Games Without Perfect Recall

In this section we consider *one-player* games without perfect recall and no moves of chance and show **NP**-hardness of the problem of determining whether a behavior strategy yielding an expected payoff of at least a given rational number exists. In contrast, it is straightforward to see that the corresponding problem for pure strategies is in **P**: For each leaf of the game, one checks if this leaf can be reached by a sequence of actions so that the same action is taken in all nodes in a given information set. This results strengthens the result of Koller and Megiddo [3, Proposition 2.6] who showed **NP**-hardness of the problem of determining whether some behavior strategy in a *two-player* game without perfect recall guarantees a certain expected payoff (against any strategy of the opponent). Also, our reduction is heavily based on their reduction but uses imperfect recall to eliminate one of the players. Before giving the proof, we give a simple example showing that an optimal strategy may require irrational behavior probabilities (therefore, strictly speaking, “finding” an optimal strategy is not a well-defined computational problem which leads to considering the stated decision problem instead). A corresponding two-player example was given by Koller and Megiddo [3, Example 2.12]. Our one-player game of Fig. 3 is in fact somewhat simpler than their example. All nodes in the game are included in the same information set. The player can choose either L or R. Thus, a behavior strategy is given by a single probability p_L with $p_R = 1 - p_L$. By construction, the expected payoff is $-2p_L^3 - (1 - p_L)^3$. This is maximized for $p_L = \sqrt{2} - 1$.

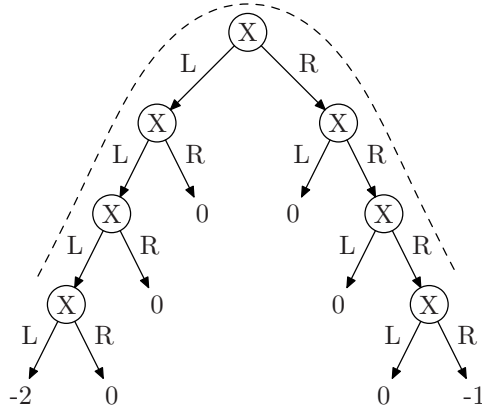


Fig. 3. A one-player game where the rational behavior is irrational

Theorem 3. *The following problem is NP-hard: Given a rational number v and a one-player extensive form game without chance nodes and a rational number v , does some behavior strategy ensure expected payoff at least v ?*

Proof. The proof is by reduction from 3SAT. Given a 3-CNF formula F with m clauses we construct a game G as follows.

Assume without loss of generality that m is a power of 2, $m = 2^k$. First G will consist of a complete binary tree of depth $2k$, whose nodes are contained in a single information set. If on the path from the root to a node, the same choice is made in step $2(i - 1) + 1$ and $2i$ for some $i \in \{1, \dots, k\}$, the game is terminated and the player receives payoff 0. Otherwise, we will associate a clause to the node in the following way: For $i = 1, \dots, k$ we interpret the choices made at step $2(i - 1) + 1$ and $2i$ as defining a binary choice. With the choices (left,right) we associate the bit 0, and with choices (right,left) we associate the bit 1. Having defined in this way k bits, we may associate a uniquely determined clause with the node.

From this node we let the player, for each of the three variables in the clause, select a truth value. If one of these choices satisfies the clause, the player receives payoff 1, and 0 otherwise. We place the nodes corresponding to the same variable in a single information set. In particular, the player does not know the clause.

The proof is now concluded by the following claim: The player can obtain expected payoff $\frac{1}{m}$ if and only if F is satisfiable.

Assume first that F is satisfiable. The player will make the first $2k$ choices by choosing left with probability $\frac{1}{2}$. The rest of the choices are made according to a satisfying assignment to F . With probability $(\frac{1}{2})^k = \frac{1}{m}$, the player gets to a node corresponding to a clause, and will obtain payoff 1. The expected payoff is therefore $\frac{1}{m}$.

Assume on the other hand that the player can obtain expected payoff $\frac{1}{m}$. Suppose that the player chooses left with probability p in the first $2k$ choices. The probability that the player reaches a node associated with a given clause

is $(p(1-p))^k \leq \frac{1}{m^2}$, independently of the (p) node. Since the player can in fact obtain expected payoff $\frac{1}{m}$, we have that at every node associated with a clause the player must obtain payoff 1, and thus his strategy gives a satisfying assignment to F .

6 Determining Whether a Two-Player Game Without Perfect Recall has an Equilibrium

Our final hardness result again uses a reduction very similar to Koller and Megiddo [3, Proposition 2.6]. In this case, we use the imperfect recall to force Player 1 to use an almost pure strategy.

Theorem 4. *The following problem is NP-hard: Given a two-player zero-sum extensive form game without chance nodes, does the game possess a Nash equilibrium in behavior strategies?*

Proof. The proof is by reduction from 3SAT. Given a 3-CNF formula F with m clauses we construct a zero-sum two-player game G as follows.

Player 1 (the max-player) starts the game by making two actions, each time choosing a clause of F . We put all corresponding $m + 1$ nodes (the root plus m nodes in the next layer) of the game in one information set. If he fails to choose the same clause twice, he receives a payoff of $-m^3$ and the game stops. Otherwise, Player 2 (the min-player) then selects a truth value for each of the three variables in the clause. We place all nodes of Player 2 corresponding to the same variable in a single information set. If one of the choices of Player 2 satisfies the clause, Player 1 receives payoff 0. If none of them do, Player 1 receives payoff 1.

The proof is now concluded by the following claim: G has an equilibrium in behavior strategies if and only if F is satisfiable.

Assume first that F is satisfiable. G then has the following equilibrium (which happens to be pure): Player 2 plays according to a satisfying assignment while Player 1 uses an arbitrary pure strategy. The payoff is 0 for both players and no player can modify their behavior to improve this so we have an equilibrium.

Next assume that G has an equilibrium. We shall argue that F has a satisfying assignment. We first observe that Player 1 in equilibrium must have expected payoff at least 0. If not, he could switch to an arbitrary pure strategy and would be guaranteed a payoff of at least 0. Now look at the two actions (i.e., clauses) that Player 1 is most likely to choose. Let clause i be the most likely and let clause j be the second-most likely. If Player 1 chooses i and then j he gets a payoff of $-m^3$. His maximum possible payoff is 1 and his expected payoff is at least 0. Hence, we must have that $-m^3 p_i p_j + 1 \geq 0$. Since $p_i \geq 1/m$, we have that $p_j \leq 1/m^2$. Since clause j was the second most likely choice, we in fact have that $p_i \geq 1 - (m-1)(1/m^2) > 1 - 1/m$. Thus, there is one clause that Player 1 plays with probability above $1 - 1/m$. Player 2 could then guarantee an expected payoff of less than $1/m$ for Player 1 by playing any assignment satisfying this clause. Since we are actually playing an equilibrium, this would not decrease the payoff of Player 1 so Player 1 currently has an expected payoff less than $1/m$. Now look at the assignment defined by the most likely choices

of Player 2 (i.e, the choices he makes with probability at least $\frac{1}{2}$, breaking ties in an arbitrary way). We claim that this assignment satisfies F . Suppose not. Then there is some clause not satisfied by F . If Player 1 changes his current strategy to the pure strategy choosing this clause, he obtains an expected payoff of at least $(1/2)^3 \geq 1/m$ (supposing, wlog, that $m \geq 8$). This contradicts the equilibrium property and we conclude that the assignment in fact does satisfy F .

Acknowledgments

We would like to thank Daniel Andersson, Lance Fortnow, and Bernhard von Stengel for helpful comments and discussions.

References

1. Blair, J.R.S., Mutchler, D., van Lent, M.: Perfect recall and pruning in games with imperfect information. *Computational Intelligence* 12, 131–154 (1996)
2. Chen, X., Deng, X.: Settling the complexity of two-player Nash equilibrium. In: 47th Annual Symposium on Foundations of Computer Science, pp. 261–272 (2006)
3. Koller, D., Megiddo, N.: The complexity of two-person zero-sum games in extensive form. *Games and Economic Behavior* 4, 528–552 (1992)
4. Koller, D., Megiddo, N., von Stengel, B.: Fast algorithms for finding randomized strategies in game trees. In: *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, pp. 750–759 (1994)
5. Kuhn, H.W.: Extensive games and the problem of information. *Annals of Mathematical Studies* 28, 193–216 (1953)
6. von Stengel, B., Forges, F.: Extensive form correlated equilibrium: Definition and computational complexity. Technical Report LSE-CDAM-2006-04, London School of Economics, Centre for Discrete and Applicable Mathematics (2006)

Efficient Testing of Forecasts

Ching-Lueh Chang¹ and Yuh-Dauh Lyuu²

¹ Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

d95007@csie.ntu.edu.tw

² Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

lyuu@csie.ntu.edu.tw

Abstract. Each day a weather forecaster predicts a probability of each type of weather for the next day. After n days, all the predicted probabilities and the real weather data are sent to a test which decides whether to accept the forecaster as possessing predicting power. Consider tests such that forecasters who know the distribution of nature are passed with high probability. Sandroni shows that any such test can be passed by a forecaster who has no prior knowledge of nature [San03], provided that the duration n is known to the forecaster in advance. On the other hand, Fortnow and Vohra [FV06] show that Sandroni's result may require forecasters with high computational complexity and is thus of little practical relevance in some cases. We consider forecasters who select a deterministic Turing-machine forecaster according to an arbitrary distribution and then use that machine for all future forecasts. We show that forecasters even more powerful than the above ones are required for Sandroni's result. Also, we show that Sandroni's result does not apply when the duration n is not known to the forecaster in advance.

1 Introduction

The weather metaphor motivates the investigations in this paper. A weather forecaster predicts the probability of rain on each day. How do we measure the forecaster's predicting power? Dawid [Daw82] proposes testing whether the announced forecasts are *well calibrated* in the following sense:

Suppose that, in a long (conceptually infinite) sequence of weather forecasts, we look at all those days for which the forecast probability of precipitation was, say, close to some given value ω and (assuming these form an infinite sequence) determine the long run proportion p of such days on which the forecast event (rain) in fact occurred. The plot of p against ω is termed the forecaster's empirical calibration curve. If the curve is the diagonal $p = \omega$, the forecaster may be termed (empirically) *well calibrated*.

A forecaster who knows the distribution of nature can produce well calibrated forecasts [Daw82].

Foster and Vohra [FV98] show that a forecaster can produce forecasts that will be well calibrated on any distribution adopted by nature. Fudenberg and Levine [DF99], Lehrer [Leh01], Sandroni, Smorodinsky and Vohra [SSV03] consider stronger forms of calibration tests. They show that a forecaster can pass these stronger tests on any distribution of nature. Thus, to pass these calibration tests, a forecaster needs no prior knowledge about the distribution of nature in that he does not need to assume anything about the distribution of nature [San03]. In this sense, if a forecaster passes a test on any distribution of nature, it means that the forecaster needs no prior knowledge about the distribution of nature to pass that test.

Ideally, we want a test to reject forecasters who have no prior knowledge about the distribution of nature. As a corollary, it is desirable that a test rejects each forecaster on some distribution of nature, for otherwise some forecaster passes the test on all distributions of nature, meaning it needs no prior knowledge about nature to pass the test. A second desirable property is that a test passes forecasters who know the distribution of nature. These two desirable properties are not simultaneously satisfied by calibration tests (see [FV98, DF99, Leh01, SSV03]). Sandroni [San03] shows that no test satisfies both desirable properties as long as the duration is made known beforehand to the forecaster. In this known-duration case, if a test is such that forecasters who know the distribution of nature are accepted with high probability, then there exists a forecaster that can pass that test on any distribution of nature.

Fortnow and Vohra [FV06] consider computational bounds on the forecaster and the test. They construct efficient tests that pass forecasters who know the distribution of nature, whereas only computationally powerful forecasters have a chance to pass these tests on all possible distributions of nature. This result implies that Sandroni's result requires the absence of computational constraints on the forecaster.

There is also closely related literature on theory testing [DF06, OS06b, OS06a]. In these models, an expert proposes a distribution of a stochastic process, which is tested for truthfulness. Some papers [ANW06, FS07] discuss identifying which of several experts knows the true distribution of the stochastic process.

In this paper, we present two results. The first proceeds along the computational perspective of Fortnow and Vohra [FV06], who show that Sandroni's [San03] result may require forecasters with high computational complexity. We consider the family of forecasters who select a deterministic Turing-machine forecaster of any time complexity according to an arbitrary distribution and then use that machine for all future forecasts. We show that Sandroni's result requires forecasters even more powerful than the above. We achieve it by exhibiting a test such that forecasters who know the distribution of nature are passed with high probability, whereas each of the above-mentioned forecasters is rejected with high probability on some n -day weather sequence for all sufficiently large n . Unlike the results in [FV06] where Turing-machine forecasters with a certain computational complexity are rejected on some distributions of

nature, the above-mentioned forecasters may adopt uncomputable distributions over Turing-machines of arbitrary time complexity.

Our second result shows that Sandroni's [San03] result does not apply when the duration n of forecasting is unknown beforehand to the forecaster. We do so by exhibiting a test which passes forecasters who know the distribution of nature, whereas any forecaster is rejected on some n -day weather sequence for infinitely many durations n .

Our paper is organized as follows. Section 2 gives formal definitions. Section 3-4 present our results. Section 5 concludes the paper.

2 Definitions

The following setup extends that proposed by Sandroni [San03]. A finite state space $S \equiv \{1, \dots, K\}$ categorizes the outcome on each period into one of $1, \dots, K$ where $K \geq 2$. For $\ell \in \mathbb{N}$, denote by S^ℓ the ℓ -dimensional Cartesian product of S and $S^* \equiv \bigcup_{\ell \geq 0} S^\ell$. Let $n \in \mathbb{N}$ be the duration of forecasting (in periods), which may or may not be known to the forecaster and the test. Given a finite outcome sequence $s \in S^*$, the data generating process assigns a probability distribution over the outcomes for the next period. A null outcome sequence is denoted λ .

A distribution over the outcomes of S is called a forecast. The set of distributions over S is denoted ΔS [San03]. A (possibly randomized) forecaster F announces a forecast for the $(t + 1)$ th period given the outcomes of the first t periods and the previous forecasts by F for these t periods. Formally, the input to a forecaster F is an outcome sequence s and a forecast sequence f where $(s, f) \in \bigcup_{t \geq 0} S^t \times (\Delta S)^t$. F 's output is a forecast for the following period's outcome. Let $s = (s_1, \dots, s_{n-1}) \in S^{n-1}$. If the data generating process adopts s as the outcome sequence for the first $n - 1$ periods, the n forecasts announced by a forecaster F are denoted

$$F(\lambda; 1), F(s_1; 2), \dots, F(s_1, \dots, s_{n-1}; n).$$

Here $F(s_1, \dots, s_{i-1}; i)$ is F 's forecast for the i th period. It depends only on the first $i - 1$ outcomes in s and the forecasts

$$F(\lambda; 1), \dots, F(s_1, \dots, s_{i-2}; i - 1)$$

for them, for $i \geq 1$. For convenience, we write

$$F(s) \equiv (F(\lambda; 1), F(s_1; 2), \dots, F(s_1, \dots, s_{n-1}; n)) \quad (1)$$

for the full forecast sequence (we may also write $F(s_1, \dots, s_{n-1})$ as $s = (s_1, \dots, s_{n-1})$). When the forecaster is a Turing machine, we assume a reasonable encoding for its output format. For example, we may let it output a forecast as a vector consisting of its predicted probabilities of the outcomes in S .

Let \mathcal{P} be the distribution adopted by the data generating process. Given any outcome sequence, \mathcal{P} determines the probability distribution over S for the next

period. Given \mathcal{P} and an outcome sequence $s = (s_1, \dots, s_n)$, the correct forecast sequence $\hat{f}(s) = (\hat{f}_1, \dots, \hat{f}_n)$ is one such that \hat{f}_j equals the probability distribution that the data generating process assigns for the j th period conditioned on (s_1, \dots, s_{j-1}) being the outcome sequence for the first $j - 1$ periods, where $1 \leq j \leq n$. Sensibly, \hat{f}_j depends only on \mathcal{P} and (s_1, \dots, s_{j-1}) , not on future outcomes (s_j, \dots, s_n) . We may write \hat{f}_j as $\hat{f}_j(s_1, \dots, s_{j-1})$ to make the dependency explicit. According to this notation,

$$\hat{f}(s) \equiv \left(\hat{f}_1(\lambda), \hat{f}_2(s_1), \dots, \hat{f}_n(s_1, \dots, s_{n-1}) \right). \tag{2}$$

A test is deterministic. It receives an outcome sequence s and a corresponding forecast sequence f where $(s, f) \in \bigcup_{t \geq 0} S^t \times (\Delta S)^t$ and decides whether to accept the forecaster as possessing predicting power. A natural criterion is that, whatever the distribution \mathcal{P} adopted by the data generating process, an outcome sequence $s = (s_1, \dots, s_n)$ together with the correct forecast sequence $\hat{f}(s)$ must be accepted with high probability. Here the probability is taken over the random variables s_1, \dots, s_n being the outcome sequence whose distribution is determined by the data generating process. A test satisfying this criterion is said to pass the truth with high probability [FV06].

A few comparisons with Sandroni’s [San03] definitions can be made. In Sandroni’s definition, the duration n is known to the forecaster and the test; the data after the n th period are simply ignored. The forecaster receives its input from $\bigcup_{t=0}^{n-1} S^t \times (\Delta S)^t$, and the test receives its input from $S^n \times (\Delta S)^n$. This paper, however, considers the consequences on the same forecaster (and test) over arbitrarily large duration n . Thus, in our definition the forecaster receives its input from $\bigcup_{t \geq 0} S^t \times (\Delta S)^t$, and the test receives its input from $\bigcup_{n \geq 0} S^n \times (\Delta S)^n$.

3 Forecasters with Arbitrary Time Complexity

We first review previous theorems. The following theorem is due to Sandroni [San03] (for the interpretations of this theorem please refer to their paper).

Theorem 1. ([San03]) *Let $S \equiv \{1, \dots, K\}$ be the finite state space, $0 < \epsilon < 1$, and $n \in \mathbb{N}$ be the duration of forecasting. Consider forecasters receiving input from $\bigcup_{t=0}^{n-1} S^t \times (\Delta S)^t$ and tests receiving input from $S^n \times (\Delta S)^n$. If a test T_n passes the truth with probability at least $1 - \epsilon$, there is a forecaster F_n (which may be randomized) that is accepted by T_n with probability at least $1 - \epsilon$ on any data generating process.*

The existence of the forecaster implied in Theorem 1 is shown via Fan’s minimax theorem [Fan53]. The proof is nonconstructive, and there seem no reasons for the implied forecaster to be efficient. Indeed, Fortnow and Vohra [FV06] proceed along this computational perspective and show the following.

Theorem 2. ([FV06]) *Let $S \equiv \{1, \dots, K\}$ be the finite state space. For any time-constructible $t(n)$, there is a test T of time complexity at most $\text{poly}(n) t(n)$ and an infinite sequence $s^* = (s_1^*, s_2^*, \dots)$ over S with the following properties:*

1. For each duration n , the test T passes the truth with high probability.
2. For any deterministic Turing-machine forecaster F with time complexity $t(n)$, T rejects $((s_1^*, \dots, s_n^*), F(s_1^*, \dots, s_{n-1}^*))$ for sufficiently large n .

Theorem 2 complements Theorem 1 by giving a test which passes the truth with high probability, whereas any deterministic forecaster of time complexity $t(n)$ is rejected on some outcome sequence if the duration is sufficiently long. Fortnow and Vohra [FV06] also show that Theorem 2 carries over to randomized forecasters of time complexity $t(n)$.

Theorem 3. ([FV06]) *Let $S \equiv \{1, \dots, K\}$ be the finite state space. For any time-constructible $t(n)$, there is a test T of time complexity at most $\text{poly}(n)t(n)$ and an infinite sequence $s^* = (s_1^*, s_2^*, \dots)$ over S with the following properties:*

1. For each duration n , the test T passes the truth with high probability.
2. For any randomized Turing-machine forecaster F with time complexity $t(n)$, T rejects $((s_1^*, \dots, s_n^*), F(s_1^*, \dots, s_{n-1}^*))$ with high probability for sufficiently large n .

Clearly, for the test in Theorem 3, the forecaster implied in Theorem 1 must be highly complicated—it cannot simply be the same randomized $t(n)$ -time Turing machine for all durations n . Fortnow and Vohra [FV06] also show tests such that the forecasters implied in Theorem 1 can be used to do factorization or even solve PSPACE-complete problems, a strong indication of the high complexity of the forecasters in Theorem 1. Based on these results, Fortnow and Vohra [FV06] conclude that Theorem 1 may be of little practical relevance in some cases.

We are now ready to describe our results. In Theorem 2, the test T has a running time of $\text{poly}(n)t(n)$ and is guaranteed to reject any $t(n)$ -time forecasters when the data generating process adopts s^* as the outcome sequence. The test, therefore, is more complex in terms of running time than the forecasters it is to reject. The $\text{poly}(n)t(n)$ running time of T is inherent in the proof of [FV06] in that T simulates deterministic Turing machines of time complexity $t(n)$. We improve upon Theorem 2 by allowing T to run in $\text{poly}(n)$ time and reject every Turing-computable deterministic forecaster of any time complexity—not just $t(n)$. For our test T , the forecasters implied in Theorem 1 can not be the same deterministic Turing machine (of any time complexity) for all durations n .

Lemma 4. *Let $S \equiv \{1, \dots, K\}$ be the finite state space. There is a polynomial-time test T and an infinite sequence $s^* = (s_1^*, s_2^*, \dots)$ over S with the following properties:*

1. For each duration n , the test T passes the truth with high probability.
2. For any Turing-computable deterministic forecaster F , T rejects

$$((s_1^*, \dots, s_n^*), F(s_1^*, \dots, s_{n-1}^*))$$

for sufficiently large n .

Proof. Consider the doubly fractal sequence AW

$$1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, \dots$$

Let $h(j)$ be the j th element of the sequence. Note that $h(j) \leq j$. We begin by describing the test T . The input to T is some outcome sequence (s_1, \dots, s_n) and a forecast sequence (f_1, \dots, f_n) where f_i is the forecast probability distribution over S for the i th period. For each $1 \leq j \leq n$ the test T marks j if s_j equals the smallest outcome among those with the smallest forecast probability in f_j . Now T rejects if for some $1 \leq k \leq \log n$, every $j \leq n$ with $h(j) = k$ gets marked; otherwise, T accepts. It is clear that T runs in polynomial time.

We proceed to construct s^* . Let F_1, F_2, \dots be an enumeration of deterministic Turing machines that compute recursive functions. Such machines must halt on all inputs. Pick s_1^* arbitrarily from S . Inductively, once $(s_1^*, \dots, s_{j-1}^*)$ is determined, set $s_j^* \in S$ to be the outcome with the smallest forecast probability in $F_{h(j)}(s_1^*, \dots, s_{j-1}^*; j)$. If there are ties, s_j^* should be the one with the smallest value. In this way s^* is constructed. The intuition is that s_j^* is set to foil the forecast of $F_{h(j)}$ as much as possible.

We now argue item 1. Let the data generating process adopt an arbitrary distribution \mathcal{P} . Let $s = (s_1, \dots, s_n)$ be the outcome sequence. Here s and thus $\hat{f}(s)$ (defined in Eq. 2) consist of random variables whose distributions are determined by the data generating process. On input s and $\hat{f}(s)$, what is the probability that T marks every $1 \leq j \leq n$ with $h(j) = k$, for a particular $k \leq \log n$? Conditioned on any particular realization (w_1, \dots, w_{j-1}) of the outcome sequence for the first $j - 1$ periods where $h(j) = k$, let $u \in S$ be the smallest outcome among those with the smallest probability assigned by $\hat{f}_j(w_1, \dots, w_{j-1})$. Then j gets marked if and only if s_j turns out to be u , which happens with probability at most $1/K$ since u is least probable among all K possible outcomes. Since the conditional realization (w_1, \dots, w_{j-1}) is arbitrary, the probability that T marks every $1 \leq j \leq n$ with $h(j) = k$ is at most

$$(1/K)^{|\{1 \leq j \leq n \mid h(j) = k\}|}.$$

For $k \leq \log n$, it is not hard to see that

$$|\{1 \leq j \leq n \mid h(j) = k\}| > \sqrt{n}.$$

Therefore, the probability that T marks every $1 \leq j \leq n$ with $h(j) = k$ for some $k \leq \log n$ is at most

$$\sum_{k=1}^{\log n} (1/K)^{\sqrt{n}} = o(1).$$

That is, the correct forecast sequence is rejected only with $o(1)$ probability. This completes item 1.

We now move on to item 2. Fix a k and consider F_k . From the construction of s^* , the forecast sequence $F_k(s_1^*, \dots, s_{n-1}^*)$ made by F_k is such that for every j

with $h(j) = k$, s_j^* is the smallest among the outcomes to which $F_k(s_1^*, \dots, s_{j-1}^*; j)$ assigns the smallest probability. Therefore, on input (s_1^*, \dots, s_n^*) and the forecast sequence $F_k(s_1^*, \dots, s_{n-1}^*)$ announced by F_k , each j with $h(j) = k$ gets marked by T and thus T rejects if n satisfies $k \leq \log n$. \square

The next theorem generalizes Lemma 4 by including a class of randomized forecasters.

Theorem 5. *Let $S \equiv \{1, \dots, K\}$ be the finite state space. There is a polynomial-time test T and an infinite sequence $s^* = (s_1^*, s_2^*, \dots)$ over S with the following properties:*

1. *For each duration n , the test T passes the truth with high probability.*
2. *Consider any forecaster F that adopts an arbitrary distribution over deterministic Turing-machine forecasters to select one machine for use, and then uses that same machine for all future forecasts. The test T rejects*

$$((s_1^*, \dots, s_n^*), F(s_1^*, \dots, s_{n-1}^*))$$

with high probability for sufficiently large n .

Proof. T and s^* are as in Lemma 4, which already shows that T passes the truth with high probability for each duration n . Let F adopt a distribution \mathcal{Q} over deterministic Turing-machine forecasters to select one machine for use, and then uses that same machine for all future forecasts. Let F_1, F_2, \dots be an enumeration of deterministic Turing-machine forecasters. Clearly,

$$\sum_{i=1}^{\infty} \Pr [F \text{ selects } F_i] = 1.$$

For any $\epsilon > 0$, there is an $m \in \mathbb{N}$ such that

$$\sum_{i=1}^m \Pr [F \text{ selects } F_i] > 1 - \epsilon. \tag{3}$$

Assume the data generating process adopts s^* as the outcome sequence. Lemma 4 guarantees that T rejects all F_1, \dots, F_m for sufficiently long duration n . Once F selects a machine for forecasting, it stays with that machine thereafter. Hence inequality (3) shows that, with probability more than $1 - \epsilon$, T rejects F . \square

4 Impossibility of Working for All Durations

A natural generalization of Theorem 1 to the case where the duration n is unknown to the forecaster is to have the forecaster implied in Theorem 1 pass the test with high probability for all durations n and data generating processes. This is because a forecaster who does not know the duration n beforehand is

guaranteed to pass a test with high probability if and only if it is so for every duration n (note that $n \in \mathbb{N}$ may be arbitrary). Formally, given an arbitrary test T that passes the truth with high probability, is it possible to find a forecaster F such that for each duration n and each data generating process, F is accepted by T with high probability? However, this is impossible as the following theorem shows.

Theorem 6. *Let $S \equiv \{1, \dots, K\}$ be the finite state space and $0 < \epsilon < 1$. There is a polynomial-time test T with the following properties:*

1. *For each duration n , the test T passes the truth with probability greater than $1 - \epsilon$.*
2. *For every randomized forecaster F and $\delta > 0$, there are infinitely many $n \in \mathbb{N}$ and outcome sequences $s^* = (s_1^*, \dots, s_n^*)$ such that T accepts $(s^*, F(s_1^*, \dots, s_{n-1}^*))$ with probability at most $1/2 + \delta$.*

Proof. We first describe the test T . Consider an arbitrary outcome sequence $s = (s_1, \dots, s_n)$ and a forecast sequence $f = (f_1, \dots, f_n)$. Denote by $f_i^{[1]}$ the probability f_i assigns to the outcome $1 \in S$. On input (s, f) , the test T rejects only in the following two cases:

case 1. $s_1 = \dots = s_n = 1$ and $\prod_{i=1}^n f_i^{[1]} < \epsilon/2$.

case 2. $s_1 = \dots = s_{n-1} = 1, s_n \neq 1$, and $f_n^{[1]} > 1 - \epsilon/2$.

The test T clearly runs in polynomial time.

To argue item 1, fix $n \in \mathbb{N}$ and a distribution \mathcal{P} adopted by the data generating process. Let $s = (s_1, \dots, s_n)$ be the outcome sequence for the first n periods and $\hat{f}(s)$ be the corresponding correct forecast sequence. Here s and thus $\hat{f}(s)$ are random variables whose distributions are determined by the data generating process. By the definition of the correct forecast sequence $\hat{f}(s)$, the probability that $s_1 = \dots = s_n = 1$ is

$$\prod_{i=1}^n \Pr [s_i = 1 \mid s_1 = \dots = s_{i-1} = 1] = \prod_{i=1}^n \hat{f}_i^{[1]}(1^{i-1}),$$

where $\hat{f}_i^{[1]}(1^{i-1})$ is the probability assigned to the outcome $1 \in S$ in $\hat{f}_i(1^{i-1})$. Note that $\hat{f}_i^{[1]}(1^{i-1})$ is not a random variable but a fixed number determined by \mathcal{P} for $1 \leq i \leq n$. Now, feed $(s, \hat{f}(s))$ to T . The test T rejects due to case 1 only if s_1, \dots, s_n all turn out to be 1 and $\prod_{i=1}^n \hat{f}_i^{[1]}(1^{i-1}) < \epsilon/2$. But

$$\prod_{i=1}^n \hat{f}_i^{[1]}(1^{i-1})$$

is precisely the probability that $s_1 = \dots = s_n = 1$. Hence, if $\prod_{i=1}^n \hat{f}_i^{[1]}(1^{i-1}) < \epsilon/2$ holds, T rejects due to case 1 with probability less than $\epsilon/2$. Similarly, T rejects due to case 2 only if s_1, \dots, s_{n-1} turn out to be 1 but s_n turns out otherwise, and $\hat{f}_n^{[1]}(1^{n-1}) > 1 - \epsilon/2$. Assume $\hat{f}_n^{[1]}(1^{n-1}) > 1 - \epsilon/2$. When conditioned on

$s_1 = \dots = s_{n-1} = 1$, the probability that s_n turns out 1 is $\hat{f}_n^{[1]}(1^{n-1}) > 1 - \epsilon/2$, so that s_n turns out otherwise with probability less than $\epsilon/2$. Therefore, the probability that T rejects due to case 2 is less than $\epsilon/2$. To sum up, the probability that T rejects $(s, \hat{f}(s))$ on either case is less than $\epsilon/2 + \epsilon/2 = \epsilon$. This completes item 1.

We now move on to item 2. Let F be an arbitrary (possibly randomized) forecaster and $\delta > 0$. Let $m > d_{\epsilon,\delta}$ for some $d_{\epsilon,\delta}$ to be determined later. Assume the data generating process picks the outcome sequence $1^m \in S^m$ for the first m periods. T rejects $(1^m, F(1^{m-1}))$ because of case 1 if $\prod_{i=1}^m F^{[1]}(1^{i-1}; i) < \epsilon/2$. Thus, either

$$\Pr [T \text{ accepts } (1^m, F(1^{m-1}))] \leq \frac{1}{2} + \delta, \tag{4}$$

or

$$\Pr \left[\prod_{i=1}^m F^{[1]}(1^{i-1}; i) \geq \epsilon/2 \right] > \frac{1}{2} + \delta, \tag{5}$$

where the probability is taken over the random variables in $F(1^{m-1})$. Assume inequality (5) holds. The event $\prod_{i=1}^m F^{[1]}(1^{i-1}; i) \geq \epsilon/2$ implies

$$\left| \{i \mid F^{[1]}(1^{i-1}; i) \leq 1 - \epsilon/2, 1 \leq i \leq m\} \right| \leq c_\epsilon$$

where

$$c_\epsilon \equiv \frac{\log(\epsilon/2)}{\log(1 - \epsilon/2)}.$$

The above and inequality (5) imply

$$\Pr \left[\left| \{i \mid F^{[1]}(1^{i-1}; i) \leq 1 - \epsilon/2, 1 \leq i \leq m\} \right| \leq c_\epsilon \right] > \frac{1}{2} + \delta$$

where the probability is taken over the random variables in $F(1^{m-1})$. Now set $d_{\epsilon,\delta} \equiv c_\epsilon(1 + 2\delta)/\delta$. The above inequality trivially implies

$$\Pr \left[\left| \{i \mid F^{[1]}(1^{i-1}; i) \leq 1 - \epsilon/2, m - d_{\epsilon,\delta} < i \leq m\} \right| \leq c_\epsilon \right] > \frac{1}{2} + \delta,$$

which is equivalent to

$$\Pr \left[\left| \{i \mid F^{[1]}(1^{i-1}; i) > 1 - \epsilon/2, m - d_{\epsilon,\delta} < i \leq m\} \right| \geq d_{\epsilon,\delta} - c_\epsilon \right] > \frac{1}{2} + \delta \tag{6}$$

where the probability is taken over the random variables in $F(1^{m-1})$.

Now add the random variable r uniformly distributed over $\{m - d_{\epsilon,\delta} + 1, m - d_{\epsilon,\delta} + 2, \dots, m\}$ and independent of the random variables in $F(1^{m-1})$. Inequality (6) and the independence of r from the random variables in $F(1^{m-1})$ imply

$$\Pr \left[F^{[1]}(1^{r-1}; r) > 1 - \epsilon/2 \right] > \left(\frac{1}{2} + \delta \right) \frac{d_{\epsilon,\delta} - c_\epsilon}{d_{\epsilon,\delta}} \tag{7}$$

where the probability is taken over r and the random variables in $F(1^{m-1})$. Due to the independence of r from the random variables in $F(1^{m-1})$, we have

$$\Pr \left[F^{[1]}(1^{r-1}; r) > 1 - \epsilon/2 \right] = \sum_{i=m-d_{\epsilon,\delta}+1}^m \frac{\Pr \left[F^{[1]}(1^{i-1}; i) > 1 - \epsilon/2 \right]}{d_{\epsilon,\delta}} \tag{8}$$

where the probability on the left-hand side is taken over r and the random variables in $F(1^{m-1})$, and those within the summation are over the random variables in $F(1^{m-1})$. Inequalities (7) and (8) imply the existence of a number $i(m) \in \{m - d_{\epsilon,\delta} + 1, m - d_{\epsilon,\delta} + 2, \dots, m\}$ with

$$\Pr \left[F^{[1]}(1^{i(m)-1}; i(m)) > 1 - \epsilon/2 \right] > \left(\frac{1}{2} + \delta \right) \cdot \frac{d_{\epsilon,\delta} - c_\epsilon}{d_{\epsilon,\delta}} = \frac{1}{2} + \frac{\delta}{2} \tag{9}$$

where the probability is taken over the random variables in $F(1^{m-1})$.

Consider the sequence $1^{i(m)-1}2 \in S^*$ standing for $\overbrace{1 \cdots 1}^{i(m)-1} 2$. Observe that inequality (9) holds also when the probability is taken over the random variables in $F(1^{i(m)-1})$ because 1^m and $1^{i(m)-1}$ share a common prefix, $1^{i(m)-1}$. Inequality (9) therefore says that, if the data generating process adopts the outcome sequence $1^{i(m)-1}2$ for the first $i(m)$ periods, then with probability more than $1/2 + \delta/2$ the forecast that F makes for the $i(m)$ th period attaches more than $1 - \epsilon/2$ to the probability on the outcome 1. But then T rejects $(1^{i(m)-1}2, F(1^{i(m)-1}))$ with probability more than $1/2 + \delta/2$ (see case 2). In summary, for each $m > d_{\epsilon,\delta}$, either inequality (4) holds, or T rejects $(1^{i(m)-1}2, F(1^{i(m)-1}))$ with probability more than $1/2 + \delta/2$. \square

Our proof of Theorem 6 actually shows the following stronger result.

Theorem 7. *Let $S \equiv \{1, \dots, K\}$ be the finite state space and $0 < \epsilon < 1$. There is a polynomial-time test T with the following properties:*

1. *For each duration n , the test T passes the truth with probability greater than $1 - \epsilon$.*
2. *Consider an arbitrary, possibly randomized forecaster F and $\delta > 0$. Let $d_{\epsilon,\delta} \equiv (1 + 2\delta) \log(\epsilon/2) / (\delta \log(1 - \epsilon/2))$. For sufficiently large $m \in \mathbb{N}$, there is an $n \in \{m - d_{\epsilon,\delta} + 1, m - d_{\epsilon,\delta} + 2, \dots, m\}$ and an outcome sequence $s^* = (s_1^*, \dots, s_n^*) \in S^n$ such that T accepts $(s^*, F(s_1^*, \dots, s_{n-1}^*))$ with probability at most $1/2 + \delta$.*

Thus, for the test T in Theorem 7 the outcome sequences on which a forecaster (computable or not) cannot perform well are not rare. One difference of Theorem 7 from OS06a is that T may accept sequence pairs with prefixes rejected by T .

5 Conclusion

We have built computationally efficient tests with various desirable properties. The tests that we construct pass the truth with high probability, and it is hard

in various ways to pass these tests for all data generating processes. Thus, it is hard for a forecaster to pass our tests without prior knowledge on the underlying data generating process.

Unlike most previous works except that by Fortnow and Vohra [FV06], our results take a computational perspective on forecast testing. Our tests run in polynomial time, and our first result requires the forecaster to have enormous computational power to pass the test on all data generating processes. As suggested by Fortnow and Vohra [FV06], we believe that taking a computational perspective may shed some light on many other problems previously studied without computational considerations.

References

- [ANW06] Al-Najjar, N.I., Weinstein, J.: Comparative testing of experts, Levine's Working Paper Archive 321307000000000590, Department of Economics, UCLA (2006)
- [AW] Adams-Watters, F.T.: <http://www.research.att.com/~njas/sequences/a002260.txt>.
- [Daw82] Dawid, A.P.: The well calibrated Bayesian. *Journal of the American Statistical Association* 77(379), 605–613 (1982)
- [DF99] Levine, D., Fudenberg, D.: Conditional universal consistency. *Games and Economic Behavior* 29, 104–130 (1999)
- [DF06] Dekel, E., Feinberg, Y.: Non-Bayesian testing of a stochastic prediction. *Review of Economic Studies* 73(4), 893–906 (2006)
- [Fan53] Fan, K.: Minimax theorems. *Proceedings of the National Academy of Science USA*. 39, 42–47 (1953)
- [FS07] Feinberg, Y., Stewart, C.: Testing multiple forecasters. Research Paper Series 1957, Graduate School of Business, Stanford University (2007)
- [FV98] Foster, D.P., Vohra, R.V.: Asymptotic calibration. *Biometrika* 85(2), 379–390 (1998)
- [FV06] Fortnow, L., Vohra, R.V.: The complexity of forecast testing, Tech. Report TR06-149, Electronic Colloquium on Computational Complexity (2006)
- [Leh01] Lehrer, E.: Any inspection rule is manipulable. *Econometrica* 69(5), 1333–1347 (2001)
- [OS06a] Olszewski, W., Sandroni, A.: Counterfactual predictions, Tech. report, Northwestern University, Department of Economics (2006)
- [OS06b] Olszewski, W., Sandroni, A.: Strategic manipulation of empirical tests, Tech. report, Northwestern University, Department of Economics (2006)
- [San03] Sandroni, A.: The reproducible properties of correct forecasts. *International Journal of Game Theory* 32(1), 151–159 (2003)
- [SSV03] Sandroni, A., Smorodinsky, R., Vohra, R.V.: Calibration with many checking rules. *Mathematics of Operations Research* 28(1), 141–153 (2003)

When Does Greedy Learning of Relevant Attributes Succeed?

— A Fourier-Based Characterization —

Jan Arpe* and Rüdiger Reischuk

Institut für Theoretische Informatik, Universität zu Lübeck
Ratzeburger Allee 160, 23538 Lübeck, Germany
{arpe, reischuk}@tcs.uni-luebeck.de

Abstract. We introduce a new notion called *Fourier-accessibility* that allows us to precisely characterize the class of Boolean functions for which a standard greedy learning algorithm successfully learns all relevant attributes. If the target function is Fourier-accessible, then the success probability of the greedy algorithm can be made arbitrarily close to one. On the other hand, if the target function is *not* Fourier-accessible, then the error probability tends to one. Finally, we extend these results to the situation where the input data are corrupted by random attribute and classification noise and prove that greedy learning is quite robust against such errors.

1 Introduction

For many application areas, greedy strategies are natural and efficient heuristics. In some cases, such as simple scheduling problems, greedy strategies find a global optimum (see, e.g., [21], Chap. 4). For the vast majority of optimization problems, however, greedy heuristics do not achieve optimal solutions for all inputs. In such a case, one can sometimes show that the greedy algorithm at least achieves a nontrivial approximation to an optimal solution. A prominent example is the greedy algorithm for the SET COVER problem [20, 16, 27, 17].

A different approach is to ask “What is the subset of the input space for which a greedy algorithm outputs an optimal solution?”. This question has rarely been answered. One notable exception is the characterization of transportation problems using the Monge property by Shamir and Dietrich [26].

We investigate the performance of greedy algorithms for the problem of *relevant feature selection*. Confronted with an unknown *target function* $f : \{0, 1\}^n \rightarrow \{0, 1\}$ that is only specified by randomly drawn examples $(x^k, f(x^k))$, $x^k \in \{0, 1\}^n$, $k = 1, \dots, m$, the task is to detect which variables x_i (also referred to as *attributes* or *features*) are relevant to f . This problem is central to many data mining applications; specifically, if f is a so-called *d-junta*, which means that it only depends on a small number d of all n attributes. A survey of this topic has been provided by Blum and Langley [13].

To infer relevant attributes from randomly drawn examples, the key task is to find a minimal set of attributes R admitting a consistent hypothesis h (i.e., $h(x^k) = f(x^k)$)

* Supported by DFG research grant Re 672/4.

for all k) that depends only on the variables in R . By standard arguments [14], once the sample size m exceeds $\text{poly}(2^d, \log n)$, with high probability there remains only one such hypothesis—the target function itself. Finding such a set R is equivalent to solving the following SET COVER instance. The ground set is the set of all pairs $\{k, \ell\}$ such that $f(x^k) \neq f(x^\ell)$. A pair $\{k, \ell\}$ may be covered by an attribute x_i if $x_i^k \neq x_i^\ell$. The goal is to cover the ground set by as few attributes as possible. Using this reduction, we can apply the greedy heuristic for SET COVER: the algorithm, which we call GREEDY, successively selects the attribute that covers most of the remaining edges and deletes them [20, 16].

For relevant feature selection, this approach has been proposed by Almuallim and Dietterich [4] and Akutsu and Bao [1]. Experimental results have been obtained in various areas [4, 2, 3, 15]. Akutsu et al. [3] have shown how to implement GREEDY such that its running time is only $O(d \cdot m \cdot n)$.

In this paper, we are mainly concerned with uniformly distributed attributes. For this case, Akutsu et al. [3] have proven that with high probability, GREEDY successfully infers the relevant variables for the class of Boolean monomials and that a small sample of size $\text{poly}(2^d, \log n)$ already suffices. Fukagawa and Akutsu [18] have extended this result to functions f that are unbalanced with respect to all of their relevant variables (i.e., for x uniformly chosen at random, $\Pr[f(x) = 1 | x_i = 0] \neq \Pr[f(x) = 1 | x_i = 1]$ for each relevant x_i).

Our first major result is a concise characterization of the class of target functions for which GREEDY is able to infer the relevant variables. This class properly contains the functions mentioned above. The new characterization is based on a property of the Fourier spectrum of the target function, which we call *Fourier-accessibility*. Recall that for $I \subseteq [n]$, the *Fourier coefficient* $\hat{f}(I)$ is equal to the correlation between $f(x)$ and the parity $\bigoplus_{i \in I} x_i$ (see Sect. 2 for a precise definition). A function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is

Table 1. Examples of Boolean functions and their Fourier spectra

$f(x_1, x_2, x_3)$	$\hat{f}(\emptyset)$	$\hat{f}(1)$	$\hat{f}(2)$	$\hat{f}(3)$	$\hat{f}(\{1, 2\})$	$\hat{f}(\{1, 3\})$	$\hat{f}(\{2, 3\})$	$\hat{f}(\{1, 2, 3\})$
$f_1 = x_1 \oplus (x_2 \wedge x_3)$	1/2	-1/4	0	0	-1/4	-1/4	0	1/4
$f_2 = (x_1 \oplus x_2) \wedge x_3$	1/4	0	0	-1/4	-1/4	0	0	1/4

Fourier-accessible if for each relevant variable, one can find a sequence $\emptyset \subsetneq I_1 \subsetneq \dots \subsetneq I_s \subseteq [n]$ such that $i \in I_s$ and for all $j \in \{1, \dots, s\}$, $|I_j \setminus I_{j-1}| = 1$ and $\hat{f}(I_j) \neq 0$.

We prove that GREEDY correctly infers all relevant variables of Fourier-accessible d -juntas from $m = \text{poly}(2^d, \log n, \log(1/\delta))$ uniformly distributed examples with probability at least $1 - \delta$. On the other hand, it is shown that if a function f is *not* Fourier-accessible, then the error probability of GREEDY is at least $1 - d^2/(n - d)$. In particular, this probability tends to 1 if d is fixed and $n \rightarrow \infty$, or if $d \rightarrow \infty$ and $n \in \omega(d^2)$. Thus, the average-case analysis of the greedy algorithm results in a *dichotomy*: for a given function, *either* the relevant variables are inferred correctly with high probability, *or* with high probability at least some relevant variables are not detected at all.

Coping with errors in the input data has been well studied in numerical analysis, but hardly in discrete algorithms. We have shown in [7, 8] that the relevant attributes of a

d -junta can still be learned efficiently if the input data are corrupted by random noise. In Sect. 6 we describe how to extend our analysis to noisy situations and show that greedy learning is highly fault-tolerant.

There is a long tradition of relating algorithmic learning problems to spectral properties of Boolean functions, see, e.g., [22, 23, 12]. Specifically, Mossel et al. [24] have combined spectral and algebraic methods to reduce the worst-case running time for learning the class of all n -ary d -juntas to roughly $n^{0.7 \cdot d}$ (a trivial approach is to test all $\Theta(n^d)$ sets of potentially relevant variables). The novelty of our analysis lies in the following. While the greedy algorithm investigated in this paper does *not* exploit any properties of the Fourier spectrum explicitly, we show that Fourier-accessibility is necessary and sufficient for this algorithm to work successfully.

There is a simple Fourier-based algorithm that learns the relevant attributes of Fourier-accessible functions: It estimates all first-level coefficients $\hat{f}(i)$ until it finds a nonzero coefficient, which implies that x_i is relevant. Subsequently, it recurses for the subfunctions $f_{x_i=0}$ and $f_{x_i=1}$ (see also [24]). The point is that this simple greedy algorithm coincidentally also succeeds for exactly those functions. While one may argue in favour of the Fourier-based algorithm that it can easily be generalized to cope with functions with vanishing Fourier coefficients at low levels, it is explained in [6] how the greedy algorithm can also be extended naturally to cope with such functions as well.

This paper is organized as follows. The terminology and the learning model are introduced in Sect. 2. The reduction to SET COVER and the GREEDY algorithm are presented in Sect. 3. Sect. 4 provides three major lemmata used in the proof of our main results for GREEDY, which are presented in Sect. 5. In Sect. 6 we study the robustness of GREEDY against corruption of the input data. In Sect. 7 issues of further research are discussed. Due to space constraints, most proofs have been omitted. A technical report for the noise-free case is available [9]. A detailed exposition of the topics presented in this paper including all proofs can be found in the first author's PhD thesis [6].

2 Preliminaries

For $n \in \mathbb{N}$, let $[n] = \{1, \dots, n\}$. We consider the problem of inferring the relevant variables of an unknown function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ from randomly drawn examples. Variable x_i is *relevant* to f if $f_{x_i=0} \neq f_{x_i=1}$, where $f_{x_i=a}$ denotes the restriction of f with variable x_i set to a . The set of variables that are relevant to f is denoted by $\text{rel}(f)$, whereas $\text{irrel}(f)$ denotes the set of variables that are *irrelevant* (i.e., not relevant) to f . A function with $|\text{rel}(f)| \leq d$ is called a d -junta. The restriction of f to its relevant variables is called its *base function*.

We assume that an algorithm for inferring the relevant variables receives a sequence S of randomly generated *examples* (x^k, y^k) , $k \in [m]$, where $x^k \in \{0, 1\}^n$ is drawn according to the uniform distribution and $y^k = f(x^k) \in \{0, 1\}$. Such a sequence is called a *sample* for f of size m . If for another function h , $y^k = h(x^k)$ for all $k \in [m]$, h is said to be *consistent* with S .

If x is randomly generated, then $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is a Bernoulli random variable, thus we will use the notation $\Pr[f = b] = \Pr_x[f(x) = b]$ for $b \in \{0, 1\}$ and $\text{Var}(f) =$

$\Pr[f = 0] \Pr[f = 1]$. The following well-known Chernoff bound can, e.g., be found in [5].

Lemma 1 (Chernoff Bound). *Let X be a random variable that is binomially distributed with parameters n and p , and let $\mu = pn$ be the expectation of X . Then for all ϵ with $0 \leq \epsilon \leq 1$, $\Pr[|X - \mu| > \epsilon n] < 2e^{-2\epsilon^2 n}$.*

The space $\mathbb{R}^{\{0,1\}^n}$ of real-valued functions on the hypercube with inner product $\langle f, g \rangle = 2^{-n} \sum_{x \in \{0,1\}^n} f(x)g(x)$ is a Hilbert space of dimension 2^n . It has an orthonormal basis $(\chi_I \mid I \subseteq [n])$, where $\chi_I(x) = (-1)^{\sum_{i \in I} x_i}$ for $x \in \{0, 1\}^n$, see, e.g., [11]. Let $f : \{0, 1\}^n \rightarrow \mathbb{R}$ and $I \subseteq [n]$. The Fourier coefficient of f at I is $\hat{f}(I) = 2^{-n} \sum_{x \in \{0,1\}^n} f(x) \cdot \chi_I(x)$. Thus, $\hat{f}(I)$ measures the correlation between $f(x)$ and $\chi_I(x)$. If $I = \{i\}$, we write $\hat{f}(i)$ instead of $\hat{f}(\{i\})$. We have the Fourier expansion formula $f(x) = \sum_{I \subseteq [n]} \hat{f}(I) \cdot \chi_I(x)$ for all $x \in \{0, 1\}^n$.

Definition 1 (Fourier support). *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$. The Fourier support of f is $\text{supp}(\hat{f}) = \{I \subseteq [n] \mid \hat{f}(I) \neq 0\}$. The Fourier support graph $\text{FSG}(f)$ of f is the subgraph of the n -dimensional Hamming cube induced by $\text{supp}(\hat{f})$.*

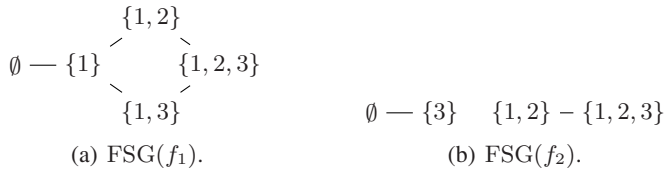


Fig. 1. Fourier support graphs of functions f_1 and f_2 presented in Table 1

Fourier coefficients are connected to relevant variables as follows (cf. [24, 8]):

Lemma 2. *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$. Then for all $i \in [n]$, x_i is relevant to f if and only if there exists $I \subseteq [n]$ such that $i \in I$ and $\hat{f}(I) \neq 0$.*

Hence, whenever we find a nonzero Fourier coefficient $\hat{f}(I)$, we know that all variables $x_i, i \in I$, are relevant to f . Moreover, all relevant variables can be detected in this way, and we only have to check out subsets of size at most $d = |\text{rel}(f)|$. However, there are $\Theta(n^d)$ such subsets, an amount that one would generally like to reduce. The best known learning algorithm to date that for all d -juntas is guaranteed to find the relevant attributes runs in time roughly $n^{0.7 \cdot d}$ [24]. In contrast, greedy heuristics require only time polynomial in n with an exponent independent of d . For our characterization of the functions to which GREEDY is applicable, we introduce the concept of *Fourier-accessibility*.

Definition 2 (Fourier-accessible). *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ and $i \in [n]$. Variable x_i is accessible (w.r.t. f) if there exists a sequence $\emptyset = I_0 \subsetneq I_1 \subsetneq \dots \subsetneq I_s \subseteq [n]$ such that (1) $i \in I_s$, (2) for all $j \in [s]$, $|I_j \setminus I_{j-1}| = 1$, and (3) for all $j \in [s]$, $\hat{f}(I_j) \neq 0$. The set of variables that are accessible with respect to f is denoted by $\text{acc}(f)$, whereas the set of inaccessible variables with respect to f is denoted by $\text{inacc}(f)$. The function f is called Fourier-accessible if and only if every variable that is relevant to f is also accessible, i.e., $\text{acc}(f) = \text{rel}(f)$.*

Equivalently, x_i is accessible if and only if there exists $I \in \text{supp}(\hat{f})$ with $i \in I$ such that there is a path in $\text{FSG}(f)$ from \emptyset to I . Since $\hat{f}(\emptyset) = \Pr[f(x) = 1]$, $\emptyset \in \text{supp}(\hat{f})$ whenever $f \not\equiv 0$. Hence f is Fourier-accessible if and only if the union of all subsets $I \in \text{supp}(\hat{f})$ that belong to the connected component of \emptyset in $\text{FSG}(f)$ equals $\text{rel}(f)$.

Throughout the paper, if f is clear from the context, we call a variable that is relevant to f simply *relevant*. Similarly, a variable that is accessible with respect to f is simply called *accessible*. Simple examples of a Fourier-accessible function f_1 and a non-Fourier-accessible function f_2 are given in Table 1. The corresponding Fourier support graphs are presented in Fig. 1.

In our algorithm analyses, we will consider the *expanded attribute space* of attributes $x_I = \bigoplus_{i \in I} x_i$ for $I \subseteq [n]$. The connection between these expanded attributes and the functions χ_I used to define the Fourier transform is given by $\chi_I(x) = (-1)^{x_I}$.

3 The Reduction to Set Cover and the Greedy Algorithm

With a sample $S = (x^k, y^k)_{k \in [m]} \in (\{0, 1\}^n \times \{0, 1\})^m$, we associate the *functional relations graph* $G_S = (V, E)$ which is defined as follows (see also [3, 7]). Its vertices correspond to the examples of S , i.e., $V = [m]$. They are partitioned into the subset of examples $V^{(0)}$ with $y^k = 0$, and the examples $V^{(1)}$ with $y^k = 1$. G_S is the complete bipartite graph with the vertex set partition $[m] = V^{(0)} \cup V^{(1)}$. Given S , our primary goal is to determine a set of variables $R \subseteq \{x_1, \dots, x_n\}$ such that there exists *some* function $g : \{0, 1\}^n \rightarrow \{0, 1\}$ with $\text{rel}(g) \subseteq R$ that is consistent with the sample. In this case, R is said to *explain the sample*. Note that g may not be identical to the original function f , nor may the set R contain all relevant variables of f .

In order to find an explaining set of variables, we have to specify, for each edge $\{k, \ell\} \in E$, a relevant variable that differs in x^k and x^ℓ . Such a variable is said to *explain the edge*. Formally, an edge $\{k, \ell\} \in E$ may be *covered* by attribute x_i if and only if $x_i^k \neq x_i^\ell$. The set of edges that may be covered by x_i is denoted by E_i . A set R of variables thus explains the sample S if and only if these variables explain all edges. The previous discussion is formally summarized by the following lemma:

Lemma 3. *Let $S \in (\{0, 1\}^n \times \{0, 1\})^m$ be a sample and $R \subseteq \{x_1, \dots, x_n\}$. Then R explains S if and only if $E = \cup_{x_i \in R} E_i$, where E is the edge set of the functional relations graph G_S .*

The lemma provides a reduction from the problem of inferring small sets of explaining variables to the problem of finding a small cover of E by sets from E_1, \dots, E_n . This allows us to use algorithms for the set cover problem to find explaining variables. The best known and most generic algorithm for this problem is a greedy algorithm that successively picks a set that covers the largest amount of elements not covered so far. This algorithm, which we call **GREEDY**, is defined as follows.

If there are several sets of maximum cardinality in step 7, **GREEDY** picks one of them at random. The notion of success for **GREEDY** is captured as follows.

Definition 3 (λ -success). *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$, S be a sample for f , and $\lambda \geq 1$. **GREEDY** is λ -successful on input S if and only if $|\text{GREEDY}(S)| \leq \lambda \cdot |\text{rel}(f)|$ and*

Algorithm 1.. GREEDY

```

1: input  $S = ((x_1^k, \dots, x_n^k), y^k)_{k \in [m]}$ 
2:  $E \leftarrow \{\{k, \ell\} \mid k, \ell \in [m], y^k \neq y^\ell\}$ 
3:  $R \leftarrow \emptyset$ 
4: while  $E \neq \emptyset$  do
5:   for  $i = 1$  to  $n$  do
6:      $E_i \leftarrow \{\{k, \ell\} \in E \mid x_i^k \neq x_i^\ell\}$ 
7:   select  $x_i \notin R$  with maximum  $|E_i|$ 
8:    $E \leftarrow E \setminus E_i$ 
9:    $R \leftarrow R \cup \{x_i\}$ 
10: output GREEDY( $S$ ) =  $R$ 

```

GREEDY(S) \supseteq rel(f). GREEDY is successful (or succeeds) if and only if it is λ -successful, i.e., GREEDY(S) = rel(f), otherwise we say that it fails. GREEDY λ -fails if and only if it is not λ -successful.

4 Key Lemmata for the Algorithm Analysis

In this section, we provide three key lemmata that will be used in the proofs of our main results in Sect. 5. For technical reasons, it is useful to consider the edge sets $E_I = \{\{k, \ell\} \in E \mid x_I^k \neq x_I^\ell\}$ corresponding to the attributes from the expanded attribute space. Since x_I^k and x_I^ℓ differ if and only if the number of $i \in I$ with $x_i^k \neq x_i^\ell$ is odd, we obtain that $E_I = \Delta_{i \in I} E_i$, where Δ denotes the symmetric difference.

Suppose that GREEDY has put the variables x_{i_1}, \dots, x_{i_s} into R after s rounds. Hence, all edges in $E' = E_{i_1} \cup \dots \cup E_{i_s}$ have been covered. The number of remaining edges that can be covered by variable x_i in the next round is $|E_i \setminus E'|$. Provided that x_{i_1}, \dots, x_{i_s} are all relevant, we would like to estimate the set size $|E_i \setminus E'|$ in dependence of properties of f . As we do not see any direct way of doing so, we take a detour via the cardinalities of the sets E_I . These turn out to be quite efficiently approximable, as we will show in Lemma 5. But let us first show how to express the cardinality of $E_i \setminus E'$ in terms of the cardinalities of the sets $E_I, I \subseteq \{i_1, \dots, i_s\}$:

Lemma 4. *Let $S \in (\{0, 1\}^n \times \{0, 1\})^m$ be a sample and $G_S = (V, E)$ be the corresponding functional relations graph. Let $R \subsetneq [n]$ and $i^* \in [n] \setminus R$ and define $E' = \bigcup_{i \in R} E_i$. Then $|E_{i^*} \setminus E'| = 2^{-|R|} \sum_{I \subseteq R} (|E_{I \cup \{i^*\}}| - |E_I|)$.*

Now we are concerned with the estimation of the cardinalities $|E_I|, I \subseteq [n]$. For $a, b \in \{0, 1\}$, let $\alpha_I^{ab} = \Pr[x_I = a \wedge f(x) = b]$, where $x \in \{0, 1\}^n$ is drawn according to the uniform distribution. It follows that $\alpha_I^{a0} + \alpha_I^{a1} = \Pr[x_I = a] = 1/2$ for $I \neq \emptyset$ and $\alpha_I^{0b} + \alpha_I^{1b} = \Pr[f(x) = b]$ for all $I \subseteq [n]$.

A (noise-free) sample of size m consists of the outcomes of m independent draws of $x^k \in \{0, 1\}^n$ and the corresponding classifications $y^k = f(x^k) \in \{0, 1\}$. In the following, all probabilities and expectations are taken with respect to the random experiment of “drawing a sample of size m ” for an arbitrary but fixed m . For all $I \subseteq [n]$ and all pairs of example indices $k, \ell \in [m]$ with $k \neq \ell$, the probability that $\{k, \ell\} \in E_I$ is

$\Pr[x_I^k \neq x_I^\ell \wedge y^k \neq y^\ell] = 2(\alpha_I^{00}\alpha_I^{11} + \alpha_I^{10}\alpha_I^{01})$. Since there are $\frac{1}{2}(m-1)m$ such pairs, the expectation of $|E_I|$ is $\alpha_I(m-1)m$ with $\alpha_I = \alpha_I^{00}\alpha_I^{11} + \alpha_I^{10}\alpha_I^{01}$.

We prove a Chernoff style mass concentration for the cardinalities $|E_I|$. It shows that for a sufficiently large sample size, $|E_I|$ is likely to be close to $\alpha_I \cdot m^2$.

Lemma 5. *There exist $c_1, c_2 > 0$ such that for every $f : \{0, 1\}^n \rightarrow \{0, 1\}$, given a uniformly distributed sample S of size m for f , for all $I \subseteq [n]$ and all $\epsilon \in [0, 1]$, $\Pr[||E_I| - \alpha_I m^2| > \epsilon m^2] < c_1 e^{-c_2 \epsilon^2 m}$.*

Before stating the third lemma, let us briefly take a closer look at the cardinalities $|E_i|$ for irrelevant variables x_i . Since for these, the value of x_i is independent of the classification $f(x)$, $\alpha_i^{ab} = \frac{1}{2} \Pr[f(x) = b]$. Consequently, $\alpha_i = \frac{1}{2} \Pr[f(x) = 0] \Pr[f(x) = 1] = \frac{1}{2} \text{Var}[f]$. Hence, the expectation of $|E_i|$ is $\frac{1}{2} \text{Var}[f]m(m-1) \approx \frac{1}{2} \text{Var}[f]m^2$. The following lemma generalizes this result to arbitrary $I \subseteq [n]$, revealing an unexpected relationship between the cardinalities $|E_I|$ and the Fourier coefficients $\hat{f}(I)$. Recall that for $I \subseteq [n]$ with $I \not\subseteq \text{rel}(f)$, $\hat{f}(I) = 0$ by Lemma 2.

Lemma 6. *Let $I \subseteq [n]$ with $I \neq \emptyset$. Then $\alpha_I = (\text{Var}[f] + \hat{f}(I)^2)/2$.*

5 Analysis of Greedy

In this section, we state and prove our main results. Let us start with the positive result, the class of functions for which GREEDY is successful.

Theorem 1. *There is a polynomial p such that the following holds. Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a Fourier-accessible concept, $d = |\text{rel}(f)|$, and $\delta > 0$. Let S be a uniformly distributed sample for f of size $m \geq p(2^d, \log n, \log(1/\delta))$. Then $\text{GREEDY}(S) = \text{rel}(f)$ with probability at least $1 - \delta$.*

Proof. We only provide a brief sketch here, the full proof can be found in [6, 9].

First we can show that with probability at least $1 - \delta/2$, GREEDY outputs at least d variables, provided that m is sufficiently large. Once this has been shown, let the sequence of variables output by GREEDY start with x_{i_1}, \dots, x_{i_d} . By Lemma 5 it happens with probability at least $\rho = 1 - n^d \cdot c_1 \cdot e^{-c_2 \epsilon^2 m}$ that for all $I \subseteq [n]$ such that $1 \leq |I| \leq d$, we have $||E_I| - \alpha_I m^2| \leq \epsilon m^2$. For this case, we show by induction that the variables x_{i_1}, \dots, x_{i_s} are all relevant for $s \in [d]$. This implies that GREEDY halts exactly after d steps since E can always be covered by the sets E_i with $x_i \in \text{rel}(f)$. For $s = 0$, $R_0 = \emptyset \subseteq \text{rel}(f)$. For the induction step, we pick an $i^* \in \text{rel}(f) \setminus R_s$ and an $I^* \subseteq R_s$ such that $\hat{f}(I^* \cup \{i^*\}) \neq 0$ and hence $|\hat{f}(I^* \cup \{i^*\})| \geq 2^{-d}$ (since the Fourier coefficients of d -juntas are always multiples of 2^{-d}). Such an i^* exists since f is Fourier-accessible. Now we use Lemmata 4, 5, and 6 and obtain that for a suitable choice of ϵ in Lemma 5 every $x_j \in \text{irrel}(f)$ satisfies $|E_{i^*}^{(s)}| > |E_j^{(s)}|$. Consequently, in step $s + 1$, GREEDY prefers the relevant variable x_{i^*} to all irrelevant variables. Finally, we have to choose m such that $\rho \geq 1 - \delta/2$. □

Example 1. The function $f_1 : \{0, 1\}^3 \rightarrow \{0, 1\}$ in Table 1 is Fourier-accessible. By Theorem 1, for any function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ that has f_1 as its base function,

GREEDY succeeds with probability at least $1 - \delta$ for sample size polynomial in 2^d , $\log n$, and $\log(1/\delta)$.

If a function is not Fourier-accessible, then one of its relevant variables is not accessible. The proof of Theorem 1 shows that GREEDY first outputs all accessible variables with high probability. Once all of these have been output, the intuition is that the non-accessibility of the other relevant variables makes them statistically indistinguishable from the irrelevant variables. In particular, each inaccessible but relevant variable will be selected by GREEDY with the same probability as each irrelevant variable. Assuming that the number of irrelevant variables is much larger than the number of relevant ones, it becomes very likely that GREEDY picks an irrelevant variable and thus fails. The following result describes the class of functions for which GREEDY fails.

Theorem 2. *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a function that is not Fourier-accessible and $\lambda \geq 1$. Given a sample S for f of arbitrary size, GREEDY λ -fails on input S with probability at least $1 - \frac{\lambda d^2}{n - \lambda d}$, where $d = |\text{rel}(f)|$.*

Corollary 1. *Let $p_\lambda(n, d)$ denote the probability that for any given function f with $|\text{rel}(f)| = d$ that is not Fourier-accessible and for any uniformly distributed sample S for f , GREEDY λ -fails. Then for fixed $\lambda \geq 1$,*

- (a) for fixed d , $\lim_{n \rightarrow \infty} p_\lambda(n, d) = 1$ and
- (b) for $d \rightarrow \infty$ and $n = n(d) \in \omega(d^2)$, $\lim_{d \rightarrow \infty} p_\lambda(n, d) = 1$.

Example 2. The function $f_2 : \{0, 1\}^3 \rightarrow \{0, 1\}$ in Table 1 is not Fourier-accessible. By Theorem 2 for any function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ that has f_2 as its base function, GREEDY fails with probability at least $1 - \frac{9}{n-3}$.

Note that Theorem 2 not only says that GREEDY (with high probability) fails for functions that are not Fourier-accessible, but that GREEDY even fails to find all relevant variables of the target function in $\lambda \cdot |\text{rel}(f)|$ rounds for any $\lambda \geq 1$. In addition, note that the claim in Theorem 2 is independent of the sample size.

In the literature, it has often been emphasized that GREEDY has a “logarithmic approximation guarantee” (see [1, 3, 13, 18]), i.e., given a sample S for f of size m , GREEDY finds a set of at most $(2 \ln m + 1) \cdot |\text{rel}(f)|$ variables that explain S . Theorem 2 shows that if f is not Fourier-accessible, then with probability at least $\frac{(2 \ln m + 1)d^2}{n - (2 \ln m + 1)d}$, these variables *do not contain all relevant variables* (where $d = |\text{rel}(f)|$). Thus, GREEDY *misses* some relevant variable with high probability, provided that $m \in 2^{o(n)}$. Hence the positive approximability properties of the greedy strategy for the SET COVER problem do not translate to the learning situation. The fact that GREEDY outputs at most $(2 \ln m + 1) \cdot |\text{rel}(f)|$ variables only guarantees that any sample of size m can be explained by this amount of more or less arbitrary variables.

6 Robustness Against Noise

The technical analysis of GREEDY and the Fourier spectrum for d -juntas in the previous section can be extended to the situation where the input data contain errors.

We can show that GREEDY is *extremely robust* against noise, which will be modelled as follows. Instead of receiving suitable examples $(x, f(x))$, the learning algorithm now obtains *noisy examples* of the form $(x \oplus \xi, f(x) \oplus \zeta)$, where in the *noise vector* $\xi = (\xi_1, \dots, \xi_n) \in \{0, 1\}^n$ each ξ_i is set to 1 independently with probability p_i , and the *classification noise bit* $\zeta \in \{0, 1\}$ is set to 1 with probability η . To avoid that some attribute or the classification is turned into a purely random bit, the noise has to be bounded away from $1/2$: we require that there exist $\gamma_a, \gamma_b > 0$ such that $p_i \leq (1 - \gamma_a)/2$ for all $i \in [n]$ and $\eta \leq (1 - \gamma_b)/2$. Let P denote the product distribution on $\{0, 1\}^n$ induced by the probabilities p_1, \dots, p_n . A sample S for a function f that is corrupted by such a noise process is called a (P, η) -noisy sample.

We can show that given a (P, η) -noisy sample S for a Fourier-accessible function f of size polynomial in $2^d, \log(n/\delta), \gamma_a^{-d}$, and γ_b , GREEDY still outputs all relevant variables of f . It may be the case that the sets E_i that correspond to the relevant variables do not suffice to explain all edges in E due to noise. Even worse, it may happen that some edges cannot be explained at all: the sample may contain contradictive examples. For this reason, we have to employ a variant of GREEDY that gets the number d of relevant attributes as a parameter and outputs a set of d variables that is supposed to contain the relevant ones. In other words, the `while`-loop in line 4 of Algorithm 1 is replaced with the statement “`while` $|R| \leq d$ `do`”. We denote this algorithm by GREEDY_d .

We adjust the definition of the probabilities α_I^{ab} and define, for $a, b \in \{0, 1\}$, $\beta_I^{ab} = \Pr[x_I \oplus \xi_I = a \wedge f(x) \oplus \zeta = b]$, where $\Pr[\xi_i = 1] = p_i, \Pr[\zeta = 1] = \eta$, and $\xi_I = \bigoplus_{i \in I} \xi_i$. While in the noise-free scenario, the expectation of $|E_I|$ is $\alpha_I(m - 1)m$, the expectation of $|E_I|$ is now equal to $\beta_I(m - 1)m$ with $\beta_I = \beta_I^{00}\beta_I^{11} + \beta_I^{10}\beta_I^{01}$. The next lemma is completely analogous to Lemma 5:

Lemma 7. *There exist $c_1, c_2 > 0$ such that for every $f : \{0, 1\}^n \rightarrow \{0, 1\}$, given a uniformly distributed (P, η) -noisy sample S of size m for f , for all $I \subseteq [n]$ and $\epsilon \in [0, 1]$, $\Pr[||E_I| - \beta_I m^2| > \epsilon m^2] < c_1 e^{-c_2 \epsilon^2 m}$.*

Proving an analog of Lemma 6 requires some more computation:

Lemma 8. *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$, $I \subseteq [n]$ with $I \neq \emptyset$, and $\lambda_I = \prod_{i \in I} (1 - 2p_i)$. Then $\beta_I = \frac{1}{2} \left((1 - 2\eta)^2 \cdot \text{Var}[f] + \eta \cdot (1 - \eta) + (1 - 2\eta)^2 \cdot \lambda_I^2 \cdot \hat{f}(I)^2 \right)$*

Theorem 1 generalizes to the scenario of noisy data as follows:

Theorem 3. *There is a polynomial p such that the following holds. Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a Fourier-accessible function, $d = |\text{rel}(f)|$, and $\delta > 0$. Let S be a uniformly distributed (P, η) -noisy sample S for f of size $m \geq p(2^d, \log n, \log(1/\delta), \gamma_a^d, \gamma_b)$. Then $\text{GREEDY}_d(S) = \text{rel}(f)$ with probability at least $1 - \delta$.*

So far, we cannot exclude that there may be (fixed) noise distributions for which *strictly more* functions may be learned than can be learned without noise (compare to the situation of noisy circuits as discussed in [25]). However, we can show that Theorem 2 and Corollary 1 also hold for uniformly distributed (P, η) -noisy samples. The proofs are similar to those for the noise-free case.

The high fault tolerance of GREEDY can be further generalized to a situation where we do not have to assume statistical independence for the corruption of individual attributes. Instead of flipping each attribute value independently with probability p_i , let

the noise vectors ξ be drawn according to an arbitrary distribution $P : \{0, 1\}^n \rightarrow [0, 1]$ that satisfies $\Pr[\xi_I = 1] \leq \frac{1}{2}(1 - \gamma_a^{|I|})$ for all $I \subseteq [n]$ with $1 \leq |I| \leq d$. All results of this section are still valid in this setting (in Lemma 8 the definition of λ_I has to be replaced with $1 - 2 \Pr[\xi_I = 1]$). In fact, product distributions with $p_i \leq \frac{1}{2}(1 - \gamma_a)$ are a special case of this scenario. Further details of this most general result have to be omitted due to space constraints, but can be found in [6].

7 Concluding Remarks

The first issue left for future research is the investigation of the performance of the greedy algorithm in variations of the learning scenario considered in this paper: attributes and classifications may take more than two values, attributes may be non-uniformly distributed, etc.

For non-uniform attribute distributions—although a generic notion of Fourier coefficients can be given [10, 19]—Lemma 5 with a similar definition of α_I does not hold any more. It is easy to find examples such that (a) there are $x_i, x_j \in \text{irrel}(f)$ such that the expected sizes of E_i and E_j differ or (b) there are $x_i \in \text{rel}(f)$ and $x_j \in \text{irrel}(f)$ such that the expected sizes of E_i and E_j are equal, although $\hat{f}(i) = \hat{f}(j) = 0$. Thus, a completely different analysis is needed for such a setting. Again, we refer to [6, 9] for more details.

The second issue is to stick to the learning scenario and investigate variants of the greedy heuristic. If an edge is labeled by exactly one variable, then this variable has to be selected in order to explain the sample. For this reason, Almuallim and Dietterich [4] proposed to assign the weight $\sum_{e \in E_i} \frac{1}{c(e)-1}$ to x_i (where $c(e)$ is equal to the number of variables that can cover e) and then find a set cover by selecting variables of maximum weight. Since for $n \gg |\text{rel}(f)|$, each edge is labeled by roughly $n/2$ irrelevant variables, such a weighting is unlikely to help much during the first rounds of the algorithm. Thus, it seems unlikely that there are functions for which this heuristic outperforms the algorithm analyzed in this paper.

References

- [1] Akutsu, T., Bao, F.: Approximating Minimum Keys and Optimal Substructure Screens. In: Cai, J.-Y., Wong, C.K. (eds.) COCOON 1996. LNCS, vol. 1090, pp. 290–299. Springer, Heidelberg (1996)
- [2] Akutsu, T., Miyano, S., Kuhara, S.: Algorithms for Identifying Boolean Networks and Related Biological Networks Based on Matrix Multiplication and Fingerprint Function. *J. Comput. Biology* 7(3-4), 331–343 (2000)
- [3] Akutsu, T., Miyano, S., Kuhara, S., Simple, A.: A Simple Greedy Algorithm for Finding Functional Relations: Efficient Implementation and Average Case Analysis. *Theoret. Comput. Sci.* 292(2), 481–495 (2003)
- [4] Almuallim, H., Dietterich, T.G.: Learning Boolean Concepts in the Presence of Many Irrelevant Features. *Artificial Intelligence* 69(1-2), 279–305 (1994)
- [5] Alon, N., Spencer, J.: *The Probabilistic Method*. Wiley-Intersci. Ser. Discrete Math. Optim. John Wiley and Sons, Chichester (1992)

- [6] Arpe, J.: Learning Concepts with Few Unknown Relevant Attributes from Noisy Data. PhD thesis, Institut für Theoretische Informatik, Universität zu Lübeck (2006)
- [7] Arpe, J., Reischuk, R.: Robust Inference of Relevant Attributes. In: Gavaldá, R., Jantke, K.P., Takimoto, E. (eds.) ALT 2003. LNCS (LNAI), vol. 2842, pp. 99–113. Springer, Heidelberg (2003)
- [8] Arpe, J., Reischuk, R.: Learning Juntas in the Presence of Noise. In: Cai, J.-Y., Cooper, S.B., Li, A. (eds.) TAMC 2006. LNCS, vol. 3959, pp. 387–398. Springer, Heidelberg, Invited to appear in special issue of TAMC 2006 in Theoret. Comput. Sci., Series A (2006)
- [9] Arpe, J., Reischuk, R.: When Does Greedy Learning of Relevant Attributes Succeed?—A Fourier-based Characterization. Technical Report ECCC TR06-065, Electronic Colloquium on Computational Complexity (2006)
- [10] Bahadur, R.R.: A Representation of the Joint Distribution of Responses to n Dichotomous Items. In: Solomon, H. (ed.) Studies in Item Analysis and Prediction, pp. 158–168. Stanford University Press, Stanford (1961)
- [11] Bernasconi, A.: Mathematical Techniques for the Analysis of Boolean Functions. PhD thesis, Università degli Studi di Pisa, Dipartimento di Ricerca in Informatica (1998)
- [12] Blum, A., Furst, M., Jackson, J.C., Kearns, M., Mansour, Y., Rudich, S.: Weakly Learning DNF and Characterizing Statistical Query Learning Using Fourier Analysis. In: Proc. 26th STOC 1994, pp. 253–262 (1994)
- [13] Blum, A., Langley, P.: Selection of Relevant Features and Examples in Machine Learning. Artificial Intelligence 97(1-2), 245–271 (1997)
- [14] Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M.K.: Occam's Razor. Inform. Process. Lett. 24(6), 377–380 (1987)
- [15] Boros, E., Horiyama, T., Ibaraki, T., Makino, K., Yagiura, M.: Finding Essential Attributes from Binary Data. Ann. Math. Artif. Intell. 39(3), 223–257 (2003)
- [16] Chvátal, V.: A Greedy Heuristic for the Set Covering Problem. Math. Oper. Res. 4(3), 233–235 (1979)
- [17] Feige, U.: A Threshold of $\ln n$ for Approximating Set Cover. J. ACM 45(4), 634–652 (1998)
- [18] Fukagawa, D., Akutsu, T.: Performance Analysis of a Greedy Algorithm for Inferring Boolean Functions. Inform. Process. Lett. 93(1), 7–12 (2005)
- [19] Furst, M.L., Jackson, J.C., Smith, S.W.: Improved Learning of AC^0 Functions. In: Proc. 4th COLT 1991, pp. 317–325
- [20] Johnson, D.S.: Approximation Algorithms for Combinatorial Problems. J. Comput. System Sci. 9(3), 256–278 (1974)
- [21] Kleinberg, J., Tardos, É.: Algorithm Design. Addison-Wesley, Reading (2005)
- [22] Linial, N., Mansour, Y., Nisan, N.: Constant Depth Circuits, Fourier Transform, and Learnability. J. ACM 40(3), 607–620 (1993)
- [23] Mansour, Y.: Learning Boolean Functions via the Fourier Transform. In: Roychodhury, V., Siu, K.-Y., Orlitsky, A. (eds.) Theoretical Advances in Neural Computation and Learning, pp. 391–424. Kluwer Academic Publishers, Dordrecht (1994)
- [24] Mossel, E., O'Donnell, R.W., Servedio, R.A.: Learning functions of k relevant variables. J. Comput. System Sci. 69(3), 421–434 (2004)
- [25] Reischuk, R.: Can Large Fanin Circuits Perform Reliable Computations in the Presence of Noise? Theoretical Comput. Sci. 240(4), 319–335 (2000)
- [26] Shamir, R., Dietrich, B.: Characterization and Algorithms for Greedily Solvable Transportation Problems. In: Proc. 1st SODA 1990, pp. 358–366
- [27] Slavík, P.: A Tight Analysis of the Greedy Algorithm for Set Cover. In: Proc. 28th STOC 1996, pp. 435–441.

The Informational Content of Canonical Disjoint NP-Pairs

Christian Glaßer¹, Alan L. Selman^{2,*}, and Liyu Zhang²

¹ Lehrstuhl für Informatik IV, Universität Würzburg, Am Hubland,
97074 Würzburg, Germany

glasser@informatik.uni-wuerzburg.de

² Department of Computer Science and Engineering, University at Buffalo,
Buffalo, NY 14260

{selman,lzhang7}@cse.buffalo.edu

Abstract. We investigate the connection between propositional proof systems and their canonical pairs. It is known that simulations between proof systems translate to reductions between their canonical pairs. We focus on the opposite direction and study the following questions.

Q1: Where does the implication $[can(f) \leq_m^{pp} can(g) \Rightarrow f \leq_s g]$ hold, and where does it fail?

Q2: Where can we find proof systems of different strengths, but equivalent canonical pairs?

Q3: What do (non-)equivalent canonical pairs tell about the corresponding proof systems?

Q4: Is every NP-pair (A, B) , where A is NP-complete, strongly many-one equivalent to the canonical pair of some proof system?

In short, we show that both parts of Q1 and Q2 can be answered with ‘everywhere’, which generalize previous results by Pudlák and Beyersdorff. Regarding Q3, inequivalent canonical pairs tell that the proof systems are not “very similar”, while equivalent, P-inseparable canonical pairs tell that they are not “very different”. We can relate Q4 to the open problem in structural complexity that asks whether unions of disjoint NP-complete sets are NP-complete. This demonstrates a new connection between proof systems, disjoint NP-pairs, and unions of disjoint NP-complete sets.

1 Introduction

One reason it is important to study canonical pairs of propositional proof systems (proof systems) is their role in connecting proof systems with disjoint NP-pairs (NP-pairs) [7]. Razborov [13] first defined the canonical pair, $can(f) = (SAT^*, REF(f))$, for every proof system f . He showed that if there exists an optimal proof system f , then its canonical pair is a complete pair for DisjNP. In a recent paper [8], we show that every NP-pair is polynomial-time many-one equivalent to the canonical pair of some proof system. So the degree structure of the class of NP-pairs and of all canonical pairs is identical.

* Research partially supported by NSF grant CCR-0307077.

Beyersdorff [1] studies proof systems and their canonical pairs from a proof theoretic point of view. He defines the subclasses DNPP(P) of NP-pairs that are representable in some proof system P and shows that the canonical pairs of P are complete for DNPP(P). This interesting result tells us that for certain meaningful subclasses of NP-pairs, complete pairs do exist. Beyersdorff also compares the simulation order of proof systems with the hardness of their canonical pairs, which we will address in this paper too.

Encouraged by these exciting results on proof systems and their canonical pairs, we continue this line of research and concentrate on the following correspondence between proof systems and NP-pairs. For proof systems f and g ,

$$f \leq_s g \Rightarrow \text{can}(f) \leq_m^{pp} \text{can}(g). \quad (1)$$

Pudlák [12] and Beyersdorff [1] give counter examples for the converse. This raises the following questions which we investigate in this paper.

Q1: Where does the following implication hold, and where does it fail?

$$\text{can}(f) \leq_m^{pp} \text{can}(g) \Rightarrow f \leq_s g \quad (2)$$

Q2: Where can we find proof systems of different strengths whose canonical pairs are equivalent?

Q3: What do (non-)equivalent canonical pairs tell about the corresponding proof systems?

Moreover, it is known that every NP-pair is many-one equivalent to the canonical pair of some proof system [8]. Here we investigate the same question for strongly many-one reductions. It is easy to see that this question must be restricted to pairs whose first component is NP-complete.

Q4: Is every NP-pair (A, B) , where A is NP-complete, strongly many-one equivalent to the canonical pair of some proof system?

Theorem 3 addresses the first part of Q1: The theorem asserts that, for any two disjoint NP-pairs (A, B) and (C, D) , there are proof systems f and g such that $\text{can}(f) \equiv_m^{pp}(A, B)$, $\text{can}(g) \equiv_m^{pp}(C, D)$, and implication (2) holds nontrivially.

Corollary 2 addresses the second part of Q1: The following assertion is equivalent to the reasonable assumption that optimal proof systems do not exist. For every proof system f there is a proof system g such that f and g is a counter example to implication (2). More strongly, there is an infinite chain of proof systems g_0, g_1, \dots , such that $f <_s g_0 <_s g_1 <_s \dots$, but the canonical pairs of all of these proof systems are many-one equivalent. In this way, we address Q2.

In section 4 we answer Q3 in different ways. Equivalent canonical pairs do not tell much about the mere simulation order of two proof systems (Theorem 5). However, inequivalent canonical pairs tell us that the corresponding proof systems do not simulate each other except on a P-subset of TAUT (Proposition 3). Hence these proof systems are not “very similar”. In contrast, equivalent,

P-inseparable canonical pairs tell us that none of the corresponding proof systems is almost everywhere super-polynomially stronger than the other one (Theorem 6). So these proof systems are not “very different”.

In section 5 we can relate Q4 to the open problem in structural complexity 3.6 that asks whether unions of disjoint NP-complete sets are NP-complete. We show under the hypothesis $NP \neq coNP$ that if Q4 has an affirmative answer, then unions of disjoint NP-complete sets are NP-complete. This demonstrates a new connection between proof systems, NP-pairs, and problems in structural complexity. Finally, in section 6 we obtain connections between proof systems and the Turing-degrees of their canonical pairs.

2 Preliminaries

A disjoint NP-pair is a pair (A, B) of nonempty sets A and B such that $A, B \in NP$ and $A \cap B = \emptyset$. Let $DisjNP$ denote the class of all disjoint NP-pairs.

Given a disjoint NP-pair (A, B) , a *separator* is a set S such that $A \subseteq S$ and $B \subseteq \bar{S}$; we say that S *separates* (A, B) . Let $Sep(A, B)$ denote the set of all separators of (A, B) . For disjoint NP-pairs (A, B) , the fundamental question is whether $Sep(A, B)$ contains a set belonging to P . In that case the pair is *P-separable*; otherwise, the pair is *P-inseparable*. There is evidence 4.5 that P-inseparable disjoint NP-pairs exist, and this will be our main hypothesis in the paper. The following proposition summarizes known results.

Proposition 1

1. $P \neq NP \cap coNP$ implies that $DisjNP$ contains a P-inseparable pair.
2. $P \neq UP$ implies that $DisjNP$ contains a P-inseparable pair. 4.
3. If $DisjNP$ contains P-inseparable pairs, then it contains a P-inseparable pair whose components are NP-complete. 4.

While it is probably the case that $DisjNP$ contains P-inseparable pairs, there is an oracle relative to which $P \neq NP$ and P-inseparable pairs in $DisjNP$ do not exist 9. So $P \neq NP$ probably is not a sufficiently strong hypothesis to show the existence of P-inseparable pairs in $DisjNP$. On the other hand, if there exist secure public-key cryptosystems (for example, if RSA cannot be cracked in polynomial-time), then there exist P-inseparable disjoint NP-pairs 4.

All reducibilities in the paper are polynomial time computable. We review the notions of reducibilities between disjoint pairs. The original notions are nonuniform 4, here we state the equivalent uniform versions 4.5.

Definition 1. Let (A, B) and (C, D) be disjoint pairs.

1. (A, B) is many-one reducible in polynomial-time to (C, D) , $(A, B) \leq_m^{pp}(C, D)$, if there exists a polynomial-time computable function f such that $f(A) \subseteq C$ and $f(B) \subseteq D$.
2. (A, B) is Turing reducible in polynomial-time to (C, D) , $(A, B) \leq_T^{pp}(C, D)$, if there exists a polynomial-time oracle Turing machine M such that for every separator S of (C, D) , $L(M, S)$ is a separator of (A, B) .

Köbler, Meßner, and Torán [10] define the following stronger version of many-one reductions between disjoint NP-pairs:

Definition 2. Let (A, B) and (C, D) be disjoint pairs. (A, B) is strongly many-one reducible in polynomial-time to (C, D) , $(A, B) \leq_{\text{sm}}^{\text{pp}}(C, D)$, if there exists a polynomial-time computable function f such that $f(A) \subseteq C$, $f(B) \subseteq D$, and $f(A \cup B) \subseteq C \cup D$.

Definition 3. A disjoint pair (A, B) is \leq_m^{pp} -hard for NP if for every separator L of (A, B) , $\text{SAT} \leq_m^p L$.

Definition 4. For any disjoint pair (A, B) , the polynomial-time Turing-degree (Turing-degree for short) of (A, B) is defined as

$$\mathbf{d}(A, B) = \{(C, D) \mid (C, D) \text{ is a disjoint pair and } (A, B) \equiv_T^{\text{pp}}(C, D)\}.$$

In an earlier paper [8] we investigated the restriction of Turing-degrees of disjoint pairs on DisjNP and showed that every countable distributive lattice can be embedded into the interval between any two comparable but inequivalent restricted Turing-degrees of disjoint NP-pairs. It follows trivially that every countable distributive lattice can be embedded into the interval between any two comparable but inequivalent Turing-degrees of disjoint pairs if both degrees contain some disjoint NP-pair.

Let SAT denote the set of satisfiable formulas and let UNSAT $\stackrel{\text{df}}{=} \overline{\text{SAT}}$. Moreover, let TAUT denote the set of tautologies. Cook and Reckhow [2] defined a propositional proof system (proof system for short) to be a function $f : \Sigma^* \rightarrow \text{TAUT}$ such that f is onto and f is polynomial-time computable. For every tautology α , if $f(w) = \alpha$, then we say w is an f -proof of α .

The canonical NP-pair (canonical pair for short) of f [13][12] is the disjoint NP-pair $(\text{SAT}^*, \text{REF}(f))$, denoted by $\text{can}(f)$, where

$$\text{SAT}^* = \{(x, 0^n) \mid x \in \text{SAT}\} \quad \text{and}$$

$$\text{REF}(f) = \{(x, 0^n) \mid \neg x \in \text{TAUT and } \exists y[|y| \leq n \text{ and } f(y) = \neg x]\}.$$

Conversely, for every disjoint NP-pair (A, B) , we can define a proof system $f_{A,B}$ as follows. Let $\langle \cdot, \cdot \rangle$ be a polynomial-time computable, polynomial-time invertible pairing function such that $|\langle v, w \rangle| = 2|vw|$. Choose a g that is polynomial-time computable and polynomial-time invertible such that $A \leq_m^p \text{SAT}$ via g . Let N be an NP-machine that accepts B in time p .

$$f_{A,B}(z) \stackrel{\text{df}}{=} \begin{cases} \neg g(x) & : \text{ if } z = \langle x, w \rangle, |w| = p(|x|), N(x) \text{ accepts along path } w \\ x & : \text{ if } z = \langle x, w \rangle, |w| \neq p(|x|), |z| \geq 2^{|x|}, x \in \text{TAUT} \\ \text{true} & : \text{ otherwise} \end{cases}$$

Clearly, $f_{A,B}$ is a propositional proof system for every disjoint NP-pair (A, B) .

Theorem 1 ([8]). For every $(A, B) \in \text{DisjNP}$, $(A, B) \equiv_m^{\text{pp}} \text{can}(f_{A,B})$.

Let f and f' be two propositional proof systems. We say that f simulates f' ($f' \leq_s f$) if there is a polynomial p and a function $h : \Sigma^* \rightarrow \Sigma^*$ such that for every $w \in \Sigma^*$, $f(h(w)) = f'(w)$ and $|h(w)| \leq p(|w|)$. Furthermore, if the

function h can be computed in polynomial-time, f p -simulates f' ($f' \leq_p f$). A proof system is (p) -optimal if it (p) -simulates every other proof system.

In Section 4, we will need the following generalization of the concept “simulation”. We say that f simulates f' on a subset S of TAUT, if there is a polynomial p and a function $h : \Sigma^* \rightarrow \Sigma^*$ such that for every $w \in \Sigma^*$, $f'(w) \in S$ implies that $f(h(w)) = f'(w)$ and $|h(w)| \leq p(|w|)$. Moreover, f simulates f' except on a subset S of TAUT, if f simulates f' on TAUT $- S$. Obviously, a proof system f simulates a proof system f' if and only if f simulates f' on TAUT.

We use $f <_s g$ to denote that $f \leq_s g$ and $g \not\leq_s f$. We use $(A, B) <_m^{pp}(C, D)$ to denote that $(A, B) \leq_m^{pp}(C, D)$ and $(C, D) \not\leq_m^{pp}(A, B)$.

3 Proof Systems and Many-One Degrees of Canonical Pairs

We recall the fundamental relation between proof systems and canonical pairs.

Proposition 2 ([12,8]). *Let f and g be proof systems.*

$$f \leq_s g \Rightarrow \text{can}(f) \leq_m^{pp} \text{can}(g)$$

In this section, we investigate the converse of the above proposition. We show results that address both parts of Q1 and Q2.

We start our investigations with the observation that refuting an implication that is slightly weaker than (2) is equivalent to proving the existence of P-inseparable disjoint NP-pairs. This is done by a purely complexity theoretic proof that does not rely on specific properties of concrete proof systems.

Theorem 2. *The following statements are equivalent.*

1. P-inseparable disjoint NP-pairs exist.
2. There exist proof systems f and g such that $\text{can}(f) <_m^{pp} \text{can}(g) \not\Rightarrow f \leq_s g$.

Proof. If P-inseparable disjoint NP-pairs do not exist, then all canonical pairs of proof systems are P-separable and hence are equivalent. This shows $2 \Rightarrow 1$.

For the other direction, assume that P-inseparable disjoint NP-pairs exist and define the following set of propositional formulas.

$$\text{EASY} \stackrel{\text{df}}{=} \{x \mid x \text{ is a propositional formula such that } x = (b \vee \bar{b} \vee y) \text{ for a suitable variable } b \text{ and a suitable formula } y\}$$

EASY is a subset of TAUT. Also, $\text{EASY} \in \text{P}$. Let $\text{true} \stackrel{\text{df}}{=} (b \vee \bar{b} \vee b)$ and define a proof system as follows.

$$f(z) \stackrel{\text{df}}{=} \begin{cases} x & : \text{ if } z = \langle x, \varepsilon \rangle \text{ and } x \in \text{EASY} \\ x & : \text{ if } z = \langle x, y \rangle \text{ and } |y| > 2^{|x|} \text{ and } x \in \text{TAUT} \\ \text{true} & : \text{ otherwise.} \end{cases}$$

Note that f is a proof system. Observe that the elements in EASY are the only tautologies that have polynomial-size f -proofs. All other tautologies do not have

polynomial-size f -proofs. This makes $\text{can}(f)$ P-separable which is witnessed by the following separator:

$$S = \{(x, 0^n) \mid [n \leq 2^{|x|} \text{ and } \neg x \notin \text{EASY}] \text{ or } [n > 2^{|x|} \text{ and } x \in \text{SAT}]\}$$

By assumption there exists a P-inseparable disjoint NP-pair (A, B) . Hence, by Theorem 1 there exists a proof system g' such that $\text{can}(g')$ and (A, B) are many-one equivalent. Now define another proof system.

$$g(z) \stackrel{\text{df}}{=} \begin{cases} g'(w) & : \text{ if } z = 0w \text{ and } g'(w) \notin \text{EASY} \\ \text{true} & : \text{ if } z = 0w \text{ and } g'(w) \in \text{EASY} \\ x & : \text{ if } z = 1w, w = \langle x, y \rangle, |y| = 2^{|x|}, \text{ and } x \in \text{EASY} \\ \text{true} & : \text{ otherwise.} \end{cases}$$

Note that g is a proof system. Observe that formulas in $\text{EASY} - \{\text{true}\}$ do not have polynomial-size g -proofs. It follows that g does not simulate f , since f provides polynomial-size proofs for elements in EASY .

Now we verify that $\text{can}(g') \leq_m^{pp} \text{can}(g)$ via the reduction that maps $(x, 0^n)$ to $(x, 0^{n+1})$. If $(x, 0^n) \in \text{SAT}^*$, then $(x, 0^{n+1}) \in \text{SAT}^*$ and we are done. Let $(x, 0^n) \in \text{REF}(g')$. So there exists some w such that $|w| \leq n$ and $g'(w) = (\neg x)$. Note that $(\neg x) \notin \text{EASY}$, since formulas in EASY do not start with a negation. From the definition of g it follows that $g(0w) = g'(w) = (\neg x)$. So $(x, 0^{n+1}) \in \text{REF}(g)$.

So $\text{can}(g') \leq_m^{pp} \text{can}(g)$ and therefore, $(A, B) \leq_m^{pp} \text{can}(g)$. Hence $\text{can}(g)$ is P-inseparable. This shows $\text{can}(f) <_m^{pp} \text{can}(g)$. □

The examples given by Pudlák [12] and Beyersdorff [1] show that the simulation order of proof systems is not necessarily reflected by the reducibility of their canonical pairs. However, as the next theorem shows, the canonical pairs of proof systems that satisfy implication (2) in a non-trivial way, vary over all degrees of disjoint NP-pairs. More precisely, for each pair of many-one degrees of disjoint NP-pairs, there do exist proof systems whose canonical pairs lie in the respective degrees such that their simulation order is consistent with the reducibility of the canonical pairs. This answers the first part of Q1 in the sense that implication (2) can be satisfied non-trivially for arbitrary canonical pairs.

Theorem 3. *Let $(A, B), (C, D) \in \text{DisjNP}$ such that $(A, B) \leq_m^{pp} (C, D)$. Then there exist proof systems f_1 and f_2 such that $f_1 \leq_p f_2$, $\text{can}(f_1) \equiv_m^{pp} (A, B)$, and $\text{can}(f_2) \equiv_m^{pp} (C, D)$.*

Proof. Let $\langle \cdot, \cdot \rangle$ be a polynomial-time computable, polynomial-time invertible pairing function such that $|\langle v, w \rangle| = 2|vw|$. Choose g_1 that is polynomial-time computable and polynomial-time invertible such that $A \leq_m^p \text{SAT}$ via g_1 . Let N_1 be an NP-machine that accepts B in time p_1 . Define the following function f_1 .

$$f_1(z) \stackrel{\text{df}}{=} \begin{cases} \neg g_1(x) & : \text{ if } z = \langle x, w \rangle, |w| = p_1(|x|), N_1(x) \text{ accepts along path } w \\ x & : \text{ if } z = \langle x, w \rangle, |w| \neq p_1(|x|), |z| \geq 2^{|x|}, x \in \text{TAUT} \\ \text{true} & : \text{ otherwise} \end{cases}$$

The proof of Theorem 1 shows that f_1 is a proof system and $can(f_1) \equiv_m^{pp} (A, B)$. Now choose g_2 that is polynomial-time computable and polynomial-time invertible such that $C \leq_m^p SAT$ via g_2 . Let N_2 be an NP-machine that accepts D in time p_2 . Without loss of generality, we assume for every $n \geq 0$, $p_1(n) \neq p_2(n)$ and $range(g_1) \cap range(g_2) = \emptyset$. Define the following function f_2 .

$$f_2(z) \stackrel{df}{=} \begin{cases} \neg g_1(x) & : \text{ if } z = \langle x, w \rangle, |w| = p_1(|x|), N_1(x) \text{ accepts along path } w \\ \neg g_2(x) & : \text{ if } z = \langle x, w \rangle, |w| = p_2(|x|), N_2(x) \text{ accepts along path } w \\ x & : \text{ if } z = \langle x, w \rangle, |w| \neq p_i(|x|) \text{ for } i=1, 2, |z| \geq 2^{|x|}, x \in TAUT \\ true & : \text{ otherwise} \end{cases}$$

Clearly f_2 is also a proof system, since for every tautology y , $f_2(\langle y, 0^{2^{|y|}} \rangle) = y$. Also, we notice that each f_1 -proof z is also an f_2 -proof for the same tautology except for $z \in \{ \langle x, w \rangle \mid |w| = p_2(|x|) \wedge |\langle x, w \rangle| \geq 2^{|x|} \wedge x \in TAUT \}$, which is a finite set. So, $f_1 \leq_p f_2$.

It remains to show $can(f_2) \equiv_m^{pp} (C, D)$. We only show $can(f_2) \leq_m^{pp} (C, D)$. The proof for $(C, D) \leq_m^{pp} can(f_2)$ is the same as that for $(A, B) \leq_m^{pp} can(f_1)$, for which we refer the reader to [8].

Let g many-one reduce (A, B) to (C, D) . Choose elements $c \in C$ and $d \in D$. Define a reduction function h as follows.

```

1  input (y, 0^n)
2  if n ≥ 2^{|y|+1} then
3      if y ∈ SAT then output c else output d
4  endif
5  if g1-1(y) exists then output g(g1-1(y))
6  if g2-1(y) exists then output g2-1(y)
7  output c

```

Line 3 needs quadratic time in n . So h is polynomial-time computable.

Assume $(y, 0^n) \in SAT^*$. Then $y \in SAT$. If we reach line 3, then we output $c \in C$. Otherwise we reach line 5. If $g_1^{-1}(y)$ exists (hence, $g_2^{-1}(y)$ does not exist, since the ranges of g_1 and g_2 are disjoint), then $g_1^{-1}(y) \in A$ and so, $g(g_1^{-1}(y)) \in C$. Otherwise we reach line 6. If $g_2^{-1}(y)$ exists, then $g_2^{-1}(y) \in C$ as $y \in SAT$. So in all cases (output in line 5, 6 or 7), we output an element in C .

Assume $(y, 0^n) \in REF(f_2)$ (in particular $y \in UNSAT$). So there exists z such that $|z| \leq n$ and $f(z) = \neg y$. If we reach line 3, then we output $d \in D$. Otherwise we reach line 5. So far we have $\neg y \neq true$ and $|z| \leq n < 2^{|y|+1}$. Therefore, $f(z) = \neg y$ must be due to line 1 or line 2 in the definition of f_2 . It follows that either $g_1^{-1}(y)$ exists or $g_2^{-1}(y)$ exists (but not both). If $g_1^{-1}(y)$ exists, then $g_1^{-1}(y) \in B$ (by line 1 of f_2 's definition) and we output $g(g_1^{-1}(y))$, which belongs to D . Otherwise, $g_2^{-1}(y)$ exists and we output $g_2^{-1}(y)$, which belongs to D as well (by line 2 of f_2 's definition). This shows $can(f_2) \leq_m^{pp} (C, D)$ via h . \square

The proof system g constructed in Theorem 2 might seem “pathological”, since tautologies from an easy subset of TAUT have proofs of super-polynomial length. One might wonder whether Theorem 2 can be proved without such pathology. The corresponding proof systems are formalized as follows.

Definition 5. A proof system f is well-behaved if for every polynomial-time decidable $S \subseteq \text{TAUT}$ there exists a polynomial p such that for all $x \in S$,

$$\min\{|w| \mid f(w) = x\} \leq p(|x|).$$

However, well-behaved proof systems probably do not exist. Meßner [11] shows that the existence of well-behaved proof systems implies the existence of optimal proof systems which we believe not to exist. So it is probably the case that no proof system is well-behaved and therefore, every proof system has long proofs on some polynomial-time decidable subset of TAUT. This shows that the proof system constructed in Theorem 2 is not uncommon. Even more, we can apply the arguments used in Theorem 2 to every non-well-behaved proof system.

Theorem 4. Let f be a proof system that is not well-behaved. For every $(A, B) \in \text{DisjNP}$, there exists a proof system g such that $\text{can}(g) \equiv_m^{pp}(A, B)$ and $g \not\leq_s f$.

With help of Theorem 4 we can now give an answer to Q2: All non-well-behaved proof systems provide examples for proof systems that have equivalent canonical pairs, but that differ with respect to their strengths. Moreover, we can answer the second part of Q1 in the sense that all non-well-behaved proof systems provide counter examples for implication (2).

Corollary 1. For every proof system f that is not well-behaved, there exists a proof system g such that $\text{can}(f) \equiv_m^{pp} \text{can}(g)$ and $f <_s g$. In particular,

$$\text{can}(g) \leq_m^{pp} \text{can}(f) \not\Rightarrow g \leq_s f.$$

If we assume that optimal proof systems do not exist, then Corollary 1 provides even stronger answers: With regard to Q1, all proof systems provide counter examples for the implication (2). With regard to Q2, all proof systems provide examples that have equivalent canonical pairs, but that differ with respect to their strengths. Even more, each proof system is the origin of an infinite, strictly ascending chain of proof systems whose canonical pairs are equivalent.

Corollary 2. The following statements are equivalent.

1. Optimal proof systems do not exist.
2. For every proof system f there exists a proof system g such that $\text{can}(f) \equiv_m^{pp} \text{can}(g)$ and $f <_s g$.
3. For every proof system f there exists an infinite chain of proof systems g_0, g_1, \dots such that $f <_s g_0 <_s g_1 <_s \dots$ and $\text{can}(f) \equiv_m^{pp} \text{can}(g_0) \equiv_m^{pp} \text{can}(g_1) \dots$.
4. For every proof system f there exists a proof system g such that

$$\text{can}(g) \leq_m^{pp} \text{can}(f) \not\Rightarrow g \leq_s f.$$

4 Proof Systems with Equivalent Canonical Pairs

We have seen in the last section that the degree structure of canonical pairs does not necessarily reflect the simulation order of the corresponding proof systems. In this section we study the related question Q3.

We first show that equivalent canonical pairs do not tell much about the simulation order of two proof systems.

Theorem 5. *For every disjoint NP-pair (A, B) , there exist proof systems f, g , and h such that*

- $can(f) \equiv_m^{pp} can(g) \equiv_m^{pp} can(h) \equiv_m^{pp} (A, B)$,
- $f <_s g$ and $f <_s h$,
- $g \not<_s h$ and $h \not<_s g$.

However, from another point of view, the proof systems defined in the proof of Theorem 5 are actually quite “similar” to each other. They differ only super-polynomially on an easy subset of TAUT. More precisely, the construction of proof systems with equivalent canonical pairs but arbitrary simulation order hinges on the following fact.

Proposition 3. *If proof systems f and g simulate each other except on a P-subset of TAUT, then $can(f) \equiv_m^{pp} can(g)$.*

So here the question is whether we can construct proof systems f and g with equivalent canonical pairs such that the proof systems are “very different”. For example, do there exist proof systems f and g such that $can(f) \equiv_m^{pp} can(g)$ and f is almost everywhere super-polynomially stronger than g ? The following theorem shows that such an extreme difference is only possible for proof systems whose canonical pairs are P-separable.

Theorem 6. *Let f and g be proof systems such that $can(g) \leq_m^{pp} can(f)$. If for almost all tautologies x and for every polynomial p , the length of the shortest f -proof of x is not bounded by p in the length of the shortest g -proof of x , then $can(f)$ and $can(g)$ are P-separable.*

Let us summarize what we have seen: Proposition 3 says that if two proof systems are “very similar”, then they have equivalent canonical pairs. Theorem 6 tells us that if two proof systems are “very different” from each other, then either they have P-separable canonical pairs or their canonical pairs are inequivalent.

We continue to follow the question to what extent proof systems can differ, while still having equivalent canonical pairs. Under the hypothesis that P-inseparable disjoint NP-pairs exist, we show that Proposition 3 does not hold when the P-subset is replaced with an NP-subset (Corollary 3). So altering f -proofs on a P-subset of TAUT does not change the many-one degree of $can(f)$, but altering f -proofs on an NP-subset of TAUT can do so.

Theorem 7. *Let f be a proof system such that $can(f)$ is not \leq_m^{pp} -complete for DisjNP. Then there exists a proof system f' such that $can(f) <_m^{pp} can(f')$ and f and f' simulate each other except on an NP-subset of TAUT.*

Corollary 3. *The following statements are equivalent.*

1. P-inseparable NP-pairs exist.
2. There exist proof systems f and g whose canonical pairs are not many-one equivalent, but that simulate each other except on an NP-subset of TAUT.

Corollary 4. *If $P \neq NP \cap coNP$, then there exist proof systems f and g whose canonical pairs are not many-one equivalent, but that simulate each other except on an NP-subset of TAUT.*

Under the hypothesis that P-inseparable disjoint NP-pairs exist, we can show that proof systems whose difference cannot be “covered” by any P-subset of TAUT may still have equivalent canonical pairs. Hence, the converse of Proposition 3 does not hold, unless P-inseparable disjoint NP-pairs do not exist.

Theorem 8. *Let (A, B) be a P-inseparable NP-pair. Then there exist proof systems f and f' such that $can(f) \equiv_m^{pp} can(f') \equiv_m^{pp} (A, B)$ and for every P-subset S of TAUT it holds that f and f' do not simulate each other on $TAUT - S$.*

5 Strongly Many-One Degrees of Canonical Pairs

Every disjoint NP-pair is many-one equivalent to the canonical pair of some proof system 8. We ask the same question for strongly many-one reductions. Note that if a disjoint NP-pair (A, B) is strongly many-one equivalent to the canonical pair of some proof system, then A must be NP-complete. So we arrive at question Q4 which is closely related to the following open problem 3.

Q5: Is the union of two disjoint NP-complete sets NP-complete?

For this, we first translate Q4 into the question whether certain NP-pairs are many-one hard for NP (Corollary 5). From this we show under the hypothesis $NP \neq coNP$ that if Q4 has an affirmative answer, then Q5 has an affirmative answer. It suffices to demand that Q4 has answer ‘yes’ only for $A = SAT$.

Theorem 9. *Let (A, B) be a disjoint NP-pair. If $(A, \overline{A \cup B})$ is \leq_m^{pp} -hard for NP, then there exists a proof system f such that $(SAT^*, REF(f)) \equiv_{sm}^{pp} (A, B)$.*

Proposition 4. *Let (A, B) be a disjoint NP-pair such that $A \cup B \neq \Sigma^*$. If there exists a proof system f such that $(SAT^*, REF(f)) \equiv_{sm}^{pp} (A, B)$, then $(A, \overline{A \cup B})$ is \leq_m^{pp} -hard for NP.*

Corollary 5. *The following are equivalent for a disjoint NP-pair (A, B) where $A \cup B \neq \Sigma^*$.*

1. $(A, \overline{A \cup B})$ is \leq_m^{pp} -hard for NP.
2. There exists a proof system f such that $(SAT^*, REF(f)) \equiv_{sm}^{pp} (A, B)$.

Corollary 6. *Assume $NP \neq coNP$. If for all disjoint NP-pairs (SAT, B) there exists a proof system f such that $(SAT^*, REF(f)) \equiv_{sm}^{pp} (SAT, B)$, then unions of disjoint NP-complete sets are NP-complete.*

6 Proof Systems and Turing-Degrees of Canonical Pairs

We consider the connection between proof systems and the more general Turing-degrees of their canonical pairs.

Proposition 5. *Let f and g be proof systems such that $\text{can}(f) \leq_T^{pp} \text{can}(g)$. Then there exists a proof system g' such that $\text{can}(g') \equiv_T^{pp} \text{can}(g)$ and $f \leq_p g'$.*

Corollary 7. *Let $\mathbf{d}_1 < \mathbf{d}_2$ be two Turing-degrees of disjoint NP-pairs. Then for every proof system f such that $\text{can}(f) \in \mathbf{d}_1$, there exists a proof system g such that $\text{can}(g) \in \mathbf{d}_2$ and $f <_s g$.*

Proposition 6. *For all $(A, B), (C, D) \in \text{DisjNP}$ such that $(A, B) <_T^{pp} (C, D)$, there exist proof systems f and g such that $\text{can}(f) \equiv_m^{pp} (A, B)$, $\text{can}(g) \equiv_m^{pp} (C, D)$, $f \not\leq_s g$, and $g \not\leq_s f$.*


References

1. Beyersdorff, O.: Disjoint NP-pairs from propositional proof systems. In: Cai, J.-Y., Cooper, S.B., Li, A. (eds.) TAMC 2006. LNCS, vol. 3959, pp. 236–247. Springer, Heidelberg (2006)
2. Cook, S.A., Reckhow, R.A.: The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic* 44(1), 36–50 (1979)
3. Glaßer, C., Pavan, A., Selman, A.L., Sengupta, S.: Properties of NP-complete sets. *SIAM Journal on Computing* 36(2), 516–542 (2006)
4. Grollmann, J., Selman, A.L.: Complexity measures for public-key cryptosystems. *SIAM Journal on Computing* 17(2), 309–335 (1988)
5. Glaßer, C., Selman, A.L., Sengupta, S., Zhang, L.: Disjoint NP-pairs. *SIAM Journal on Computing* 33(6), 1369–1416 (2004)
6. Glaßer, C., Selman, A.L., Travers, S., Wagner, K.W.: The complexity of unions of disjoint sets. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, Springer, Heidelberg (2007)
7. Glaßer, C., Selman, A.L., Zhang, L.: Survey of disjoint NP-pairs and relations to propositional proof systems. In: Goldreich, O., Rosenberg, A.L., Selman, A.L. (eds.) *Theoretical Computer Science*. LNCS, vol. 3895, Springer, Heidelberg (2006)
8. Glaßer, C., Selman, A.L., Zhang, L.: Canonical disjoint NP-pairs of propositional proof systems. *Theoretical Computer Science* 370, 60–73 (2007)
9. Homer, S., Selman, A.L.: Oracles for structural properties: The isomorphism problem and public-key cryptography. *Journal of Computer and System Sciences* 44(2), 287–301 (1992)
10. Köbler, J., Messner, J., Torán, J.: Optimal proof systems imply complete sets for promise classes. *Information and Computation* 184(1), 71–92 (2003)
11. Meßner, J.: On the Simulation order of proof systems. PhD thesis, Universität Ulm, Abteilung Theoretische Informatik (December 2000)
12. Pudlák, P.: On reducibility and symmetry of disjoint NP-pairs. *Theoretical Computer Science* 295, 323–339 (2003)
13. Razborov, A.A.: On provably disjoint NP-pairs. Technical Report TR94-006, Electronic Computational Complexity Colloquium (1994)

On the Representations of NC and Log-Space Real Numbers^{*}

Fuxiang Yu

Computer Science Department
Stony Brook University
Stony Brook, New York, U.S.A.
fuxiang@cs.sunysb.edu

Abstract. We study the representations of NC and Log-space real numbers in this paper. We show that the classes of the NC and Log-space real numbers under the general left cut representation are among the most expressive representations.  On the other hand, although the general left cut representation and the Cauchy function representation have the same expressive power in P, the expressive power of the Cauchy function representation is weaker than that of the general left cut representation in NC if $\mathbf{P}_1 \neq \mathbf{NC}_1$. In addition, although the expressive power of the standard left cut representation is weaker than that of the Cauchy function representation in P, the expressive powers of these two representations are incomparable in NC if $\mathbf{P}_1 \neq \mathbf{NC}_1$. Similar results hold in Log-space.

Keywords: Complexity, representations of real numbers, Cauchy function, left cut, P, NC, Log-space, expressive power.

1 Introduction

The computability and complexity of real numbers have been widely studied, since real numbers are the main objects of continuous computation. The first issue is how to represent a real number, and the second issue is how to compute the representations of a real number effectively or efficiently.

For a given real number $x \in [0, 1]$, there are many representations. To name a few, x can be represented by (1) a Cauchy sequence $\{x_n\}$ of rational numbers such that for all n , $|x_n - x| < 2^{-n}$; (2) the standard left cut, which is the set of rational numbers that are no more than x ; and (3) the binary expansion $x = (0.b_1b_2\cdots)_2$. These representations and many others are, mathematically, equivalent to each other.

^{*} This material is based upon work supported by National Science Foundation under grant No. 0430124.

¹ We say a class A of languages is more expressive than another class B of languages, if $B \subsetneq A$. Now we also say A has more expressive power. For example, **EXP** has more expressive power than **P**.

However, when computability and complexity are considered, it becomes more complicated and interesting. For example, a recent work of Chen et al. [3] showed that the primitive recursive versions of these representations can lead to different notions of primitive recursive real numbers.

When studying the complexity of real numbers, *dyadic numbers* $0.b_1b_2 \dots b_n$ (i.e., finite binary fractions) are the basic computational objects. Now a *Cauchy function representation* of a real number $x \in [0, 1]$ is a function ϕ from natural numbers to dyadic numbers such that for all n , $\phi(n)$ has n bits and $|\phi(n) - x| < 2^{-n}$. A number x is *Cauchy polynomial-time computable* if a Cauchy function representation ϕ of x is polynomial-time computable. Similarly, we can define the *standard left cut representation* of x and the *binary expansion representation* of x using dyadic numbers as the basic objects. Ko [7] has shown that the Cauchy function representation is more expressive, unconditionally, than the standard left cut representation and the binary expansion as long as polynomial-time computability is concerned. On the other hand, if for every Cauchy function representation ϕ of a real number x , we define a set $LC_\phi = \{d \leq \phi(n) : n \in \mathbb{N}, d \text{ is a dyadic number of } n \text{ bits.}\}$, called the *general left cut* of x associated with ϕ , then the general left cut representation is as expressive as the Cauchy function representation in **P**. There are also some other representations, for example, Ko [6] presented some interesting results on the continuous fraction representation.

In this paper, we study the parallel-time complexity issues of real numbers. More precisely, we investigate which representation is the most expressive in parallel-time complexity classes such as **NC** and **L** (short for *Log-Space*). We consider four representations: standard left cut, general left cut, binary expansion, and the Cauchy function representation. As expected, the general left cut representation is among the most expressive representations; however, we still obtain some interesting results, which are summarized below.

- (1) The general left cut representation is the most expressive representation in **NC** and **L**.
- (2) There exists a real number x that has a **NC** (or **L**) computable Cauchy function representation but the standard left cut of x is not **NC** (or respectively, **L**) computable.
- (3) If $\mathbf{P}_1 \neq \mathbf{NC}_1$ (or $\mathbf{P}_1 \neq \mathbf{L}_1$), then there exists a real number x whose standard left cut is **NC** (or respectively, **L**) computable, but no Cauchy function representation of x is **NC** (or respectively, **L**) computable.

In other words, If $\mathbf{P}_1 \neq \mathbf{NC}_1$ (or $\mathbf{P}_1 \neq \mathbf{L}_1$), then the expressive powers of the Cauchy function representation and the standard left cut representation are incomparable under **NC** (or respectively, **L**). As it seems that the Cauchy function representation has been used as one main representation in parallel computation of real numbers (see, e.g., Ko [7], Hoover [5], and Yu [8]), this result should not be ignored.

2 Notations

2.1 Representations

This paper involves notions used in both discrete computation and continuous computation. The basic computational objects in discrete computation are integers and strings in $\{0, 1\}^*$. The length of a string w is denoted $\ell(w)$.

The basic computational objects in continuous computation are dyadic rationals $\mathbb{D} = \{m/2^n : m \in \mathbb{Z}, n \in \mathbb{N}\}$. Each dyadic rational d has infinitely many binary representations with arbitrarily many trailing zeros. For each such representation s , we write $\ell(s)$ to denote its length. If the specific representation of a dyadic rational d is understood (often the shortest binary representation), then we write $\ell(d)$ to denote the length of this representation. We let \mathbb{D}_n denote the class of dyadic rationals with at most n bits in the fractional part of its binary representation.

We use \mathbb{R} to denote the set of real numbers. We say a function $\phi : \mathbb{N} \rightarrow \mathbb{D}$ is a *Cauchy function representation* of a real number x , if (i) for all $n \geq 0$, $\phi(n) \in \mathbb{D}_n$, and (ii) for all $n \geq 0$, $|\phi(n) - x| \leq 2^{-n}$. For any $x \in \mathbb{R}$, there is a unique function $b_x : \mathbb{N} \rightarrow \mathbb{D}$ that binary converges to x and satisfies the condition $x - 2^{-n} < b_x(n) \leq x$ for all $n \geq 0$. We call this function b_x the *standard Cauchy function* for x . The binary expansion of x is represented by b_x . For any Cauchy function representation ϕ of x , there exists a general left cut representation $LC_\phi = \{d \in \mathbb{D}_n : n \in \mathbb{N}, d \leq \phi(n)\}$ associated with ϕ , and LC_{b_x} , also denoted LC_x , is called the standard left cut.

2.2 Complexity Classes

In this paper we consider mainly the circuit complexity class **NC** as well as complexity classes defined based on Turing machines listed as follows (see, e.g., Du and Ko [4]).

P : the class of sets accepted by deterministic polynomial-time (Turing) machines.

L : the class of sets accepted by deterministic Turing machines restricted to use an amount of memory logarithmic in the size of the input.

Recall that $\mathbf{NC} = \cup_{i \geq 0} \mathbf{NC}^i$, where \mathbf{NC}^i is the class of languages $A \subset \{0, 1\}^*$ such that there exists a circuit family $\{C_n\}$ with the following properties (see, e.g., Du and Ko [4]).

1. there exists a Turing machine M that constructs (the encoding of) each C_n in log-space.
2. for all n , C_n has n input nodes with each node fan-in 2 and fan-out 1 and accepts $A_n = A \cap \{0, 1\}^n$.
3. there exist a polynomial function p and a constant $k > 0$, such that for all n , the size $size(C_n)$ of C_n is no more than $p(n)$ and the depth $depth(C_n)$ of C_n is no more than $k \log^i n$.

We call $\{C_n\}$ an \mathbf{NC}^i circuit family.

These complexity classes have properties such as $\mathbf{NC}^1 \subseteq \mathbf{L} \subseteq \mathbf{NC} \subseteq \mathbf{P}$. The interested readers are referred to Du and Ko [4].

NC functions can be viewed as an extension of **NC** languages. Namely, now for $i \geq 0$ and an \mathbf{NC}^i circuit family $\{C_n\}$, each node in C_n is allowed to have multiple fan-outs, and a function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ is computable by $\{C_n\}$ if for each n and each $x \in \{0, 1\}^n$, C_n computes $f(x)$. In other words, f is \mathbf{NC}^i computable if the language $A_f = \{\langle x, i \rangle : \text{the } i\text{-th bit of } f(x) \text{ is } 1\}$ is in \mathbf{NC}^i . For more details, see, for example, Allender et al. [2].

If \mathcal{C} is a complexity class of sets, we use \mathcal{C}_1 to denote the complexity class $\{A \in \mathcal{C} : A \subseteq \{0\}^*\}$, called the unary version of \mathcal{C} . Similarly, if FC is a class of functions, FC_1 is the corresponding class of functions whose domain is $\{0\}^*$. As shown in Ko [7], the unary classes are closely related to complexity classes of real numbers, because Cauchy functions are unary functions.

We define **NC** computable real numbers under different representations:

NC_{SLC}: the class of real numbers whose standard left cuts are **NC** computable.

NC_{BE}: the class of real numbers whose standard Cauchy functions are **NC** computable.

NC_{GLC}: the class of real numbers x such that there exists a general left cut of x that is **NC** computable.

NC_{CF}: the class of real numbers x such that there exists a Cauchy function of x that is **NC** computable.

Similarly for **L**, we define **L_{SLC}**, **L_{BE}**, **L_{GLC}**, **L_{CF}**, and for **P**, we define **P_{SLC}**, **P_{BE}**, **P_{GLC}**, **P_{CF}**.

3 Main Results

In this section, we present the main results. The first part contains results that do not rely on assumptions that $\mathbf{P}_1 \neq \mathbf{L}_1$ and $\mathbf{P}_1 \neq \mathbf{NC}_1$, and the second part contains results that rely on these assumptions. The main result is that, if $\mathbf{P}_1 \neq \mathbf{L}_1$ (or $\mathbf{P}_1 \neq \mathbf{NC}_1$), then the four classes of Log-space (or respectively, **NC**) computable real numbers under four different representations are all distinct classes (see Figure 1), while **L_{GLC}** is the most powerful one.

These results are more complicated than the complexity of these representations in **P**, as shown in Ko [7]:

$$\mathbf{P}_{SLC} = \mathbf{P}_{BE} \subsetneq \mathbf{P}_{GLC} = \mathbf{P}_{CF}.$$

3.1 Absolute Results

We first present some *absolute* results, which do not rely on assumptions on the complexity classes such as $\mathbf{P}_1 \neq \mathbf{L}_1$. Note that we can obtain trivial results $\mathbf{L}_{BE} \subseteq \mathbf{NC}_{BE} \subseteq \mathbf{P}_{BE}$ and so on, since $\mathbf{L} \subseteq \mathbf{NC} \subseteq \mathbf{P}$. We omit such results.

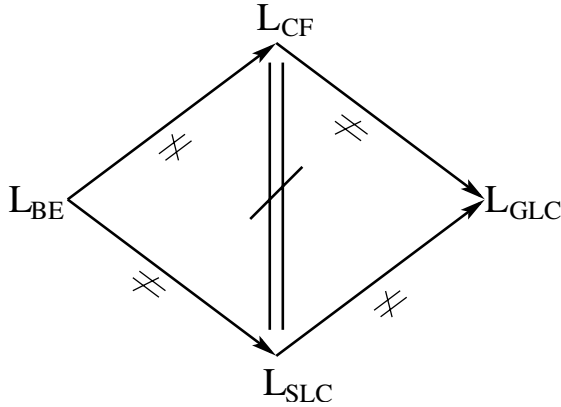


Fig. 1. Assuming $P_1 \neq L_1$

Theorem 3.1. $NC_{BE} \subseteq NC_{CF}, L_{BE} \subseteq L_{CF}$.

Proof. This is obvious since the standard Cauchy function b_x of a real number x is a Cauchy function of x . □

Theorem 3.2. $NC_{BE} \subseteq NC_{SLC}, L_{BE} \subseteq L_{SLC}$.

Proof. For any number $x \in NC_{BE}$, we have that b_x is in NC . If $x \in \mathbb{D}$, then $x \in NC_{SLC}$. Assume that $x \notin \mathbb{D}$. For any $n \in \mathbb{N}$ and $d \in \mathbb{D}_n$, it is in NC to compare $b_x(n)$ and d . Because $x \notin \mathbb{D}$, $d \leq b_x(n)$ implies that $d < x$, and $d > b_x(n)$ implies that $d > x$. This completes the proof for $NC_{BE} \subseteq NC_{SLC}$. Similarly, we can prove that $L_{BE} \subseteq L_{SLC}$. □

The following theorem extends Theorem 2.8 of Ko [7].

Theorem 3.3. *There exists a real number $x \in L_{CF} - P_{SLC}$. Thus, $L_{CF} - L_{SLC} \neq \emptyset$ and $NC_{CF} - NC_{SLC} \neq \emptyset$.*

Proof. We first define a function $T(n)$ inductively: $T(1) = 1$, and $T(n + 1) = 2^{2^{2^{T(n)}}}$. By a simple diagonalization, we can find a set $A \subseteq \{0\}^*$ such that A is computable in space $T(n)$ but not in space $\log T(n)$ (see, for example, Aho et al. [1]). Without loss of generality, let $0 \in A$. Define

$$x = \sum_{i=1}^{\infty} (2 \cdot \chi_A(0^i) - 1) 2^{-2^{T(i)}}.$$

First we show that $x \in L_{CF}$. We can compute, for each n , a dyadic rational $\phi(n)$ as follows:

- (1) find the integer k such that $T(k) \leq \log n < T(k + 1)$.
- (2) compute $\phi(n) = \sum_{i=1}^k (2 \cdot \chi_A(0^i) - 1) 2^{-2^{T(i)}}$.

It is clear that the value $\phi(n)$ computed above differs from x by at most $2^{-(2^{T(k+1)}-1)} \leq 2^{-n}$. So the above procedure computes a function $\phi \in CF_x$. Furthermore, since $T(k) \leq \log n$, the computation of $\phi(n)$ can be done in space $O(\log n)$. Thus, $x \in \mathbf{L}_{CF}$.

Next we show that $x \notin \mathbf{P}_{SLC}$. Assume, by way of contradiction, that SLC_x is computable in polynomial time. We will find a Turing machine M computing $\chi_A(0^n)$ in space $\log T(n)$.

Let M_a be a Turing machine that computes χ_A in space $T(n)$. The new machine for $\chi_A(0^n)$ works as follows. First, it simulates M_A on inputs $0, 0^2, \dots, 0^{n-1}$ and computes $d = \sum_{i=1}^{n-1} (2\chi_A(0^i) - 1)2^{-2^{T(i)}}$. Then it determines whether $d \in SLC_x$ and concludes that $\chi_A(0^n) = 1$ iff $d \in SLC_x$.

It is clear that the above machine M indeed computes χ_A . Assume that LC_x is computable in time $p(n)$ and hence in space $2^{p(n)}$. Then the space usage of the machine M is bounded by $O(\sum_{i=1}^{n-1} T(i)) + 2^{p(2^{T(n-1)})} < 2^{2^{2^{T(n-1)}}} = \log T(n)$ for almost all n , which contradicts to the fact that A is not computable in space $\log T(n)$.

In the above we have constructed a number $x \in \mathbf{L}_{CF} - \mathbf{P}_{SLC}$. As $\mathbf{L}_{CF} \subseteq \mathbf{NC}_{CF}$ and $\mathbf{L}_{SLC} \subseteq \mathbf{NC}_{SLC} \subseteq \mathbf{P}_{SLC}$, we have $\mathbf{L}_{CF} - \mathbf{L}_{SLC} \neq \emptyset$ and $\mathbf{NC}_{CF} - \mathbf{NC}_{SLC} \neq \emptyset$. □

Next we consider general left cuts.

Theorem 3.4. $\mathbf{NC}_{CF} \subseteq \mathbf{NC}_{GLC}, \mathbf{L}_{CF} \subseteq \mathbf{L}_{GLC}$.

Proof. For a number $x \in \mathbf{NC}_{CF}$, let $\phi \in CF_x$ be in **NC**. Note that the general left cut L_ϕ of x associated with ϕ is $\{d \in \mathbb{D}_n : n \in \mathbb{N}, d \leq \phi(n)\}$. Similar to the proof of Theorem 3.2, it is in **NC** to compare a dyadic number d and $\phi(n)$. □

3.2 Results Under Assumptions $\mathbf{P}_1 \neq \mathbf{L}_1$ and $\mathbf{P}_1 \neq \mathbf{NC}_1$

We use a specific coding system for the instantaneous descriptions (IDs) of the computation of a time-bounded Turing machine so that we can discuss the simulation of the machine (see Ko [7]). We assume that our machine M works on two tape symbols: 0 and 1 (and a special blank symbol), has k states q_1, \dots, q_k , uses a single tape, and has a time bound ψ . For each string $s \in \{0, 1\}^*$ of length $\ell(s) = n$, each ID of the computation of $M(s)$ is encoded by a string in $\{0, 1\}^*$ of length $2\psi(n) + 2k + 4$: we encode type symbols 0 and 1 by 01 and 10, respectively, and the blank symbol by 00, and the state symbol q_i by $11(01)^i(10)^{k-i}11$, where the state symbol appears just to the left of the tape symbol that is currently scanned by the tape head. (To see this in another way, at any moment, there are at most $\psi(n)$ symbols (of 0, 1 and the blank) in the computation, whose length is at most $2\psi(n)$ in our encoding, and the state is of length $2k + 4$.) Thus, for any input s of length n , the computation of $M(s)$ is encoded by a $(\psi(n) + 1) \cdot (2\psi(n) + 2k + 4)$ -bit string $\alpha_0\alpha_1 \dots \alpha_{\psi(n)}$, where α_i is the code of the i -th ID in the computation of $M(s)$. Note that we fix the length of the codes for IDs to be exactly $2\psi(n) + 2k + 4$, and fix the number of IDs in the computation to be exactly $\psi(n) + 1$.

We say a function $\phi : \mathbb{N} \rightarrow \mathbb{N}$ is *log-space computable* if it is log-space computable when both inputs and outputs are written in the binary form. Note that if ϕ is log-space computable, then it is also log-space computable when both inputs and outputs are written in the unary form.

We use the following lemma (Lemma 4.15 of Ko [7]).

Lemma 3.5. *Let M be a Turing machine having k states and having a time bound ψ , which is log-space computable. Then for any input st of length $\ell(s) = \ell(t) = \psi(n) + 2k + 4$ such that s is an ID of $M(u)$ for some string u of length n , we can determine, in log space, whether t is the successor of s in the computation of $M(u)$, or t is less than the successor ID, or t is larger than the successor ID. Furthermore, we can compute, in log space, the maximum m such that the first m bits of t agree with those of the successor ID of s .*

Theorem 3.6. *If $\mathbf{P}_1 \neq \mathbf{L}_1$, then $\mathbf{L}_{SLC} - \mathbf{L}_{CF} \neq \emptyset$.*

Proof. Let $T \in \mathbf{P}_1 - \mathbf{L}_1$ be computed by a deterministic Turing machine M in time $p(n)$. Assume that p is log-space computable. We use the encoding system described above. Assume that M has k states and so on input 0^n , an ID of $M(0^n)$ is of length $2\psi(n) + 2k + 4$. We let $s_{n,i}$, $0 \leq i \leq p(n)$, be the i -th ID of the computation of $M(0^n)$. Define a real number x whose binary expansion is

$$x = 0.01\tau(s_{1,0})\tau(s_{1,1}) \cdots \tau(s_{1,p(1)})\tau(s_{2,0}) \cdots \tau(s_{2,p(2)}) \cdots,$$

where τ is the local translation function defined by $\tau(0) = 01$, $\tau(1) = 10$, and $\tau(ab) = \tau(a)\tau(b)$ for all $a, b \in \{0, 1\}^+$.

We first show that $x \in \mathbf{P}_{CF} - \mathbf{L}_{CF}$. Let $q(n) = 2p(n) + 2k + 4$ and $r(n) = \sum_{i=1}^n 2q(i) \cdot (p(i) + 1)$. From $T \in \mathbf{P}_1$, it is easy to see that $x \in \mathbf{P}_{CF}$. Furthermore, note that for each 0^n , we need only $\tau(s_{n,p(n)})$, or, from the $(r(n) - 2q(n) + 3)$ -rd bit to the $r(n) + 2$ bit of the binary expansion of x , to determine whether $0^n \in T$. Therefore, if $x \in \mathbf{L}_{CF}$, then we can compute, in log space, an approximation value d to x such that $|d - x| \leq 2^{r(n)+4}$, and by our coding system, the first $r(n) + 2$ bits of d must be identical to those of x and so we can determine whether $0^n \in T$ in log space, which contradicts to the assumption of $T \in \mathbf{P}_1 - \mathbf{L}_1$.

Next we show that $x \in \mathbf{L}_{SLC}$, that is, $SLC_x = \{d \in \mathbb{D} : d < x\}$ is in \mathbf{L} . We only need to consider dyadic numbers of even lengths, because for a dyadic number d of an odd length, we can add a trailing zero to d . For a dyadic number $d \in \mathbb{D}_{2n}$, we can tell whether $d < x$ or $d > x$ in log space (note that since $x \notin \mathbf{L}_{CF}$, $x \notin \mathbb{D}$ and $d \neq x$). We achieve this by generating the initial IDs $s_{i,0}$ and applying Lemma 3.5 successively to each substring uv of d , where u is already been verified to be equal to $\tau(s_{i,j})$ for some j and $\ell(v) = \ell(u)$; furthermore, we check whether $v > \tau(w)$, $v = \tau(w)$ or $v < \tau(w)$, where w is the next ID of u . If $v > \tau(w)$, $d > x$; if $v < \tau(w)$, $d < x$; if $v = \tau(w)$, we continue with the process. If all bits of d agree with the first $2n$ bits of x , $d < x$. □

Corollary 3.7. *If $\mathbf{P}_1 \neq \mathbf{NC}_1$, then $\mathbf{NC}_{SLC} - \mathbf{NC}_{CF} \neq \emptyset$.*

Proof. The proof is the same as that of Theorem 3.6, except that we let $T \in \mathbf{P}_1 - \mathbf{NC}_1$ and the constructed number $x \in \mathbf{L}_{SLC} \subseteq \mathbf{NC}_{SLC}$. □

Corollary 3.8. *If $\mathbf{P}_1 \neq \mathbf{L}_1$, then $\mathbf{L}_{BE} \subsetneq \mathbf{L}_{SLC}$.*

Proof. Because we have $\mathbf{L}_{BE} \subseteq \mathbf{L}_{CF}$, $\mathbf{L}_{BE} \subseteq \mathbf{L}_{SLC}$ and $\mathbf{L}_{SLC} - \mathbf{L}_{CF} \neq \emptyset$. □

Corollary 3.9. (a) *If $\mathbf{P}_1 \neq \mathbf{L}_1$, then \mathbf{L}_{CF} and \mathbf{L}_{SLC} are incomparable.*
 (b) *If $\mathbf{P}_1 \neq \mathbf{NC}_1$, then \mathbf{NC}_{CF} and \mathbf{NC}_{SLC} are incomparable.*

Proof. Statement (a) follows Theorems 3.6 and 3.3. Statement (b) follows Corollary 3.7 and Theorem 3.3. □

Corollary 3.10. (a) *If $\mathbf{P}_1 \neq \mathbf{L}_1$, then $\mathbf{L}_{CF} \subsetneq \mathbf{L}_{GLC}$.*
 (b) *If $\mathbf{P}_1 \neq \mathbf{NC}_1$, then $\mathbf{NC}_{CF} \subsetneq \mathbf{NC}_{GLC}$.*

In summary, if $\mathbf{P}_1 \neq \mathbf{L}_1$, then \mathbf{L}_{SLC} , \mathbf{L}_{BE} , \mathbf{L}_{GLC} and \mathbf{L}_{CF} are four distinct classes (see Figure 1). If $\mathbf{P}_1 \neq \mathbf{NC}_1$, then \mathbf{NC}_{SLC} , \mathbf{NC}_{BE} , \mathbf{NC}_{GLC} and \mathbf{NC}_{CF} are four distinct classes. The converse also holds, since Ko [7] has shown that $\mathbf{P}_{CF} = \mathbf{L}_{CF} \Leftrightarrow \mathbf{P}_1 = \mathbf{L}_1$ and $\mathbf{P}_{CF} = \mathbf{NC}_{CF} \Leftrightarrow \mathbf{P}_1 = \mathbf{NC}_1$. We state it as a theorem.

Theorem 3.11. *The following are equivalent:*

- (a) $\mathbf{P}_1 \neq \mathbf{L}_1$.
- (b) \mathbf{L}_{SLC} , \mathbf{L}_{BE} , \mathbf{L}_{GLC} and \mathbf{L}_{CF} are four distinct classes.

*Similar results hold for **NC** real numbers.*

The following corollary is an *absolute* result.

Corollary 3.12. $\mathbf{L}_{CF} \neq \mathbf{L}_{SLC}$, $\mathbf{NC}_{CF} \neq \mathbf{NC}_{SLC}$.

Proof. If $\mathbf{L}_{CF} = \mathbf{L}_{SLC}$, then from Theorem 3.11, $\mathbf{P}_1 = \mathbf{L}_1$, and from Ko’s results, $\mathbf{L}_{CF} = \mathbf{P}_{CF}$, which implies $\mathbf{L}_{SLC} = \mathbf{P}_{CF}$ and furthermore $\mathbf{P}_{SLC} = \mathbf{P}_{CF}$. However, from Theorem 2.8 of Ko [7], $\mathbf{P}_{SLC} \subsetneq \mathbf{P}_{CF}$. Therefore, $\mathbf{L}_{CF} \neq \mathbf{L}_{SLC}$. We can prove in a similar way that $\mathbf{NC}_{CF} \neq \mathbf{NC}_{SLC}$. □

4 Conclusion

In this paper we have shown that the four representations of real numbers, standard left cut, general left cut, binary expansion and Cauchy function representation, have distinct expressive powers in **L** and **NC**, unless $\mathbf{L}_1 = \mathbf{P}_1$ or $\mathbf{NC}_1 = \mathbf{P}_1$, respectively. As the expressive powers of these four representations only fall into two categories in **P**, our results show that the expressive powers of these representations are more complicated in the parallel-time complexity world unless some sequential-time complexity class collapses to some parallel-time complexity class. Note that whether $\mathbf{L} = \mathbf{P}$ and whether $\mathbf{NC} = \mathbf{P}$ are important open questions, maybe just second to the **P** versus **NP** question, in the complexity theory.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The design and analysis of computer algorithms. Addison-Wesley, Reading (1974)
2. Allender, E., Loui, M.C., Regan, K.W.: Other Complexity Classes and Measures. Chapter 29. In: Atallah, M.J. (ed.) Algorithms and Theory of Computation Handbook, CRC Press, Boca Raton (1999)
3. Chen, Q., Su, K., Zheng, X.: Primitive recursiveness of real numbers under different representation. In: CCA. Electronic Notes in Theoretical Computer Science, vol. 167, Elsevier, Amsterdam (2007)
4. Du, D.-Z., Ko, K.-I.: Theory of Computational Complexity. John Wiley & Sons, Chichester (2000)
5. Hoover, H.J.: Feasible real functions and arithmetic circuits. SIAM J. Comput. 19(1), 182–204 (1990)
6. Ko, K.-I.: On the continued fraction representation of computable real numbers. Theor. Comput. Sci. 47(3), 299–313 (1986)
7. Ko, K.-I.: Complexity Theory of Real Functions. Birkhäuser, Boston (1991)
8. Yu, F.: On some complexity issues of NC analytic functions. In TAMC. In: Cai, J.-Y., Cooper, S.B., Li, A. (eds.) TAMC 2006. LNCS, vol. 3959, pp. 375–386. Springer, Heidelberg (2006)

Bounded Computable Enumerability and Hierarchy of Computably Enumerable Reals*

Xizhong Zheng

¹ Department of Computer Science, Jiangsu University, Zhenjiang 212013, China

² Theoretische Informatik, BTU Cottbus, D-03044 Cottbus, Germany

zheng@informatik.tu-cottbus.de

Abstract. The computable enumerability (c.e., for short) is one of the most important notion in computability theory and is regarded as the first weakening of the computability. In this paper, we explore further possible weakening of computable enumerability. By restricting numbers of possible big jumps in an increasing computable sequence of rational numbers which converges to a c.e. real number we introduce the notion of h -bounded c.e. reals and then shown that it leads naturally to an Ershov-style hierarchy of c.e. reals. However, the similar idea does not work for c.e. sets. We show that there is a computability gap between computable reals and the reals of c.e. binary expansions.

Keywords: c.e. sets, c.e. reals, bounded c.e. reals, Ershov's Hierarchy.

1 Introduction

A set A is *computably enumerable* (c.e., for short) if it can be enumerated by an effective procedure. That is, the elements of A can be enumerated effectively one after another, if A is not empty. The computable enumerability is one of the most important notion in the computability theory. This is not only because a lot of problems in mathematics and computer science correspond to the c.e. sets, but also the c.e. sets are regarded as the first natural weakening of the computable sets.

The definition of c.e. sets is generalized by Putnam [6], Gold [5] and Ershov [4] to k -c.e. for any constants k and h -c.e. for any functions h . The main idea behind these definitions is to allow the *mind-changes* or, as Putnam called, *trial and error* in the effective enumerations. For example, in an effective procedure to enumerate a set A , some number n may be enumerated into A at a stage, be removed from A at a later stage, and possibly be put into A afterwards again, and so on. If the number of such kind of *mind-changes* is bounded by a constant k for each n , then A is called k -c.e. A is called h -c.e. for a function h if the number of mind-changes about n is bounded by $h(n)$ for all n . An h -c.e. set is also called ω -c.e. if h is a computable function. Ershov [4] shows that, if $f(n) < g(n)$ for

* This work is supported by DFG (446 CHV 113/240/0-1) and NSFC (10420130638).

infinitely many n , then there is a g -c.e. set which is not f -c.e. This leads to a nice *Ershov's hierarchy* of the Δ_2^0 -sets:

$$\mathbf{EC} = 0\text{-CE} \subseteq k\text{-CE} \subsetneq (k + 1)\text{-CE} \subsetneq \omega\text{-CE} \subsetneq \Delta_2^0,$$

for all constant k , where **EC** (for effectively computable), $k\text{-CE}$ and $\omega\text{-CE}$ denote the classes of computable, k -c.e. and ω -c.e. sets, respectively. Especially, the class 1-CE is just the class of c.e. sets. The 2-c.e. sets are usually called d -c.e. because they are the differences of c.e. sets. This hierarchy is even valid for the corresponding classes of Turing degrees as shown by Cooper [1]. Namely, there is a $(k + 1)$ -c.e. degree which is not k -c.e.; There is an ω -c.e. degree which is not k -c.e. for any constant k , and there is a Δ_2^0 -degree which is not ω -c.e.

The Ershov's hierarchy classifies Δ_2^0 -sets nicely to different classes according to the different levels of enumerability. However, from the computability point of view, it has also some unnatural properties. For example, the classes $k\text{-CE}$ for $k > 0$ are not symmetrical. That is, a set and its complement do not necessarily belong to the same class, although they have exactly the same computability. The classes of Ershov's hierarchy, as binary expansions, do not correspond to the classes of reals of weaker computability defined by computable sequences of rational numbers (see e.g., [13,15]). These problems disappear if we consider the *h -bounded computability* (h -b.c.) introduced in section 2. In this case, we are only interested in how many mind-changes are necessary to approximate a set and do not demand an empty set to begin with.

To explore the notion weaker than c.e. we have to look at other properties which are "stronger" than simply counting the changes of membership of n in an approximation to the set A . A natural candidate here is considering the changes of the initial segments $A_s \upharpoonright n$ instead of $A_s(n)$ in an approximation (A_s) to A . Accordingly we define the notion of *h -initially-bounded computable* (h -i.b.c., for short) sets. Unfortunately, the classes of h -i.b.c. sets collapse to the set of computable sets for any constant functions h . Therefore, h -i.b.c. sets do not lead to an Ershov hierarchy of c.e. sets and we do not achieve a reasonable notion weaker than c.e. by this approach.

As a new approach we consider the computable enumerability of reals. By definition c.e. reals are limits of computable increasing sequences of rational numbers. By restricting the numbers of possible big jumps in an increasing sequence of rational numbers converges to a c.e. real we introduce the notion of *h -bounded c.e.* (h -b.c.e.) reals. We can see that the k -b.c.e. reals form an Ershov-style hierarchy of c.e. reals for constants k . This hierarchy is valid even in the sense of Turing degrees. A very interesting result here is that non-computable k -b.c.e. reals do not have c.e. binary expansions. This means that there are reals which have some weak computability between the computable reals and strongly c.e. reals (the reals of c.e. binary expansions).

The paper is organized as follows. Section 2 introduces the new notions of h -bounded computable and h -initially bounded computable sets and discusses their basic properties. In section 3 we consider h -bounded c.e. real numbers and show an Ershov hierarchy of k -b.c.e. reals. In the last section 4 we extend the

Ershov hierarchy of classes of k -b.c.e. reals to the classes of Turing degrees which contain k -b.c.e. real numbers.

2 Bounded Computable Sets

This section discusses the bounded computability of sets. Firstly the notion of h -c.e. sets is symmetrized to the notion of h -bounded computable sets. Then we introduce the notion of initially-bounded computable sets to explore the possible hierarchy of c.e. sets. We will see that an Ershov style hierarchy for initially bounded computable sets fails.

By definition, a set A is c.e. (computably enumerable) means that there is a computable sequence (A_s) of finite sets which converges to A such that $A_0 = \emptyset$ and $A_s \subseteq A_{s+1}$ for all s . This has been generalized by Putnam [6], Gold [5], and Ershov [4] to the following: A is called h -computably enumerable (h -c.e.) if A has an h -enumeration (A_s) which is a computable sequence of finite sets converging to A such that $A = \emptyset$ and

$$(\forall n)(|\{s : A_s(n) \neq A_{s+1}(n)\}| \leq h(n)). \tag{1}$$

We identify in this paper a set with its characteristic function. Thus we have $A(n) = 1$ if $n \in A$ and $A(n) = 0$ if $n \notin A$. Alternatively, a set A is h -c.e. iff there is a computable function f such that $f(0, n) = 0$, $\lim_s f(s, n) = A(n)$ and

$$(\forall n)(|\{s : f(s, n) \neq f(s + 1, n)\}| \leq h(n)). \tag{2}$$

Obviously, c.e. sets are just the h -c.e. sets for the constant function $h(n) \equiv 1$. It is worth noting that the condition $f(0, n) = 0$ in the second definition of c.e. sets is necessary because otherwise the 1-c.e. sets were not necessarily c.e. This corresponds to requiring $A_0 = \emptyset$ in the first definition which is however superfluous because A_0 is a finite set and the h -c.e. is invariable under the change on a finite set. Here the computable sequence (A_s) is regarded as an enumeration instead of an approximation of A . If we are more interested in how effectively a set can be approximated, we should consider the following variation.

Definition 1. A set A is called h -bounded computable (h -b.c., for short) if there is a computable sequence (A_s) of finite sets which converges to A such that

$$(\forall n)(|\{s \geq n : A_{s+1}(n) \neq A_s(n)\}| \leq h(n)). \tag{3}$$

The additional condition $s \geq n$ in (3) ignores inessential changes of $A_s(n)$ before the stage $s := n$. This can be replaced equivalently by $s \geq g(n)$ for an increasing computable function g . We summarize some important properties of h -b.c. sets without proof as follows.

Theorem 1. Let h be a function and let A be a set.

1. If A is h -c.e. or co- h -c.e., then A is h -b.c.
2. If A is h -b.c., then A is h' -c.e. for $h'(n) := h(n) + 1$.

3. If A is h -b.c., then so is the complement \overline{A} .
4. A is h -b.c. iff there is a computable function f such that $\lim_s f(s, n) = A(n)$ and $|\{s : f(s, n) \neq f(s + 1, n)\}| \leq h(n)$ for all n .
5. A is h -b.c. iff there is a computable function g and a computable sequence (A_s) of finite sets such that $(\forall n)(|\{s \geq g(n) : A_{s+1}(n) \neq A_s(n)\}| \leq h(n))$.

Analogous to the well known fact that a set A is computable if and only if A as well as its complement \overline{A} are c.e., we have the following result.

Theorem 2. For any functions h and $h_1(n) := h(n) + 1$, a set A is h -b.c. if and only if both A and its complement \overline{A} are h_1 -c.e.

Proof. We need only to prove the direction “ \Leftarrow ”. Suppose that (A_s) and (B_s) are h_1 -enumerations of A and \overline{A} , respectively. Assume w.l.o.g. that $A_0 = B_0 = \emptyset$. Since $\lim_s A_s(n) = A(n) \neq \overline{A}(n) = \lim_s B_s(n)$ for all n , we can define a total computable increasing function $v : \mathbb{N} \rightarrow \mathbb{N}$ by

$$\begin{cases} v(0) := \min\{s : A_s(0) \neq B_s(0)\}; \\ v(n + 1) := \min\{s > v(n) : A_s(n + 1) \neq B_s(n + 1)\}. \end{cases}$$

Then we define a computable function f by

$$f(s, n) := \begin{cases} A_{v(n)}(n) & \text{if } A_{v(n)}(n) = 1 \ \& \ s \leq v(n); \\ A_s(n) & \text{if } A_{v(n)}(n) = 1 \ \& \ s > v(n); \\ 1 \dot{-} B_{v(n)}(n) & \text{if } A_{v(n)}(n) = 0 \ \& \ s \leq v(n); \\ 1 \dot{-} B_s(n) & \text{if } A_{v(n)}(n) = 0 \ \& \ s > v(n). \end{cases}$$

Obviously, we have $\lim_s f(s, n) = A(n)$ for all n . Given an n , suppose that $A_{v(n)}(n) = 1$. If $s \geq v(n)$ such that $f(s, n) \neq f(s + 1, n)$, then $A_s(n) \neq A_{s+1}(n)$. Since $A_{v(n)}(n) = 1 \neq 0 = A_0(n)$ and (A_s) is an h_1 -enumeration, there are at most $h_1(n) - 1$ such stages $s \geq v(n)$. The same holds if $A_{v(n)}(n) = 0$. This implies that

$$(\forall n)(|\{s \geq v(n) : f(s, n) \neq f(s + 1, n)\}| \leq h(n)).$$

Because $f(s, n) = f(s + 1, n)$ hold for all $s < v(n)$, we have actually

$$(\forall n)(|\{s : f(s, n) \neq f(s + 1, n)\}| \leq h(n)),$$

and hence, by Theorem 4, A is h -b.c. □

From Theorem 1 we have seen that h -bounded computability is not very far from the h -computably enumerability. The notion introduced in the next definition is essentially different from h -computable enumerability. Here we count the changes of the initial segment $A_s \upharpoonright n$ instead of the changes of the single membership $A_s(n)$ for different indices s .

Definition 2. Let $h : \mathbb{N} \rightarrow \mathbb{N}$ be a function and let $A \subseteq \mathbb{N}$ be a set.

1. A sequence (A_s) of finite sets converges h -initially-bounded effectively to A if $\lim_{s \rightarrow \infty} A_s = A$ and

$$(\forall n) (|\{s \geq n : A_s \upharpoonright n \neq A_{s+1} \upharpoonright n\}| \leq h(n)). \tag{4}$$

2. A set A is called h -initially-bounded computable (h -i.b.c., for short) if there is a computable sequence of finite sets which converges h -initially-bounded effectively to A .

For constant function $h \equiv k$, we call h -i.b.c. sets simply k -i.b.c. Thus A is computable iff it is 0-i.b.c. Analogous to h -b.c. sets, the condition $s \geq n$ in (4) can be replaced by $s \geq g(n)$ for an increasing computable function g .

The following proposition can be proved straightforwardly.

Proposition 1. *Let $h : \mathbb{N} \rightarrow \mathbb{N}$ be a function and let A be a set.*

1. *If A is h -c.e., then it is g -i.b.c. for the function $g(n) := \sum_{i < n} h(i)$;*
2. *If A is h -i.b.c., then it is g -c.e. for the function $g(n) := h(n + 1) + 1$;*
3. *If A is h -i.b.c., then so is its complement \bar{A} .*
4. *If A and B are h -i.b.c. and g -i.b.c., respectively, then $A \cap B, A \cup B$ and $A \setminus B$ are $(g + h)$ -i.b.c.*

Given a constant k , any k -c.e. set is h -i.b.c. for $h(n) := kn$ by Proposition 1. Actually, even for the function g whose distance to the function $h(n) := kn$ is bounded by a constant, any k -c.e. set is also g -i.b.c. as shown in the next theorem. This condition is even necessary.

Theorem 3. *Let k be a constant.*

1. *If A is k -c.e., then it is h_c -i.b.c. for any constant c , where $h_c(n) := kn \div c$.*
2. *If h is a nondecreasing computable function such that*

$$(\forall c)(\exists n)(h(n) + c < kn), \tag{5}$$

then there is a k -c.e. set which is not h -i.b.c.

Proof. Let (A_s) be a k -enumeration of A . That is, (A_s) is a computable sequence of finite sets satisfying (1) for $h(n) \equiv k$ such that $A_0 = \emptyset, \lim_s A_s = A$. Let N be a natural number such that $A_s \upharpoonright c = A \upharpoonright c$ hold for all $s \geq N$. Define a computable sequence (B_s) of finite sets by $B_s := A_{N+s}$. The initial segment $B_s \upharpoonright n$ can change only at the positions $m \in [c, n)$ and at most $k(n \div c) \leq h_c(n)$ times. Therefore (B_s) converges h_c -initial-bounded effectively to A and hence A is h_c -i.b.c.

Let h be a nondecreasing computable function which satisfies condition (5). Therefore, we can define an increasing computable sequence (v_s) inductively as follows:

$$\begin{cases} v_0 := 0 \\ v_{n+1} := (\mu t)(kt > kv_n + h(t)). \end{cases}$$

Let $I_n := [v_n; v_{n+1})$. Then the (I_s) is a computable sequence of disjoint intervals of natural numbers such that $k \cdot l(I_n) > h(v_{n+1})$ for all n , where $l(I_n)$ is the length of the interval I_n .

Let (V_e) be a computable enumeration of all approximable sets and $(V_{e,s})_s$ be a computable sequence of finite sets which approximates the set V_e . Now we can construct a k -c.e. set A which is not h -i.b.c., i.e., A satisfies, for all e , the following requirements:

$$R_e \quad : \quad (V_{e,s}) \text{ converges } h\text{-initially-bounded effectively to } V_e \implies A \neq V_e.$$

The set A is constructed in stages such that, for any e , there is an $n_e \in I_e$ which witnesses the inequality $A(n_e) \neq V_e(n_e)$. The witness n_e can be chosen in the following way. We reserve the interval I_e exclusively for the requirement R_e . Suppose that $A \cap I_e = \emptyset$ and let $n_e := v_e$ at the stage $s := v_{e+1}$. We put n_e into A if it is not yet in V_e . If n_e enters V_e , then delete n_e from A and it can enter A again if n_e leaves V_e at a later stage. In order to guarantee that A is k -c.e., we do such kind of actions at most k times for the same n_e . After that, if $V_e(n_e)$ changes again, then we define $n_e := n_e + 1$ and let this new witness be outside of A or put it into A at the beginning depending on if n_e is already in V_e or not. Repeat the above procedure again for the new n_e , and so on. We continue this process until $n_e = v_{e+1}$. If this process stops for some $n_e < v_{e+1}$, then n_e is a right witness for R_e . Otherwise, if n_e arrives v_{e+1} , then this means that the initial segment $V_{e,s} \upharpoonright v_{e+1}$ changes at least $k \cdot l(I_e) > h(v_{e+1})$ times and the sequence $(V_{e,s})$ does not converge h -initially bounded effectively. In this case, the requirement R_e is satisfied trivially. More precisely, the set A can be constructed by a priority methods without injury. \square

As an immediately corollary of Proposition [1](#), any c.e. set is h_c -i.b.c. for the function $h_c(n) := n \dot{-} c$. However, if $h \in o(\text{id})$, then there is a c.e. set which is not h -i.b.c., where id is the identity function. This implies especially that there is a c.e. set which is not k -i.b.c. for any constant k . Actually, the next theorem shows that any non-computable c.e. sets are not k -i.b.c. for any constant k .

Theorem 4. *Let k be a constant. Any k -i.b.c. set is computable.*

Proof. Let A be a k -i.b.c. set and let (A_s) be a computable sequence of finite sets which converges h -initially-bounded effectively to A . Suppose w.l.o.g. that k is the least constant such that

$$|\{s \geq n : A_s \upharpoonright n \neq A_{s+1} \upharpoonright n\}| = k$$

for infinitely many n . Thus, for any $m \in \mathbb{N}$, we can always effectively find an $n > m$ and an index $s_0 > n$ such that $A_s \upharpoonright n$ changes already k times between the stages n and s_0 . Therefore, $A_s(m)$ does not change any more after stage s_0 and we have $A(m) = A_{s_0}(m)$. This implies that A is computable. \square

Theorem [4](#) implies that the classes of k -i.b.c. sets collapse to the the first level—the class of computable sets. Therefore the k -i.b.c. sets do not lead to an Ershov-style hierarchy of c.e. sets.

3 Bounded C.E. Real Numbers

In this section we turn to the computability of real numbers. Motivated from the notions of h -i.b.c. sets and h -effectively computable real numbers of [12] we introduce the notion of h -bounded c.e. reals and show that this leads naturally to an Ershov-style hierarchy of c.e. reals.

A real number x is called *computably enumerable* (c.e., for short) if there is an increasing computable sequence (x_s) of rational numbers which converges to x . Equivalently x is c.e. iff its (left) Dedekind cut $L_x := \{r \in \mathbb{Q} : r < x\}$ is a c.e. set of rational numbers. As it is pointed out in Soare [8,9], the c.e. real numbers correspond naturally to c.e. sets and can be regarded as the first weakening of computable real numbers (see [3,11,14]). We explore now the weakening of c.e. reals by bounding the number of big jumps in the approaching sequences.

Definition 3. *Let h be a function.*

1. *An increasing sequence (x_s) converges to x h -effectively if, for all n , the length of the index-chain $n = s_0 < s_1 < s_2 < \dots < s_k$ which satisfies*

$$(\forall i < k) (x_{s_{i+1}} - x_{s_i} > 2^{-n})$$

is bounded by $k \leq h(n)$.

2. *A real number x is called h -bounded c.e. (h -b.c.e., for short) if there is an increasing computable sequence (x_s) of rational numbers which converges to x h -effectively.*

The class of all h -b.c.e. real numbers is denoted by h -**BCE**. Especially, if $h(n) \equiv k$ is a constant function, then k -**BCE** denotes the class of all k -b.c.e. real numbers. We call a real *bounded computably enumerable* (b.c.e., for short) if it is k -b.c.e. for a constant k . The class of all b.c.e. real numbers is denoted by **BCE**. The next lemma follows immediately from the definition.

Lemma 1. *1. A real number x is computable if and only if it is 0-b.c.e.
 2. Any c.e. real number x is h -b.c.e. for the function $h(n) := 2^n$.*

Now we show that k -b.c.e. reals form an Ershov-style hierarchy.

Theorem 5. *For any constant k , there is a $(k + 1)$ -b.c.e. real number which is not k -b.c.e.*

Proof. For any constant k , we are going to construct an increasing computable sequence (x_s) of rational numbers which converges $(k + 1)$ -effectively to a non- k -b.c.e. real number x . To this end, the limit x has to satisfy all the following requirements

$$R_e : (\varphi_e(s))_s \text{ is increasing and converges } k\text{-effectively to } y_e \implies x \neq y_e$$

where (φ_e) is a computable enumeration of all partial computable functions $\varphi_e : \subseteq \mathbb{N} \rightarrow \mathbb{Q}$.

The sequence (x_s) is constructed in stages. To satisfy a single requirement R_e , we choose a rational interval I^{e-1} of the length $2^{-n_{e-1}}$ for some natural number n_{e-1} and divide it equidistantly into at least $2k+5$ subintervals I_0, I_1, \dots, I_{m_e} of the same length 2^{-n_e} for some natural number n_e . Our goal is to find a *witness interval* I^e for R_e from I_0, I_1, \dots, I_{m_e} such that any element of I^e satisfy the requirement R_e . As default, let $I^e := I_1$ be the first candidate of I^e and define x_s to be the middle point of I_1 at the beginning. If at some stage s_0 we find that there is a $t_0 \geq n_e$ such that $\varphi_e(t_0) \in I_1$, then $I^e := I_3$ is the new candidate of the witness interval and redefine x_{s_0+1} to be the middle point of the interval I_3 . And if at a later stage $s_1 > s_0$, we find a $t_1 > t_0$ such that $\varphi_e(t_1) \in I_3$, then change x_{s_1+1} to the middle point of $I^e := I_5$ which is the new current candidate of witness interval and so on. Obviously, at most $k+1$ redefinitions guarantee that the limit $x := \lim_{s \rightarrow \infty} x_s$ is different from the possible limit $\lim_{s \rightarrow \infty} \varphi_e(s)$ if it converges k -effectively and we achieve finally a correct witness interval I^e . On the other hand, the sequence (x_s) converges $(k+1)$ -effectively.

In order to satisfy all requirements simultaneously, a finite injury priority construction suffices. Here we mention only two important points. Firstly, the candidates of witness interval for R_e are chosen from the subintervals of the current candidate of witness interval for R_{e-1} . Secondly, the default candidate of witness interval for R_e (the interval I_0 explained above) should be chosen in such a way that the current x_s which is defined from the current candidate interval for R_{e-1} should be exactly its middle point. This avoids the extra jumps of the sequence (x_s) and guarantees that the constructed sequence converges $(k+1)$ -effectively indeed. \square

It is natural to ask whether b.c.e. reals exhaust all c.e. real numbers. We will give a negative answer to this question. Actually we achieve a more stronger result. Recall that a real x is called *strongly c.e.* (see [2]) if it has a c.e. binary expansion A , i.e., $x = x_A := \sum_{i \in A} 2^{-(i+1)}$. As pointed out by Jockusch (see [8]) the class of strongly c.e. real numbers form a proper subset of c.e. real numbers. The next theorem implies that there is a c.e. real number which is not b.e.c.

Theorem 6. *If a real number is both b.c.e. and strongly c.e., then it must be computable.*

Proof. Let x be a b.c.e. real number for some constant k with a c.e. binary expansion A (i.e., $x = x_A$), and suppose that x is irrational. Then we have a strictly increasing computable sequence (x_s) of rational numbers which converges to x k -effectively for some constant k as well as a computable sequence (A_s) of finite sets such that $A_s \subset A_{s+1}$ and $A = \lim_{s \rightarrow \infty} A_s$. Since both sequences (x_s) and (x_{A_s}) are strictly increasing, we can define inductively two computable increasing functions $u, v : \mathbb{N} \rightarrow \mathbb{N}$ as follows.

$$\left\{ \begin{array}{l} u(0) := 0 \\ v(0) := \min\{s : x_{A_s} > x_0\} \\ u(n+1) := \min\{s > u(n) : x_s > x_{A_{v(n)}}\} \\ v(n+1) := \min\{s > v(n) : x_{A_s} > x_{u(n)}\}. \end{array} \right.$$

Let $y_s := x_{u(s)}$ and $B_s := A_{v(s)}$ for all s . Then (y_s) and (x_{B_s}) are computable subsequences of (x_s) and (x_{A_s}) , respectively which satisfy the following

$$y_0 < x_{B_0} < y_1 < x_{B_1} < y_2 < x_{B_2} < \dots$$

As a subsequence of (x_s) , the sequence (y_s) converges k -effectively to x too. That is, for any n , the length of any index-chain $s_0 < s_1 < s_2 < \dots < s_t$ such that $y_{s_{i+1}} - y_{s_i} > 2^{-n}$ for all $i < t$ is bounded by $t \leq k$.

Given a natural number n . If at a stage $s + 1 > n$ some element less than n is enumerated into B_{s+1} , i.e. $B_s \upharpoonright n \neq B_{s+1} \upharpoonright n$, then we have $x_{B_{s+1}} - x_{B_s} > 2^{-n}$, and hence $y_{s+2} - y_s > 2^{-n}$. This means that the initial segment $B_s \upharpoonright n$ can be changed after the stage n at most $2k$ times. That is, we have

$$|\{s > n : B_s \upharpoonright n \neq B_{s+1} \upharpoonright n\}| \leq 2k, \tag{6}$$

for all natural numbers n . Therefore, the computable sequence (B_s) converges $2k$ -initially bounded effectively to A and the set A is $2k$ -i.b.c. By theorem 4, A is computable and hence the real x is computable. \square

Although the proof of Theorem 6 is very simple, the result seems quite surprising. As we have mentioned before, the computable enumerability of a set is the first weakening of the computability. Therefore, the strongly computable enumerability of a real number can be regarded as the first weakening of the computability of real numbers with respect to binary expansion. However, Theorem 6 shows that there is a gap between computability and strong computability of real numbers if we consider how effectively a real number can be approximated by computable sequence of rational numbers.

4 Hierarchy of Turing Degrees

In this section we discuss the hierarchy Turing degrees which contain k -b.c.e. real numbers for different constant k . This strengthens the hierarchy of Theorem 5. To simplify the notation we identify a real number x with its characteristic binary sequence in this section.

Let (Φ_e) be an effective enumeration of computable partial functionals. By definition, a real number x is *Turing reducible* to another real number y (denoted by $x \leq_T y$) if there is an index e such that $x(n) = \Phi_e^y(n)$ for all n . Two real numbers x and y are *Turing equivalent* (notation $x \equiv_T y$) if $x \leq_T y$ and $y \leq_T x$ hold. In other words, x is Turing equivalent to y if there are indices i and j such that $x(n) = \Phi_i^y(n)$ and $y(n) = \Phi_j^x(n)$ for all n . From these it is not difficult to find (possibly different) indices i and j such that

$$(\forall n) (x \upharpoonright n = \Phi_i^y(n) \ \& \ y \upharpoonright n = \Phi_j^x(n)). \tag{7}$$

We say that x is (i, j) -Turing equivalent to y (denote by $x \equiv_T^{(i,j)} y$) if they satisfy condition (7). For the (i, j) -Turing equivalence relates more closely to the topological property of real numbers. The following important technical lemma will be used later.

Lemma 2 (Rettinger and Zheng [7]). *For any rational interval I_0 and any natural numbers i, j, t there are two open rational intervals $I \subseteq I_0$ and J such that*

$$(\forall x, y) \left(x \equiv_T^{(i,j)} y \implies (x \in I \implies y \in J) \ \& \ (y \in J \implies x \in I_0) \right). \quad (8)$$

We say that an intervals I is (i, j) -reducible to another interval J (denoted by $I \preceq^{(i,j)} J$) if they satisfy the following condition

$$(\forall x, y) \left(x \in I \ \& \ x \equiv_T^{(i,j)} y \implies y \in J \right).$$

By Lemma 2, there are $I \subseteq I_0$ and J such that $I \preceq^{(i,j)} J$ for any given interval I_0 . If all elements of I_0 are not (i, j) -Turing equivalent to some element, then this holds trivially. Actually, the Lemma 2 holds even in an more effective sense. Namely, if there exists $x \in I_0$ which is (i, j) -Turing equivalent to some y , then the intervals I and J which satisfy condition (7) can be effectively found (see Lemma 2.2 of [7] for details). This fact will be used in the proof of the following theorem.

Theorem 7. *For any constant k , there is a $(k + 1)$ -b.c.e. real number which is not Turing equivalent to any k -b.c.e. real numbers.*

Proof. We construct an increasing computable sequence (x_s) of rational numbers which converges $(k + 1)$ -effectively to a non- k -b.c.e. real number x . The limit x has to satisfy, for all i, j, k , all the following requirements

$$R_{\langle i,j,k \rangle} : \quad (\varphi_k(s)) \text{ converges } k\text{-effectively to } y_k \implies x \not\equiv_T^{(i,j)} y_k.$$

We explain firstly the idea how a single requirement R_e for $e := \langle i, j, k \rangle$ can be satisfied. Choose arbitrarily a rational interval I_{e-1} . We want to find a rational subinterval $I_e \subseteq I_{e-1}$ such that all $x \in I_e$ satisfy R_e . This interval I_e is called a *witness interval* of R_e .

Divide the interval I_{e-1} into at least $2k + 5$ rational subintervals I^0, I^1, I^2, \dots with the same length 2^{-n_e} for some natural number n_e . According to (the effective version of) Lemma 2, either we can find an interval I_{2t+1} (for some $t \leq k$) whose elements do not (i, j) -Turing equivalent to any real number, or we can find rational subintervals $I^t \subseteq I^t$ and rational intervals J^t such that $I^t \preceq^{(i,j)} J^t$ for $t = 1, 3, \dots, 2k + 3$. W.l.o.g. we can assume that the distances between any pair of J -intervals is larger than 2^{-m_e} for some natural number m_e .

The sequence (x_s) can be constructed in stages. At the beginning we choose I_e^1 as default candidate of the witness interval of R_e and let x_s be its middle point. We do not change x_s as long as the sequence $(\varphi_k(t))$ does not enters the interval J^1 after the stage $s := m_e$. Otherwise, if at some stage s_0 we find a $t_0 > m_e$ such that $\varphi_k(t_0) \in J^1$, then change the candidate of witness interval of R_e to the interval I_e^3 and redefine x_{s_0+1} as the middle point of I_e^3 . If at a late stage $s_1 > s_0$, we find a new $t_1 > t_0$ such that $\varphi_k(t_1) \in J^3$, then change the candidate interval to I_e^5 and redefine x_{s_1+1} as its middle point, and so on. We allow at

most $k + 1$ times such kind of redefinition which suffices to guarantee that the limit of the sequence (x_s) is different to the possible limit of $\lim_{s \rightarrow \infty} \varphi_k(s)$ if it converges k -effectively.

We need only a standard finite injury priority construction to construct the computable sequence (x_s) and x_s is chosen from the actually smallest witness intervals defined at the stage s . The details are omitted here. \square

References

1. Cooper, B.S.: Degrees of Unsolvability. Ph.D thesis, Leicester University, Leicester, England (1971)
2. Downey, R.G.: Some computability-theoretic aspects of reals and randomness. In: The Notre Dame lectures. Assoc. Symbol. Logic. Lect. Notes Log., vol. 18, pp. 97–147. Urbana, IL (2005)
3. Downey, R.G., Hirschfeldt, D.R.: Algorithmic Randomness and Complexity. Springer, Heidelberg, Monograph to be published
4. Ershov, Y.L.: A certain hierarchy of sets. i, ii, iii. (Russian). Algebra i Logika. 7(1), 47–73 (1968), 7(4), 15–47 (1968), 9, 34–51 (1970)
5. Gold, E.M.: Limiting recursion. J. Symbolic Logic 30, 28–48 (1965)
6. Putnam, H.: Trial and error predicates and the solution to a problem of Mostowski. J. Symbolic Logic 30, 49–57 (1965)
7. Rettinger, R., Zheng, X.: A hierarchy of Turing degrees of divergence bounded computable real numbers. J. Complexity 22(6), 818–826 (2006)
8. Soare, R.I.: Cohesive sets and recursively enumerable Dedekind cuts. Pacific J. Math. 31, 215–231 (1969)
9. Soare, R.I.: Recursion theory and Dedekind cuts. Trans. Amer. Math. Soc. 140, 271–294 (1969)
10. Soare, R.I.: Recursively enumerable sets and degrees. A study of computable functions and computably generated sets. Perspectives in Mathematical Logic. Springer, Heidelberg (1987)
11. Weihrauch, K.: Computable Analysis, An Introduction. Springer, Heidelberg (2000)
12. Zheng, X.: Classification of the computably approximable real numbers. Theory of Computing Systems (to appear)
13. Zheng, X.: Recursive approximability of real numbers. Mathematical Logic Quarterly 48(Suppl. 1), 131–156 (2002)
14. Zheng, X.: Computability Theory of Real Numbers. Habilitation's thesis, BTU Cottbus, Germany (February 2005)
15. Zheng, X., Rettinger, R.: Weak computability and representation of reals. Mathematical Logic Quarterly 50(4/5), 431–442 (2004)

Streaming Algorithms Measured in Terms of the Computed Quantity^{*}

Shengyu Zhang

California Institute of Technology
Computer Science Department and Institute for Quantum Information,
1200 E California Bl, MC 107-81, Pasadena, CA 91125, USA
shengyu@caltech.edu

Abstract. The last decade witnessed the extensive studies of algorithms for data streams. In this model, the input is given as a sequence of items passing only once or a few times, and we are required to compute (often approximately) some statistical quantity using a small amount of space. While many lower bounds on the space complexity have been proved for various tasks, almost all of them were done by reducing the problems to the cases where the desired statistical quantity is at one extreme end. For example, the lower bound of triangle-approximating was showed by reducing the problem to distinguishing between graphs without triangle and graphs with only one triangle.

However, data in many practical applications are not in the extreme, and/or usually we are interested in computing the statistical quantity only if it is in some range (and otherwise reporting “too large” or “too small”). This paper takes this practical relaxation into account by putting the computed quantity itself into the measure of space complexity. It turns out that all three possible types of dependence of the space complexity on the computed quantity exist: as the quantity goes from one end to the other, the space complexity can goes from max to min, remains at max, or goes to somewhere between.

1 Introduction

Data stream is a very natural and important model for massive data sets in many applications, where the input data is given as a stream of items with only one or a few passes, and usually we want to determine or approximate some statistical quantity of the input stream. See [16] for an excellent and comprehensive survey.

Many algorithms are designed and many lower bounds on the space complexities are proven for various types of problems such as, just to name a few, frequency moments [1, 7, 14, 3, 6], vector distance [11, 17], and some graph problems [2, 4, 9, 13]. Almost all lower bounds were proved by reducing the problem to the cases where the desired statistical quantity is at an extreme end (and this

^{*} This work was mostly done when the author was a graduate student in Computer Science Department at Princeton University, supported in part by NSF grants CCR-0310466 and CCF-0426582.

is further reduced to the communication complexity of some related problems). For example, the lower bound for approximating the number of triangles was proved by a reduction to distinguishing between the graph containing 0 and 1 triangle; the lower bound for the infinity frequency moment F_∞^* was proved by reducing the problem to distinguishing between $F_\infty^* = 1$ and $F_\infty^* = 2$.

Despite its theoretical correctness, this reduction to extreme cases can be misleading for many practical applications for at least the following two reasons. First, the extreme case may not happen at all in practice. For example, the number of triangles in a graph has many implications in various applications. In a social network, the number of triangles characterizes the average strength of the ties in the community [8, 18]. But note that in most (if not all) practical communities, there are a large number of triangles, and those extreme cases (0 and 1 triangle) are never the case. As another example, in many applications such as data mining, the number of common neighbors of two vertices in a graph shows the amount of common interest. A canonical example is that if two commodities have a large number of common buyers, then putting these two commodities close to each other in a supermarket will make more sales for both of them. Similar to the triangle example, the maximal number of common neighbors from data in practice is always large.

The second reason is from the user side. Even if some data happen to have the quantity at extreme, we are not interested in it in this case. For example, if two commodities have a very small number (such as one) of common buyers, then it barely means anything because that buyer may just happen to buy them. Therefore, we have a *threshold range* in mind within which we care about the quantity; if the quantity is outside the range, we will be satisfied if the algorithm can report “too low” or “too high”.

Due to these two reasons, it is natural to ask the following question: is the hardness of a problem essentially due to the extreme cases? To answer this question, we study the space complexity in terms of both input size and the threshold range. In particular, for any input size n and any possible quantity value $q(n)$, the stream space complexity $s(n, q(n))$ is, roughly speaking, the minimal space used to compute $f(x)$ for all inputs in $\{x : f(x) = \Theta(q(n))\}$.

This question has been occasionally studied implicitly. In [2], Bar-Yosseff, Kumar, and Sivakumar initialized the study of graph problems in the adjacency stream model, where the graph is given by a sequence of edges (i, j) in an arbitrary order¹. In particular they studied the problem of approximating the number of triangles, giving a one-pass algorithm using $O(\frac{1}{\epsilon^3} \log \frac{1}{\delta} (1 + \frac{T_1+T_2}{T_3})^3 \log n)$ space, where T_i is the number of unordered triples containing i edges. Unfortunately, they could not show when it is better than the naive sampling algorithm (which uses $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta} (1 + \frac{T_0+T_1+T_2}{T_3}))$ space), and they asked this as an open problem. They also gave an $\Omega(n^2)$ lower bound for general graph, by reducing the problem at one extreme end (distinguishing between graphs with no triangle *vs.*

¹ They also proposed the incidence stream model, where each item in the stream is a vertex with all its neighbors. In this paper we only study the adjacency stream model.

with one triangle) to the communication complexity of some Boolean function. They then ask as another open problem for a lower bound in terms of T_1, T_2, T_3 [2]. Jowhari and Ghodsi [13] later proved a lower bound of $\Omega(n/T_3)$. In this paper we will show that their algorithm is always asymptotically worse than the naive sampling algorithm (for any graph) by proving $\left(1 + \frac{T_1+T_2}{T_3}\right)^3 \geq \Omega\left(1 + \frac{T_0+T_1+T_2}{T_3}\right)$ using algebraic graph theoretical arguments. Also, we prove a lower bound of $\min\{\Omega(n^3/T_3), \Omega(n^2)\}$, which matches the naive sampling algorithms, and the proof is much simpler than the previous (weaker) one [2]. It should be noted that subsequent papers [13, 5] improve the upper bound and finally [5] achieve $O\left(\frac{1}{\epsilon^2} \log \frac{1}{\delta} \cdot \left(1 + \frac{T_1+T_2}{T_3}\right)\right)$. So our lower bound does not mean that the naive sampling algorithm is always the best, but that it is the best if the algorithm aims at dealing with all the graphs with T_3 in the known range.

For the problem of computing the maximal number of common neighbors, previously Buchsbaum, Giancarlo and Westbrook [4] gave a lower bound of $\Omega(n^{3/2}\sqrt{c})$ to compute the exact value, where c is the max number of the common neighbors. In this paper, after observing a matching upper bound, we consider the approximation version of the problem, showing that approximating the number needs $\tilde{\Theta}(n^{3/2}/\sqrt{c})$ space. Compared to the triangle counting example where the space complexity $\Theta(\min\{n^3/T_3, n^2\})$ drops from the maximum possible value ($\Theta(n^2)$) to the minimum possible value (constant), in this common neighbor example, the space complexity drops from some large value $\Theta(n^{3/2})$ to some small value $\Theta(n)$. Note that the $\tilde{\Theta}(n)$ space capacity is a well-studied model (called the semi-stream model) for graph problems, which is interesting [9] partly because $\tilde{\Theta}(n)$ is affordable in some Internet applications but higher space is not.

Not surprisingly, there are also many other problems whose space complexity, though first proved by considering extreme inputs, remains hard even if the computed quantity is not at extreme. We will give a simple example in this category too.

2 Preliminaries and Definitions

We say that an algorithm \mathcal{A} (ϵ, δ) -approximates the function f on input x if $\Pr[|\mathcal{A}(x) - f(x)| \leq \epsilon f(x)] \geq 1 - \delta$. In this paper, we will think of ϵ and δ as small constants. A graph $G = (V, E)$ is given in the adjacency streaming model if the input is a sequence of edges $(i, j) \in E$ in an arbitrary order.

2.1 Formulation of the Notion

The most naive way to formulate the notion in Section 1 is to define the space complexity $s(n, q)$ to be the minimum space needed to compute the quantity

² A lower bound in terms of all T_1, T_2 and T_3 does not seem quite justified: after all, for an unknown given graph, we do not know what T_i 's are. But a lower bound in terms of mere T_3 is well-justified for the two reasons mentioned earlier.

$f(x)$ for all those x satisfying $f(x) = q$. However, this is obviously a useless definition because if we already know that $f(x) = q$ then we do not need any computation. Thus we need to be a little more careful about the definition.

Definition 1. *A function f has stream space complexity $\Theta(s(n, q(n)))$ if for any constants $c_2 > c_1 > 0$, the best algorithm (ϵ, δ) -approximating f on any input in $\{x : c_1q(n) \leq f(x) \leq c_2q(n)\}$ uses space $\Theta(s(n, q(n)))$.*

Several comments are in order. First, as mentioned in Section 1, we may desire that for those inputs that are not in the range, the algorithm outputs “too high” or “too low”. Actually, in most (if not all) cases, the algorithm working for Definition 1 can be easily modified (with a multiplicative constant factor cost added) such that for any constants d_1 and d_2 with $c_1 < d_1 < d_2 < c_2$ and $d_2 - d_1 \geq 2\epsilon$, the algorithm has the following additional property: it outputs “too low” if $f(x) < d_1q(n)$ and “too high” if $f(x) > d_2q(n)$; for those inputs x with $c_1q(n) \leq f(x) \leq d_1f(x)$ or $d_2q(n) \leq f(x) \leq c_2f(x)$, either an ϵ -approximation or a “too low/high” is considered correct. Second, distinguishing between $f(x) \leq (1 - \epsilon)cq$ and $f(x) \geq (1 + \epsilon)cq$ with success probability $1 - \delta$ is clearly a relaxation of the above task for any constant c (since we can let $d_1 = (1 - \epsilon)c$ and $d_2 = (1 + \epsilon)c$). The lower bounds showed in this paper apply to this easier task.

A basic fact that will be used in the proofs is as follows. The problem Index is a streaming problem where the input is an n -bit string x followed by an index $i \in [n]$, and the task is to output x_i with success probability at least $1 - \delta$.

Fact 1. *The Index problem needs $(1 - 2\delta)n$ bits of memory.*

A generalization of the fact is to consider k bits instead of just one bit. In the problem k -Index, the input is an n -bit string x followed by k indices $i_1, \dots, i_k \in [n]$. The task is to distinguish between “ $x_{i_j} = 1, \forall j = 1, \dots, i_k$ ” and “ $x_{i_j} = 0, \forall j = 1, \dots, i_k$ ” with success probability at least $1 - \delta$.

Fact 2. *The k -Index problem needs $(1 - 2\delta)n/k$ bits of memory.*

This is easy to see by repeating each bit in Fact 1 k times. Also, a simple random sampling argument shows an $O(\frac{n}{k} \log \frac{1}{2\delta})$ upper bound for the number of memory cells.

3 Three Types of Dependence of the Space Complexity on the Computed Quantity

In this section, we will show three types of dependence of space complexity $s(n, q(n))$ on $q(n)$. In Section 3.1, we show a dependence which is the strongest possible: as $q(n)$ goes from one end (constant) to the other ($\Theta(n^3)$), the space complexity $s(n, q(n))$ drops from the maximal possible value ($\Theta(n^2)$) to the minimal possible value (constant). In Section 3.2, we show a weaker dependence: $s(n, q(n))$ drops from $\Theta(n^{3/2})$ to $\Theta(n)$. In Section 3.3, we show one example in which $s(n, q(n))$ is independent of $q(n)$.

3.1 Strong Dependence

The first problem that we study is triangle counting: Given a graph in the adjacency streaming model, (ϵ, δ) -approximate the number of triangles in the graph. Recall that T_i is the number of unordered triples of vertices with i edges, and thus T_3 is the number of triangles. The following theorem gives lower bounds that match the naive upper bounds: $O(n^3/T_3)$ for $T_3 \geq \frac{n}{3(1+\epsilon)}$ (by random sampling) and $O(n^2)$ space for $T_3 < \frac{n}{3(1+\epsilon)}$ (by storing all the edges).

Theorem 1. *Any streaming algorithm distinguishing between $T_3 \leq (1 - \epsilon)t$ and $T_3 \geq (1 + \epsilon)t$ with error probability $\delta = 1/3$ needs $\Omega(n^3/t)$ space for $t \geq \frac{n}{3(1+\epsilon)}$ and $\Omega(n^2)$ space for $t < \frac{n}{3(1+\epsilon)}$.*

Proof. Consider the case $t \geq \frac{n}{3(1+\epsilon)}$ first. Let the input graph G consist of 2 parts. One part is an $(n/3, n/3)$ -bipartite graph $H = (L, R, E_H)$ (where L and R are left and right side vertex sets), and another part $J = (V_J, E_J)$ contains $n/3$ vertices. Partition L into $n/3k$ blocks $L_1, \dots, L_{n/3k}$, each of size $k = \sqrt{3(1 + \epsilon)t/n}$; similarly partition $R = R_1 \cup \dots \cup R_{n/3k}$. (See Figure [1](#)) Denote by $H_{i,j}$ the subgraph $(L_i, R_j, E_H|_{L_i \times R_j})$. Now let the stream first give the graph H , with the promise that each subgraph $H_{i,j}$ is either empty or complete. Clearly it needs $(n/3k)^2$ bits of information to specify H . We claim that the streaming algorithm needs to basically keep all these $(n/3k)^2$ bits of information in order to approximate the number of triangles in the whole graph.

Actually, we claim that for any (i, j) , by choosing the rest of the graph in an appropriate way, we can know whether $H_{i,j}$ is empty or complete with probability $1 - \delta$. Suppose we want to know whether $H_{i,j}$ is empty or complete, we let the remaining stream contain all edges in $\{(a, b) : a \in L_i \cup R_j, b \in V_J\}$. If $H_{i,j}$ is

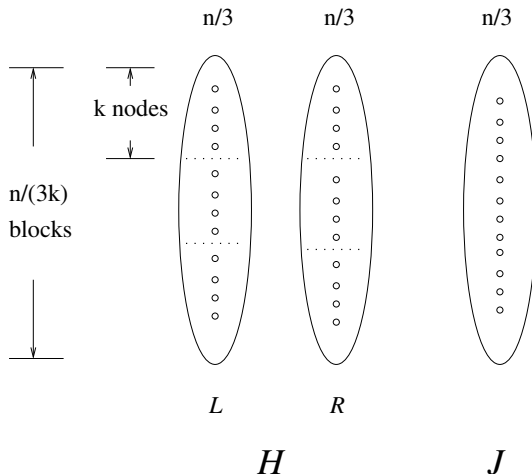


Fig. 1. A graph for illustration of the proof of triangle counting problem

complete, then G contains $k^2n/3 = (1 + \epsilon)t$ triangles; if H_i is empty, then G contains no triangle. Since the algorithm can distinguish between $T_3 \leq (1 - \epsilon)t$ and $T_3 \geq (1 + \epsilon)t$ with probability $1 - \delta$, it follows that after the first half of the stream (that specifies H) has passed, we can extract, for any (i, j) , the one bit information about whether $H_{i,j}$ is empty or complete with probability $1 - \delta$. Therefore by Fact [1](#), we need

$$(1 - 2\delta) \left(\frac{n}{3k}\right)^2 = \frac{(1 - 2\delta)n^3}{27(1 + \epsilon)t} = \Omega(n^3/t) \tag{1}$$

bits of memory.

Note that in the above analysis, we implicitly require that block size $k \geq 1$ and the number of blocks $n/(3k) \geq 1$, for which we need $\frac{n}{3(1+\epsilon)} \leq t \leq \frac{(1/2-\delta)n^3}{27(1+\epsilon)}$. For $t > \frac{(1/2-\delta)n^3}{27(1+\epsilon)}$, the lower bound is trivially true. For $t < \frac{n}{3(1+\epsilon)}$, let $k = 1$ and then the graph has $n/3$ triangles if $H_{i,j}$ is complete. Similar arguments give the lower bound of $(1/2 - \delta)n^2/9 = \Omega(n^2)$, which completes our proof.

Another open question asked in [2](#) is about the comparison of their algorithm and the naive random sampling one. Their algorithm uses $O(\frac{1}{\epsilon^3} \log \frac{1}{\delta} (1 + \frac{T_1+T_2}{T_3})^3 \log n)$ space, and they asked when the algorithm is better than the naive sampling algorithm which uses $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta} (1 + \frac{T_0+T_1+T_2}{T_3}))$ space. We now show by some simple algebraic graph theory arguments that the algorithm in [2](#) is always no better than the naive random sampling one.

Proposition 1. *For any graph we have*

$$\left(1 + \frac{T_1 + T_2}{T_3}\right)^3 \geq \Omega\left(1 + \frac{T_0 + T_1 + T_2}{T_3}\right), \tag{2}$$

Proof. First observe that $T_0 + T_1 + T_2 + T_3 = \binom{n}{3}$ and that $T_1 + 2T_2 + 3T_3 = m(n - 2)$ where m is the number of edges in the graph. The latter implies that $T_1 + T_2 + T_3 = \Theta(mn)$. Thus it is enough to prove that $T_3^2 = O(m^3)$, which can be easily done using algebraic arguments as follows. Suppose A is the adjacency matrix of the graph, then $Tr(A^3) = 6T_3$. Now notice that $Tr(A^3) = \sum_i \sum_k a_{ik} b_{ki}$ where $B = [b_{ij}]_{ij} = A^2$. It is easy to see that B is also symmetric, so $Tr(A^3) = \sum_{ik} a_{ik} b_{ik} \leq \sqrt{(\sum_{ik} a_{ik}^2)(\sum_{ik} b_{ik}^2)}$. Note that $\sum_{ik} a_{ik}^2 = \sum_{ik} a_{ik} = 2m$, and $\sqrt{\sum_{ik} b_{ik}^2} = \|B\|_2 = \|A \cdot A\|_2 \leq \|A\|_2^2 = 2m$, we thus have $T_3 \leq (2m)^{3/2}/6$, as desired.

As mentioned in Section [1](#), new algorithms are known ([13](#), [5](#)) which are better than the naive sampling algorithm for some graphs. So the above proposition is mainly of discrete math interest.

3.2 Weak Dependence

The second problem that we study is max common neighbor counting: Given a graph in the adjacency stream model, (ϵ, δ) -approximate the maximum number

of common neighbors, *i.e.* $mcn(G) = \max_{u,v} |\{w : (u,w) \in E, (v,w) \in E\}|$. In [4], it is showed that computing the *exact* value of $mcn(G)$ needs $\Omega(n^{3/2}\sqrt{c})$ space, where c is the max number of common neighbors. In this paper, after observing a matching upper bound for this exact counting problem, we consider the approximate version of the problem and show that the space complexity of approximating $mcn(G)$ is $\tilde{O}(n^{3/2}/\sqrt{c})$.

In both the upper and lower bounds, we will use the following theorem in extremal graph theory. Denote by $ex(n, H)$ is the maximal number of edges that an n -vertex graph can have without containing H as a subgraph. The upper bound is by Kovari, Sos and Turan [15], and the lower bound is by Furedi [12].

Theorem 2. $\frac{1}{2}\sqrt{tn}^{3/2} - O(n^{4/3}) \leq ex(n, K_{2,t+1}) \leq \frac{1}{2}\sqrt{tn}^{3/2} + n/4$.

Now there is a very easy algorithm: keep all the edge information until the number of edges exceeds $\frac{1}{2}\sqrt{tn}^{3/2} + n/4$, in which case the graph contains $K_{2,t+1}$ for sure; otherwise, use the kept edge information to decide whether $mcn(G) \geq t$.

Now we give the algorithm to approximate $mcn(G)$ as in the Algorithm **Approx-mcn(G)** box. Its analysis is given by the theorem below.

Algorithm **Approx-mcn(G)**
 Input: a data stream of edges $(i, j) \in E$ of a graph G in arbitrary order, two constants $a \in (0, 1)$ and $b > 1$.
 Output: an (ϵ, δ) -approximate of $mcn(G)$ if $mcn(G) \in [ac, bc]$.

1. Use a counter to count the total number m of edges. Stop and output “ $mcn(G) > bc$ ” if $m > M \equiv \frac{1}{2}n^{3/2}\sqrt{bc} + n/4$.
2. Randomly pick (with replacement) $t = \frac{1}{ae^2}(\log \frac{3n^2}{\delta})\frac{2n}{c}$ vertices v_1, \dots, v_t . Denote this multi-set by T .
3. Keep all edges incident to T . (If the number exceeds $\frac{6Mt}{\delta n}$, output FAIL.)
4. Use the kept edge information to get

$$c' = \max_{u,v \in V-T} \sum_{i=1}^t \mathbf{1}[v_i \text{ is a common neighbor of } u \text{ and } v]$$
 where $\mathbf{1}[\phi]$ is the indicator variable for the event ϕ .
5. Output $c'n/t$ as estimate to $mcn(G)$. If $c' = 0$, output $mcn(G) < ac$.

Theorem 3. For any c and any constants $a \in (0,1)$ and $b > 1$, Algorithm **Approx-mcn(G)** (ϵ, δ) -approximates $mcn(G)$ for those G with $mcn(G) \in [ac, bc]$, and the algorithm uses space $O(n^{3/2} \log^2 n/\sqrt{c})$.

Proof. First it is obvious that if the number of edge exceeds M which is larger than $ex(n, K_{2,bc})$, then $mcn(G) > bc$ for sure. Now consider $m = |E| < M$. By Markov’s Inequality, the total degree of vertices in T is at most

$$\frac{3}{\delta} \frac{2mt}{n} \leq \frac{6Mt}{\delta n} = O\left(\frac{n^{3/2}(\log n + \log \frac{1}{\delta})}{\epsilon^2 \delta \sqrt{c}}\right) \tag{3}$$

with probability $1 - \delta/3$. Now assume $mcn(G) = s \in [ac, bc]$, then $\exists u_0, v_0$ sharing a set S_0 of s common neighbors. Fix u_0, v_0 and S_0 . Since

$$\Pr[u_0 \in T \text{ or } v_0 \in T] \leq 2t/n \leq \delta/3 \tag{4}$$

if $c \geq \frac{12}{a\delta\epsilon^2} \log \frac{3n^2}{\delta}$. Let $X_i(u, v)$ be the indicator random variable for the event “the i -th vertex picked is a common neighbor of u and v ”, and let $X(u, v) = \sum_{i=1}^t X_i(u, v)$. Then under the condition that $u_0, v_0 \notin T$, we have $X(u_0, v_0) \leq c'$. Now by Chernoff’s bound,

$$\Pr\left[X(u_0, v_0) < \frac{(1 - \epsilon)ts}{n}\right] < e^{-\frac{\epsilon^2 ts}{2n}} \leq e^{-\frac{\epsilon^2 t ac}{2n}} = \delta/(3n^2). \tag{5}$$

Therefore, $\Pr[c'n/t < (1 - \epsilon)s] \leq \delta/(3n^2) < \delta/3$. On the other hand, by the definition of c' , we have

$$\Pr[c'n/t > (1 + \epsilon)s] = \Pr[c' > (1 + \epsilon)st/n] \tag{6}$$

$$= \Pr[\exists u, v \in V - T, \text{ s.t. } X(u, v) > (1 + \epsilon)st/n] \tag{7}$$

$$\leq n^2 \cdot \Pr[X(u, v) > (1 + \epsilon)st/n \mid u, v \notin T] \tag{8}$$

$$\leq n^2 \cdot \delta/(3n^2) = \delta/3. \tag{9}$$

Putting all things together, the algorithm outputs an ϵ -approximation with probability at least $1 - \delta$.

The analysis of space that the algorithm uses is as follows. It needs $O(\log n)$ to store a vertex v or an edge (u, v) , and the algorithm needs to store t vertices and $6Mt/(\delta n)$ edges. Step 4 is space efficient since we can reuse space to check each pair (u, v) . Thus the total number of bits used in the algorithm is $O(\frac{Mt}{\delta n} \log n) = O\left(\frac{n^{3/2}(\log n + \log \frac{1}{\delta})}{\epsilon^2 \delta \sqrt{c}} \log n\right)$, which is $O\left(\frac{n^{3/2} \log^2 n}{\sqrt{c}}\right)$ if ϵ and δ are constants.

We can also prove a matching lower bound as follows.

Theorem 4. *Distinguishing between $mcn(G) \leq (1 - \epsilon)c$ and $mcn(G) \geq (1 + \epsilon)c$ with small constant error probability δ needs $\Omega(n^{3/2}/\sqrt{c})$ space for small constant ϵ (say $\epsilon = 0.1$).*

Proof. Consider the extremal graph H with $n - (1 - \epsilon)c - 1$ vertices and no $K_{2, (1 - \epsilon)c + 1}$ as a subgraph. For each vertex, partition its neighbors into subsets of size $2\epsilon c$, with possibly one subset of smaller size. The total number of edges in the regular (*i.e.* not smaller) subsets is at least

$$\sqrt{(1 - \epsilon)c(n - (1 - \epsilon)c - 1)^{3/2}}/2 - (n - (1 - \epsilon)c - 1)2\epsilon c - O(n^{4/3}) = \Omega(\sqrt{cn}^{3/2}) \tag{10}$$

if $\epsilon \leq 0.1$. Now consider the graph G with n vertices, $n - (1 - \epsilon)c - 1$ of which are for the graph H , and the rest $(1 - \epsilon)c + 1$ vertices are denoted by u and S with $|S| = (1 - \epsilon)c$. (See Figure 2)

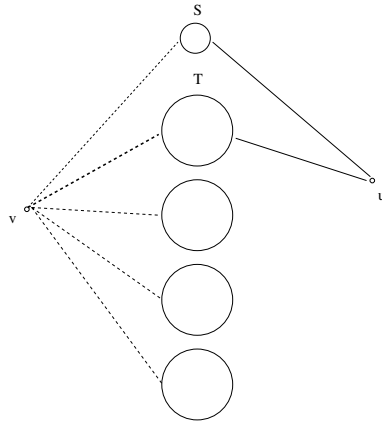


Fig. 2. A graph for illustration of the proof of max common neighbor counting problem

We will use Fact 2 to show the lower bound. Let the streaming first provide (a subgraph of) H , where for any $v \in H$ and each of its $2\epsilon c$ -subsets T , the edges from v to T may all exist or all not. Fix the content of the memory of the algorithm. Then for any fixed $v \in H$ and any of its $2\epsilon c$ -subsets T , we can know whether v and T are connected or not by providing the remaining stream (and running the streaming algorithm) in the following way. Connect v and S , u and $S \cup T$. Now note that if there is no edge between v and T , then an easy case-by-case study shows that there are no two vertices in the whole graph G sharing $(1 - \epsilon)c + 1$ common neighbors. If, on the other hand, v connects to all points in T , then clearly u and v share $(1 + \epsilon)c$ common neighbors. Thus if we can distinguish the $mcn(G) \leq (1 - \epsilon)c$ and $mcn(G) \geq (1 + \epsilon)c$ with probability $1 - \delta$, then we can distinguish between the two cases that the edges between v and T all exist and all of them do not exist with success probability $1 - \delta$, which need $\Omega(n^{3/2} \sqrt{c})/2\epsilon c = \Omega(n^{3/2}/\sqrt{c})$ space by Fact 2.

3.3 Independence

Not surprisingly, there are also many problems whose hardness is not due to the extreme case in nature, though the previous lower bounds were proved by considering the extreme cases. We just mention one simple example here to end this section. The problem is to estimate the distance between two vertices on a graph: For two fixed vertices u, v on a graph G which is given in the adjacency stream model, (ϵ, δ) -approximate $d(u, v)$, the distance between u and v on the graph G . It is not hard to see that distinguishing between $d(u, v) \leq 3$ and $d(u, v) \geq n/2$ needs $\Omega(n)$ bits of memory. Actually, consider the graph consisting of two parts. One is a $n/2$ -long path connecting u and v , and another part contains $n/4 - 1$ disjoint edges $(u_1, v_1), \dots, (u_{n/4-1}, v_{n/4-1})$. We first stream in these two parts, but each edge (u_i, v_i) may or may not exist. Then to know whether a particular edge (u_i, v_i) exists or not, we connect (u, u_i) and (v, v_i) . If

(u_i, v_i) exists, then the $d(u, v) = 3$; otherwise it is $n/2$. Thus we need $n/4 - 1$ bits of memory to distinguish these two cases. Note that in [10], a $2t + 1$ spanner is constructed using $O(tn^{1+1/t} \log^2 n)$ space thus the graph distance problem can be approximated up to a factor of $2t + 1$ by using the same amount of space, which implies that the $\Omega(n)$ lower bound is almost optimal for large constant approximation.

4 Discussions

Previous research on streaming algorithms mainly focused on designing space-efficient algorithms for important tasks; usually, log or even constant space complexity is desired. There may be more problems that, though may be very important in practical applications, did not get well studied theoretically simply because a high lower bound can be easily shown (by considering extreme inputs). This paper studies some problems which are hard for general case but easy if the computed quantity is within some range that we care about and/or the practical data are actually in.

Clearly, the same question can be asked for general algorithms. And within the domain of streaming algorithms, the problems studied in this paper happen to be those on graphs, but we believe that there are many more other problems having the same interesting phenomena.

Acknowledgement

The author thanks Sanjeev Arora, Moses Charikar, Yaoyun Shi and Martin Strauss for listening to the results and giving valuable comments.

References

- [1] Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences* 58(1), 137–147 (1999)
- [2] Bar-Yossef, Z., Kumar, R., Sivakumar, D.: Reductions in streaming algorithms, with an application to counting triangles in graphs. In: *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pp. 623–632. ACM Press, New York (2002)
- [3] Bar-Yossef, Z., Jayram, T., Kumar, R., Sivakumar, D.: Information statistics approach to data stream and communication complexity. In: *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pp. 209–218. IEEE Computer Society Press, Los Alamitos (2002)
- [4] Buchsbaum, A., Giancarlo, R., Westbrook, J.: On finding common neighborhoods in massive graphs. *Theoretical Computer Science* 299, 707–718 (2003)
- [5] Buriol, L., Frahling, G., Leonardi, S., Marchetti-Spaccamela, A., Sohler, C.: Counting Triangles in Data Streams. In: *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 253–262. ACM Press, New York (2006)

- [6] Chakrabarti, A., Khot, S., Sun, X.: Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In: Proceedings of the 18th IEEE Conference on Computational Complexity, pp. 107–117. IEEE Computer Society Press, Los Alamitos (2003)
- [7] Coppersmith, D., Kumar, R.: An improved data stream algorithm for frequency moments. In: Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 151–156. ACM Press, New York (2004)
- [8] Derenyi, I., Palla, G., Vicsek, T.: Clique percolation in random networks. *Physical Review Letters* 94, 160–202 (2005)
- [9] Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., Zhang, J.: On graph problems in a semi-streaming model. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 531–543. Springer, Heidelberg (2004)
- [10] Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., Zhang, J.: Graph Distances in the Streaming Model: The Value of Space. In: Proceedings of the 16th Symposium on Discrete Algorithms (SODA), pp. 745–754 (2005)
- [11] Feigenbaum, J., Kannan, S., Strauss, M., Viswanathan, M.: An approximate L1 difference algorithm for massive data streams. In: Proceedings of the 40th Annual Symposium on Foundations of Computer Science, pp. 501–511 (1999)
- [12] Füredi, Z.: New asymptotics for bipartite Turan numbers. *Journal of Combinatorial Theory, Series A* 75, 141–144 (1996)
- [13] Jowhari, H., Ghodsi, M.: New Streaming Algorithms for Counting Triangles in Graphs. In: Proceedings of the Eleventh International Computing and Combinatorics Conference, pp. 710–716 (2005)
- [14] Indyk, P., Woodruff, D.: Optimal approximations of the frequency moments of data streams. In: Proceedings of the 37th ACM Symposium on Theory of Computing, pp. 202–208. ACM Press, New York (2005)
- [15] Kovari, T., Sos, V.T., Turan, P.: On a problem of K. Zarankiewicz. *Colloq. Math.*, vol. 3, pp. 50–57 (1954)
- [16] Muthukrishnan, S.: *Data Streams: Algorithms and Applications*. Roundations and Trends in Theoretical Computer Science, vol. 1(2), pp. 117–236 (2005)
- [17] Saks, M., Sun, X.: Space lower bounds for distance approximation in the data stream model. In: Proceedings on 34th Annual ACM Symposium on Theory of Computing, pp. 360–369. ACM Press, New York (2002)
- [18] Shi, X., Adamic, L., Strauss, M.: Networks of strong ties. *Physica A* 378(1), 33–47 (2007)

A Randomized Approximation Algorithm for Parameterized 3-D Matching Counting Problem*

Yunlong Liu^{1,2}, Jianer Chen^{1,3}, and Jianxin Wang¹

¹ College of Information Science and Engineering, Central South University,
Changsha 410083, P.R. China

² School of Further Education, Hunan Normal University,
Changsha 410012, P.R. China

³ Department of Computer Science Texas A&M University,
College Station, TX 77843, USA

hnsdlyl@163.com, chen@cs.tamu.edu, jxwang@mail.csu.edu.cn

Abstract. The computational complexity of counting the number of matchings of size k in a given triple set remains open, and it is conjectured that the problem is infeasible. In this paper, we present a fixed parameter tractable randomized approximation scheme (FPTRAS) for the problem. More precisely, we develop a randomized algorithm that, on given positive real numbers ϵ and δ , and a given set S of n triples and an integer k , produces a number h in time $O(5.48^{3k} n^2 \ln(2/\delta)/\epsilon^2)$ such that

$$\text{prob}[(1 - \epsilon)h_0 \leq h \leq (1 + \epsilon)h_0] \geq 1 - \delta$$

where h_0 is the total number of matchings of size k in the triple set S . Our algorithm is based on the recent improved color-coding techniques and the Monte-Carlo self-adjusting coverage algorithm developed by Karp and Luby.

1 Introduction

Counting, like decision, function and enumeration, is a kind of fundamental computation in classical complexity theory, and is an important branch in theoretical computer science. Apart from being mathematically interesting, counting is closely related to important practical applications, such as data processing, reliability analyzing, artificial intelligence, and statistical physics [13].

Counting graph matchings is a central problem in computer science and has received much attention since the seminal work of Valiant, who proved that counting the perfect matchings in a bipartite graph is $\#P$ -complete [14]. Valiant also proved that counting perfect matchings in a general graph is also $\#P$ -complete [15]. Vadhan [13] proved that the problems of counting matchings and maximal matchings are all $\#P$ -complete even restricted to planar bipartite graphs whose

* This work is supported by the National Natural Science Foundation of China (60433020) and the Program for New Century Excellent Talents in University (NCET-05-0683)

degree is bounded by a small constant or to k -regular graphs for a constant k . Recently, Flum and Grohe set up a framework for a parameterized complexity theory of counting problems and especially formulated the problem of counting parameterized graph matchings, i.e., the p -#MATCHING problem. [6]. The computational complexity of p -#MATCHING remains open and is conjectured to be intractable (i.e., #W[1]-hard) [6]. Despite the fact that all the above mentioned counting problems are intractable, many effective approaches have been proposed. Especially, there have been considerable advances in recent years in the design of efficient approximation algorithms for counting graph matchings (see, e.g., [3,5,11] for a survey) or counting parameterized graph matchings [2].

In the current paper, we are focused on a generalization of the bipartite graph matching problem, the 3-D MATCHING problem. In particular, we will study the complexity of counting parameterized 3-D matchings. We start with some definitions related to the problem.

Let A_1, A_2, A_3 be three pairwise disjoint finite symbol sets. A *triple* in $A_1 \times A_2 \times A_3$ is given as (a_1, a_2, a_3) , where $a_i \in A_i$ for $i = 1, 2, 3$. A *matching* M in a triple set S is a subset of triples in S such that no two triples in M share a common symbol. A matching is a k -*matching* if it contains exactly k triples.

Definition 1. (Parameterized) 3-D MATCHING: Given a pair (S, k) , where S is a triple set and k is an integer, either construct a k -matching in S or report that no such a matching exists.

In its general form, 3-D MATCHING is one of the six basic NP-complete problems according to Garey and Johnson [8]. In recent years, 3-D MATCHING has become a focus in the study of parameterized algorithms. A series of improved algorithms has been developed for the problem [7,9,12]. Currently the best parameterized algorithm for 3-D MATCHING is due to Liu *et al.* [12], and has its running time bounded by $O(2.77^{3k} n^{O(1)})$.

Our focus in the current paper is to count the number of k -matchings in a given triple set, defined as follows.

Definition 2. (Parameterized counting) 3-D MATCHING : Given a pair (S, k) , where S is a triple set and k is an integer, count the number of distinct k -matchings in S . Let p -#3D MATCHING denote this problem.

The p -#3D MATCHING problem is a generalization of the problem of counting parameterized matchings in bipartite graphs, and the latter has been conjectured to be infeasible [6]. Therefore, p -#3D MATCHING has no known efficient algorithms. This fact makes approximation algorithms for the problem interesting and meaningful.

2 Preliminaries

Fix three finite symbol sets A_1, A_2, A_3 . For a triple $\rho = (a_1, a_2, a_3)$ in $A_1 \times A_2 \times A_3$, denote by $\text{Val}(\rho)$ the set $\{a_1, a_2, a_3\}$, and let $\text{Val}^i(\rho) = \{a_i\}$ for $1 =$

1, 2, 3. For a collection S of triples, define $\text{Val}(S) = \bigcup_{\rho \in S} \text{Val}(\rho)$, and $\text{Val}^i(S) = \bigcup_{\rho \in S} \text{Val}^i(\rho)$, for $i = 1, 2, 3$. The symbols in $\text{Val}^i(S)$ will be called the *i-th dimension symbols* in S .

Our algorithms take the advantages of the recently developed improved algorithms for the technique of *color-coding*. For this, we briefly review the necessary terminologies and results.

Definition 3. Let F be a finite set. A *k-coloring* of F is a function mapping F to the set $\{1, 2, \dots, k\}$. A subset W of F is colored *properly* by a coloring f if no two elements in W are colored with the same color under f .

Definition 4. A collection \mathcal{C} of *k-colorings* of a finite set F is a *k-color coding scheme* if for every subset W of k elements in F , there is a *k-coloring* in \mathcal{C} that colors W properly. The *size* of the *k-color coding scheme* \mathcal{C} is the number of *k-colorings* in \mathcal{C} .

The concept of color-coding was introduced by Alon, Yuster, and Zwick [1], who proved that for a set of n elements there exists a *k-color coding scheme* whose size is bounded by $O(2^{O(k)n})$. Progress has been made recently. Chen *et al.* [4] presented a construction that gives a *k-color coding scheme* of size $O(6.4^k n)$ for a set of n elements, and developed an algorithm of running time $O(6.4^k n)$ that generates such a *k-color coding scheme*.

Theorem 1. ([4]) For any finite set F of n elements and any integer k , $n \geq k$, there is a *k-color coding scheme* \mathcal{C} of size $O(6.4^k n)$ for the set F . Moreover, such a *k-color coding scheme* can be constructed in time $O(6.4^k n)$.

Before proceeding to details, we give a brief description of the ideas of our algorithms. Let (S, k) be an input instance of the *p-3D MATCHING* problem, where S is a set of n triples. Let H be the set of all *k-matchings* in S . Our objective is to compute the value $|H|$.

Let F be the set of symbols that are in either 2nd-dimension or 3rd-dimension in S . We construct a $(2k)$ -color coding scheme $\mathcal{C} = \{f_1, \dots, f_m\}$ for the set F such that the size m of \mathcal{C} is bounded by $O(6.4^{2k} n)$. For a $(2k)$ -coloring f_i in \mathcal{C} and a matching M in S , we say that M is *properly colored* by f_i if all symbols in the 2nd and the 3rd dimensions in M are colored with distinct colors under the coloring f_i .

For $1 \leq i \leq m$, let H_i be the subset of H such that H_i consists all *k-matchings* in S that are colored properly by the $(2k)$ -matching f_i . Since \mathcal{C} is a $(2k)$ -color coding scheme for the set F , for every *k-matching* M in S , there is a $(2k)$ -coloring f_i that properly colors M . Therefore, we have $H = \bigcup_{i=1}^m H_i$.

This gives the standard UNION OF SETS problem studied by Karp, Luby, and Madras [10]: given a collection of m sets H_1, H_2, \dots, H_m , compute the value $|H|$, where $H = \bigcup_{i=1}^m H_i$. This problem has been thoroughly studied in [10]. In particular, the following theorem is proved [1].

¹ The theorem stated here is more detailed than the one presented in [10]. It is straightforward to follow the original proof in [10] to derive this version of the theorem. Therefore, the proof of the theorem is omitted.

Theorem 2. (Karp-Luby [10]) *Let H_1, \dots, H_m be m sets such that*

- (1) *for each i , the value $|H_i|$ can be computed in time t_1 ;*
- (2) *for each i , in time t_2 we can randomly pick an element in H_i with a probability $1/|H_i|$; and*
- (3) *for each i and for any $v \in H_i$, we can decide if $v \in H_i$ in time t_3 .*

Then we can construct a randomized algorithm that for two given positive real numbers ϵ and δ , and for given m sets satisfying the conditions (1)-(3), generates a number h in time $O(mt_1 + mt_2t_3 \ln(2/\delta)/\epsilon^2)$ such that

$$\text{prob}[(1 - \epsilon)|H| \leq h \leq (1 + \epsilon)|H|] \geq 1 - \delta,$$

where $H = \bigcup_{i=1}^m H_i$.

Therefore, to develop an effective randomized approximation algorithm for the p -#3D MATCHING problem, we only need to show

- (1) how to count the number $|H_i|$ of k -matchings in S that are properly colored by the $(2k)$ -coloring f_i , for each i ;
- (2) how to randomly pick, with uniform probability, a k -matching that is properly colored by the $(2k)$ -coloring f_i , for each i ; and
- (3) how to determine if a k -matching is properly colored by the $(2k)$ -coloring f_i , for each i .

In the rest of this paper, we present algorithms that solve the above three problems. These algorithms combined with Theorem 2 give an algorithm that effectively computes an approximation of the number of k -matchings in a given triple set S .

3 Dealing with Properly Colored k -Matchings

As given in the previous section, for the given input instance (S, k) of the p -#3D MATCHING problem, let f_i be a $(2k)$ -coloring of the symbols in the 2nd and the 3rd dimensions in S , and let H_i be the set of all k -matchings in S that are properly colored by f_i .

3.1 Computing the Value $|H_i|$

We present an algorithm $CM1(S, k, f_i)$ to count the distinct k -matchings in S that are properly colored by the $(2k)$ -coloring f_i , i.e., $CM1(S, k, f_i)$ computes the value $|H_i|$. Figure 1 gives the algorithm in detail, where C in the pair (C, N) denotes a color set, and N denotes the number of k -matchings whose 2nd and 3rd dimension symbols colored exactly by the color set C , and $cl(\text{Val}^i(\rho))$ denotes the color of the symbol in column- i in the triple ρ , where $i = 2, 3$.

Theorem 3. *The algorithm $CM1(S, k, f_i)$ runs in time $O(2^{2k}kn)$ and returns exactly the number of k -matchings in S that are properly colored by f_i .*

Proof. For each i , $1 \leq i \leq r$, let S_i be the set of triples in S' whose symbols in column-1 are among $\{x_1, x_2, \dots, x_i\}$. By induction on i , we first prove that

Algorithm CM1(S, k, f_i)

Input: a $(2k)$ -coloring f_i of the symbols in the 2nd and 3rd dimensions in the triple set S

Output: The number of k -matchings in S that are properly colored by f_i ;

1 remove any triples in S in which any two symbols have the same color ;

2 let the set of remaining triples be S' ;

3 let the symbols in $\text{Val}^1(S')$ be x_1, x_2, \dots, x_r ;

4 $\mathcal{Q}_{old} = \{(\emptyset, 0)\}$; $\mathcal{Q}_{new} = \{(\emptyset, 0)\}$;

5 **for** $i = 1$ **to** r **do**

5.1 **for** each pair (C, N) in \mathcal{Q}_{old} **do**

5.2 **for** each $\rho \in S'$ with $\text{Val}^1(\rho) = x_i$ **do**

5.3 **if** $C \cap \{cl(\text{Val}^2(\rho)) \cup cl(\text{Val}^3(\rho))\} = \emptyset$

5.4 **then** $\{C' = C \cup \{cl(\text{Val}^2(\rho)) \cup cl(\text{Val}^3(\rho))\}$;

5.5 **if** $N = 0$ **then** $N' = N + 1$ **else** $N' = N$;

5.6 **if** the number of colors in C' is no more than $2k$

5.7 **then if** there exists another pair (C'', N'') in \mathcal{Q}_{new} in which
 C'' is exactly the same color set as C'

5.8 **then** replace N'' in the pair (C'', N'')
with $N'' = N'' + N'$;

5.9 **else** add (C', N') to \mathcal{Q}_{new} directly ; }

5.10 $\mathcal{Q}_{old} = \mathcal{Q}_{new}$;

6 **if** there exists (C, N) in \mathcal{Q}_{old} in which C contains $2k$ colors **then** return N
else return 0 .

Fig. 1. An algorithm computing $|H_i|$

for all $h \leq k$, if S_i has t h -matchings properly colored by the same color set C containing $2h$ distinct colors, then after the i -th execution of the for-loop in step 5 of the algorithm, the super-collection \mathcal{Q}_{old} must contain a pair (C, t) .

The initial case $i = 0$ is trivial since $\mathcal{Q}_{old} = \{(\emptyset, 0)\}$.

Now consider a general value $i \geq 1$. Suppose that S_i has t h -matchings ($h \leq k$) properly colored by the same color set C . Let T be the set of triples whose symbol in the 1st dimension is x_i . Moreover, suppose that among the t h -matchings in S_i , exactly u of them are not in the set T , and $t - u$ of them are in T . By induction, the triple set $S_{i-1} = S_i - T$ has exactly u h -matchings properly colored by the color set C . Moreover, there are exactly $t - u$ $(h - 1)$ -matchings in S_{i-1} properly colored by $2(h - 1)$ colors that can be combined with the two colors in a triple in T to form the color set C . Let these sets of $2(h - 1)$ colors be C_1, \dots, C_g . By the inductive hypothesis, after the $(i - 1)$ -st execution of the for-loop in step 5, the super-collection \mathcal{Q}_{old} must contain a pair (C, u) and the pairs $(C_1, N_1), \dots, (C_g, N_g)$, where N_j denotes the number of $(h - 1)$ -matchings colored properly by the color set C_j , for $1 \leq j \leq g$. It is easy to see that $N_1 + \dots + N_g = t - u$. Therefore, during the i -th execution of the for-loop in step 5, each of the pairs $(C_1, N_1), \dots, (C_g, N_g)$ will respectively be combined with the colors of a corresponding triple in T to form the color set C . At the same time, the corresponding N_j is added to N in (C, N) . After the

i -th execution of the for-loop in step 5, the value of N in (C, N) will become $N = u + (N_1 + \dots + N_g) = u + (t - u) = t$. Therefore, the pair (C, t) is included in \mathcal{Q}_{old} by step 5.10, and thus \mathcal{Q}_{old} contains the pair (C, t) .

There are two special cases: (1) $u = 0$, it means that each h -matching that uses exactly the same color set C contains one triple in T , and it also means that $S_i - T$ does not have an h -matching that uses exactly the color set C . After the $(i - 1)$ -st execution of the for-loop in step 5, the super-collection \mathcal{Q}_{new} contains no pair (C, u) , and the pair (C, N_1) produced during the i -th execution of the for-loop in step 5 will directly be added to \mathcal{Q}_{new} by step 5.9. (2) $u = t$, it means that each h -matching that uses exactly the color set C does not contain a triple in T , and it also means that $S_i - T$ has t h -matchings which use exactly the same color set C . By the inductive hypothesis, after the $(i - 1)$ -st execution of the for-loop in step 5, the pair (C, t) will have already existed in \mathcal{Q}_{old} .

Now let $i = r$, it can be concluded that for any $h \leq k$, if the collection S_r (i.e the original collection S) contains w h -matchings properly colored by the same color set C , then after the r -th execution of the for-loop in step 5, the super-collection \mathcal{Q}_{old} must contain the pair (C, w) . In particular, if the triple set S contains t k -matchings properly colored by $2k$ colors (i.e., properly colored by the $(2k)$ -coloring f_i), then at the end of the algorithm, the super-collection \mathcal{Q}_{old} contains a pair (C, t) , where C is exactly the entire color set used by f_i .

The time complexity of the algorithm $CM1(S, k, f_i)$ can be analyzed as the following. For each $0 \leq h \leq k$ and for each color set C containing $2h$ different colors, the super-collection \mathcal{Q}_{old} keeps at most one pair (C, N) . Since there are $\binom{2k}{2h}$ different subsets of $2h$ colors over a total of $2k$ colors, the total number of pairs (C, N) in \mathcal{Q}_{old} is bounded by $\sum_{h=0}^k \binom{2k}{2h} \leq 2^{2k}$. For each i , $1 \leq i \leq k$, we examine each pair (C, N) in \mathcal{Q}_{old} in step 5.3 and check if we can construct a larger (C, N) by adding the colors of triple ρ to C . This can be done for each pair (C, N) in time $O(k)$. Step 5.7 can be done in time $O(k)$ by constructing an array of size 2^{2k} to record the status of all color sets. Although in step 5 there are double for-loops (i.e step 5 and step 5.2), the total time of these double for-loops is $O(n)$. Thus, step 5 takes time $O(2^{2k}kn)$. Since step 5 is the dominating step of the algorithm, the algorithm $CM1(S, k, f_i)$ runs in time $O(2^{2k}kn)$. \square

3.2 Random Sampling in the Set H_i

For a $(2k)$ -coloring f_i in our $(2k)$ -color coding scheme \mathcal{C} , let H_i be the set of all k -matchings in S that are colored properly by f_i . In this section, we present a sampling algorithm, which randomly picks a k -matching in H_i with a probability $1/|H_i|$. Figure 2 gives the algorithm in detail, where C in the pair (C, N, M) denotes a color set, N denotes the number of matchings colored exactly by the color set C , and M denotes one matching randomly chosen from these matchings colored by C . In step 5.8, $p(M = M')$ denotes the probability of matching M' being set to M .

Theorem 4. *The algorithm $CM2(S, k, f_i)$ runs in time $O(2^{2k}kn)$, and returns a k -matching properly colored by f_i with a probability $1/|H_i|$.*

Algorithm CM2(S, k, f_i)

Input: f_i is a $(2k)$ -coloring on the symbols in the 2nd and 3rd dimensions in S

Output: A random k -matching in S that is properly colored by f_i ;

```

1 remove any triples in  $S$  in which any two symbols have the same color ;
2 let the set of remaining triples be  $S'$  ;
3 let the symbols in  $\text{Val}^1(S')$  be  $x_1, x_2, \dots, x_r$  ;
4  $\mathcal{Q}_{old} = \{(\emptyset, 0, \emptyset)\}$ ;  $\mathcal{Q}_{new} = \{(\emptyset, 0, \emptyset)\}$ ;
5 for  $i = 1$  to  $r$  do
5.1 for each group  $(C, N, M)$  in  $\mathcal{Q}_{old}$  do
5.2   for each  $\rho \in S'$  with  $\text{Val}^1(\rho) = x_i$  do
5.3     if  $C \cap \{cl(\text{Val}^2(\rho)) \cup cl(\text{Val}^3(\rho))\} = \emptyset$ 
5.4       then  $\{C' = C \cup \{cl(\text{Val}^2(\rho)) \cup cl(\text{Val}^3(\rho))\}\}$ ;  $M' = M \cup \{\rho\}$  ;
5.5         if  $N = 0$  then  $N' = N + 1$  else  $N' = N$  ;
5.6         if the number of triples in  $M'$  is no more than  $k$ 
5.7           then if there exists another group  $(C'', N'', M'')$  in  $\mathcal{Q}_{new}$ 
                    in which  $C''$  is exactly the same color set as  $C'$ 
5.8             then{ replace  $M''$  in  $(C'', N'', M'')$  with
                     $M = \text{random}(M', M'')$  such that  $p(M = M') =$ 
                     $N'/(N' + N'')$  and  $p(M = M'') = N''/(N' + N'')$ ;
                    replace  $N''$  in  $(C'', N'', M'')$  with  $N'' = N' + N''$ ;
5.9             else add  $(C', N', M')$  to  $\mathcal{Q}_{new}$  directly ; }
5.10
5.11  $\mathcal{Q}_{old} = \mathcal{Q}_{new}$  ;
6 if there exists  $(C, N, M)$  in  $\mathcal{Q}_{old}$  where  $C$  contains  $2k$  colors then return  $M$ 
else return "no such a matching exists".

```

Fig. 2. An algorithm for randomly choosing a properly colored k -matching from S

Proof. For each i , let S_i be the set of triples in S' whose symbols in column-1 are among $\{x_1, x_2, \dots, x_i\}$. We first prove by induction on i that for all $h \leq k$, if S_i has t h -matchings properly colored by the same color set C which contains $2h$ distinct colors, then the super-collection \mathcal{Q}_{old} must contain a group (C, t, M) after the i -th execution of the for-loop in step 5 of the algorithm, where M is an h -matching randomly chosen from the previous t h -matchings with probability $1/t$. In other words, any h -matching colored by C in S_i can be randomly set to M with the same probability.

Since the algorithm $\text{CM2}(S, k, f_i)$ is similar to the algorithm $\text{CM1}(S, k, f_i)$, we will only concentrate on the difference, that is, the proof that any h -matching colored by C in S_i can be randomly set to M with the same probability.

Suppose that S_i has t h -matchings ($h \leq k$) properly colored by the same color set C and that T is the set of triples whose first symbol is x_i . Suppose that among the t h -matchings, u of them are not in T , and $t - u$ of them are in T . Then $S_{i-1} = S_i - T$ has u h -matchings properly colored by the same color set C . By the induction, after the $(i - 1)$ -st execution of the for-loop in step 5, the super-collection \mathcal{Q}_{old} contains a 3-tuple (C, u, M_0) , where M_0 is an h -matching in S_{i-1} randomly picked with a probability $1/u$. Moreover, S_{i-1} has $t - u$ $(h - 1)$ -matchings properly colored by $2(h - 1)$ colors that can be combined

with the two colors in a triple in T to form the color set C . Let these sets of $2(h - 1)$ colors be C_1, \dots, C_g . Let the number of $(h - 1)$ -matchings in S_{i-1} that use the color set C_j be $N_j, 1 \leq j \leq g$. Obviously, $N_1 + \dots + N_g = t - u$. Then by induction, after the $(i - 1)$ -st execution of the for-loop in step 5, for each $j, 1 \leq j \leq g$, the super-collection \mathcal{Q}_{old} contains a 3-tuple (C_j, N_j, M_j) , where N_j is total number of $(h - 1)$ -matchings in S_{i-1} that are properly colored by the set C_j , and M_j is an $(h - 1)$ -matching in S_{i-1} properly colored by the color set C_j and randomly picked with a probability $1/N_j$. Furthermore, let $\rho_j, 1 \leq j \leq g$, be the triple in T , whose colors when combined with C_j make the color set C .

During the i -th execution of the for-loop in step 5, according to the order assumed, the group (C, u, M) in \mathcal{Q}_{old} will firstly meet with triples in T through running g times of the for-loop in step 5.2, but it will not be modified since the condition in step 5.3 is not satisfied according to the previous assumption. While in the next for-loop in step 5.2, the group (C_1, N_1, M_1) in \mathcal{Q}_{old} will be combined with at least one triple (let it be ρ_1) in T . As a result, the probability for h -matching M_0 occurring in $(C, u + N_1, M)$ is $(1/u) * (u/(u + N_1)) = 1/(u + N_1)$, and the probability for h -matching $M_1 \cup \{\rho_1\}$ occurring in $(C, u + N_1, M)$ is $(1/N_1) * (N_1/(u + N_1)) = 1/(u + N_1)$.

After the group (C_2, N_2, M_2) in \mathcal{Q}_{old} is combined with one triple (let it be ρ_2) in T through the for-loop in step 5.2, the probability for h -matchings $M_0, M_1 \cup \{\rho_1\}$ occurring in $(C, u + N_1 + N_2, M)$ are all $(1/(u + N_1)) * ((u + N_1)/(u + N_1 + N_2)) = 1/(u + N_1 + N_2)$, and the probability for h -matching $M_2 \cup \{\rho_2\}$ occurring in $(C, u + N_1 + N_2, M)$ is $(1/N_2) * (N_2/(u + N_1 + N_2)) = 1/(u + N_1 + N_2), \dots$ Finally, after the group (C_g, N_g, M_g) in \mathcal{Q}_{old} is combined with one triple (let it be ρ_g) in T through the for-loop in step 5.2, the probability for h -matching $M_0, M_1 \cup \{\rho_1\}, M_2 \cup \{\rho_2\}, \dots, M_{g-1} \cup \{\rho_{g-1}\}$ occurring in $(C, u + N_1 + N_2 + \dots + N_{g-1} + N_g, M)$ are all $(1/(u + N_1 + \dots + N_{g-1})) * ((u + N_1 + \dots + N_{g-1})/(u + N_1 + N_2 + \dots + N_{g-1} + N_g)) = 1/(u + N_1 + N_2 + \dots + N_{g-1} + N_g) = 1/(u + t - u) = 1/t$, and the probability for h -matching $M_g \cup \{\rho_g\}$ occurring in $(C, u + N_1 + N_2 + \dots + N_{g-1} + N_g, M)$ is $(1/N_g) * (N_g/(u + N_1 + N_2 + \dots + N_{g-1} + N_g)) = 1/(u + N_1 + N_2 + \dots + N_{g-1} + N_g) = 1/(u + t - u) = 1/t$. Furthermore, $M_0, M_1 \cup \{\rho_1\}, M_2 \cup \{\rho_2\}, \dots, M_g \cup \{\rho_g\}$ are exactly h -matchings colored by color set C and they are arbitrary. Therefore, after the i -th execution of the for-loop in step 5, the super-collection \mathcal{Q}_{old} must contain a group (C, t, M) , in which M is an h -matching randomly chosen from the previous t h -matchings with probability $1/t$. If these groups in \mathcal{Q}_{old} are in other orders, the proof is similar.

Now let $i = r$, it can be concluded that for any $h \leq k$, if the collection S_r (i.e. the original collection S') contains w h -matchings properly colored by the same color set C , then at end of the algorithm, the super-collection \mathcal{Q}_{old} must contain the group (C, N, M) , in which $N = w$ and M is a k -matching randomly chosen from the previous w k -matchings with probability $1/w$.

The main difference between algorithm $CM2(S, k, f_i)$ and $CM1(S, k, f_i)$ lies in step 5.8. This step can be done by firstly randomly choosing an integer b between 1 and $N' + N''$. If $b > N'$, the matching M'' is set to M in group (C, N, M) . Otherwise, the matching M' is set to M in group (C, N, M) . Obviously, this step

can be done in time $O(1)$ and the time complexity of the rest steps is similar to that of $\text{CM1}(S, k, f_i)$. In consequence, the time complexity of the algorithm $\text{CM2}(S, k, f_i)$ is $O(2^{2k}kn)$. \square

3.3 On the Membership of the Set H_i

In this last subsection, we discuss how to decide if a k -matching is in the set H_i . In other words, we need to decide if a given k -matching M is properly colored by the $(2k)$ -coloring f_i . This is trivial: we only need to go through the symbols in the 2nd and 3rd dimensions in M and check if there are two symbols colored with the same color. This can be easily done in time $O(n)$.

Theorem 5. *For any $(2k)$ -coloring f_i in our $(2k)$ -color coding scheme \mathcal{C} , let H_i be the set of all k -matchings in S that are properly colored by f_i . We can decide for any k -matching M in S if $M \in H_i$ in time $O(kn)$.*

4 Conclusions

Putting all the discussions together, we obtain a randomized approximation algorithm for the p - \sharp 3D MATCHING problem.

Theorem 6. *There is a randomized approximation algorithm that solves the p - \sharp 3D MATCHING problem in the following sense: for two given positive real numbers ϵ and δ , and a given instance (S, k) of the p - \sharp 3D MATCHING problem, where S is a set of n triples and k is an integer, the algorithm generates a number h in time $O(5.48^{3k}n^2 \ln(2/\delta)/\epsilon^2)$ such that*

$$\text{prob}[(1 - \epsilon)h_0 \leq h \leq (1 + \epsilon)h_0] \geq 1 - \delta$$

where h_0 is the total number of k -matchings in the triple set S .

Proof. The algorithm proceeds as follows. For the given instance (S, k) of p - \sharp 3D MATCHING, let F be the set of the symbols that are in either the 2nd or the 3rd dimension in S . Obviously, $|F| \leq 2n$. The algorithm then, using Theorem 1, constructs a $(2k)$ -color coding scheme \mathcal{C} for the set F in time $O(6.4^{2k}n)$ such that the size of \mathcal{C} is also bounded by $O(6.4^{2k}n)$. Let $\mathcal{C} = \{f_1, \dots, f_m\}$, where each f_i is a $(2k)$ -coloring of the set F and $m = O(6.4^{2k}n)$. Now for each i , $1 \leq i \leq m$, we define H_i to be the set of all k -matchings that are properly colored by the $(2k)$ -coloring f_i . By Theorem 3, for each i , we can compute the number $|H_i|$ of k -matchings that are properly colored by f_i in time $t_1 = O(2^{2k}kn)$. By Theorem 4, for each i , we can randomly pick in time $t_2 = O(2^{2k}kn)$ a k -matching in H_i with a probability $1/|H_i|$. Finally, by Theorem 5, for any k -matching M in S and for any i , we can decide if M is in H_i , i.e., if M is properly colored by f_i in time $t_3 = O(kn)$. Combining all these with Theorem 2, we obtain an algorithm that for any given positive real numbers ϵ and δ , generates a number h in time

$O(6.4^{2k}n + mt_1 + mt_2t_3 \ln(2/\delta)/\epsilon^2)$ (the first term is for the construction of the $(2k)$ -color coding scheme \mathcal{C}) such that

$$\text{prob}[(1 - \epsilon)h_0 \leq h \leq (1 + \epsilon)h_0] \geq 1 - \delta$$

where $h_0 = |\bigcup_{i=1}^m H_i|$ is the total number of k -matchings in the triple set S . By replacing t_1 by $O(2^{2k}kn)$, t_2 by $O(2^{2k}kn)$, and t_3 by $O(kn)$, we conclude that the running time of the algorithm is $O(5.48^{3k}n^2 \ln(2/\delta)/\epsilon^2)$. \square

According to the literature [210], a randomized algorithm Φ_Q for a counting problem Q is a *fully polynomial time randomized approximation scheme* (FPRAS) if for any instance x of Q , and any positive real numbers ϵ and δ , the algorithm Φ_Q runs in time polynomial in $|x|$, $1/\epsilon$, and $\log(1/\delta)$, and produces a number h such that

$$\text{prob}[(1 - \epsilon)h_0 \leq h \leq (1 + \epsilon)h_0] \geq 1 - \delta,$$

where h_0 is the solution to the instance x . There have been some very interesting results in the line of research in this direction. For example, although the problem of counting the number of satisfying assignments for a DNF formula is $\#P$ -complete [15], the problem has a very nice FPRAS [10].

Arvind and Raman [2] generalized the concept of FPRAS and proposed the concept of *fixed parameter tractable randomized approximation scheme* (FPTRAS). A parameterized counting problem Q has an FPTRAS if for any instance (x, k) of Q , and any positive real numbers ϵ and δ , the algorithm Φ_Q runs in time $f(k)g(|x|, \epsilon, \delta)$, where f is a fixed recursive function and $g(|x|, \epsilon, \delta)$ is a polynomial of $|x|$, ϵ , and $\log(1/\delta)$, and produces a number h such that

$$\text{prob}[(1 - \epsilon)h_0 \leq h \leq (1 + \epsilon)h_0] \geq 1 - \delta,$$

where h_0 is the solution to the instance (x, k) . Arvind and Raman [2] have shown that there are a number of interesting problems that have FPTRAS. In terms of this terminology, the algorithm presented in the current paper is an FPTRAS for the problem p - $\#3D$ MATCHING, which adds another FPTRAS problem to the literature.

References

1. Alon, N., Yuster, R., Zwick, U.: Color-coding. *Journal of the ACM* 42, 844–856 (1995)
2. Arvind, V., Raman, V.: Approximation algorithms for some parameterized counting problems. In: Bose, P., Morin, P. (eds.) *ISAAC 2002*. LNCS, vol. 2518, pp. 453–464. Springer, Heidelberg (2002)
3. Bayati, M., Gamarnik, D., Katz, D., Nair, C., Tetali, P.: Simple deterministic approximation algorithms for counting matchings. In: *Proc. 39th Symp. on Theory of Computation (STOC 07)* (to appear)
4. Chen, J., Lu, S., Sze, S.-H., Zhang, F.: Improved algorithms for path, matching, and packing problems. In: *Proc. 18th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA 07)*, pp. 298–307. ACM Press, New York (2007)

5. Chien, S.: A determinant-based algorithm for counting perfect matchings in a general graph. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 04), pp. 728–735. ACM Press, New York (2004)
6. Flum, J., Grohe, M.: The parameterized complexity of counting problems. *SIAM J. Comput.* 33(4), 892–922 (2004)
7. Fellows, M.R., Knauer, C., Nishimura, N., Ragde, P., Rosamond, F., Stege, U., Thilikos, D., Whitesides, S.: Faster Fixed-parameter tractable algorithms for matching and packing problems. In: Albers, S., Radzik, T. (eds.) *ESA 2004*. LNCS, vol. 3221, pp. 311–322. Springer, Heidelberg (2004)
8. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman (1979)
9. Koutis, I.: A faster parameterized algorithm for set packing. *Information processing letters* 94, 7–9 (2005)
10. Karp, R., Luby, M., Madras, N.: Monte-Carlo Approximation Algorithms for Enumeration Problems. *Journal of Algorithms* 10, 429–448 (1989)
11. Sankowski, P.: Alternative algorithms for counting all matchings in graph. In: Alt, H., Habib, M. (eds.) *STACS 2003*. LNCS, vol. 2607, pp. 427–438. Springer, Heidelberg (2003)
12. Liu, Y., Lu, S., Chen, J., Sze, S.-H.: Greedy localization and color-coding: improved matching and packing algorithms. In: Bodlaender, H.L., Langston, M.A. (eds.) *IWPEC 2006*. LNCS, vol. 4169, pp. 84–95. Springer, Heidelberg (2006)
13. Vadhan, S.P.: The complexity of counting in sparse, regular, and planar graphs. *SIAM J. Comput.* 31(22), 398–427 (2002)
14. Valiant, L.: The complexity of enumeration and reliability problems. *SIAM J. Comput.* 8(3), 410–421 (1979)
15. Valiant, L.: The complexity of computing the permanent. *Theoretical Computer Science* (8), 189–201 (1979)

Optimal Offline Extraction of Irredundant Motif Bases (Extended Abstract)

Alberto Apostolico* and Claudia Tagliacollo**

Georgia Institute of Technology & Università di Padova

Abstract. The problem of extracting a basis of irredundant motifs from a sequence is considered. In previous work such bases were built incrementally for all suffixes of the input string s in $O(n^3)$, where n is the length of s . Faster, non-incremental algorithms have been based on the landmark approach to string searching due to Fischer and Paterson, and exhibit respective time bounds of $O(n^2 \log n \log |\Sigma|)$ and $O(|\Sigma|n^2 \log^2 n \log \log n)$, with Σ denoting the alphabet. The algorithm by Fischer and Paterson makes crucial use of the FFT, which is impractical with long sequences.

The algorithm presented in the present paper does not need to resort to the FFT and yet is asymptotically faster than previously available ones. Specifically, an off-line algorithm is presented taking time $O(|\Sigma|n^2)$, which is optimal for finite Σ .

Keywords and Phrases: Design and Analysis of Algorithms, Pattern Matching, Motif Discovery, Irredundant Motif, Basis.

1 Introduction

The extraction from a sequence or sequence ensemble of recurrent patterns consisting of intermixed sequences of solid characters and wildcards finds multiple applications in domains ranging from text processing to computational molecular biology (see, e.g., [11]). Like with most other problems of pattern discovery, the process is often beset by the number of candidates to be considered, which in this particular case can grow exponentially with the input size. Beginning with [8,10], notions of pattern maximality or saturation have been formulated that prove capable of alleviating this problem. This is achieved by the algebraic-flavored notion of a basis, a compact subset of the set of all patterns the elements

* Corresponding author. Dipartimento di Ingegneria dell' Informazione, Università di Padova, Padova, Italy *and* College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30318, USA. axa@dei.unipd.it Work Supported in part by the Italian Ministry of University and Research under the Bi-National Project FIRB RBIN04BYZ7, and by the Research Program of Georgia Tech.

** Work performed in part while visiting the College of Computing of the Georgia Institute of Technology.

of which can account, by suitable combination, for any other pattern in the set. The elements of the basis are called irredundant motifs, and a few algorithms have been produced to this date for the extraction of a basis from a sequence. In [2] bases are built incrementally for all suffixes of the input string s in $O(n^3)$, where n is the length of s . Faster algorithms, based on the landmark string searching algorithm by Fischer and Paterson [5], are given in [9] and [7], with respective time bounds of $O(n^2 \log n \log |\Sigma|)$ and $O(|\Sigma|n^2 \log^2 n \log \log n)$. The algorithm in [5] is based on the FFT, which is admittedly impractical for long sequences.

In this work, we design algorithms that do not make use of the FFT and yet are asymptotically faster than previously available ones. Specifically, we present here an off-line algorithm taking time $O(|\Sigma|n^2)$. In a companion paper we also present an incremental algorithm taking time $O(|\Sigma|n^2 \log n)$. The explicit description of a basis requires $\Omega(n^2)$ worst-case time and space, whence the offline algorithm described in the present paper is optimal for finite alphabets.

The paper is organized as follows. Basic definitions and properties are recaptured in the next section. Following that, we describe a basic tool used by our algorithms, which consists of a speed-up in the computation of the occurrence lists of all patterns needed in the computation of a basis. The application of these constructs to the algorithms are then discussed. The paper is self-contained and notation largely conforms to the one adopted in [3,2].

2 Preliminaries

Let Σ be a finite alphabet of *solid* characters, and let ‘ \bullet ’ $\notin \Sigma$ denote a don’t-care character, that is, a wildcard matching any of the characters in $\Sigma \cup \{\bullet\}$. A *pattern* is a string over $\Sigma \cup \{\bullet\}$ containing at least one *solid* character. We use σ to denote a generic character from Σ . For characters σ_1 and σ_2 , we write $\sigma_1 \preceq \sigma_2$ if and only if σ_1 is a don’t care or $\sigma_1 = \sigma_2$.

Given two patterns p_1 and p_2 with $|p_1| \leq |p_2|$, $p_1 \preceq p_2$ holds if $p_1[j] \preceq p_2[j]$, $1 \leq j \leq |p_1|$. We also say in this case that p_1 is a *sub-pattern* of p_2 , and that p_2 *implies* or *extends* p_1 . If, moreover, the first characters of p_1 and p_2 are matching solid characters, then p_1 is also called a *prefix* of p_2 . For example, let $p_1 = ab\bullet\bullet e$, $p_2 = ak\bullet\bullet e$ and $p_3 = abc\bullet e\bullet g$. Then $p_1 \preceq p_3$, and $p_2 \not\preceq p_3$. Note that the \preceq relation is transitive. The following operators are further introduced.

Definition 1. (\oplus) Let $\sigma_1, \sigma_2 \in \Sigma \cup \bullet$.

$$\sigma_1 \oplus \sigma_2 = \begin{cases} \sigma_1, & \text{if } \sigma_1 = \sigma_2 \\ \bullet, & \text{if } \sigma_1 \neq \sigma_2 \end{cases}$$

Definition 2. (Extended \oplus) Given patterns p_1 and p_2 , $p_1 \oplus p_2 = p_1[i] \oplus p_2[i]$, $\forall 1 \leq i \leq \min\{|p_1|, |p_2|\}$.

Definition 3. (Consensus, Meet) Given the patterns p_1, p_2 , the consensus of p_1 and p_2 is the pattern $p = p_1 \oplus p_2$. Deleting all leading and trailing don’t cares from p yields the meet of p_1 and p_2 , denoted by $[p_1 \oplus p_2]$.

For instance, $aac\bullet tgcta \oplus caact\bullet cat = \bullet a\bullet\bullet t\bullet c\bullet\bullet$, and $[aac\bullet tgcta \oplus caact\bullet cat] = a\bullet\bullet t\bullet c$. Note that a meet may be the empty word. Let now $s = s_1s_2\dots s_n$ be a sequence of n over Σ . We use suf_i to denote the suffix $s_i s_{i+1} \dots s_n$ of s .

Definition 4. (Autocorrelation) *A pattern p is an autocorrelation of s if p is the meet of s and one of its suffixes, i.e., if $p = [s \oplus suf_i]$ for some $1 < i \leq n$.*

For instance, the autocorrelations of $s = acacacacabaaba$ are: $\overline{m}_1 = s \oplus suf_2 = s \oplus suf_{11} = s \oplus suf_{14} = a$, $\overline{m}_2 = s \oplus suf_3 = acacaca\bullet a\bullet\bullet a$, $\overline{m}_3 = s \oplus suf_4 = aba$, $\overline{m}_4 = s \oplus suf_5 = acaca\bullet a$, $\overline{m}_5 = s \oplus suf_6 = s \oplus suf_9 = s \oplus suf_8 = s \oplus suf_{10} = s \oplus suf_{12} = a\bullet a$, $\overline{m}_6 = s \oplus suf_7 = aca\bullet a$.

Definition 5. (Motif) *For a sequence s and positive integer k , $k \leq |s|$, a k -motif of s is a pair (m, \mathcal{L}_m) , where m is a pattern such that $|m| \geq 1$ and $m[1]$, $m[|m|]$ are solid characters, and $\mathcal{L}_m = (l_1, l_2, \dots, l_q)$ with $q \geq k$ is the exhaustive list of the starting position of all occurrences of m in s .*

Note that both components concur to this definition: two distinct location lists correspond to two distinct motifs even if the pattern component is the same and, conversely, motifs that have different location lists are considered to be distinct. In the following, we will denote motifs by their pattern component alone, when this causes no confusion. Consider $s = abcdabcd$. Using the definition of motifs, the different 2-motifs are as follows: $m_1 = ab$ with $\mathcal{L}_{m_1} = \{1, 5\}$, $m_2 = bc$ with $\mathcal{L}_{m_2} = \{2, 6\}$, $m_3 = cd$ with $\mathcal{L}_{m_3} = \{3, 7\}$, $m_4 = abc$ with $\mathcal{L}_{m_4} = \{1, 5\}$, $m_5 = bcd$ with $\mathcal{L}_{m_5} = \{2, 6\}$ and $m_6 = abcd$ with $\mathcal{L}_{m_6} = \{1, 5\}$.

Given a motif m , a *sub-motif* of m is any motif m' that may be obtained from m by (i) changing one or more solid characters into don't care, (ii) eliminating all resulting don't cares that precede the first remaining solid character or follow the last one, and finally (iii) updating \mathcal{L}_m in order to produce the (possibly, augmented) list $\mathcal{L}_{m'}$. We also say that m is a *condensation* for any of its sub-motifs.

We are interested in motifs for which any condensation would disrupt the list of occurrences. A motif with this property has been called *maximal* or *saturated*. In intuitive terms, a motif m is maximal or saturated if we cannot make it more specific while retaining the cardinality of the list \mathcal{L}_m of its occurrences in s . More formally, in a saturated motif m no don't care of m can be replaced by a solid character that appears in all the locations in \mathcal{L}_m , nor can m be expanded by a pattern prefix or suffix without affecting the cardinality of \mathcal{L}_m .

A motif (m, \mathcal{L}_m) is redundant if m and its location list \mathcal{L}_m can be deduced from the other motifs *without* knowing the input string s . Trivially, every unsaturated motif is redundant. As it turns out, however, saturated motifs may be redundant, too. More formally:

Definition 6. *A saturated motif (m, \mathcal{L}_m) , is redundant if there exist saturated motifs (m_i, \mathcal{L}_{m_i}) $1 \leq i \leq t$, such that*

$$\mathcal{L}_m = (\mathcal{L}_{m_1} + d_1) \cup (\mathcal{L}_{m_2} + d_2) \cup \dots \cup (\mathcal{L}_{m_p} + d_t)$$

with $0 \leq d_j < |m_j|$.

Here and in the following, $(\mathcal{L} + d)$ is used to denote the list that is obtained by adding a uniform offset d to every element of \mathcal{L} . For instance, the saturated motif $m_1 = a\bullet a$ is redundant in $s = acacacacabaaba$, since $\mathcal{L}_{m_1} = \{1, 3, 5, 7, 9, 12\} = (\mathcal{L}_{m_2}) \cup (\mathcal{L}_{m_3}) \cup (\mathcal{L}_{m_4} + 1)$ where $m_2 = acac, m_3 = aba$ and $m_4 = ca\bullet a$.

Saturated motifs enjoy some special properties.

Property 1. Let (m_1, \mathcal{L}_{m_1}) and (m_2, \mathcal{L}_{m_2}) be saturated motifs. Then,

$$m_1 = m_2 \Leftrightarrow \mathcal{L}_{m_1} = \mathcal{L}_{m_2}.$$

We also know that, given a generic pattern m , it is always possible to determine its occurrence list in any sequence s . With a saturated motif m , however, it is possible in addition to retrieve the structure of m from the sole list \mathcal{L}_m in s , simply by taking:

$$m = \left[\bigoplus_{i \in \mathcal{L}_m} \text{su}f_i \right].$$

We also have:

Property 2. Let $(m_1, \mathcal{L}_{m_1}), (m_2, \mathcal{L}_{m_2})$ be motifs of s . Then,

$$m_1 \preceq m_2 \Leftrightarrow \mathcal{L}_{m_2} \subseteq \mathcal{L}_{m_1}.$$

Similarly:

Property 3. Let (m, \mathcal{L}_m) be a saturated motif of s . Then $\forall L \subseteq \mathcal{L}_m$ we have

$$m \preceq \left[\bigoplus_{k \in L} \text{su}f_k \right].$$

Let now $\text{su}f_i(m)$ denote the i th suffix of m .

Definition 7. (Coverage) *The occurrence at j of m_1 is covered by m_2 if $m_1 \preceq \text{su}f_i(m_2), j \in \mathcal{L}_{m_2} + i - 1$ for some $\text{su}f_i(m_2)$.*

For instance, $\overline{m}_6 = aca\bullet a$ with $\mathcal{L}_{\overline{m}_6} = \{1, 3, 5, 7\}$ is covered at position 5 by $\overline{m}_2 = acacaca\bullet a\bullet\bullet a, \mathcal{L}_{\overline{m}_2} = \{1, 3\}$. In fact, let m' be i th suffix of \overline{m}_3 with $i = 5$, that is, $m' = aca\bullet a\bullet\bullet a$. Then $5 \in \mathcal{L}_{\overline{m}_2} + 4$ and $\overline{m}_6 \prec m'$, which together lead to conclude that \overline{m}_6 is covered at 5 by \overline{m}_2 . An alternate definition of the notion of coverage can be based solely on occurrence lists:

Definition 8. (Coverage) *The occurrence at j of m_1 is covered by m_2 if $j \in \mathcal{L}_{m_2} + i \subseteq \mathcal{L}_{m_1}$ for some i .*

In terms of our running example, we have: $5 \in \mathcal{L}_{\overline{m}_2} + 4$ and $\mathcal{L}_{\overline{m}_2} + 4 = \{5, 7\} \subset \mathcal{L}_{\overline{m}_6} = \{1, 3, 5, 7\}$.

A maximal motif that is not redundant is called an *irredundant motif*. Hence a saturated motif (m, \mathcal{L}_m) is irredundant if the components of the pair (m, \mathcal{L}_m) cannot be deduced by the union of a number of other saturated motifs.

We use \mathcal{B}_i to denote the set of irredundant motifs in $\text{su}f_i$. Set \mathcal{B}_i is called the *basis* for the motifs of $\text{su}f_i$. In particular, \mathcal{B} is used to denote the basis of s , which coincides with \mathcal{B}_1 .

Definition 9. (Basis) *Given a sequence s on an alphabet Σ , let \mathcal{M} be the set of all saturated motifs on s . A set of saturated motifs \mathcal{B} is called a basis of \mathcal{M} iff the following hold: (1) for each $m \in \mathcal{B}$, m is irredundant with respect to $\mathcal{B} - \{m\}$, and, (2) let $\mathbf{G}(\mathcal{X})$ be the set of all the redundant maximal motifs generated by the set of motifs \mathcal{X} , then $\mathcal{M} = \mathbf{G}(\mathcal{B})$.*

In general, $|\mathcal{M}| = \Omega(2^n)$. Luckily, however, it has been established that the basis of 2-motifs has size linear in $|s|$. As will be recaptured later in the discussion, a simple proof of this fact rests on the circumstance, that all motifs in the basis are autocorrelations of the string s . Before getting to that, we discuss a crucial block of our construction, which is the efficient computation of the lists of occurrences of all meets between suffixes of s . From now on and for the remainder of this paper, treatment will be restricted to 2-motifs.

3 Searching for Pattern Occurrences

String searching, which is the problem of finding all occurrences of an assigned string into a larger text string, is one of the most battered problems of algorithmics. Among the variants of the problem, a prominent role is held by searching for *approximate* occurrences of a solid string (see, e.g., [11,6]), as well as searching for patterns with don't care of the kind considered here. A classical, $O(n \log m \log |\Sigma|)$ time solution based on the FFT was provided in 1974 in a seminal paper by Fischer and Paterson [5] that exploited the convolutive substrate of the problem. More recently, the complexity of that approach was further reduced to $O(n \log n)$ by Cole and Hariharan [4].

All the existing approaches to the extraction of bases of irredundant motifs must solve the problem of finding the occurrences of a special family of patterns with don't cares, namely, the autocorrelations of the input string s or of suffixes thereof. The incremental approach in [2] proceeds by computing those lists for consecutively increasing suffixes of each autocorrelation. This produces the basis associated with each one of the suffixes of s , at the overall cost of $O(n^3)$ time. The approaches of [9,7] compute the occurrences of autocorrelations off-line with Fischer and Paterson [5], hence at an overall cost of $O(n^2 \log n \log |\Sigma|)$. The FFT-based approach does not make use of the fact that the strings being sought are autocorrelations of the input string s . The incremental approach uses this to derive the list of occurrences of consecutive suffixes of the same autocorrelation as consecutive refinements of previously computed lists. However, none of the approaches available takes advantage of the fact, that the patterns of which the occurrences are sought *come all from the set of autocorrelations of the same string*. The analysis and exploitation of such a relationship constitutes a core contribution of the present paper. In a nutshell, if $m = suf_i \oplus suf_j$ has an occurrence at some position k in s , then this induces stringent relationships among the number of don't cares in each of the three patterns $m = suf_i \oplus suf_j$, $m' = suf_i \oplus suf_k$ and $m'' = suf_j \oplus suf_k$. The specific structure of these relationships depends on whether the alphabet is binary or larger. We examine the case of a binary alphabet first.

3.1 Binary Alphabet

Let $m = [suf_i \oplus suf_j]$ and assume an occurrence of m at k . We establish here a relationship among the number of don't cares that are found respectively in m and in the prefixes of length $|m|$ of $suf_i \oplus suf_k$ and $suf_j \oplus suf_k$, that is, $[suf_i \oplus suf_k]$ and $[suf_j \oplus suf_k]$. Such a relationship will enable us to decide whether $k \in \mathcal{L}_m$ based solely on the knowledge of those three numbers of don't cares. We use d_x to denote the number of don't cares in x and $pref_i(x)$ to denote the prefix of x of length i .

Lemma 1. *Let $m = [suf_i \oplus suf_j]$, $m' = pref_{|m|}(suf_i \oplus suf_k)$ and $m'' = pref_{|m|}(suf_j \oplus suf_k)$.*

$$k \in \mathcal{L}_m \Leftrightarrow d_m = d_{m'} + d_{m''}.$$

Proof. We show first that if m has an occurrence at k this implies the claim. Under such hypotheses, we have $i, j, k \in \mathcal{L}_m$, whence, by Property 3, also $m \preceq m' = pref_{|m|}(suf_i \oplus suf_k)$. Similarly, it must be $m \preceq m''$. Considering then homologous positions in m , m' and m'' , the following holds:

- If $m[l] = \sigma$ then, from $m \preceq m'$ and $m \preceq m''$, we get $m'[l] = m''[l] = \sigma$.
- If $m[l] = \bullet \Leftrightarrow suf_i[l] \neq suf_j[l]$, and one of the following two cases is possible:
 1. $m'[l] = \sigma \Leftrightarrow suf_i[l] = suf_k[l] \Leftrightarrow suf_j[l] \neq suf_k[l] \Leftrightarrow m''[l] = \bullet$.
 2. $m'[l] = \bullet \Leftrightarrow suf_i[l] \neq suf_k[l] \Leftrightarrow suf_j[l] = suf_k[l] \Leftrightarrow m''[l] = \sigma$.

The last one is summarized by $m[l] = \bullet \Leftrightarrow m'[l] \neq m''[l]$. Note that, since m' and m'' both result from a meet of suf_k with some other suffix of s , then $m'[l] \neq m''[l]$ implies $m'[l] = \bullet$ or $m''[l] = \bullet$.

Thus, in correspondence with every don't care of m , only one of the patterns m' and m'' will have a don't care. Since every solid character of m must also appear in homologous positions in both m' and m'' , we have that the total number of don't cares in m' and m'' equals the don't cares in m .

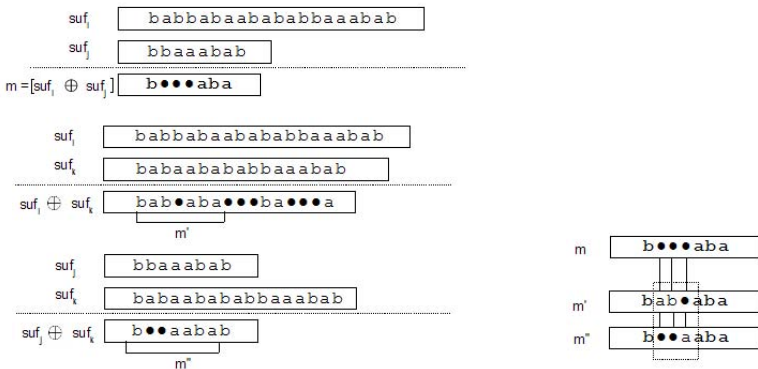


Fig. 1. Illustrating Lemma 1

To prove the converse, we show that if k is not an occurrence of m then this infringes the claimed relationship. Assume then $k \notin \mathcal{L}_m$. Hence, $\exists l$ such that $m[l] = \sigma$ and $\text{suffix}_k[l] \neq \sigma$. Since $m[l] = \sigma$, it must be $\text{suffix}_i[l] = \sigma$ and $\text{suffix}_j[l] = \sigma$, whence $m'[l] = \bullet = m''[l]$. Upon re-examining the distribution of don't cares in m' and m'' with respect to m , we have the following cases:

- $m[l] = \sigma$. This splits into:
 1. $m'[l] = \sigma \Leftrightarrow m''[l] = \sigma$.
 2. $m'[l] = \bullet \Leftrightarrow m''[l] = \bullet$.
- $m[l] = \bullet$. There is no change with respect to the first part of the proof.

We see thus that the difference with respect to the assumption $k \in \mathcal{L}_m$ is posed by some solid characters in m that become don't care in m' and m'' . Every don't care in m is balanced by corresponding don't cares in m' and m'' . However, we must now add to the equation a positive contribution that amounts to twice the number of positions of suffix_k that cause a *mismatch* with m . In other words, when $k \notin \mathcal{L}_m$ we have $d_m < d_{m'} + d_{m''}$, hence $d_m \neq d_{m'} + d_{m''}$. \square

Once the number of don't cares in every suffix of each autocorrelation of the binary string s has been tabulated, Lemma [II](#) makes it possible to decide in constant time whether or not a meet occurs at any given position of s . Tallying don't cares in every suffix of a pattern is trivially accomplished in time linear in the pattern length. Once this is done, the number of don't cares in any substring of that pattern is retrieved by a straightforward subtraction, in constant time. In conclusion, the constant time implementation of occurrence queries based on Lemma [II](#) requires only a trivial $O(n^2)$ preprocessing of s .

3.2 Alphabets with More Than 2 Symbols

The criterion offered by Lemma [II](#) can be generalized to a larger alphabet Σ by first generating, from s , $|\Sigma|$ binary instances of the problem, then handling them separately and in analogy to Lemma [II](#), and finally combining the results in a whole. To handle the instance relative to the generic $\sigma \in \Sigma$, a binary version \tilde{s} of s is built by changing every σ into a '1' and every $\sigma' \neq \sigma$ into a '0'. Upon computing any autocorrelation or meet of \tilde{s} , all 0's are replaced by don't cares, yielding what will be referred to as the corresponding *binary projection*. The overall operation will be denoted by \otimes and it is equivalent to taking the bitwise binary product or logical 'and' of the two input strings. However, separate accounting must now be kept based on the origin of each don't care. Specifically, relative to a given pattern m we keep individual tracks of:

1. the number of don't cares originating by a 1×0 , with '1' in \tilde{s} and '0' in a suffix of \tilde{s} ; this number is denoted by d_m^{10} ;
2. the number of don't cares originating by a 0×1 , denoted by d_m^{01} ;
3. the number of don't cares originating by 0×0 , denoted by q_m .

If now $m = \text{pref}_{|m|}(\text{suf}_i(\tilde{s}) \otimes \text{suf}_j(\tilde{s}))$ is one of these binary patterns, we will check its occurrence at k in analogy with Lemma 1 by comparing the don't cares in $m, m' = \text{pref}_{|m|}(\text{suf}_i(\tilde{s}) \otimes \text{suf}_k(\tilde{s}))$, and $m'' = \text{pref}_{|m|}(\text{suf}_j(\tilde{s}) \oplus \text{suf}_k(\tilde{s}))$. Only, this time we will need to distinguish among the three possible origins of the don't cares.

The following relationships among homologous positions of m, m' and m'' are readily checked.

- for any don't care of the type 00 in m , there is one of the type 00 or 01 both in m' and m'' .
- for any don't care of the type 10 in m , either:
 - m' has a don't care 10 $\Leftrightarrow m''$ has a don't care 00.
 - m' has a solid character $\Leftrightarrow m''$ has a don't care 01.
- for any don't care 01 the situation is dual, and we have one of the following:
 - m'' has a don't care 10 $\Leftrightarrow m'$ has a don't care of type 00.
 - m'' has a solid character $\Leftrightarrow m'$ has a don't care 01.
- every 1 in m has homologous occurrences both in m' and m'' .

Lemma 2. Let $m = \text{pref}_{|m|}(\text{suf}_i(\tilde{s}) \otimes \text{suf}_j(\tilde{s}))$, $m' = \text{pref}_{|m|}(\text{suf}_i(\tilde{s}) \otimes \text{suf}_k(\tilde{s}))$, $m'' = \text{pref}_{|m|}(\text{suf}_j(\tilde{s}) \otimes \text{suf}_k(\tilde{s}))$, and set

$$t = q_m - \frac{(q_{m'} + q_{m''})}{2}.$$

Then,

$$k \in \mathcal{L}_m \Leftrightarrow d_m^{10} - d_{m'}^{10} = d_{m''}^{01} - t.$$

Proof. Observe first that to every 00 don't care of m there corresponds in m' and m'' a pair of don't cares of the same or of 01 type, the latter being counted precisely by the parameter t . Moreover, by the structure of \otimes , a 00 don't care in m cannot be covered by a solid character in m' or m'' . In other words, the value of t does not depend on whether or not k is an occurrence of m .

Assume now $k \in \mathcal{L}_m$. Then, $d_m^{10} - d_{m'}^{10}$ represents the number of don't cares of m that are covered by a solid character in m' . The only such don't cares of m are of the 10 type, and the only case in which a 10 don't care may originate in m' is by having a corresponding 10 don't care in m . To any don't care covered by m' , there corresponds a 01 don't care in m'' . However, m'' will contain in general additional don't cares of 01 type, which correspond to the configuration, described earlier, where a 00 don't care of m corresponds a 01 don't care both in m' and m'' . This yields the term t in the equality.

For the second part of the proof, assume that the equality holds and yet $k \notin \mathcal{L}_m$. Imagine that, starting with an occurrence of m , one injects mismatches (that is, solid 1 characters, that are transformed into as many 0's in m' e m''). Every new mismatch causes a unit increase in $d_{m'}^{10}$, whereas d_m^{10} and $d_{m''}^{01}$ are not affected. Consequently, $t = d_{m'}^{10} + d_{m''}^{01} - d_m^{10}$ undergoes a unit increase. But this is impossible, by the invariance of t . □

Lemma 3. Let $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_{|\Sigma|}\}$ and $m_1, m_2, \dots, m_{|\Sigma|}$ the binary projections of a given meet $m = [suf_i \oplus suf_j]$.

$$k \in \mathcal{L}_m \Leftrightarrow k \in \mathcal{L}_{m_1} \cap \mathcal{L}_{m_2} \cap \dots \cap \mathcal{L}_{|\Sigma|}$$

up to a suitable shift for each list.

Proof. Since the only solid characters of m_i are precisely the occurrences of σ_i in s , we have $m_i \preceq m, \forall i = 1, \dots, |\Sigma|$. Then, $\mathcal{L}_m \subseteq \mathcal{L}_{m_i}, \forall i$, hence $\mathcal{L}_m \subseteq \mathcal{L}_{m_1} \cap \mathcal{L}_{m_2} \cap \dots \cap \mathcal{L}_{m_{|\Sigma|}}$. Assume an occurrence $l \in \mathcal{L}_{m_1} \cap \mathcal{L}_{m_2} \cap \dots \cap \mathcal{L}_{m_{|\Sigma|}}$ but $l \notin \mathcal{L}_m$. This implies the existence of a position in suf_i that causes a mismatch with a solid character, say, σ_k , of m . Consider then the projection involving σ_k . From $l \notin \mathcal{L}_m$ we get $l \notin \mathcal{L}_{m_k}$, hence $l \notin \mathcal{L}_{m_1} \cap \mathcal{L}_{m_2} \cap \dots \cap \mathcal{L}_{m_{|\Sigma|}}$. \square

Theorem 1. Let s be a string of n characters over an alphabet Σ , and m the meet of any two suffixes of s . Following an $O(|\Sigma|n^2)$ time preprocessing of s , it is possible to decide for any assigned position k whether or not k is an occurrence of m in time $O(|\Sigma|)$.

Proof. By the preceding properties and discussion. \square

4 An $O(|\Sigma|n^2)$ Off-Line Basis Computation

We are now ready to develop in full the criterion upon which our optimal algorithm is built. Recall that in order for a motif to be irredundant it must have at least one occurrence that cannot be deduced from occurrences of other motifs. In [2], such an occurrence is called *maximal* and the motif is correspondingly said to be *exposed* at the corresponding position. Clearly, every motif with a maximal occurrence is saturated. However, not every saturated motif has a maximal occurrence. In fact, the set of irredundant motifs is precisely the subset of saturated motifs with a maximal occurrence. The following known definitions and properties (see, e.g., [29]) systematize these notions and a few more important facts.

Definition 10. (Maximal occurrence) Let (m, \mathcal{L}_m) be a motif of s and $j \in \mathcal{L}_m$. Position j is a maximal occurrence for m if for no $d' \geq 0$ and $(m', \mathcal{L}_{m'})$ we have $\mathcal{L}_{m'} \subseteq (\mathcal{L}_m - d')$ with $(j - d') \in \mathcal{L}_{m'}$.

For a given $m \in \mathcal{B}$, let \mathcal{L}_m^{max} denote the list of maximal occurrences of m . The following lemma contributes an alternative definition of irredundancy, whereby the problem of identifying the basis for s translates into that of identifying the motifs with at least one maximal occurrence in s .

Lemma 4. $m \in \mathcal{B} \Leftrightarrow |\mathcal{L}_m^{max}| > 0$.

Proof. By the definition of irredundancy. \square

Lemma 5. *If $m \in \mathcal{B}$, then*

$$j \in \mathcal{L}_m^{max} \Leftrightarrow [s \oplus suf_{(max\{j,k\}-min\{j,k\})}] = m, \forall k \in \mathcal{L}_m.$$

Proof. W.l.o.g., assume $k > j$ and that $\exists k \in \mathcal{L}_m, k > j : w = [s \oplus suf_{k-j}] \neq m$. Let $w = vm'$ with $m' = [suf_j \oplus suf_k]$. Since $j, k \in \mathcal{L}_m$, it must be $m \preceq m'$, hence $m \prec m'$ from the hypothesis. Then, the occurrence at j is covered by m' (or by the motifs that cover m'). Therefore, it must be $m = m' = [suf_j \oplus suf_k], \forall k \in \mathcal{L}_m$ and then $w = vm' = vm$. Likewise, if $w \neq m$, then the occurrence at j is covered by w and thus cannot be maximal. It remains to be shown that the converse is also true, i.e., that

$$[s \oplus suf_{(max\{j,k\}-min\{j,k\})}] = m, \forall k \in \mathcal{L}_m$$

implies that j is a maximal occurrence for m . But it follows from the hypothesis that the occurrence at j cannot be covered by any motif with a list $\mathcal{L} \subset \mathcal{L}_m$, whence this occurrence is maximal. \square

Lemma 5 gives a handle to check whether a position i is a maximal occurrence for an assigned motif (m, \mathcal{L}_m) . For this, it suffices to check that $[suf_i \oplus suf_k] = m, \forall k \in \mathcal{L}_m$. We note, for future record, that this holds in particular for i equal to the smallest index in \mathcal{L}_m .

Theorem 2. *Every irredundant motif is the meet of s and one of its suffixes.*

Proof. Let $m \in \mathcal{B}$. By Lemma 4, we derive that $m \in \mathcal{B} \Leftrightarrow m$ must be exposed at some position of s . Let j be this position. It follows from Lemma 5 that for m to be exposed at j it must be $m = [s \oplus suf_{(max\{j,k\}-min\{j,k\})}], \forall k \in \mathcal{L}_m$. \square

We have thus:

Theorem 3. *The number of irredundant motifs in a string of n characters is $O(n)$.*

Proof. Immediate. \square

Lemma 6. $\sum_{m \in \mathcal{B}} |\mathcal{L}_m| < 2n$.

Proof. Let m be an irredundant motif in s , with a maximal occurrence at j . >From Lemma 5, it follows that

$$\forall k \in \mathcal{L}_m, m = [s \oplus suf_{(max\{j,k\}-min\{j,k\})}].$$

We charge each term of \mathcal{L}_m other than j to a different shift $(max\{j, k\} - min\{j, k\})$ between s and one of its suffixes, and we charge the occurrence at j to s itself. By the maximality of m , each one of the possible shifts in $\{1, 2, \dots, n - 1\}$ is only charged once, while s gets charged at most $n - 1$ times overall. \square

Lemma 6 shows that the implicit specification of the basis takes linear space. This provides a crucial pillar for an algorithm based entirely on the management of occurrence lists, as is described next.

In the off-line basis computation the only patterns at play are the autocorrelations of the input string s , since the only candidate irredundant motifs consist of the corresponding meets. The main stages of the computation are summarized as follows:

- 1 compute the occurrence lists of the autocorrelations of s ;
- 2 store these autocorrelations in a trie ;
- 3 identify the members of the basis by visiting the nodes of the trie.

In order to take advantage of Lemma 1 and its extensions, Stage 1 requires a straightforward preprocessing that consists of computing, for each $m = [s \oplus suf_i]$ the number of don't cares that are present in every suffix of m partitioned, for large alphabets, among the implied binary projections. Once this information is available, that Lemma and its extensions will support the compilation of the occurrence lists of all autocorrelations of s .

Once this is done, the collection of all autocorrelations of s are stored in a trie each node of which stores some additional information, as follows. Let us say that suf_k takes part in a autocorrelation m of s if $m = [suf_k \oplus suf_l]$ for some l . Let v be a node of the trie and denote by $index[k]$ the number of times that suf_k takes part in a autocorrelation of s that terminates at v .

Definition 11. *The index of v is $I(v) = \max_{1 \leq k \leq |s|} \{index[k]\}$.*

We assume that it is possible to know, for every node of the trie, both $I(v)$ and the suffix of s that produces it. Then, the elements of the basis are identified by comparing, for every terminal node of an autocorrelation, $I(v)$ and $|\mathcal{L}_m|$. In fact, we know that if m has a maximal occurrence at i , then $m = [suf_i \oplus suf_j], \forall j \in \mathcal{L}_m$. Therefore, if $I(v) = |\mathcal{L}_m|$ then the suffix yielding $I(v)$ is a maximal occurrence for m .

Theorem 4. *The basis of irredundant 2-motifs in a string s of n characters can be computed in time $(O|\Sigma|n^2)$.*

Proof. The dominant term in the computation is finding the occurrence lists for the autocorrelations, which was found to take time $O(|\Sigma|n^2)$. Building the trie takes $O(n^2)$ and the computation of $I(v)$ for each v can be carried out during this construction at no extra cost. The last step only calls for a constant check at each node, which charges $O(n)$ overall. \square

Since the explicit description of the $O(n)$ motifs of the basis takes space $O(n^2)$, then this algorithm is optimal for any alphabet of constant size. As it turns out, the computation of the basis may be further simplified when $|\Sigma| = 2$. This rests on some additional properties that will be described only in the full version of the paper.

5 Concluding Remarks

Several issues are still open, notable among them, the existence of an optimal algorithm for general alphabets and of an optimal incremental algorithm for alphabets of constant or unbounded size.

References

1. Apostolico, A., Galil, Z.: Pattern matching algorithms. Oxford University Press, New York (1997)
2. Apostolico, A., Parida, L.: Incremental paradigms of motif discovery. *Journal of Computational Biology* 11(1), 15–25 (2004)
3. Apostolico, A.: Pattern discovery and the algorithmics of surprise. *Artificial Intelligence and Heuristic Methods for Bioinformatics*, pp. 111–127 (2003)
4. Cole, R., Hariharan, R.: Verifying candidate matches in sparse and wildcard matching. In: STOC '02. Proceedings of the thirty-fourth annual ACM symposium on Theory of computing, pp. 592–601 (2002)
5. Fischer, M.J., Paterson, M.S.: String matching and other products. In: Karp, R. (ed.) *Proceedings of the SIAM-AMS Complexity of Computation*, Providence, R.I. American Mathematical Society, pp. 113–125 (1974)
6. Navarro, G.: A guided tour to approximate string matching. *ACM Computing Surveys* 33(1), 31–88 (2001)
7. Pelfrène, J., Abdeddaïm, S., Alexandre, J.: Extracting approximate patterns. *Journal of Discrete Algorithms* 3(2-4), 293–320 (2005)
8. Parida, L.: *Algorithmic Techniques in Computational Genomics*. PhD thesis, Department of Computer Science, New York University (1998)
9. Pisanti, N., Crochemore, M., Grossi, R., Sagot, M.-F.: Bases of motifs for generating repeated patterns with wild cards. *IEEE/ACM Trans. Comput. Biol. Bioinformatics* 2(1), 40–50 (2005)
10. Parida, L., Rigoutsos, I., Floratos, A., Platt, D., Gao, Y.: Pattern discovery on character sets and real-valued data: linear bound on irredundant motifs and an efficient polynomial time algorithm. In: *Symposium on Discrete Algorithms*, pp. 297–308 (2000)
11. Wang, J.T.L., Shapiro, B.A., Shasha, D.E.: *Pattern Discovery in Biomolecular Data: Tools, Techniques and Applications*. Oxford University Press, Oxford (1999)

Linear Algorithm for Broadcasting in Unicyclic Graphs

(Extended Abstract)

Hovhannes Harutyunyan and Edward Maraachlian

Concordia University, Department of Computer Science and Software Engineering,
Montreal, QC. H4G 1M8, Canada
haruty@cs.concordia.ca, e_maraac@cs.concordia.ca

Abstract. *Broadcasting* is an information dissemination problem in a connected network, in which one node, called the *originator*, disseminates a message to all other nodes by placing a series of calls along the communication lines of the network. Once informed, the nodes aid the originator in distributing the message. Finding the minimum broadcast time of a vertex in an arbitrary graph is NP-complete. The problem is solved polynomially only for trees. It is proved that the complexity of the problem of determining the minimum broadcast time of any vertex in an arbitrary tree $T = (V, E)$ is $\Theta(|V|)$. In this paper we present an algorithm that determines the broadcast time of any originator in an arbitrary unicyclic graph $G = (V, E)$ in $O(|V|)$ time. This, combined with the obvious lower bound, gives a $\Theta(|V|)$ solution for the problem of broadcasting in unicyclic graphs. As a byproduct, we also find a broadcast center of the unicyclic graph (a vertex in G with the minimum broadcast time).

1 Introduction

Computer networks have become essential in several aspects of modern society. The performance of information dissemination in networks often determines their overall efficiency. One of the fundamental information dissemination problems is broadcasting. Broadcasting is a process in which a single message is sent from one member of a network to all other members. Inefficient broadcasting could degrade the performance of a network seriously. Therefore, it is of a major interest to improve the performance of a network by using efficient broadcasting algorithms.

Broadcasting is an information dissemination problem in a connected network, in which one node, called the *originator*, must distribute a message to all other nodes by placing a series of calls along the communication lines of the network. Once informed, the informed nodes aid the originator in distributing the message. This is assumed to take place in discrete time units. The broadcasting is to be completed as quickly as possible, subject to the following constraints:

- Each call involves only one informed node and one of its uninformed neighbors.
- Each call requires one unit of time.

- A node can participate in only one call per unit of time.
- In one unit of time, many calls can be performed in parallel.

A *broadcast scheme* of an originator u is a set of calls that completes the broadcasting in the network originated at vertex u .

Formally, any network can be modeled as a connected graph $G = (V, E)$, where V is the set of vertices (or nodes) and E is the set of edges (or communication lines) between the vertices in graph G .

Given a connected graph $G = (V, E)$ and a message originator, vertex u , the *broadcast time* of vertex u , $b(u, G)$ or $b(u)$, is the minimum number of time units required to complete broadcasting from the vertex u . Note that for any vertex u in a connected graph G on n vertices, $b(u) \geq \lceil \log n \rceil$ since during each time unit the number of informed vertices can at most be doubled. The broadcast time $b(G)$ of the graph G is defined as $\max\{b(u) | u \in V\}$.

Determination of $b(u, G)$ or $b(u)$ for a vertex u in an arbitrary graph G is *NP*-complete [17]. The proof of *NP*-completeness is presented in [20]. Therefore, many papers have presented approximation algorithms to determine the broadcast time of any vertex in G (see [2, 3, 8, 9, 11, 12, 13, 14, 18, 19, 21]). Some of these papers give theoretical bounds on the broadcast time. Given $G = (V, E)$ and the originator u , the heuristic in [18] returns broadcast algorithm with broadcast time at most $b(u, G) + O(\sqrt{|V|})$. The best theoretical upper bound is presented in [9]. Their approximation algorithm generates a broadcast algorithm with broadcast time $O(\frac{\log(|V|)}{\log \log(|V|)})b(G)$. The heuristics [3] and [16] are the best existing heuristics for broadcasting in practice. Their performance is almost the same for commonly used interconnection networks. However, the broadcast algorithm from [16] outperforms the broadcast algorithm from [3] in three graph models from a network simulator ns-2 (see [12, 7, 22]). Also, its time complexity is $O(|E|)$, while the complexity of the algorithm from [3] is $O(|V|^2 \cdot |E|)$.

Since the problem is *NP*-complete in general, another direction for research is to design polynomial algorithms that determines the broadcast time of any vertex for a class of graphs. To the best of our knowledge, the only known result in this direction is the linear algorithm (called BROADCAST) designed in [20] which determines the broadcast time of a given vertex in any tree. They also find a broadcast scheme of a given tree $T = (V, E)$ in linear time $O(|V|)$.

In this paper we present an algorithm that determines the broadcast time of any originator in an arbitrary unicyclic graph $G = (V, E)$. The algorithm is linear, $O(|V|)$, for any originator. As a byproduct, we also find a broadcast center (a vertex in G with minimum broadcast time) of the unicyclic graph.

The paper is organized as follows. The next section will present some auxiliary results that will be necessary to understand the algorithm and prove its correctness and linearity. Section 3 presents the actual algorithm with a proof of correctness and a complexity analysis. The final section is a conclusion.

2 Definitions and Auxiliary Results

A unicyclic graph (Fig. 1) is a connected graph with only one cycle. Basically it is a tree with only one extra edge. It can also be seen as a cycle where every vertex on the cycle is the root of a tree. Denote the vertices of the cycle C_k by r_1, r_2, \dots, r_k and the tree rooted at r_i by T_i , where $1 \leq i \leq k$. We will use the following definitions and results from [20].

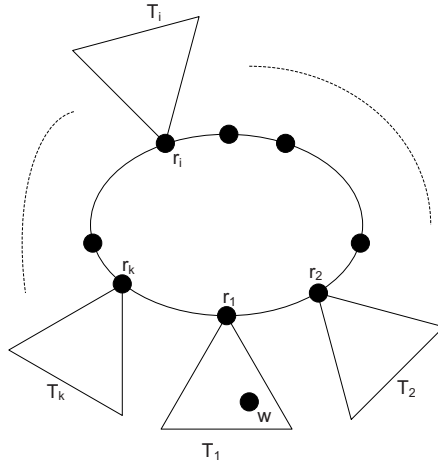


Fig. 1. A unicyclic graph where the vertices r_i , belonging to the cycle C_k , are the roots of the trees T_i for $1 \leq i \leq k$

Definition 1 ([20]). *The minimum broadcast time, $b(BC, G)$, of the graph $G = (V, E)$ is defined to be the minimum of the broadcast times of all the vertices. $b(BC, G) = \min_{u \in V} \{b(u, G)\}$.*

Definition 2 ([20]). *The broadcast center of the graph G , $BC(G)$, is defined to be the set of all vertices whose broadcast time is equal to the minimum broadcast time of the graph, $BC(G) = \{u | b(u, G) = b(BC, G)\}$.*

Theorem 1 ([20]). *Let $v \notin BC(T)$ be a vertex in a tree T such that the shortest distance from v to a vertex $x \in BC(T)$ is k . Then $b(v, T) = k + b(BC, T)$.*

Corollary 1 ([20]). *For any tree T , $BC(T)$ consists of a star with at least two vertices.*

The unicyclic graph can be converted into a tree by cutting one of the edges of the cycle $C_k = r_1, r_2, \dots, r_k, r_1$. A simple algorithm to determine the broadcast time of a vertex w in an arbitrary unicyclic graph $G = (V, E)$ would be the following:

SIMPLEBROADCASTALGORITHM(w, G):

1. Extract from G the cycle C_k and the trees T_i for $1 \leq i \leq k$ which are rooted at a vertex on C_k .
2. Cut edge (r_i, r_{i+1}) from the cycle C_k , for $i = 1, 2, \dots, k$. Denote the resulting tree by G_i .
3. Apply BROADCAST(w, G_i) for $i = 1, 2, \dots, k$ from [20] and choose the tree G_i with the minimum broadcast time $b(w, G_i)$.

The complexity of step 1 of the algorithm is $O(n)$, where $|V| = n$. The complexity of steps 2 and 3 are $O(k)$ and $O(kn)$ respectively. Thus, the total complexity of the above algorithm will be $O(kn)$, which is $O(n^2)$ in the worst case. However, $\Omega(n)$ is an obvious lower bound. In this paper we will show $\Theta(n)$ bound by describing a linear algorithm that determines the broadcast time of any vertex w in an arbitrary unicyclic graph.

Notation 1 If u and v are two vertices in graph G then (u, v) represents the edge between them, and $d(u, v)$ represents the distance between them.

Definition 3. Given trees $T_1 = (V_1, E_1), T_2 = (V_2, E_2), \dots, T_i = (V_i, E_i)$ with roots r_1, r_2, \dots, r_i respectively, the tree $T_{1,2,\dots,i} = (V, E) = T_1 \oplus T_2 \oplus \dots \oplus T_i$ is a tree where $V = V_1 \cup V_2 \cup \dots \cup V_i$ and $E = E_1 \cup E_2 \cup \dots \cup E_i \cup \{(r_1, r_2), (r_2, r_3), \dots, (r_{i-1}, r_i)\}$.

In other words, the trees T_i are connected by adding the edges $(r_1, r_2), (r_2, r_3), \dots, (r_{i-1}, r_i)$.

2.1 The Broadcast Center of the Sum of Two Trees

In this section we will describe how to find a broadcast center and calculate the minimum broadcast time of the sum of two trees.

Lemma 1. In any tree T , rooted at r , there exists a unique vertex $u \in BC(T)$, called the special broadcast center denoted as $u = SBC(T)$, such that the path joining u and r does not contain any other vertex v such that $v \in BC(T)$.

Proof. First we will show the existence of vertex u . Let v be a vertex such that $v \in BC(T)$ and let $P = v, v_1, v_2, \dots, v_k, r$ be the unique path from v to r . Due to Corollary [1] only v_1 can be in $BC(T)$. If $v_1 \in BC(T)$, then it is the required vertex, and v_1, v_2, \dots, v_k, r is the path from $SBC(T) = v_1$ to the root of T, r . If $v_1 \notin BC(T)$ then v is the required vertex and $P = v, v_1, v_2, \dots, v_k, r$ is the unique path from $SBC(T) = v$ to the root of T, r . The uniqueness of $u = SBC(T)$ immediately follows from the fact that T is a tree and no cycles are allowed in a tree.

From Lemma [1] it follows that if T_1 and T_2 are two trees with $u_1 = SBC(T_1)$ and $u_2 = SBC(T_2)$, then the path that joins u_1 and u_2 does not contain any other vertex v (different than u_1 and u_2) such that $v \in (BC(T_1) \cup BC(T_2))$.

In the remaining part of this section it is assumed that there are two trees T_1 and T_2 with roots r_1 and r_2 respectively, $T = T_1 \oplus T_2$, $u_1 = SBC(T_1)$, and $u_2 = SBC(T_2)$.

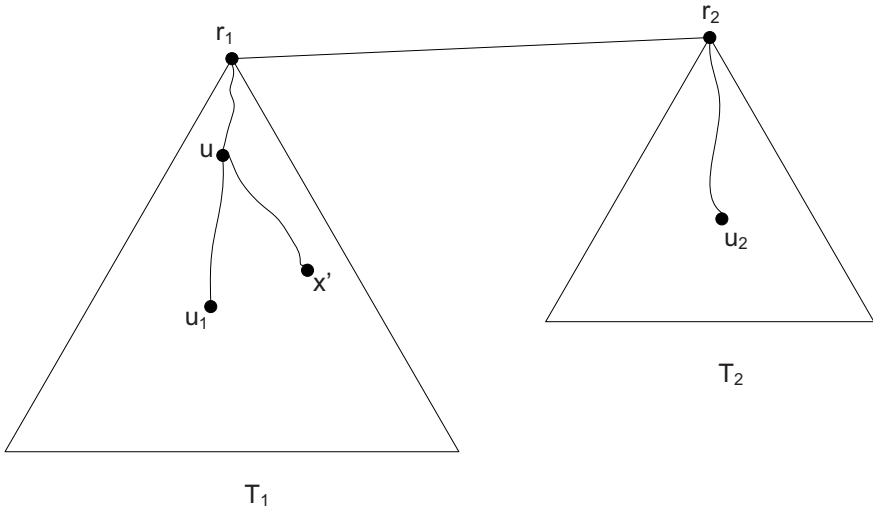


Fig. 2. Sum of two trees

Theorem 2. *There exists a vertex u such that $u \in BC(T_1 \oplus T_2)$ and u is on the path joining $u_1 = SBC(T_1)$ and $u_2 = SBC(T_2)$.*

Proof. We will prove this theorem by contradiction. Assume that there exists a vertex x' (Fig. 2) not on the path from u_1 to u_2 and such that $b(x', T) < b(u, T)$ for all vertices u on the path joining u_1 and u_2 . Without loss of generality assume that x' is in T_1 . Because T is a tree, there exists a unique path P that joins x' to r_1 . Two cases may arise:

Case 1: The path P intersects the path from u_1 to r_1 at a vertex other than u_1 . Let $u \in P$ be this intersection vertex. We denote by $t_1(x)$ the minimum time that is needed to inform all the vertices of T_1 starting at the originator x . Therefore, $b(u, T_1) = t_1(u) = d(u, u_1) + b(BC, T_1)$ and $t_1(x') = b(x', T_1) = d(x', u) + (d(u, u_1) + b(BC, T_1)) = d(x', u) + t_1(u)$. Similarly, the minimum time to inform all the vertices in T_2 starting at any originator x will be denoted by $t_2(x)$. We have $t_2(u) = d(u, r_1) + 1 + d(r_2, u_2) + b(BC, T_2) = d(u, u_2) + b(BC, T_2)$ and $t_2(x') = d(x', u) + d(u, u_2) + b(BC, T_2)$. Having calculated $t_1(x)$ and $t_2(x)$ we can calculate $b(x, T)$ since $b(x, T) = t_1(x) + 1$ if $t_1(x) = t_2(x)$ and $b(x, T) = \max\{t_1(x), t_2(x)\}$ otherwise. Since, $t_1(x') > t_1(u)$ and $t_2(x') > t_2(u)$ we conclude that $b(x', T) > b(u, T)$ which contradicts the assumption that $b(x', T) < b(u, T)$.

Case 2: The path P , joining x' and r_1 , does not intersect the path joining u_1 and r_1 . In this case the path P will merge with the path joining u_1 and r_1 at vertex u_1 . Using arguments similar to the preceding case we can show that there there is no vertex x' such that $b(x', T) < b(u_1, T)$. The details are omitted. \square

¹ Refer to Fig. 3 for an example illustrating Theorems 2 and 3.

Let u be a vertex such that $u \in BC(T_1 \oplus T_2)$. Theorem 2 confirms the existence of such a vertex on the path joining u_1 and u_2 . The position of u can be found as described in the following theorem.

Theorem 3. *Let $A = b(BC, T_1) - b(BC, T_2)$ and $B = d(r_1, u_1) + d(r_2, u_2) + 1$. Three cases may arise:*

If $B - A < 0$, then $d(u, u_1) = 0$, i.e. u coincides with u_1 .

If $A + B < 0$, then $d(u, u_1) = B$, i.e. u coincides with u_2 .

If $B - A \geq 0$ and $A + B \geq 0$, then $d(u, u_1) = \lfloor \frac{B-A}{2} \rfloor$ or $d(u, u_1) = \lceil \frac{B-A}{2} \rceil$.

Both positions of u have equal broadcast times in the tree T .

Proof. If $B - A < 0$, then we have

$$b(BC, T_1) > b(BC, T_2) + d(u_1, r_1) + 1 + d(u_2, r_2). \tag{1}$$

We will prove that $u = u_1 \in BC(T)$ by contradiction. Assume that there is another vertex u' on the path joining u_1 and u_2 such that $b(u', T) < b(u, T)$. Because of Theorem 2 we do not have to consider a vertex not belonging to the path joining u_1 and u_2 . Using the definitions of the functions $t_1(x)$ and $t_2(x)$ from above we get: $t_1(u') = d(u', u_1) + b(BC, T_1)$, $t_2(u') = d(u', r_1) + 1 + d(u_2, r_2) + b(BC, T_2)$, $t_1(u) = b(BC, T_1)$, and $t_2(u) = d(u_1, r_1) + 1 + d(u_2, r_2) + b(BC, T_2)$. Using the condition in equation 1 we get that $t_1(u) = b(BC, T_1) > t_2(u) = d(u_1, r_1) + 1 + d(u_2, r_2) + b(BC, T_2)$. Therefore, $b(u, T) = t_1(u)$. Similarly, $t_1(u') = d(u', u_1) + b(BC, T_1) > d(u', u_1) + b(BC, T_2) + d(u_1, r_1) + 1 + d(u_2, r_2)$ which implies that $t_1(u') > d(u', u_1) + b(BC, T_2) + d(u_1, r_1) + 1 + d(u_2, r_2) = 2d(u', u_1) + b(BC, T_2) + d(u', r_1) + 1 + d(u_2, r_2) = 2d(u', u_1) + t_2(u')$ which implies that $t_1(u') > t_2(u')$ and hence $b(u', T) = t_1(u')$. But we have $t_1(u') = d(u', u_1) + b(BC, T_1)$ and $t_1(u) = b(BC, T_1)$ which implies that $t_1(u') > t_1(u)$. Therefore we conclude that $b(u', T) > b(u, T)$ which contradicts the assumption.

The case $A + B < 0$ can be proved similarly.

Note that the two conditions $A + B < 0$ and $B - A < 0$ are mutually exclusive. If either one of them is satisfied the other will not be satisfied. The only remaining case is when both of them are not satisfied i.e. $A + B \geq 0$ and $B - A \geq 0$. Assume the case where $B - A$ is odd, the case if it is even can be dealt with similarly. Without loss of generality, assume that there exists a vertex $u' \in T_1$, on the path joining u_1 and u_2 , such that $b(u', T) < b(u, T)$. Two cases may arise:

Case 1: $d(u', u_1) < d(u, u_1) = \lfloor \frac{B-A}{2} \rfloor$. Since $B - A$ is assumed to be odd, $d(u, u_1) = \frac{B-A-1}{2}$. Calculating $t_1(u) = d(u, u_1) + b(BC, T_1)$ and $t_2(u) = d(u, r_1) + 1 + d(u_2, r_2) + b(BC, T_2)$ we deduce that $t_2(u) = (d(u_1, r_1) - d(u, u_1)) + 1 + d(u_2, r_2) + b(BC, T_2)$. Substituting the value of $d(u, u_1)$, and $b(BC, T_2) = b(BC, T_1) - A$ we get: $t_2(u) = [d(u_1, r_1) + 1 + d(u_2, r_2)] + b(BC, T_2) - d(u, u_1) = B + (b(BC, T_1) - A) - \frac{B-A-1}{2} = \frac{B-A+1}{2} + b(BC, T_1) = t_1(u) + 1$. Therefore, $b(u, T) = t_2(u)$. Now consider the vertex u' , $t_1(u') = d(u', u_1) + b(BC, T_1)$ and $t_2(u') = d(u', u) + d(u, r_1) + 1 + d(u_2, r_2) + b(BC, T_2)$. Since $d(u', u_1) < d(u, u_1)$, we get $t_1(u') < t_1(u)$. Moreover, $t_2(u') = d(u', u) + t_2(u) > t_1(u')$ since $t_2(u) > t_1(u) > t_1(u')$. Finally we arrive at: $b(u', T) = t_2(u') > b(u, T)$ which contradicts the assumption.

Case 2: $d(u', u_1) > d(u, u_1)$. As it was done in the previous case, we can deduce that $t_2(u) = t_1(u) + 1$ and $b(u, T) = t_2(u)$. Now consider the vertex u' . Calculating $t_1(u')$ and $t_2(u')$ we get: $t_1(u') = d(u', u) + d(u, u_1) + b(BC, T_1)$ and $t_2(u') = d(u', r_1) + 1 + d(r_2, u_2) + b(BC, T_2)$. Using that $d(u, r_1) = d(u, u') + d(u', r_1)$ we get: $t_2(u') = [d(u, r_1) - d(u, u')] + 1 + d(r_2, u_2) + b(BC, T_2)$. Hence, $t_2(u') = t_2(u) - d(u, u') = t_1(u) + 1 - d(u, u')$. On the other hand, $t_1(u') = t_1(u) + d(u, u')$. Since $d(u', u_1) > d(u, u_1)$, we conclude that $d(u, u') \geq 1$. Subtracting $t_2(u')$ from $t_1(u')$ we get: $t_1(u') - t_2(u') = t_1(u) + d(u, u') - [t_1(u) + 1 - d(u, u')]$. Therefore, $t_1(u') - t_2(u') = 2d(u, u') - 1$. Using $d(u, u') \geq 1$, we get that $t_1(u') > t_2(u') + 1$. Hence, we conclude that $b(u', T) = t_1(u')$. So, $b(u', T) = t_1(u') = t_2(u') + 2d(u, u') - 1$. Furthermore, using $t_2(u') = t_2(u) - d(u, u')$ we get $b(u', T) = t_2(u) + d(u, u') - 1 \geq t_2(u)$, which implies that $b(u', T) \geq b(u, T)$ which is a contradiction.

3 The UNICYCLICBROADCAST Algorithm

The algorithm, *UNICYCLICBROADCAST*(w, G), calculates the broadcast time, $b(w, G)$, of a given vertex w in any unicyclic graph G . For convenience we will assume that w belongs to tree T_1 .

3.1 Description of the Algorithm

INPUT: A unicyclic graph G on n vertices and the broadcast originator w .

OUTPUT: Broadcast time of the originator w in G , $b(w, G)$, and a broadcast scheme.

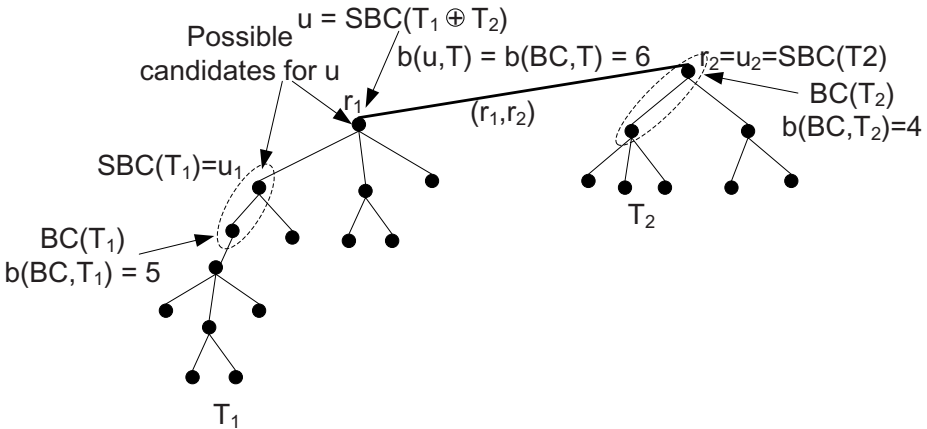


Fig. 3. An example of the sum of two trees. It shows the vertices in the broadcast center and the special broadcast center of T_1 , T_2 , and $T_1 \oplus T_2$. The minimum broadcast time of the three trees are calculated too.

UNICYCLICBROADCAST(w, G):

1. Extract from G the cycle C_k , consisting of the vertices $\{r_1, r_2, \dots, r_k\}$, and the trees T_i rooted at the vertices r_i for $1 \leq i \leq k$.
2. For all trees T_i where $1 \leq i \leq k$ calculate and save the positions of $u_i = SBC(T_i)$ relative to r_i , as well as $b(BC, T_i)$.
3. Calculate and save the distance $d(w, u_1)$ and the path joining w and u_1 .
4. Construct the trees $T_{1,2,\dots,i}$, where $2 \leq i \leq k$, and $T_{k,k-1,\dots,i}$, where $1 \leq i \leq k-1$. For each tree T , compute and store the position of the $SBC(T)$ and $b(BC, T)$.
5. Construct the spanning trees $T_{j,j+1,\dots,k,1,2,\dots,j-1}$, where $1 \leq j \leq k$. For each tree T compute and store $SBC(T)$, $b(BC, T)$, $d(w, SBC(T))$, and $b(w, T)$.
6. Out of the trees generated in the previous step, choose the spanning tree T with the minimum value of $b(w, T)$.
7. Run BROADCAST [20] to find a broadcast scheme for the originator w .

The algorithm first preprocesses the unicyclic graph and calculates the cycle C_k consisting of the vertices $\{r_1, r_2, \dots, r_k\}$ and the trees T_i rooted at r_i , where $1 \leq i \leq k$. In step 2 the path joining w to u_1 and $d(w, u_1)$ are calculated and saved. This information will be needed to calculate the broadcast time of w . Steps 3 and 4 construct several trees and calculate results that will be used in constructing the k spanning trees of the graph G . More specifically the trees, T_i , $T_{i,i+1,\dots,k}$ for $1 \leq i \leq k-1$, and $T_{1,2,\dots,i}$ for $2 \leq i \leq k$ are constructed. For each spanning tree T the position of the $SBC(T)$ and $b(BC, T)$ are calculated and stored. These results will be useful to calculate the broadcast centers and the minimum broadcast times of the spanning trees. At the end of step 4 only one spanning tree will be constructed. In step 5, the algorithm builds the remaining $k-1$ spanning trees of the unicyclic graph. It also calculates the broadcast time of w for each one of them. Note that for each spanning tree T , $b(w, T) = d(w, SBC(T)) + b(BC, T)$ where the distance $d(w, SBC(T))$ can be easily calculated by using $d(w, u_1)$ and position of $SBC(T)$ calculated in steps 2 and 4 respectively. The spanning tree that has the minimum broadcast time for w is the required result. Finally in order to obtain the optimal broadcast scheme for the originator w in the unicyclic graph G , the BROADCAST algorithm of [20] is run on the spanning tree T that had the minimum value of $b(w, T)$.

In step 5, $k-1$ spanning trees are constructed each in a constant time. The construction of each tree can be done easily by observing that the spanning tree $T_{i,i+1,\dots,k,1,2,\dots,i-1}$ is the sum of the two trees $T_{i,i+1,\dots,k}$ and $T_{1,2,\dots,i-1}$ rooted at r_k and r_1 respectively. These two trees and all the information pertinent to the calculation of the $b(w, T_{i,i+1,\dots,k,1,2,\dots,i-1})$ were calculated in step 4.

3.2 Proof of Correctness and Complexity Analysis

Theorem 4. UNICYCLICBROADCAST(w, G) generates a broadcast scheme for originator w , and finds $b(w, G)$.

Proof. Let C_k , consisting of the vertices $\{r_1, r_2, \dots, r_k\}$, represent the cycle in the unicyclic graph G . One of the spanning trees of G is obtained by removing the edge (r_k, r_1) . The resulting tree is $T_{1,2,\dots,k}$. The minimum broadcast time of $T_{1,2,\dots,k}$ is calculated iteratively by tree summations $T_{1,\dots,i-1} \oplus T_i$ rooted at r_{i-1} and r_i respectively, where $2 \leq i \leq k$. The remaining minimum spanning trees, $T_{j,j+1,\dots,k,1,2,\dots,j-1}$ where $2 \leq j \leq k$, are calculated by performing the summations $T_{j,\dots,k} \oplus T_{1,\dots,j-1}$. Theorems 2 and 3 guarantee that a broadcast center and the minimum broadcast time of all the trees $T_{j,j+1,\dots,k,1,2,\dots,j-1}$, where $1 \leq j \leq k$, are calculated correctly. Since, the trees $T_{j,j+1,\dots,k,1,2,\dots,j-1}$, where $1 \leq j \leq k$, are the all possible spanning trees of the unicyclic graph G , the algorithm correctly finds the spanning tree of the unicyclic graph G that has the minimum broadcast time of all the spanning trees. We will prove the correctness of the algorithm by contradiction. Assume that there exists a broadcast tree T' and a broadcast scheme in G that performs broadcasting in time t such that $t = d(w, BC(T')) + b(BC, T') < b(w, G)$. T' should be one of the trees $T_{j,j+1,\dots,k,1,2,\dots,j-1}$, where $1 \leq j \leq k$. But the algorithm correctly calculated the minimum broadcast time of all the spanning trees and chose the spanning tree T with the minimum value of $d(w, BC(T)) + b(BC, T)$. Therefore the existence of T' creates a contradiction.

Theorem 5. *The complexity of the UNICYCLICBROADCAST running on a unicyclic graph $G = (V, E)$ is $O(|V|)$.*

Proof. Steps 1 and 2 of the algorithm can be accomplished by a depth first search in $O(|V|)$ time. In Step 3, BROADCAST [20] is applied on the trees T_i for $1 \leq i \leq k$. The complexity of this step is $O(|V_1| + |V_2| + \dots + |V_k|) = O(|V|)$. Step 4 is of complexity $O(k)$ since there are k sums to be done, and the sum of two trees T_1 and T_2 , $T_1 \oplus T_2$, can be done in a constant time. Moreover, every time a tree T is constructed as $T = T_1 \oplus T_2$, calculating the distance between the root of T and $SBC(T)$, and $b(BC, T)$ can be done in a constant time using the positions of $SBC(T_1)$ and $SBC(T_2)$, and the broadcast times $b(BC, T_1)$ and $b(BC, T_2)$. Step 5 involves the calculation of $k - 1$ spanning trees. Each spanning tree is constructed by summing two trees, hence the complexity of this step is again $O(k)$. Step 6 chooses the spanning tree with the least minimum broadcast time hence its complexity is $O(k)$. Adding all the complexities we get that the complexity of the algorithm is $O(|V| + k)$. However, $k \leq |V|$, so this proves that the complexity of the algorithm UNICYCLICBRDCST is $O(|V|)$.

4 Conclusion

In this paper we presented an algorithm that determines the broadcast time of any vertex w in an arbitrary unicyclic graph $G = (V, E)$ in linear time, $O(|V|)$. Therefore, the complexity of the problem of determining the broadcast time of any vertex in an arbitrary unicyclic graph is $\Theta(|V|)$. The algorithm can find the minimum broadcast time of G as well as the spanning tree of G which is the broadcast tree corresponding to the minimum broadcast time. As a byproduct, the algorithm also finds a broadcast center of the unicyclic graph.

References

1. Aiello, W., Chung, F., Lu, L.: Random evolution in massive graphs. In: FOCS'01. Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, pp. 510–519 (2001)
2. Bar-Noy, A., Guha, S., Naor, J., Schieber, B.: Multicasting in Heterogeneous Networks. In: STOC'98. Proc. of ACM Symp. on Theory of Computing (1998)
3. Beier, R., Sibeyn, J.F.: A powerful heuristic for telephone gossiping. In: SIROCCO'00. Proc. of the 7th International Colloquium on Structural Information & Communication Complexity, L'Aquila, Italy, pp. 17–36 (2000)
4. Bermond, J.-C., Fraigniaud, P., Peters, J.: Antepenultimate broadcasting. Networks 26, 125–137 (1995)
5. Bermond, J.-C., Hell, P., Liestman, A.L., Peters, J.G.: Sparse broadcast graphs. Discrete Appl. Math. 36, 97–130 (1992)
6. Dinneen, M.J., Fellows, M.R., Faber, V.: Algebraic constructions of efficient broadcast networks. In: Mattson, H.F., Rao, T.R.N., Mora, T. (eds.) Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. LNCS, vol. 539, pp. 152–158. Springer, Heidelberg (1991)
7. Doar, M.B.: A better model for generating test networks. In: IEEE GLOBECOM'96, London, IEEE Computer Society Press, Los Alamitos (1996)
8. Elkin, M., Kortsarz, G.: A combinatorial logarithmic approximation algorithm for the directed telephone broadcast problem. In: STOC'02. Proc. of ACM Symp. on Theory of Computing, pp. 438–447 (2002)
9. Elkin, M., Kortsarz, G.: Sublogarithmic approximation for telephone multicast: path out of jungle. In: SODA'03. Proc. of Symposium on Discrete Algorithms, Baltimore, Maryland, pp. 76–85 (2003)
10. Farley, A.M., Hedetniemi, S.T., Proskurowski, A., Mitchell, S.: Minimum broadcast graphs. Discrete Math. 25, 189–193 (1979)
11. Feige, U., Peleg, D., Raghavan, P., Upfal, E.: Randomized broadcast in networks. In: SIGAL'90. Proc. of International Symposium on Algorithms, pp. 128–137 (1990)
12. Fraigniaud, P., Vial, S.: Approximation algorithms for broadcasting and gossiping. J. Parallel and Distrib. Comput. 43(1), 47–55 (1997)
13. Fraigniaud, P., Vial, S.: Heuristic Algorithms for Personalized Communication Problems in Point-to-Point Networks. In: SIROCCO'97. Proc. of the 4th Colloquium on Structural Information and Communication Complexity, pp. 240–252 (1997)
14. Fraigniaud, P., Vial, S.: Comparison of Heuristics for One-to-All and All-to-All Communication in Partial Meshes. Parallel Processing Letters 9(1), 9–20 (1999)
15. Harutyunyan, H.A., Liestman, A.L.: More broadcast graphs. Discrete Math. 98, 81–102 (1999)
16. Harutyunyan, H.A., Shao, B.: An Efficient Heuristic for Broadcasting in Networks. Journal of Parallel and Distributed Computing (to appear)
17. Johnson, D., Garey, M.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco, CA (1979)
18. Kortsarz, G., Peleg, D.: Approximation algorithms for minimum time broadcast. SIAM J. Discrete Math. 8, 401–427 (1995)

19. Ravi, R.: Rapid Rumor Ramification: Approximating the minimum broadcast time. In: FOCS'94. Proc. of 35th Symposium on Foundation of Computer Science, pp. 202–213 (1994)
20. Slater, P.J., Cockayne, E.J., Hedetniemi, S.T.: Information dissemination in trees. SIAM J.Comput. 10(4), 692–701 (1981)
21. Scheuerman, P., Wu, G.: Heuristic Algorithms for Broadcasting in Point-to-Point Computer Network. IEEE Transactions on Computers C-33(9), 804–811 (1984)
22. Zegura, E.W., Calvert, K., Bhattacharjee, S.: How to model an internetwork. In: INFOCOM'96. Proc. The IEEE Conf. on Computer Communications, San Francisco, CA, IEEE Computer Society Press, Los Alamitos (1996)

An Improved Algorithm for Online Unit Clustering

Hamid Zarrabi-Zadeh and Timothy M. Chan

School of Computer Science, University of Waterloo
Waterloo, Ontario, Canada, N2L 3G1
{hzarrabi, tmchan}@uwaterloo.ca

Abstract. We revisit the *online unit clustering* problem in one dimension which we recently introduced at WAOA'06: given a sequence of n points on the line, the objective is to partition the points into a minimum number of subsets, each enclosable by a unit interval. We present a new randomized online algorithm that achieves expected competitive ratio $11/6$ against oblivious adversaries, improving the previous ratio of $15/8$. This immediately leads to improved upper bounds for the problem in two and higher dimensions as well.

1 Introduction

At WAOA'06 [1], we began investigating an online problem we call *unit clustering*, which is extremely simple to state but turns out to be nontrivial surprisingly:

Given a sequence of n points on the real line, assign points to clusters so that each cluster is enclosable by a unit interval, with the objective of minimizing the number of clusters used.

In the offline setting, variations of this problem frequently appear as textbook exercises and can be solved in $O(n \log n)$ time by a simple greedy algorithm (e.g., see [3]). The problem is equivalent to finding the minimum number of points that stab a given collection of unit intervals (i.e., clique partitioning in unit interval graphs, or coloring unit co-interval graphs), and to finding the maximum number of disjoint intervals in a given collection (i.e., maximum independent set in unit interval graphs). It is the one-dimensional analog of an often-studied and important geometric clustering problem—covering a set of points in d dimensions using a minimum number of unit disks (for example, under the Euclidean or L_∞ metric) [5,6,8,11,12]. This geometric problem has applications in facility location, map labeling, image processing, and other areas.

Online versions of clustering and facility location problems are natural to consider because of practical considerations and have been extensively studied in the literature [2,4,10]. Here, input points are given one by one as a sequence over time, and each point should be assigned to a cluster upon its arrival. The main constraint is that clustering decisions are irrevocable: once formed, clusters cannot be removed or broken up.

For our one-dimensional problem, it is easy to come up with an algorithm with competitive ratio 2; for example, we can use a naïve grid strategy: build a uniform unit grid and simply place each arriving point in the cluster corresponding to the point's grid cell (for the analysis, just observe that every unit interval intersects at most 2 cells). Alternatively, we can use the most obvious greedy strategy: for each given point, open a new cluster only if the point does not “fit” in any existing cluster; this strategy too has competitive ratio 2.

In the previous paper [1], we have shown that it is possible to obtain an online algorithm with expected competitive ratio strictly less than 2 using randomization; specifically, the ratio obtained is at most $15/8 = 1.875$. This result is a pleasant surprise, considering that ratio 2 is known to be tight (among both deterministic and randomized algorithms) for the related *online unit covering* problem [21] where the position of each enclosing unit interval is specified upon its creation, and this position cannot be changed later. Ratio 2 is also known to be tight among deterministic algorithms for the problem of online coloring of (arbitrary rather than unit) co-interval graphs [7,9].

In this paper, we improve our previous result further and obtain a randomized online algorithm for one-dimensional unit clustering with expected competitive ratio at most $11/6 \approx 1.8333$. Automatically, this implies improved online algorithms for geometric unit clustering under the L_∞ metric, with ratio $11/3$ in 2D, for example.

The new algorithm is based on the approach from the previous paper but incorporates several additional ideas. A key difference in the design of the algorithm is to make more uses of randomization (the previous algorithm requires only 2 random bits). The previous algorithm is based on a clever grid approach where windows are formed from pairs of adjacent grid cells, and clusters crossing two adjacent windows are “discouraged”; in the new algorithm, crossings of adjacent windows are discouraged to a “lesser” extent, as controlled by randomization. This calls for other subtle changes in the algorithm, as well as a lengthier case analysis that needs further technical innovations.

2 The New Randomized Algorithm

In this section, we present the new randomized algorithm for the online unit clustering problem in one dimension. The competitive ratio of the algorithm is not necessarily less than 2, but will become less than 2 when combined with the naïve grid strategy as described in Section 5. Our new algorithm is based in part on our previous randomized algorithm [1], although we will keep the presentation self-contained. A key difference is to add an extra level of randomization.

Consider a uniform unit grid on the line, where each grid cell is a half-closed interval of the form $[i, i + 1)$. To achieve competitive ratio better than 2, we have to allow clusters to cross grid cells occasionally (for example, just consider the input sequence $\langle \frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \dots \rangle$, where the naïve grid strategy would require twice as many clusters as the optimum). As in the previous algorithm, we accomplish this by forming *windows* over the line each consisting of two grid cells and permit

clusters crossing two cells within a window. There are two ways to form windows over the grid; we choose which according to an initial random bit. In the previous algorithm, clusters crossing two adjacent windows are not strictly forbidden but are discouraged in some sense.

In the new algorithm, the idea, roughly speaking, is to permit more clusters crossing windows. More specifically, call the grid point lying between two adjacent windows a *border*; generate a random bit for every border, where a 1 bit indicates an *open* border and a 0 bit indicates a *closed* border. Clusters crossing closed borders are still discouraged, but not clusters crossing open borders. (As it turns out, setting the probability of border opening/closing to $1/2$ is indeed the best choice.)

The actual details of the algorithm are important and are carefully crafted. In the pseudocode below, $b(w, w')$ refers to the border indicator between windows w and w' . We say that a point *lies* in a cluster if inserting it to the cluster would not increase the length of the cluster, where the *length* of a cluster refers to the length of its smallest enclosing interval. We say that a point *fits* in a cluster if inserting it to the cluster would not cause the length to exceed 1.

RandBorder Algorithm: Partition the line into *windows* each of the form $[2i, 2i+2)$. With probability $1/2$, shift all windows one unit to the right. For each two neighboring windows w and w' set $b(w, w')$ to a randomly drawn number from $\{0, 1\}$. For each new point p , find the window w containing p , and do the following:

- 1: **if** p fits in a cluster intersecting w **then**
 - 2: put p in the “closest” such cluster
 - 3: **else if** p fits in a cluster u inside a neighboring window w' **then**
 - 4: **if** $b(w, w') = 1$ **then** put p in u
 - 5: **else if** w (completely) contains at least 1 cluster and
 w' (completely) contains at least 2 clusters
 - 6: **then** put p in u
 - 7: **if** p is not put in any cluster **then** open a new cluster for p
-

Thus, a cluster is allowed to cross the boundary of two grid cells within a window freely, but it can cross the boundary of two adjacent windows only in two exceptional cases: when the corresponding border indicator is set to 1, or when the carefully specified condition in Line 5 arises (this condition is slightly different from the one in the previous algorithm). We will see the rationale for this condition during the analysis.

To see what the “closeness” exactly means in Line 2, we define the following two preference rules:

- RULE I. If p lies in a cluster u , then u is the closest cluster to p .
- RULE II. If p lies in a cell c , then any cluster intersecting c is closer to p than any cluster contained in a neighboring cell.

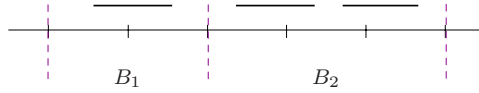


Fig. 1. Two blocks of sizes 2 and 3

The first preference rule prevents clusters from overlapping each other, and the second rule prevents clusters from unnecessarily crossing the boundary of two neighboring cells. The above preference rules and exceptional cases will be vital to the analysis.

Note that the random bits used for the border indicators can be easily generated on the fly as new borders are created.

3 Preliminaries for the Analysis

To prepare for the analysis, we first state a few definitions (borrowed from [1]).

Let σ be the input sequence. We denote by $\text{opt}(\sigma)$ the optimal offline solution obtained by the following greedy algorithm: sort all points in σ from left to right; cover the leftmost point p and all points within unit distance to it by a unit interval started at p ; and repeat the procedure for the remaining uncovered points. Obviously, the unit intervals obtained by this algorithm are disjoint.

We refer to a cluster as a *crossing cluster* if it intersects two adjacent grid cells, or as a *whole cluster* if it is contained completely in a grid cell.

For any real interval x (e.g., a grid cell or a group of consecutive cells), the *cost* of x denoted by $\mu(x)$ is defined to be the number of whole clusters contained in x plus half the number of clusters crossing the boundaries of x , in the solution produced by the RandBorder algorithm. We note that μ is additive, i.e., for two adjacent intervals x and y , $\mu(x \cup y) = \mu(x) + \mu(y)$.

A set of k consecutive grid cells containing $k - 1$ intervals from $\text{opt}(\sigma)$ is called a *block* of size k (see Fig. 1). We define $\rho(k)$ to be the expected competitive ratio of the RandBorder algorithm within a block of size k . In other words, $\rho(k)$ upper-bounds the expected value of $\mu(B)/(k - 1)$ over all blocks B of size k .

In the following, a list of objects (e.g., grid cells or clusters) denoted by $\langle x_i, \dots, x_j \rangle$ is always implicitly assumed to be ordered from left to right on the line. Moreover, $p_1 \ll p_2$ denotes the fact that point p_1 arrives before point p_2 in the input sequence.

We now establish some observations concerning the behavior of the RandBorder algorithm. Observations 1(ii) and (iii) are basically from [1] and have similar proofs (which are reproduced here for completeness' sake since the algorithm has changed); the other observations and subsequent lemmas are new and will be used multiple times in the analysis in the next section.

Observation 1.

- (i) Any interval in $\text{opt}(\sigma)$ that does not cross a closed border can (completely) contain at most one whole cluster.

- (ii) Any grid cell c can contain at most one whole cluster. Thus, we always have $\mu(c) \leq 1 + \frac{1}{2} + \frac{1}{2} = 2$.
- (iii) If a grid cell c intersects a crossing cluster u_1 and a whole cluster u_2 , then u_2 must be opened after u_1 has been opened, and after u_1 has become a crossing cluster.

Proof. (i) Let u_1 and u_2 be two whole clusters contained in the said interval and suppose that u_1 is opened before u_2 . Then all points of u_2 would be assigned to u_1 , because Lines 2 and 4 precede Line 7. (ii) holds by the same argument, because Line 2 precedes Line 7.

For (iii), let p_1 be the first point of u_1 in c and p'_1 be the first point of u_1 in a cell adjacent to c . Let p_2 be the first point of u_2 . Among these three points, p_1 cannot be the last to arrive: otherwise, p_1 would be assigned to the whole cluster u_2 instead of u_1 , because of Rule II. Furthermore, p'_1 cannot be the last to arrive: otherwise, p_1 would be assigned to u_2 instead. So, p_2 must be the last to arrive. □

Observation 2. Let u_1 be a whole cluster contained in a grid cell c , and let u_2 and u_3 be two clusters crossing the boundaries of c . Then

- (i) u_1 and u_2 cannot be entirely contained in the same interval from $\text{opt}(\sigma)$.
- (ii) there are no two intervals I_1 and I_2 in $\text{opt}(\sigma)$ such that $u_1 \cup u_2 \cup u_3 \subseteq I_1 \cup I_2$.

Proof. (i) Suppose by way of contradiction that u_1 and u_2 are entirely contained in an interval I from $\text{opt}(\sigma)$. Then by Observation □(iii), u_1 is opened after u_2 has become a crossing cluster, but then the points of u_1 would be assigned to u_2 instead: a contradiction.

(ii) Suppose that $u_1 \cup u_2 \cup u_3 \subseteq I_1 \cup I_2$, where I_1 and I_2 are the two intervals from $\text{opt}(\sigma)$ intersecting c . We now proceed as in part (i). By Observation □(iii), u_1 is opened after u_2 and u_3 have become crossing clusters, but then the points of u_1 would be assigned to u_2 or u_3 instead: a contradiction. □

Lemma 1. Let $B = \langle c_1, \dots, c_k \rangle$ be a block of size $k \geq 2$, and S be the set of all odd-indexed (or even-indexed) cells in B . Then there exists a cell $c \in S$ such that $\mu(c) < 2$.

Proof. Let $\langle I_1, \dots, I_{k-1} \rangle$ be the $k - 1$ intervals from $\text{opt}(\sigma)$ in B , where each interval I_i intersects two cells c_i and c_{i+1} ($1 \leq i \leq k - 1$). Let O represent the set of all odd integers between 1 and k . We first prove the lemma for the odd-indexed cells.

Suppose by way of contradiction that for each $i \in O$, $\mu(c_i) = 2$. It means that for each $i \in O$, c_i intersects three clusters $\langle u_i^\ell, u_i, u_i^r \rangle$, where u_i is a whole cluster, and u_i^ℓ and u_i^r are two crossing clusters. We prove inductively that for each $i \in O$, $u_i \cap I_i \neq \emptyset$ and $u_i^r \cap I_{i+1} \neq \emptyset$.

BASE CASE: $u_1 \cap I_1 \neq \emptyset$ and $u_1^r \cap I_2 \neq \emptyset$.

The first part is trivial, because c_1 intersects just I_1 , and hence, $u_1 \subseteq I_1$.

The second part is implied by Observation □(i), because u_1 and u_1^r cannot be entirely contained in I_1 .

INDUCTIVE STEP: $u_i \cap I_i \neq \emptyset \wedge u_i^r \cap I_{i+1} \neq \emptyset \Rightarrow u_{i+2} \cap I_{i+2} \neq \emptyset \wedge u_{i+2}^r \cap I_{i+3} \neq \emptyset$.
 Suppose by contradiction that $u_{i+2} \cap I_{i+2} = \emptyset$. Therefore, u_{i+2} must be entirely contained in I_{i+1} . On the other hand, $u_i^r \cap I_{i+1} \neq \emptyset$ implies that u_{i+2}^ℓ is entirely contained in I_{i+1} . But this is a contradiction, because u_{i+2} and u_{i+2}^ℓ are contained in the same interval, which is impossible by Observation 2(i). Now, suppose that $u_{i+2}^r \cap I_{i+3} = \emptyset$. Since $u_i^r \cap I_{i+1} \neq \emptyset$, and clusters do not overlap, u_{i+2}^ℓ , u_{i+2} , and u_{i+2}^r should be contained in $I_{i+1} \cup I_{i+2}$, which is impossible by Observation 2(ii).

Repeating the inductive step zero or more times, we end up at either $i = k$ or $i = k - 1$. If $i = k$, then $u_k \cap I_k \neq \emptyset$ which is a contradiction, because there is no I_k . If $i = k - 1$, then $u_{k-1}^r \cap I_k \neq \emptyset$ which is again a contradiction, because we have no I_k .

Both cases lead to contradiction. It means that there exists some $i \in \mathcal{O}$ such that $\mu(c_i) < 2$. The proof for the even-indexed cells is similar. The only difference is that we need to prove the base case for $i = 2$, which is easy to get by Observations 2(i) and 2(ii). □

Lemma 2. *Let B be a block of size $k \geq 2$.*

- (i) $\mu(B) \leq 2k - 1$.
- (ii) *If all borders strictly inside B are open, then $\mu(B) \leq 2(k - 1)$.*

Proof. (i) is a direct corollary of Lemma 1, because there are at least two cells in B (one odd-indexed and one even-indexed) that have cost at most $3/2$, and the other cells have cost at most 2.

(ii) is immediate from the fact that each block of size $k \geq 2$ contains exactly $k - 1$ intervals from $\text{opt}(\sigma)$, and that each of these $k - 1$ intervals has cost at most 2 by Observation 1(i). □

4 The Analysis

We are now ready to analyze the expected competitive ratio of our algorithm within a block of size $k \geq 2$.

Theorem 1. $\rho(2) = 27/16$.

Proof. Consider a block B of size 2, consisting of two cells $\langle c_1, c_2 \rangle$ (see Fig. 2). Let I be the single unit interval in B in $\text{opt}(\sigma)$. There are two possibilities.

CASE 1: B falls completely in one window w . Let $\langle b_1, b_2 \rangle$ be the two border indicators at the boundaries of w . Let p_0 be the first point to arrive in I . W.l.o.g., assume p_0 is in c_2 (the other case is symmetric). We consider four subcases.

- SUBCASE 1.1: $\langle b_1, b_2 \rangle = \langle 0, 0 \rangle$. Here, both boundaries of B are closed. Thus, after a cluster u has been opened for p_0 (by Line 7), all subsequent points in I are put in the same cluster u . Note that the condition in Line 5 prevents points from the neighboring windows to join u and make crossing clusters. So, u is the only cluster in B , and hence, $\mu(B) = 1$.

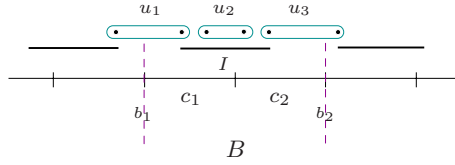


Fig. 2. Illustration of Subcase 1.3

- SUBCASE 1.2: $\langle b_1, b_2 \rangle = \langle 1, 0 \rangle$. When p_0 arrives, a new cluster u is opened, since p_0 is in c_2 , the right border is closed, and w contains < 1 cluster at the time so that the condition in Line 5 fails. Again, all subsequent points in I are put in the same cluster, and points from the neighboring windows cannot join u and make crossing clusters. Hence, $\mu(B) = 1$.
- SUBCASE 1.3: $\langle b_1, b_2 \rangle = \langle 0, 1 \rangle$. We show that $\mu(B) < 2$. Suppose by contradiction that $\mu(B) = 2$. By Observation 1(i), I cannot contain two clusters entirely. Therefore, the only way to get $\mu(B) = 2$ is that I intersects three clusters $\langle u_1, u_2, u_3 \rangle$ (from left to right, as always), where u_1 and u_3 are crossing clusters, and u_2 is entirely contained in I (see Fig. 2). By a similar argument as in the proof of Observation 1(iii), u_2 is opened after u_1 and u_3 have become crossing clusters. Let p_1 be the first point of u_1 in w , and p_2 be the first point of u_1 in the neighboring window. We have two scenarios:

- SUBSUBCASE 1.3.1: $p_1 \ll p_2$. In this case, cluster u_1 is opened for p_1 . But p_2 cannot be put in u_1 , because upon arrival of p_2 , w contains < 2 clusters, and thus, the condition in line 5 does not hold.
- SUBSUBCASE 1.3.2: $p_2 \ll p_1$. Here, cluster u_1 is opened for p_2 . But p_1 cannot be put in u_1 , because upon arrival of p_1 , w contains < 1 cluster, and hence, the condition in line 5 does not hold.

Both scenarios leads to contradiction. Therefore, $\mu(B) \leq 3/2$.

- SUBCASE 1.4: $\langle b_1, b_2 \rangle = \langle 1, 1 \rangle$. Here, Lemma 2(ii) implies that $\mu(B) \leq 2$.

Since each of the four subcases occurs with probability $1/4$, we conclude that the expected value of $\mu(B)$ in Case 1 is at most $\frac{1}{4}(1 + 1 + \frac{3}{2} + 2) = \frac{11}{8}$.

CASE 2: B is split between two neighboring windows. Let b be the single border indicator inside B . Let $\mu_0(B)$ and $\mu_1(B)$ represent the value of $\mu(B)$ for the case that b is set to 0 and 1, respectively. It is clear by Lemma 2(ii) that $\mu_1(B) \leq 2$. We rule out two possibilities:

- SUBCASE 2.1: $\mu_0(B) = 3$. Since I cannot contain both a whole cluster and a crossing cluster by Observation 2(i), the only possible scenario is that c_1 intersects two clusters $\langle u_1, u_2 \rangle$, and c_2 intersects two clusters $\langle u_3, u_4 \rangle$, where u_1 and u_4 are crossing clusters, and u_2 and u_3 are whole clusters. Let p_1 be the first point in u_2 and p_2 be the first point in u_3 . Suppose w.l.o.g. that $p_1 \ll p_2$. By Observation 1(iii), p_1 arrives after u_1 has been opened, and p_2 arrives after u_4 has been opened. But when p_2 arrives, the window

containing it contains one cluster, u_4 , and the neighboring window contains two clusters u_1 and u_2 . Therefore, p_2 would be assigned to u_2 by Line 5 instead: a contradiction.

- SUBCASE 2.2: $\mu_0(B) = 5/2$ and $\mu_1(B) = 2$. Suppose that $\mu_1(B) = 2$. Then I intersects three clusters $\langle u_1, u_2, u_3 \rangle$, where u_1 and u_3 are crossing clusters, and u_2 is completely contained in I . Let t be the time at which u_1 becomes a crossing cluster, and let $\sigma(t)$ be the subset of input points coming up to time t . By a similar argument as in the proof of Observation 1(iii), any point in $I \cap c_1$ not contained in u_1 arrives after time t . Therefore, upon receiving the input sequence $\sigma(t)$, u_1 becomes a crossing cluster no matter whether the border between c_1 and c_2 is open or closed. Using the same argument we conclude that u_3 becomes a crossing cluster regardless of the value of b . Now consider the case where $b = 0$. Since both u_1 and u_3 remain crossing clusters, $\mu_0(B)$ must be an integer (1, 2, or 3) and cannot equal $5/2$.

Ruling out these two subcases, we have $\mu_0(B) + \mu_1(B) \leq 4$ in all remaining subcases, and therefore, the expected value of $\mu(B)$ in this case is at most 2.

Since each of Cases 1 and 2 occurs with probability $1/2$, we conclude that $\rho(2) \leq \frac{1}{2}(\frac{11}{8}) + \frac{1}{2}(2) = \frac{27}{16}$. (This bound is tight: to see this just consider the block $B = [2, 4)$, and the sequence of 8 points $\langle 1.5, 2.5, 0.5, 3.5, 4.5, 2.7, 3.2, 5.5 \rangle$ for which $E[\mu(B)] = \frac{27}{16}$.) □

Theorem 2. $\rho(3) \leq 17/8$.

Proof. Consider a block B of size 3, consisting of cells $\langle c_1, c_2, c_3 \rangle$, and let b be the single border indicator strictly inside B . We assume w.l.o.g. that c_1 and c_2 fall in the same window (the other scenario is symmetric). We consider two cases.

- CASE 1: $b = 0$. We rule out the following possibilities.
 - SUBCASE 1.1: $\mu(c_2) = 2$. Impossible by Lemma 1.
 - SUBCASE 1.2: $\mu(c_1) = \mu(c_3) = 2$. Impossible by Lemma 1.
 - SUBCASE 1.3: $\mu(c_1) = 2$ and $\mu(c_2) = \mu(c_3) = 3/2$. Here, B intersects six clusters $\langle u_1, \dots, u_6 \rangle$, where u_1, u_3, u_6 are crossing clusters and u_2, u_4, u_5 are whole clusters. Let $\langle I_1, I_2 \rangle$ be the two unit intervals in B in $\text{opt}(\sigma)$. By Observation 2(i), u_3 cannot be entirely contained in I_1 . This implies that $u_4 \cup u_5 \subset I_2$. Now suppose w.l.o.g. that u_4 is opened after u_5 . By Observation 1(iii), u_4 is the last to be opened after u_3, u_5, u_6 . Consider any point p in u_4 . Upon arrival of p , the window containing p contains at least one cluster, u_3 , and the neighboring window contains two clusters u_5 and u_6 . Therefore, by the condition in Line 5, the algorithm would assign p to u_5 instead of u_4 , which is a contradiction.
 - SUBCASE 1.4: $\mu(c_1) = \mu(c_2) = 3/2$ and $\mu(c_3) = 2$. Similarly impossible.

In all remaining subcases, $\mu(B)$ is at most $2 + \frac{3}{2} + 1 = \frac{9}{2}$ or $\frac{3}{2} + \frac{3}{2} + \frac{3}{2} = \frac{9}{2}$.
- CASE 2: $b = 1$. Here, Lemma 2(ii) implies that $\mu(B) \leq 4$.

Each of Cases 1 and 2 occurs with probability $1/2$, therefore $\rho(3) \leq \frac{1}{2}(4 + \frac{9}{2})/2 = 17/8$. □

Theorem 3. $\rho(4) \leq 53/24$.

Proof. Consider a block B of size 4. We consider two easy cases.

- CASE 1: B falls completely in two windows. Let b be the single border indicator strictly inside B . Now, if $b = 1$, $\mu(B) \leq 6$ by Lemma 2(ii), otherwise, $\mu(B) \leq 7$ by Lemma 2(i). Therefore, the expected cost in this case is at most $\frac{1}{2}(6 + 7) = \frac{13}{2}$.
- CASE 2: B is split between three consecutive windows. Let $\langle b_1, b_2 \rangle$ be the two border indicators inside B . For the subcase where $\langle b_1, b_2 \rangle = \langle 1, 1 \rangle$ the cost is at most 6 by Lemma 2(ii), and for the remaining 3 subcases, the cost of B is at most 7 by Lemma 2(i). Thus, the expected cost in this case is at most $\frac{1}{4}(6) + \frac{3}{4}(7) = \frac{27}{4}$.

Since each of Cases 1 and 2 occurs with probability exactly $1/2$, we conclude that $\rho(4) \leq \frac{1}{2}(\frac{13}{2} + \frac{27}{4})/3 = \frac{53}{24}$. □

Theorem 4. $\rho(k) \leq (2k - 1)/(k - 1)$ for all $k \geq 5$.

Proof. This is a direct implication of Lemma 2(i). □

5 The Combined Algorithm

The RandBorder algorithm as shown in the previous section has competitive ratio greater than 2 on blocks of size three and more. To overcome this deficiency, we need to combine RandBorder with another algorithm that works well for larger block sizes. A good candidate for this is the naïve grid algorithm:

Grid Algorithm: For each new point p , if the grid cell containing p contains a cluster, then put p in that cluster, else open a new cluster for p .

It is easy to verify that the Grid algorithm uses exactly k clusters on a block of size k . Therefore, the competitive ratio of this algorithm within a block of size k is $k/(k - 1)$. We can now randomly combine the RandBorder algorithm with the Grid algorithm to obtain an expected competitive ratio strictly less than 2.

Combined Algorithm: With probability $8/15$ run RandBorder, and with probability $7/15$ run Grid.

Theorem 5. *The competitive ratio of the Combined algorithm is at most $11/6$ against oblivious adversaries.*

Proof. The competitive ratios of RandBorder and Grid within blocks of size 2 are 27/16 and 2, respectively. Therefore, the expected competitive ratio of the Combined algorithm is $\frac{8}{15}(\frac{27}{16}) + \frac{7}{15}(2) = \frac{11}{6}$ within a block of size 2. For larger block sizes, the expected competitive ratio of Combined is always at most 11/6, as shown in Table 1. By summing over all blocks and exploiting the additivity of our cost function $\mu(\cdot)$, we see that the expected total cost of the solution produced by Combined is at most 11/6 times the size of $\text{opt}(\sigma)$ for every input sequence σ . \square

Table 1. The competitive ratio of the algorithms within a block

Block Size	Grid	RandBorder	Combined
2	2	27/16	11/6
3	3/2	$\leq \frac{17}{8}$	$\leq 11/6$
4	4/3	$\leq \frac{53}{24}$	$\leq 9/5$
$k \geq 5$	$\frac{k}{k-1}$	$\leq \frac{2k-1}{k-1}$	$\leq \frac{23k-8}{15(k-1)}$

Remarks. Currently only a 4/3 randomized lower bound and a 3/2 deterministic lower bound are known for the one-dimensional problem [1]. Also, as a corollary to Theorem 5, we immediately get an upper bound of $(\frac{11}{12}) \cdot 2^d$ for the d -dimensional unit clustering problem under the L_∞ metric [1].

References

- Chan, T.M., Zarrabi-Zadeh, H.: A randomized algorithm for online unit clustering. In: Erlebach, T., Kaklamani, C. (eds.) WAOA 2006. LNCS, vol. 4368, pp. 121–131. Springer, Heidelberg (2007)
- Charikar, M., Chekuri, C., Feder, T., Motwani, R.: Incremental clustering and dynamic information retrieval. SIAM J. Comput. 33(6), 1417–1440 (2004)
- Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
- Fotakis, D.: Incremental algorithms for facility location and k -median. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 347–358. Springer, Heidelberg (2004)
- Fowler, R.J., Paterson, M.S., Tanimoto, S.L.: Optimal packing and covering in the plane are NP-complete. Inform. Process. Lett. 12(3), 133–137 (1981)
- Gonzalez, T.: Covering a set of points in multidimensional space. Inform. Process. Lett. 40, 181–188 (1991)
- Gyarfas, A., Lehel, J.: On-line and First-Fit colorings of graphs. J. Graph Theory 12, 217–227 (1988)
- Hochbaum, D.S., Maass, W.: Approximation schemes for covering and packing problems in image processing and VLSI. J. ACM 32, 130–136 (1985)
- Kierstead, H.A., Qin, J.: Coloring interval graphs with First-Fit. SIAM J. Discrete Math. 8, 47–57 (1995)

10. Meyerson, A.: Online facility location. In: Proc. 42nd IEEE Sympos. Found. Comput. Sci., pp. 426–433 (2001)
11. Nielsen, F.: Fast stabbing of boxes in high dimensions. *Theoret. Comput. Sci.* 246, 53–72 (2000)
12. Tanimoto, S.L., Fowler, R.J.: Covering image subsets with patches. In: Proc. 5th International Conf. on Pattern Recognition, pp. 835–839 (1980)

Linear Time Algorithms for Finding a Dominating Set of Fixed Size in Degenerated Graphs

Noga Alon^{1,*} and Shai Gutner^{2,**}

¹ Schools of Mathematics and Computer Science, Tel-Aviv University,
Tel-Aviv, 69978, Israel
noga@math.tau.ac.il.

² School of Computer Science, Tel-Aviv University, Tel-Aviv, 69978, Israel
gutner@tau.ac.il.

Abstract. There is substantial literature dealing with fixed parameter algorithms for the dominating set problem on various families of graphs. In this paper, we give a $k^{O(dk)}n$ time algorithm for finding a dominating set of size at most k in a d -degenerated graph with n vertices. This proves that the dominating set problem is fixed-parameter tractable for degenerated graphs. For graphs that do not contain K_h as a topological minor, we give an improved algorithm for the problem with running time $(O(h))^{hk}n$. For graphs which are K_h -minor-free, the running time is further reduced to $(O(\log h))^{hk/2}n$. Fixed-parameter tractable algorithms that are linear in the number of vertices of the graph were previously known only for planar graphs.

For the families of graphs discussed above, the problem of finding an induced cycle of a given length is also addressed. For every fixed H and k , we show that if an H -minor-free graph G with n vertices contains an induced cycle of size k , then such a cycle can be found in $O(n)$ expected time as well as in $O(n \log n)$ worst-case time. Some results are stated concerning the (im)possibility of establishing linear time algorithms for the more general family of degenerated graphs.

Keywords: H -minor-free graphs, degenerated graphs, dominating set problem, finding an induced cycle, fixed-parameter tractable algorithms.

1 Introduction

This paper deals with fixed-parameter algorithms for degenerated graphs. The degeneracy $d(G)$ of an undirected graph $G = (V, E)$ is the smallest number d for which there exists an acyclic orientation of G in which all the outdegrees are at most d . Many interesting families of graphs are degenerated (have bounded

* Research supported in part by a grant from the Israel Science Foundation, and by the Hermann Minkowski Minerva Center for Geometry at Tel Aviv University.

** This paper forms part of a Ph.D. thesis written by the author under the supervision of Prof. N. Alon and Prof. Y. Azar in Tel Aviv University.

degeneracy). For example, graphs embeddable on some fixed surface, degree-bounded graphs, graphs of bounded tree-width, and non-trivial minor-closed families of graphs.

There is an extensive literature dealing with fixed-parameter algorithms for the dominating set problem on various families of graphs. Our main result is a linear time algorithm for finding a dominating set of fixed size in degenerated graphs. This is the most general class of graphs for which fixed-parameter tractability for this problem has been established. To the best of our knowledge, linear time algorithms for the dominating set problem were previously known only for planar graphs. Our algorithms both generalize and simplify the classical bounded search tree algorithms for this problem (see, e.g., [2][13]).

The problem of finding induced cycles in degenerated graphs has been studied by Cai, Chan and Chan [8]. Our second result in this paper is a randomized algorithm for finding an induced cycle of fixed size in graphs with an excluded minor. The algorithm’s expected running time is linear, and its derandomization is done in an efficient way, answering an open question from [8]. The problem of finding induced cycles in degenerated graphs is also addressed.

The Dominating Set Problem. The dominating set problem on general graphs is known to be $W[2]$ -complete [12]. This means that most likely there is no $f(k) \cdot n^c$ -algorithm for finding a dominating set of size at most k in a graph of size n for any computable function $f : \mathbb{N} \rightarrow \mathbb{N}$ and constant c . This suggests the exploration of specific families of graphs for which this problem is fixed-parameter tractable. For a general introduction to the field of parameterized complexity, the reader is referred to [12] and [14].

The method of bounded search trees has been used to give an $O(8^k n)$ time algorithm for the dominating set problem in planar graphs [2] and an $O((4g + 40)^k n^2)$ time algorithm for the problem in graphs of bounded genus $g \geq 1$ [13]. The algorithms for planar graph were improved to $O(4^{6\sqrt{34k}} n)$ [1], then to $O(2^{27\sqrt{k}} n)$ [17], and finally to $O(2^{15.13\sqrt{k}} k + n^3 + k^4)$ [15]. Fixed-parameter algorithms are now known also for map graphs [9] and for constant powers of H -minor-free graphs [10]. The running time given in [10] for finding a dominating set of size k in an H -minor-free graph G with n vertices is $2^{O(\sqrt{k})} n^c$, where c is a constant depending only on H . To summarize these results, fixed-parameter tractable algorithms for the dominating set problem were known for fixed powers of H -minor-free graphs and for map graphs. Linear time algorithms were established only for planar graphs.

Finding Paths and Cycles. The foundations for the algorithms for finding cycles, presented in this paper, have been laid in [4], where the authors introduce the color-coding technique. Two main randomized algorithms are presented there, as follows. A simple directed or undirected path of length $k - 1$ in a graph $G = (V, E)$ that contains such a path can be found in $2^{O(k)}|E|$ expected time in the directed case and in $2^{O(k)}|V|$ expected time in the undirected case. A simple directed or undirected cycle of size k in a graph $G = (V, E)$ that contains such a cycle can be found in either $2^{O(k)}|V||E|$ or $2^{O(k)}|V|^\omega$ expected time, where

$\omega < 2.376$ is the exponent of matrix multiplication. These algorithms can be derandomized at a cost of an extra $\log |V|$ factor. As for the case of even cycles, it is shown in [23] that for every fixed $k \geq 2$, there is an $O(|V|^2)$ algorithm for finding a simple cycle of size $2k$ in an undirected graph (that contains such a cycle). Improved algorithms for detecting given length cycles have been presented in [5] and [24]. The authors of [5] describe fast algorithms for finding short cycles in d -degenerated graphs. In particular, C_3 's and C_4 's can be found in $O(|E| \cdot d(G))$ time and C_5 's in $O(|E| \cdot d(G)^2)$ time.

Finding Induced Paths and Cycles. Cai, Chan and Chan have recently introduced a new interesting technique they call *random separation* for solving fixed-cardinality optimization problems on graphs [8]. They combine this technique together with color-coding to give the following algorithms for finding an induced graph within a large graph. For fixed constants k and d , if a d -degenerated graph G with n vertices contains some fixed induced tree T on k vertices, then it can be found in $O(n)$ expected time and $O(n \log^2 n)$ worst-case time. If such a graph G contains an induced k -cycle, then it can be found in $O(n^2)$ expected time and $O(n^2 \log^2 n)$ worst-case time. Two open problems are raised by the authors of the paper. First, they ask whether the $\log^2 n$ factor incurred in the derandomization can be reduced to $\log n$. A second question is whether there is an $O(n)$ expected time algorithm for finding an induced k -cycle in a d -degenerated graph with n vertices. In this paper, we show that when combining the techniques of random separation and color-coding, an improved derandomization with a loss of only $\log n$ is indeed possible. An $O(n)$ expected time algorithm finding an induced k -cycle in graphs with an excluded minor is presented. We give evidence that establishing such an algorithm even for 2-degenerated graphs has far-reaching consequences.

Our Results. The main result of the paper is that the dominating set problem is fixed-parameter tractable for degenerated graphs. The running time is $k^{O(dk)}n$ for finding a dominating set of size k in a d -degenerated graph with n vertices. The algorithm is linear in the number of vertices of the graph, and we further improve the dependence on k for the following specific families of degenerated graphs. For graphs that do not contain K_h as a topological minor, an improved algorithm for the problem with running time $(O(h))^{hk}n$ is established. For graphs which are K_h -minor-free, the running time obtained is $(O(\log h))^{hk/2}n$. We show that all the algorithms can be generalized to the weighted case in the following sense. A dominating set of size at most k having minimum weight can be found within the same time bounds.

We address two open questions raised by Cai, Chan and Chan in [8] concerning linear time algorithms for finding an induced cycle in degenerated graphs. An $O(n)$ expected time algorithm for finding an induced k -cycle in graphs with an excluded minor is presented. The derandomization performed in [8] is improved and we get a deterministic $O(n \log n)$ time algorithm for the problem. As for finding induced cycles in degenerated graphs, we show a deterministic $O(n)$ time algorithm for finding cycles of size at most 5, and also explain why this is unlikely to be possible to achieve for longer cycles.

Techniques. We generalize the known search tree algorithms for the dominating set problem. This is enabled by proving some combinatorial lemmas, which are interesting in their own right. For degenerated graphs, we bound the number of vertices that dominate many elements of a given set, whereas for graphs with an excluded minor, our interest is in vertices that still need to be dominated and have a small degree.

The algorithm for finding an induced cycle in non-trivial minor-closed families is based on random separation and color-coding. Its derandomization is performed using known explicit constructions of families of (generalized) perfect hash functions.

2 Preliminaries

The paper deals with undirected and simple graphs, unless stated otherwise. Generally speaking, we will follow the notation used in [7] and [11]. For an undirected graph $G = (V, E)$ and a vertex $v \in V$, $N(v)$ denotes the set of all vertices adjacent to v (not including v itself). We say that v *dominates* the vertices of $N(v) \cup \{v\}$. The graph obtained from G by deleting v is denoted $G - v$. The subgraph of G induced by some set $V' \subseteq V$ is denoted by $G[V']$.

A graph G is *d-degenerated* if every induced subgraph of G has a vertex of degree at most d . It is easy and known that every d -degenerated graph $G = (V, E)$ admits an acyclic orientation such that the outdegree of each vertex is at most d . Such an orientation can be found in $O(|E|)$ time. A d -degenerated graph with n vertices has less than dn edges and therefore its average degree is less than $2d$.

For a directed graph $D = (V, A)$ and a vertex $v \in V$, the set of out-neighbors of v is denoted by $N^+(v)$. For a set $V' \subseteq V$, the notation $N^+(V')$ stands for the set of all vertices that are out-neighbors of at least one vertex of V' . For a directed graph $D = (V, A)$ and a vertex $v \in V$, we define $N_1^+(v) = N^+(v)$ and $N_i^+(v) = N^+(N_{i-1}^+(v))$ for $i \geq 2$.

An edge is said to be *subdivided* when it is deleted and replaced by a path of length two connecting its ends, the internal vertex of this path being a new vertex. A *subdivision* of a graph G is a graph that can be obtained from G by a sequence of edge subdivisions. If a subdivision of a graph H is the subgraph of another graph G , then H is a *topological minor* of G . A graph H is called a *minor* of a graph G if it can be obtained from a subgraph of G by a series of edge contractions.

In the parameterized dominating set problem, we are given an undirected graph $G = (V, E)$, a parameter k , and need to find a set of at most k vertices that dominate all the other vertices. Following the terminology of [2], the following generalization of the problem is considered. The input is a *black and white graph*, which simply means that the vertex set V of the graph G has been partitioned into two disjoint sets B and W of black and white vertices, respectively, i.e., $V = B \uplus W$, where \uplus denotes disjoint set union. Given a black and white graph $G = (B \uplus W, E)$ and an integer k , the problem is to find a set of at most k vertices that dominate the black vertices. More formally, we ask whether there

is a subset $U \subseteq B \uplus W$, such that $|U| \leq k$ and every vertex $v \in B - U$ satisfies $N(v) \cap U \neq \emptyset$. Finally we give a new definition, specific to this paper, for what it means to be a *reduced* black and white graph.

Definition 1. A black and white graph $G = (B \uplus W, E)$ is called *reduced* if it satisfies the following conditions:

- W is an independent set.
- All the vertices of W have degree at least 2.
- $N(w_1) \neq N(w_2)$ for every two distinct vertices $w_1, w_2 \in W$.

3 Algorithms for the Dominating Set Problem

3.1 Degenerated Graphs

The algorithm for degenerated graphs is based on the following combinatorial lemma.

Lemma 1. Let $G = (B \uplus W, E)$ be a d -degenerated black and white graph. If $|B| > (4d + 2)k$, then there are at most $(4d + 2)k$ vertices in G that dominate at least $|B|/k$ vertices of B .

Proof. Denote $R = \{v \in B \cup W \mid |(N_G(v) \cup \{v\}) \cap B| \geq |B|/k\}$. By contradiction, assume that $|R| > (4d + 2)k$. The induced subgraph $G[R \cup B]$ has at most $|R| + |B|$ vertices and at least $\frac{|R|}{2} \cdot (\frac{|B|}{k} - 1)$ edges. The average degree of $G[R \cup B]$ is thus at least

$$\frac{|R|(|B| - k)}{k(|R| + |B|)} \geq \frac{\min\{|R|, |B|\}}{2k} - 1 > 2d.$$

This contradicts the fact that $G[R \cup B]$ is d -degenerated. □

Theorem 1. There is a $k^{O(dk)}n$ time algorithm for finding a dominating set of size at most k in a d -degenerated black and white graph with n vertices that contains such a set.

Proof. The pseudocode of algorithm *DominatingSetDegenerated*(G, k) that solves this problem appears below. If there is indeed a dominating set of size at most k , then this means that we can split B into k disjoint pieces (some of them can be empty), so that each piece has a vertex that dominates it. If $|B| \leq (4d + 2)k$, then there are at most $k^{(4d+2)k}$ ways to divide the set B into k disjoint pieces. For each such split, we can check in $O(kdn)$ time whether every piece is dominated by a vertex. If $|B| > (4d + 2)k$, then it follows from Lemma 1 that $|R| \leq (4d + 2)k$. This means that the search tree can grow to be of size at most $(4d + 2)^k k!$ before possibly reaching the previous case. This gives the needed time bound. □

Algorithm 1. *DominatingSetDegenerated*(G, k)

Input: Black and white d -degenerated graph $G = (B \uplus W, E)$, integers k, d
Output: A set dominating all vertices of B of size at most k or *NONE* if no such set exists

```

if  $B = \emptyset$  then
   $\perp$  return  $\emptyset$ 
else if  $k = 0$  then
   $\perp$  return NONE
else if  $|B| \leq (4d + 2)k$  then
   $\perp$  forall possible ways of splitting  $B$  into  $k$  (possibly empty) disjoint pieces
     $B_1, \dots, B_k$  do
     $\perp$  if each piece  $B_i$  has a vertex  $v_i$  that dominates it then
       $\perp$  return  $\{v_1, \dots, v_k\}$ 
     $\perp$  return NONE
else
   $R \leftarrow \{v \in B \cup W \mid |(N_G(v) \cup \{v\}) \cap B| \geq |B|/k\}$ 
  forall  $v \in R$  do
     $\perp$  Create a new graph  $G'$  from  $G$  by marking all the elements of  $N_G(v)$  as
      white and removing  $v$  from the graph
     $D \leftarrow \text{DominatingSetDegenerated}(G', k - 1)$ 
    if  $D \neq \text{NONE}$  then
       $\perp$  return  $D \cup \{v\}$ 
   $\perp$  return NONE

```

3.2 Graphs with an Excluded Minor

Graphs with either an excluded minor or with no topological minor are known to be degenerated. We will apply the following useful propositions.

Proposition 1. [6,18] *There exists a constant c such that, for every h , every graph that does not contain K_h as a topological minor is ch^2 -degenerated.*

Proposition 2. [19,21,22] *There exists a constant c such that, for every h , every graph with no K_h minor is $ch\sqrt{\log h}$ -degenerated.*

The following lemma gives an upper bound on the number of cliques of a prescribed fixed size in a degenerated graph.

Lemma 2. *If a graph G with n vertices is d -degenerated, then for every $k \geq 1$, G contains at most $\binom{d}{k-1}n$ copies of K_k .*

Proof. By induction on n . For $n = 1$ this is obviously true. In the general case, let v be a vertex of degree at most d . The number of copies of K_k that contain v is at most $\binom{d}{k-1}$. By the induction hypothesis, the number of copies of K_k in $G - v$ is at most $\binom{d}{k-1}(n - 1)$. \square

We can now prove our main combinatorial results.

Theorem 2. *There exists a constant $c > 0$, such that for every reduced black and white graph $G = (B \uplus W, E)$, if G does not contain K_h as a topological minor, then there exists a vertex $b \in B$ of degree at most $(ch)^h$.*

Proof. Denote $|B| = n > 0$ and $d = ch^2$ where c is the constant from Proposition 1. Consider the vertices of W in some arbitrary order. For each such vertex $w \in W$, if there exist two vertices $b_1, b_2 \in N(w)$, such that b_1 and b_2 are not connected, add the edge $\{b_1, b_2\}$ and remove the vertex w from the graph. Denote the resulting graph $G' = (B \uplus W', E')$. Obviously, $G'[B]$ does not contain K_h as a topological minor and therefore has at most dn edges. The number of edges in the induced subgraph $G'[B]$ is at least the number of white vertices that were deleted from the graph, which means that at most dn were deleted so far.

We now bound $|W'|$, the number of white vertices in G' . It follows from the definition of a reduced black and white graph that there are no white vertices in G' of degree smaller than 2. The graph G' cannot contain a white vertex of degree $h - 1$ or more, since this would mean that the original graph G contained a subdivision of K_h . Now let w be a white vertex of G' of degree k , where $2 \leq k \leq h - 2$. The reason why w was not deleted during the process of generating G' is because $N(w)$ is a clique of size k in $G'[B]$. The graph G' is a reduced black and white graph, and therefore $N(w_1) \neq N(w_2)$ for every two different white vertices w_1 and w_2 . This means that the neighbors of each white vertex induce a different clique in $G'[B]$. By applying Lemma 2 to $G'[B]$, we get that the number of white vertices of degree k in G' is at most $\binom{d}{k-1}n$. This means that $|W'| \leq \left[\binom{d}{1} + \binom{d}{2} + \dots + \binom{d}{h-3} \right] n$. We know that $|W| \leq |W'| + dn$ and therefore $|E| \leq d(|B| + |W|) \leq d \left[3d + \binom{d}{2} + \dots + \binom{d}{h-3} \right] n$. Obviously, there exists a black vertex of degree at most $2|E|/n$. The result now follows by plugging the value of d and using the fact that $\binom{n}{k} \leq \left(\frac{en}{k}\right)^k$. □

Theorem 3. *There exists a constant $c > 0$, such that for every reduced black and white graph $G = (B \uplus W, E)$, if G is K_h -minor-free, then there exists a vertex $b \in B$ of degree at most $(c \log h)^{h/2}$.*

Proof. We proceed as in the proof of Theorem 2 using Proposition 2 instead of Proposition 1. □

Theorem 4. *There is an $O(h)^{hk}n$ time algorithm for finding a dominating set of size at most k in a black and white graph with n vertices and no K_h as a topological minor.*

Proof. The pseudocode of algorithm *DominatingSetNoMinor*(G, k) that solves this problem appears below. Let the input be a black and white graph $G = (B \uplus W, E)$. It is important to notice that the algorithm removes vertices and edges in order to get a (nearly) reduced black and white graph. This can be done in time $O(|E|)$ by a careful procedure based on the proof of Theorem 2 combined with radix sorting. We omit the details which will appear in the full version of the paper. The time bound for the algorithm now follows from Theorem 2. □

Algorithm 2. *DominatingSetNoMinor*(G, k)

Input: Black and white (K_h -minor-free) graph $G = (B \uplus W, E)$, integer k
Output: A set dominating all vertices of B of size at most k or *NONE* if no such set exists

```

if  $B = \emptyset$  then
   $\perp$  return  $\emptyset$ 
else if  $k = 0$  then
   $\perp$  return NONE
else
  Remove all edges of  $G$  whose two endpoints are in  $W$ 
  Remove all white vertices of  $G$  of degree 0 or 1
  As long as there are two different vertices  $w_1, w_2 \in W$  with
   $N(w_1) = N(w_2)$ ,  $|N(w_1)| < h - 1$ , remove one of them from the graph
  Let  $b \in B$  be a vertex of minimum degree among all vertices in  $B$ 
  forall  $v \in N_G(b) \cup \{b\}$  do
    Create a new graph  $G'$  from  $G$  by marking all the elements of  $N_G(v)$  as
    white and removing  $v$  from the graph
     $D \leftarrow \text{DominatingSetNoMinor}(G', k - 1)$ 
    if  $D \neq \text{NONE}$  then
       $\perp$  return  $D \cup \{v\}$ 
   $\perp$  return NONE

```

Theorem 5. *There is an $(O(\log h))^{hk/2}n$ time algorithm for finding a dominating set of size at most k in a black and white graph with n vertices which is K_h -minor-free.*

Proof. The proof is analogous to that of Theorem 4 using Theorem 3 instead of Theorem 2. □

3.3 The Weighted Case

In the weighted dominating set problem, each vertex of the graph has some positive real weight. The goal is to find a dominating set of size at most k , such that the sum of the weights of all the vertices of the dominating set is as small as possible. The algorithms we presented can be generalized to deal with the weighted case without changing the time bounds. In this case, the whole search tree needs to be scanned and one cannot settle for the first valid solution found.

Let $G = (B \uplus W, E)$ be the input graph to the algorithm. In algorithm 1 for degenerated graphs, we need to address the case where $|B| \leq (4d + 2)k$. In this case, the algorithm scans all possible ways of splitting B into k disjoint pieces B_1, \dots, B_k , and it has to be modified, so that it will always choose a vertex with minimum weight that dominates each piece. In algorithm 2 for graphs with an excluded minor, the criterion for removing white vertices from the graph is modified so that whenever two vertices $w_1, w_2 \in W$ satisfy $N(w_1) = N(w_2)$, the vertex with the bigger weight is removed.

4 Finding Induced Cycles

4.1 Degenerated Graphs

Recall that $N_i^+(v)$ is the set of all vertices that can be reached from v by a directed path of length exactly i . If the outdegree of every vertex in a directed graph $D = (V, A)$ is at most d , then obviously $|N_i^+(v)| \leq d^i$ for every $v \in V$ and $i \geq 1$.

Theorem 6. *For every fixed $d \geq 1$ and $k \leq 5$, there is a deterministic $O(n)$ time algorithm for finding an **induced** cycle of length k in a d -degenerated graph on n vertices.*

Proof. Given a d -degenerated graph $G = (V, E)$ with n vertices, we orient the edges so that the outdegree of all vertices is at most d . This can be done in time $O(|E|)$. Denote the resulting directed graph $D = (V, A)$. We can further assume that $V = \{1, 2, \dots, n\}$ and that every directed edge $\{u, v\} \in A$ satisfies $u < v$. This means that an out-neighbor of a vertex u will always have an index which is bigger than that of u . We now describe how to find cycles of size at most 5.

To find cycles of size 3 we simply check for each vertex v whether $N^+(v) \cap N_2^+(v) \neq \emptyset$. Suppose now that we want to find a cycle $v_1 - v_2 - v_3 - v_4 - v_1$ of size 4. Without loss of generality, assume that $v_1 < v_2 < v_4$. We distinguish between two possible cases.

- $v_1 < v_3 < v_2 < v_4$: Keep two counters C_1 and C_2 for each pair of vertices. For every vertex $v \in V$ and every unordered pair of distinct vertices $u, w \in N^+(v)$, such that u and w are not connected, we raise the counter $C_1(\{u, w\})$ by one. In addition to that, for every vertex $x \in N^+(v)$ such that $u, w \in N^+(x)$, the counter $C_2(\{u, w\})$ is incremented. After completing this process, we check whether there are two vertices for which $(C_1(\{u, w\}) - C_2(\{u, w\})) > 0$. This would imply that an induced 4-cycle was found.
- $v_1 < v_2 < v_3 < v_4$ or $v_1 < v_2 < v_4 < v_3$: Check for each vertex v whether the set $\{v\} \cup N^+(v) \cup N_2^+(v) \cup N_3^+(v)$ contains an induced cycle.

To find an induced cycle of size 5, a more detailed case analysis is needed. It is easy to verify that such a cycle has one of the following two types.

- There is a vertex v such that $\{v\} \cup N^+(v) \cup N_2^+(v) \cup N_3^+(v) \cup N_4^+(v)$ contains the induced cycle.
- The cycle is of the form $v - x - u - y - w - v$, where $x \in N^+(v)$, $u \in N^+(x) \cap N^+(y)$, and $w \in N^+(v) \cap N^+(y)$. The induced cycle can be found by defining counters in a similar way to what was done before. We omit the details. □

The following simple lemma shows that a linear time algorithm for finding an induced C_6 in a 2-degenerated graph would imply that a triangle (a C_3) can be found in a general graph in $O(|V| + |E|) \leq O(|V|^2)$ time. It is a long standing open question to improve the natural $O(|V|^\omega)$ time algorithm for this problem [16].

Lemma 3. *Given a linear time algorithm for finding an induced C_6 in a 2-degenerated graph, it is possible to find triangles in general graphs in $O(|V|+|E|)$ time.*

Proof. Given a graph $G = (V, E)$, subdivide all the edges. The new graph obtained G' is 2-degenerated and has $|V| + |E|$ vertices. A linear time algorithm for finding an induced C_6 in G' actually finds a triangle in G . By assumption, the running time is $O(|V| + |E|) \leq O(|V|^2)$. \square

4.2 Minor-Closed Families of Graphs

Theorem 7. *Suppose that G is a graph with n vertices taken from some non-trivial minor-closed family of graphs. For every fixed k , if G contains an **induced** cycle of size k , then it can be found in $O(n)$ expected time.*

Proof. There is some absolute constant d , so that G is d -degenerated. Orient the edges so that the maximum outdegree is at most d and denote the resulting graph $D = (V, E)$. We now use the technique of random separation. Each vertex $v \in V$ of the graph is independently removed with probability $1/2$, to get some new directed graph D' . Now examine some (undirected) induced cycle of size k in the original directed graph D , and denote its vertices by U . The probability that all the vertices in U remained in the graph and all vertices in $N^+(U) - U$ were removed from the graph is at least $2^{-k(d+1)}$.

We employ the color-coding method to the graph D' . Choose a random coloring of the vertices of D' with the k colors $\{1, 2, \dots, k\}$. For each vertex v colored i , if $N^+(v)$ contains a vertex with a color which is neither $i - 1$ nor $i + 1 \pmod k$, then it is removed from the graph. For each induced cycle of size k , its vertices will receive distinct colors and it will remain in the graph with probability at least $2k^{1-k}$.

We now use the $O(n)$ time algorithm from [4] to find a multicolored cycle of length k in the resulting graph. If such a cycle exists, then it must be an induced cycle. Since k and d are constants, the algorithm succeeds with some small constant probability and the expected running time is as needed. \square

The next theorem shows how to derandomize this algorithm while incurring a loss of only $O(\log n)$.

Theorem 8. *Suppose that G is a graph with n vertices taken from some non-trivial minor-closed family of graphs. For every fixed k , there is an $O(n \log n)$ time deterministic algorithm for finding an **induced** cycle of size k in G .*

Proof. Denote $G = (V, E)$ and assume that G is d -degenerated. We derandomize the algorithm in Theorem 7 using an $(n, dk + k)$ -family of perfect hash functions. This is a family of functions from $[n]$ to $[dk + k]$ such that for every $S \subseteq [n]$, $|S| = dk + k$, there exists a function in the family that is 1-1 on S . Such a family of size $e^{dk+k}(dk + k)^{O(\log(dk+k))} \log n$ can be efficiently constructed [20]. We think of each function as a coloring of the vertices with the $dk + k$ colors

$C = \{1, 2, \dots, dk + k\}$. For every combination of a coloring, a subset $L \subseteq C$ of k colors and a bijection $f : L \rightarrow \{1, 2, \dots, k\}$ the following is performed. All the vertices that got a color from $c \in L$ now get the color $f(c)$. The other vertices are removed from the graph.

The vertices of the resulting graph are colored with the k colors $\{1, 2, \dots, k\}$. Examine some induced cycle of size k in the original graph, and denote its vertices by U . There exists some coloring c in the family of perfect hash functions for which all the vertices in $U \cup N^+(U)$ received different colors. Now let L be the k colors of the vertices in the cycle U and let $f : L \rightarrow [k]$ be the bijection that gives consecutive colors to vertices along the cycle. This means that for this choice of c , L , and f , the induced cycle U will remain in the graph as a multicolored cycle, whereas all the vertices in $N^+(U) - U$ will be removed from the graph.

We proceed as in the previous algorithm. Better dependence on the parameters d and k can be obtained using the results in [3]. \square

5 Concluding Remarks

- The algorithm for finding a dominating set in graphs with an excluded minor, presented in this paper, generalizes and improves known algorithms for planar graphs and graphs with bounded genus. We believe that similar techniques may be useful in improving and simplifying other known fixed-parameter algorithms for graphs with an excluded minor.
- An interesting open problem is to decide whether there is a $2^{O(\sqrt{k})}n^c$ time algorithm for finding a dominating set of size k in graphs with n vertices and an excluded minor, where c is some absolute constant that does not depend on the excluded graph. Maybe even a $2^{O(\sqrt{k})}n$ time algorithm can be achieved.

References

1. Alber, J., Bodlaender, H.L., Fernau, H., Kloks, T., Niedermeier, R.: Fixed parameter algorithms for DOMINATING SET and related problems on planar graphs. *Algorithmica* 33(4), 461–493 (2002)
2. Alber, J., Fan, H., Fellows, M.R., Fernau, H., Niedermeier, R., Rosamond, F.A., Stege, U.: A refined search tree technique for dominating set on planar graphs. *J. Comput. Syst. Sci.* 71(4), 385–405 (2005)
3. Alon, N., Cohen, G.D., Krivelevich, M., Litsyn, S.: Generalized hashing and parent-identifying codes. *J. Comb. Theory, Ser. A* 104(1), 207–215 (2003)
4. Alon, N., Yuster, R., Zwick, U.: Color-coding. *Journal of the ACM* 42(4), 844–856 (1995)
5. Alon, N., Yuster, R., Zwick, U.: Finding and counting given length cycles. *Algorithmica* 17(3), 209–223 (1997)
6. Bollobás, B., Thomason, A.: Proof of a conjecture of Mader, Erdős and Hajnal on topological complete subgraphs. *Eur. J. Comb.* 19(8), 883–887 (1998)
7. Bondy, J.A., Murty, U.S.R.: Graph theory with applications. American Elsevier Publishing, New York (1976)

8. Cai, L., Chan, S.M., Chan, S.O.: Random separation: A new method for solving fixed-cardinality optimization problems. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 239–250. Springer, Heidelberg (2006)
9. Demaine, E.D., Fomin, F.V., Hajiaghayi, M., Thilikos, D.M.: Fixed-parameter algorithms for (k, r) -center in planar graphs and map graphs. *ACM Transactions on Algorithms* 1(1), 33–47 (2005)
10. Demaine, E.D., Fomin, F.V., Hajiaghayi, M., Thilikos, D.M.: Subexponential parameterized algorithms on bounded-genus graphs and H -minor-free graphs. *Journal of the ACM* 52(6), 866–893 (2005)
11. Diestel, R.: *Graph theory*, 3rd edn. Graduate Texts in Mathematics, vol. 173. Springer, Heidelberg (2005)
12. Downey, R.G., Fellows, M.R.: *Parameterized complexity*. Monographs in Computer Science. Springer, Heidelberg (1999)
13. Ellis, J.A., Fan, H., Fellows, M.R.: The dominating set problem is fixed parameter tractable for graphs of bounded genus. *J. Algorithms* 52(2), 152–168 (2004)
14. Flum, J., Grohe, M.: *Parameterized complexity theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2006)
15. Fomin, F.V., Thilikos, D.M.: Dominating sets in planar graphs: branch-width and exponential speed-up. In: *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 168–177. ACM Press, New York (2003)
16. Itai, A., Rodeh, M.: Finding a minimum circuit in a graph. *SIAM Journal on Computing* 7(4), 413–423 (1978)
17. Kanj, I.A., Perkovic, L.: Improved parameterized algorithms for planar dominating set. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 399–410. Springer, Heidelberg (2002)
18. Komlós, J., Szemerédi, E.: Topological cliques in graphs II. *Combinatorics, Probability & Computing* 5, 79–90 (1996)
19. Kostochka, A.V.: Lower bound of the Hadwiger number of graphs by their average degree. *Combinatorica* 4(4), 307–316 (1984)
20. Naor, M., Schulman, L.J., Srinivasan, A.: Splitters and near-optimal derandomization. In: *36th Annual Symposium on Foundations of Computer Science*, pp. 182–191 (1995)
21. Thomason, A.: An extremal function for contractions of graphs. *Math. Proc. Cambridge Philos. Soc.* 95(2), 261–265 (1984)
22. Thomason, A.: The extremal function for complete minors. *J. Comb. Theory, Ser. B* 81(2), 318–338 (2001)
23. Yuster, R., Zwick, U.: Finding even cycles even faster. *SIAM Journal on Discrete Mathematics* 10(2), 209–222 (1997)
24. Yuster, R., Zwick, U.: Detecting short directed cycles using rectangular matrix multiplication and dynamic programming. In: *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 254–260 (2004)

Single-Edge Monotonic Sequences of Graphs and Linear-Time Algorithms for Minimal Completions and Deletions^{*}

Pinar Heggernes and Charis Papadopoulos

Department of Informatics, University of Bergen, N-5020 Bergen, Norway
{pinar,charis}@ii.uib.no

Abstract. We study graph properties that admit an increasing, or equivalently decreasing, sequence of graphs on the same vertex set such that for any two consecutive graphs in the sequence their difference is a single edge. This is useful for characterizing and computing minimal completions and deletions of arbitrary graphs into having these properties. We prove that threshold graphs and chain graphs admit such sequences. Based on this characterization and other structural properties, we present linear-time algorithms both for computing minimal completions and deletions into threshold, chain, and bipartite graphs, and for extracting a minimal completion or deletion from a given completion or deletion. Minimum completions and deletions into these classes are NP-hard to compute.

1 Introduction

A graph property \mathcal{P} is called *monotone* if it is closed under any edge or vertex removal. Equivalently, a property is monotone if it is closed under taking subgraphs that are not necessarily induced. A property is *hereditary* if it is closed under taking induced subgraphs. Every monotone property is hereditary but the converse is not true. For example bipartiteness and planarity are monotone properties, as they are characterized through forbidden subgraphs that are not necessarily induced, whereas perfectness is a hereditary but not monotone property. Some of the most well-studied graph properties are monotone [13] or hereditary [14]. If a given monotone (hereditary) property is equivalent to belonging to a graph class then this graph class is called monotone (hereditary).

In this work, we introduce *sandwich monotonicity* of graph properties and graph classes. We say that a graph property \mathcal{P} is *sandwich monotone* if \mathcal{P} satisfies the following: Given two graphs $G_1 = (V, E)$ and $G_2 = (V, E \cup F)$ that both satisfy \mathcal{P} , the edges in F can be ordered $f_1, f_2, \dots, f_{|F|}$ such that when single edges of F are added to G_1 one by one in this order (or removed from G_2 in the reverse order), the graph obtained at each step satisfies \mathcal{P} . Every monotone property is clearly sandwich monotone as well. However every hereditary

^{*} This work is supported by the Research Council of Norway through grant 166429/V30.

property is not necessarily sandwich monotone. Here, we are interested in identifying non-monotone hereditary graph classes that are sandwich monotone. Until now, sandwich monotonicity of only two such classes has been shown: chordal graphs [28] and split graphs [15], and it has been an open question which other graph classes are sandwich monotone [2]. Simple examples exist to show that the hereditary classes of perfect graphs, comparability graphs, permutation graphs, cographs, trivially perfect graphs, and interval graphs are not sandwich monotone. In this extended abstract we show that threshold graphs and chain graphs are sandwich monotone.

Our main motivation for studying sandwich monotonicity comes from the problem of adding edges to or deleting edges from a given arbitrary graph so that it satisfies a desired property. For example, a *chordal completion* is a chordal supergraph on the same vertex set, and a *bipartite deletion* is a spanning bipartite subgraph of an arbitrary graph. Completions and deletions into other graph classes are defined analogously. A completion (deletion) is *minimum* if it has the smallest possible number of added (deleted) edges. Unfortunately minimum completions and deletions into most interesting graph classes, including threshold, chain, and bipartite graphs, are NP-hard to compute [13,7,22,25,29]. However, minimum completions (deletions) are a subset of minimal completions (deletions), and hence we can search for minimum among the set of minimal. A completion (deletion) is *minimal* if no subset of the added (deleted) edges is enough to give the desired property when added to (deleted from) the input graph.

Given as input an arbitrary graph, there are two problems related to minimal completions (deletions) : 1. Computing a minimal completion (deletion) of the given input graph into the desired graph class, 2. Extracting a minimal completion (deletion) which is a subgraph (supergraph) of a given arbitrary completion (deletion) of the input graph into the desired class. A solution of problem 2 requires a characterization of minimal completions (deletions) into a given class, and readily gives a solution of problem 1. A solution of problem 1 might generate only a subset of all possible minimal completions (deletions), and does not necessarily solve problem 2. Solution of problem 2 in polynomial time is known only for completions into chordal [4], split [15], and interval [18] graphs, and for deletions into chordal [9], split [16], and planar [10,20] graphs. As an example of usefulness of a solution of problem 2, various characterizations of minimal chordal completions [21,6] have made it possible to design approximation algorithms [25] and fast exact exponential time algorithms [11] for computing minimum chordal completions. A solution of problem 2 also allows the computation of minimal completions that are not far from minimum in practice [5].

For a graph property \mathcal{P} that is sandwich monotone, problem 2 of extracting a minimal completion from a given completion of an input graph into \mathcal{P} has always a polynomial time solution if \mathcal{P} can be recognized in polynomial time. Given G and a supergraph G_2 of G satisfying \mathcal{P} , if G_2 is not a minimal completion, then a minimal completion G_1 exists sandwiched between G and G_2 . Hence by trying one by one all edges of G_2 that are not in G for removal, one obtains

a minimal extraction algorithm with a number of iterations that is quadratic in the number of edges appearing in G_2 but not in G . Similarly, problem 2 of extracting a minimal deletion from a given deletion can also be solved with a number of iterations that is quadratic in the number of deleted edges.

In this extended abstract, by showing that threshold graphs and chain graphs are sandwich monotone, we thus establish that minimal completions and deletions of arbitrary graphs into these graph classes can be computed in polynomial time. Even more interesting, we give linear-time algorithms for minimal completions into threshold graphs and minimal deletions into bipartite and chain graphs. This does not follow from sandwich monotonicity in general. Furthermore, we solve the extraction version of these problems (problem 2 above) in linear time for these graph classes. Linear-time minimal completion algorithms have been known only into two classes previously; split [15] and proper interval [27] graphs. The only linear-time minimal extraction algorithm known is the one into split graphs [15]. Linear-time minimal deletion algorithms are known only into split [16] and planar graphs (which are monotone) [10,20].

Notation and Terminology: We consider simple undirected graphs. For a graph $G = (V, E)$, $V(G) = V$ and $E(G) = E$, with $n = |V|$ and $m = |E|$. For $S \subseteq V$, the *subgraph of G induced by S* is denoted by $G[S]$. Moreover, we denote by $G - S$ the graph $G[V \setminus S]$ and by $G - v$ the graph $G[V \setminus \{v\}]$. We distinguish between *subgraphs* and *induced subgraphs*. By a *subgraph* of G we mean a graph G' on the same vertex set containing a subset of the edges of G , and we denote it by $G' \subseteq G$. If G' contains a proper subset of the edges of G , we write $G' \subset G$. We write $G - uv$ to denote the graph $(V, E \setminus \{uv\})$.

The *neighborhood* of a vertex x of G is $N_G(x) = \{v \mid xv \in E\}$. The *degree* of x in G is $d_G(x)$. For $S \subseteq V$ $N_G(S) = \bigcup_{x \in S} N_G(x) \setminus S$. The size of a largest clique in G is $\omega(G)$. A chordless cycle on k vertices is denoted by C_k and a chordless path on k vertices is denoted by P_k . The graph consisting of only two disjoint edges is denoted by $2K_2$. The *complement* \overline{G} of a graph G consists of all vertices and all non-edges of G . For a graph class \mathcal{C} , the class *co- \mathcal{C}* is the set of graphs \overline{G} for which $G \in \mathcal{C}$. A class \mathcal{C} is *self-complementary* if $\mathcal{C} = \text{co-}\mathcal{C}$.

Given an arbitrary graph $G = (V, E)$ and a graph class \mathcal{C} , a \mathcal{C} *completion* of G is a graph $H = (V, E \cup F)$ such that $H \in \mathcal{C}$, and H is a *minimal \mathcal{C} completion* of G if $(V, E \cup F')$ fails to be in \mathcal{C} for every $F' \subset F$. Similarly, $H = (V, E \setminus D)$ is a \mathcal{C} *deletion* of G if $H \in \mathcal{C}$, and H is minimal if $(V, E \setminus D')$ fails to be in \mathcal{C} for every $D' \subset D$. The edges added to the original graph in order to obtain a completion are called *fill edges*, and the edges removed from the original graph in order to obtain a deletion are called *deleted edges*.

2 Minimal Completions and Deletions into Sandwich Monotone Graph Classes

Definition 1. *A graph class \mathcal{C} is sandwich monotone if the following is true for any pair of graphs $G = (V, E)$ and $H = (V, E \cup F)$ in \mathcal{C} with $E \cap F = \emptyset$: There is*

an ordering $f_1, f_2, \dots, f_{|F|}$ of the edges in F such that in the sequence of graphs $G = G_0, G_1, \dots, G_{|F|} = H$, where G_{i-1} is obtained by removing edge f_i from G_i , (or equivalently, G_i is obtained by adding edge f_i to G_{i-1}), every graph belongs to \mathcal{C} .

Minimal completions and deletions into sandwich monotone graph classes have the following algorithmically useful characterization, as observed on chordal graphs [28] and split graphs [15] previously.

Observation 1. *Let \mathcal{C} be a graph class and let \mathcal{P} be the property of belonging to \mathcal{C} . The following are equivalent:*

- (i) \mathcal{C} is sandwich monotone.
- (ii) $\text{co-}\mathcal{C}$ is sandwich monotone.
- (iii) A \mathcal{C} completion is minimal if and only if no single fill edge can be removed without destroying the property \mathcal{P} .
- (iv) A \mathcal{C} deletion is minimal if and only if no single deleted edge can be added without destroying the property \mathcal{P} .

Observation 2. *Let \mathcal{C} be a sandwich monotone graph class. Given a polynomial time algorithm for the recognition of \mathcal{C} , there is a polynomial time algorithm for extracting a minimal \mathcal{C} completion (deletion) of an arbitrary graph G from any given \mathcal{C} completion (deletion) of G .*

Even though we know that minimal \mathcal{C} completions and deletions can be computed in polynomial time for a sandwich monotone graph class \mathcal{C} , the actual running time is not necessarily practical. In the following sections, we give linear-time algorithms for computing and extracting minimal completions into threshold, and deletions into bipartite and chain graphs.

For the sandwich monotone graph classes previously studied for completions and deletions, linear-time algorithms exist for computing and extracting minimal split completions [15] and deletions [16], and minimal planar deletions [10,20]. As a comparison, although chordal graphs are sandwich monotone and they can be recognized in linear time, the best known running time is $O(n^{2.376})$ for a minimal chordal completion algorithm [19], and $O(\Delta m)$ for a minimal chordal deletion algorithm [9], where Δ is the largest degree in the input graph.

Minimal Bipartite Deletions: A graph is *bipartite* if its vertex set can be partitioned into two independent sets, called a *bipartition*. The bipartition of a bipartite graph is unique if and only if the graph is connected. Bipartite graphs are exactly the class of graphs that do not contain cycles of odd length. It is well known that simple modifications of breadth-first search (BFS) can be used to find an odd cycle in a graph or provide a bipartition of it in linear time. Bipartite graphs are monotone, and hence also sandwich monotone. Computing minimum bipartite deletions is NP-hard problem [13].

If an arbitrary input graph G is connected then there exists a connected bipartite deletion of G , since G has a spanning tree, and trees are bipartite. If G is not connected, then the following result can be applied to each connected component of G separately.

Theorem 1. *Let $H = (V, E \setminus D)$ be a bipartite deletion of an arbitrary graph $G = (V, E)$ with $D \subseteq E$. A minimal bipartite deletion $H' = (V, E \setminus D')$ of G , such that $D' \subseteq D$ can be computed in $O(n + m)$ time.*

Corollary 1. *Any minimal bipartite deletion of an arbitrary graph can be computed in $O(n + m)$ time.*

3 Minimal Threshold Completions

A graph $G = (V, E)$ is called a *threshold graph* if there exist nonnegative real numbers w_v for $v \in V$, and t such that for every $I \subseteq V$, $\sum_{v \in I} w_v \leq t$ if and only if I is an independent set [8,14,23].

A graph is a *split graph* if its vertex set can be partitioned into a clique K and an independent set S , in which case (S, K) is a (not necessarily unique) *split partition* of the graph. Split graphs can be recognized and a split partition can be computed in linear time [14]. It is known that a graph is split if and only if it does not contain any vertex subset inducing $2K_2$, C_4 , or C_5 [12]. Hence the next theorem states that a graph is threshold if and only if it is split and does not contain any vertex set inducing a P_4 .

Theorem 2 ([8]). *A graph is a threshold graph if and only if it does not contain any vertex set inducing $2K_2$, C_4 , or P_4 .*

Consequently, in a disconnected threshold graph there is at most one connected component that contains an edge. An ordering $v_1, v_2, \dots, v_{|S|}$ of a subset $S \subseteq V(G)$ of vertices is called *nested neighborhood ordering* if it has the property that $(N_G(v_1) \setminus S) \subseteq (N_G(v_2) \setminus S) \subseteq \dots \subseteq (N_G(v_{|S|}) \setminus S)$.

Theorem 3 ([23]). *A graph is a threshold graph if and only if it is split and the vertices of the independent set have a nested neighborhood ordering in any split partition.*

Threshold graphs can be recognized and a nested neighborhood ordering can be computed in linear time, since sorting the vertices of the independent set by their degrees readily gives such an ordering for threshold graphs [14]. It is NP-hard to compute minimum threshold completions of arbitrary graphs; even split graphs [26].

A split partition of a threshold graph is never unique [14]. Here we define a *threshold partition* (S, K) of a threshold graph in a unique way. Note first that all vertices of degree more than $\omega(G) - 1$ must belong to K , and all vertices of degree less than $\omega(G) - 1$ must belong to S . The set of vertices of degree exactly $\omega(G) - 1$ is either an independent set or a clique [15]. If this set is a clique, we place all of these vertices in K , and if it is an independent set we place them in S . We refine the sets S and K further as follows: $(S_0, S_1, S_2, \dots, S_\ell)$ is a partition of S such that S_0 is the set of isolated vertices, and $N(S_1) \subset N(S_2) \subset \dots \subset N(S_\ell)$, where ℓ is as large as possible. Hence all vertices in S_i have the same degree for $0 \leq i \leq \ell$. This also defines a partition $(K_1, K_2, \dots, K_\ell)$ of K , where $K_1 = N(S_1)$

and $K_i = N(S_i) \setminus N(S_{i-1})$ for $2 \leq i \leq \ell$. The remaining vertices of K form another set $K_{\ell+1} = K \setminus N(S_\ell)$. Again, all vertices in K_i have the same degree for $1 \leq i \leq \ell + 1$. By definition, a graph is threshold if and only if its vertex set admits such a threshold partition. Moreover the threshold partition of a threshold graph is unique and all sets S_i and K_i , $1 \leq i \leq \ell$, are non-empty except possibly the sets S_0 and $K_{\ell+1}$. If $K_{\ell+1} = \emptyset$ then S_ℓ contains at least two vertices, and if $K_{\ell+1} \neq \emptyset$ then $K_{\ell+1}$ contains at least two vertices [14].

Lemma 1. *Let G be a threshold graph with threshold partition $((S_0, \dots, S_\ell), (K_1, \dots, K_{\ell+1}))$ and let uv be an edge satisfying the following: either $u \in S_i$ and $v \in K_i$ for some $i \in \{1, \dots, \ell\}$, or $u, v \in K_{\ell+1}$. Then $G - uv$ is a threshold graph.*

Proof. Assume first that $u \in S_i$ and $v \in K_i$ for some i . Removing uv from G results in a split graph since we remove an edge between the clique and the independent set of the split partition. In the graph $G' = G - uv$ we have that $N_{G'}(S_{i-1}) \subseteq N_{G'}(u) \subset N_{G'}(S_i \setminus \{u\})$. Thus the new independent set has a nested neighborhood ordering also in G' , and hence G' is threshold by Theorem 3.

Assume now that $u, v \in K_{\ell+1}$. We describe the threshold partition of G' : If $K_{\ell+1}$ contains more than two vertices then the new threshold partition has the sets (K_i, S_i) , $1 \leq i \leq \ell$, unchanged, and it also contains the new sets $K'_{\ell+1} = K_{\ell+1} \setminus \{u, v\}$ and $S'_{\ell+1} = \{u, v\}$. If $K_{\ell+1}$ has exactly two vertices, then every set remains as before, except $K_{\ell+1}$ which is removed and S_ℓ which now becomes $S'_\ell = S_\ell \cup \{u, v\}$. It is easy to check that removal of uv results in exactly the described threshold partition for G' and thus G' is a threshold graph.

For simplicity, we will call an edge uv of G that satisfies the condition in Lemma 1 a candidate edge of G .

Lemma 2. *Let $G = (V, E)$ and $G' = (V, E \cup F)$ be two threshold graphs such that $F \cap E = \emptyset$ and $F \neq \emptyset$. At least one edge in F is a candidate edge of G' .*

Proof. Let $((S'_0, \dots, S'_\ell), (K'_1, \dots, K'_{\ell+1}))$ be the threshold partition of G' . Assume for a contradiction that F does not contain any candidate edge, and let $uv \in F$. Since uv cannot be a candidate edge, it is of one of the following three types: (i) $u, v \in K'_i$, for some i satisfying $1 \leq i \leq \ell$, (ii) $u \in K'_i$ and $v \in K'_j$, for some i and j satisfying $1 \leq i < j \leq \ell + 1$, or (iii) $u \in K'_i$ and $v \in S'_j$, for some i and j satisfying $1 \leq i < j \leq \ell$. (Recall that there are no edges uv in G' with $u \in K'_i$ and $v \in S'_j$ where $j < i$ by the definition of threshold partition.)

If uv is of type (i), then since K'_i is non-empty, there is a vertex $x \in S'_i$ such that ux and uv are edges of G' . Since there are no candidate edges in F , $ux, uv \in E$. Assume first that $i \neq \ell$. Then there is a vertex $y \in K'_\ell$ such that uy and vy are edges of G' and xy is not an edge of G' . If both uy and vy belong to E , then $\{u, x, v, y\}$ induces a C_4 in G . If exactly one of uy and vy belongs to E , say uy , and the other belongs to F , then $\{y, u, x, v\}$ induces a P_4 in G . If both uy and vy belong to F , then there is a vertex $z \in S'_i$ such that $\{x, u, z, y\}$ induces a $2K_2$ in G , since zy is a candidate edge of G' and hence cannot be in F . Hence all possibilities lead to a contradiction since G is a threshold graph and

cannot contain any of the mentioned induced subgraphs. If $i = \ell$ and $K'_{\ell+1} \neq \emptyset$ then there are at least two vertices y and z in $K'_{\ell+1}$ that can substitute the role of y and z as in the case $i \neq \ell$, since yz is a candidate edge of G' and hence must belong to E . If $i = \ell$ and $K'_{\ell+1} = \emptyset$ then we know that there are at least two vertices $y, z \in S'_\ell$, and that $uy, uz, vy, vz \in E$ (since they are candidate edges). Hence $\{u, y, v, z\}$ induces a C_4 in G , contradicting that G is threshold.

If uv is of type (ii), assume first that $j \neq \ell + 1$. Then we know that there is a vertex $x \in S'_i$ and a vertex $y \in S'_j$ such that $ux, vy \in E$ since they are candidate edges. We know that xv is not an edge of G' . If $yu \in E$ then $\{x, u, y, v\}$ induces a P_4 in G , and if $yu \in F$ then the same set induces a $2K_2$ in G , contradicting in both cases that G is threshold. If $j = \ell + 1$ then we know that there is at least one more vertex $z \neq v$ in $K'_{\ell+1}$ where $vz \in E$ (since it is a candidate edge). If uz belongs to F then $\{v, z, u, x\}$ induces a $2K_2$ in G . Otherwise this set induces a P_4 in G , because zx and vx are not edges in G or G' , contradicting that G is threshold.

If uv is of type (iii), then we know that there are vertices $x \in S'_i$ and $y \in K'_j$ such that $ux, vy \in E$ by the same arguments as before. If $uy \in E$ then $\{x, u, y, v\}$ induces a P_4 in G , and if $uy \in F$, then this set induces a $2K_2$ in G . Hence by Theorem 2, we reach a contradiction in all possible cases. Consequently, either G is not threshold, or F must contain a candidate edge, and the proof is complete.

Theorem 4. *Threshold graphs are sandwich monotone.*

Proof. Given G and G' as in the premise of Lemma 2, we know by Lemmas 1 and 2 that there is an edge $f \in F$ such that $G' - f$ is threshold. Now the same argument can be applied to G and $G' = G' - f$ with $F = F \setminus \{f\}$ repeatedly, until G' becomes equal to G .

Theorem 5. *Let G be an arbitrary graph and H be a threshold completion of G . H is a minimal threshold completion of G if and only if no fill edge is a candidate edge of H .*

Proof. If H is a minimal threshold completion of G , then there cannot be any fill edge that is a candidate edge of H , because otherwise the removal of this edge would result in a threshold graph by Lemma 1, contradicting that H is minimal. If H is not a minimal threshold completion of G , then there exists a minimal threshold completion M of G such that $E(G) \subseteq E(M) \subset E(H)$. By Lemma 2 there is an edge $e \in E(H) \setminus E(M)$ that is a candidate edge of H .

Now given any threshold completion H of an arbitrary graph G , let us consider the problem of extracting a minimal threshold completion H' of G such that $G \subseteq H' \subseteq H$.

Theorem 6. *Let $H = (V, E \cup F)$ be a threshold completion of an arbitrary graph $G = (V, E)$ with $F \cap E = \emptyset$. A minimal threshold completion $H' = (V, E \cup F')$ of G , such that $F' \subseteq F$, can be computed in $O(n + m + |F|)$ time.*

For the proof of Theorem 6 we give Algorithm `Extr_Min_Threshold` which describes a way of computing H' .

Algorithm. Extr_Min_Threshold

Input: A graph $G = (V, E)$ and a threshold graph $H = (V, E \cup F)$ with $F \cap E = \emptyset$.

Output: A minimal threshold completion $H' = (V, E \cup F')$ of G such that $F' \subset F$.
 $H' = H$; Unmark all vertices of H' ;

Let $(S = (S_0, S_1, \dots, S_\ell), K = (K_1, K_2, \dots, K_{\ell+1}))$ be the threshold partition of H' ;

While there is an unmarked vertex v such that $d_{H'}(v) \leq \omega(H') - 1$ **do**

 Pick an unmarked vertex v of minimum $d_{H'}(v)$;

If $v \in S$ **then** compute the index j for which $v \in S_j$;

Else $j = \ell + 1$;

 Find a vertex $u \in N_G(v)$ of minimum $d_{H'}(u)$;

 Compute $U = \{u' \in N_G(v) \mid d_{H'}(u') = d_{H'}(u)\}$;

 Compute the index i for which $U \subseteq K_i$;

 Remove all edges between v and $(K_i \setminus U) \cup K_{i+1} \cup \dots \cup K_j$ from H' ;

 Update the threshold-partition of H' and mark v ;

Return H' ;

Next we show how to obtain a minimal threshold completion H of G directly. The motivation for this is that we now compute H in time linear in the size of G . The idea behind our approach is the following: Compute a minimal split completion of G using the algorithm of [15], and then compute a minimal threshold completion of the computed split completion by giving the vertices of the independent set a nested neighborhood ordering. Computing a minimal split completion G' with split partition (K, S) of G can be done in $O(n + m)$ time by the algorithm given in [15]. After that we compute an order of the vertices of S such that $d(v_1) \geq d(v_2) \geq \dots \geq d(v_{|S|})$. The important point is that $d_{G'}(v) = d_G(v)$ for each $v \in S$, since all fill edges of G' have both endpoints in K . Then for each vertex v_i we make it adjacent to the vertices of $N(\{v_{i+1}, v_{i+2}, \dots, v_{|S|}\})$, starting from v_1 . We show in the proof of the following theorem that this indeed results in a minimal threshold completion of G .

Theorem 7. *A minimal threshold completion of an arbitrary graph G can be computed in $O(n + m)$ time.*

4 Minimal Chain Deletions

Yannakakis introduced *chain graphs* and defined a bipartite graph to be a chain graph if one of the sides of the bipartition has nested neighborhood ordering [30]. He also showed that one side has this property if and only if both sides have the property. Chain graphs can be recognized in linear time [23]. It is also known that a graph is a chain graph if and only if it does not contain a vertex set inducing $2K_2$, C_3 , or C_5 as an induced subgraph [23]. Computing a minimum chain deletion of a bipartite graph is an NP-hard problem [29].

Theorem 8 ([23]). *A graph G is a chain graph if and only if G is bipartite and turning one side of the bipartition into a clique gives a threshold graph for any bipartition of G .*

By Theorem 8, a chain graph G can have at most one connected component that contains an edge. Isolated vertices can belong to any side of the bipartition. We will here define a unique way of partitioning the vertices of a chain graph that we call *chain partition*, similar to threshold partition. Define X_0 to be the set of all isolated vertices of G . The remaining vertices induce a connected bipartite graph which thus has a unique bipartition (X, Y) . Partition X into $(X_1, X_2, \dots, X_\ell)$ where $N_G(X_1) \subset N_G(X_2) \subset \dots \subset N_G(X_\ell)$, and ℓ is as large as possible. Hence vertices of X_i have the same neighborhood, for each i . This defines a partition of Y into $(Y_1, Y_2, \dots, Y_\ell)$, where $Y_i = N_G(X_i) \setminus N_G(X_{i-1})$, $2 \leq i \leq \ell$. Observe that each set X_i contains at least one vertex which implies that the set Y_i is also a non-empty set. If there are only isolated vertices in G , we let $\ell = 0$. The chain partition is unique (upon exchanging X with Y).

Theorem 9. *Chain graphs are sandwich monotone.*

Proof. Let $G = (V, E)$ and $G' = (V, E \cup F)$ be two chain graphs such that $E \cap F = \emptyset$ and $F \neq \emptyset$. We will show that there is an edge $f \in F$ that can be removed from G' so that the resulting graph remains a chain graph, from which the result follows by induction on the edges in F .

Let $((X'_0, X'_1, \dots, X'_\ell), (Y'_1, Y'_2, \dots, Y'_\ell))$ be the chain-partition of G' . First we prove that if F contains an edge xy such that $x \in X'_i$ and $y \in Y'_i$ for some $i \in \{1, \dots, \ell\}$ then removing xy from G' results in a chain graph. Let G'' be the graph that we obtain after removing xy ; that is, $G'' = G' - xy$. In G'' we have that $N_{G''}(X'_{i-1}) \subset N_{G''}(x) \subset N_{G''}(X'_i \setminus \{x\}) \subset N_{G''}(X'_{i+1})$. Thus the nested neighborhood ordering is maintained which implies that G'' is a chain graph.

Now we prove that F contains at least one edge with one endpoint in X'_i and one endpoint in Y'_i , for some i satisfying $1 \leq i \leq \ell$. Assume for a contradiction that there are no edges in F with one endpoint in X'_i and the other in Y'_i . Hence for every edge $xy \in F$, $x \in X'_j$ and $y \in Y'_i$, where $1 \leq i < j \leq \ell$. Since X'_j and Y'_i are non-empty, X'_i and Y'_j are non-empty, too by definition. Let $x' \in X'_i$ and $y' \in Y'_j$. Edges xy' and $x'y$ cannot belong to F and hence they are edges in E by our assumption. Edge $x'y'$ is not an edge of G' by the definition of chain partition, and hence it is not an edge of G either. Since xy is not an edge of G , $\{x, y', x', y\}$ induces a $2K_2$ in G contradicting that G is a chain graph.

Lemma 3. *Let $G = (V, E)$ be an arbitrary graph, and let H be a chain deletion of G with chain partition $(X = (X_0, X_1, \dots, X_\ell), Y = (Y_1, \dots, Y_\ell))$. H is a minimal chain deletion of G if and only if no deleted edge uv has the following property: $u \in X_{i-1}$ and $v \in Y_i$ for some $1 \leq i \leq \ell$, or $u \in X_\ell$ and $v \in X_0$.*

Next we consider the problem of extracting a minimal chain deletion from any chain deletion H of an arbitrary graph G . We briefly describe such an algorithm. Let (X, Y) be the chain partition of H . At the beginning we unmark all vertices of X and at each step we pick an unmarked vertex $v \in X$ of largest degree in H . We find deleted edges incident to v not having the properties given in Lemma 3. If the deleted edges can be added to H without destroying the chain property then we add them and proceed with the next unmarked vertex of X of largest degree in H .

Theorem 10. *Let $H = (V, E \setminus D)$ be a chain deletion of an arbitrary graph $G = (V, E)$ with $D \subseteq E$. A minimal chain deletion $H' = (V, E \setminus D')$ of G , such that $D' \subseteq D$ can be computed in time $O(n + m)$.*

Corollary 2. *Any minimal chain deletion of an arbitrary graph graph can be computed in $O(n + m)$ time.*

5 Concluding Remarks and Open Questions

In this extended abstract we proved sandwich monotonicity of threshold and chain graphs. Moreover, we gave linear-time algorithms for computing minimal threshold completions, minimal bipartite deletions, and minimal chain deletions; and for extracting these from any given threshold completion, bipartite deletion, or chain deletion, respectively. In fact, the results presented in this extended abstract, by careful but not difficult argumentation, lead to linear-time algorithms also for computing **minimal threshold deletions** and **minimal co-bipartite completions** of arbitrary graphs, and **minimal chain completions** of bipartite graphs (see also [24]); as well as extracting these from any given such completion or deletion. Details of these results are left to a full version [17].

It has been shown recently that minimum weakly chordal completions of arbitrary graphs are NP-hard to compute [7]. We would like to know whether weakly chordal graphs are sandwich monotone. Also, we repeat the open question of [2]: are chordal bipartite graphs sandwich monotone? In addition, we would like to know whether minimum chordal bipartite completions of bipartite graphs are NP-hard to compute.

References

1. Alon, N., Shapira, A.: Every monotone graph property is testable. In: Proceedings of STOC 2005 - 37th Annual Symposium on Theory of Computing, pp. 128–137 (2005)
2. Bakonyi, M., Bono, A.: Several results on chordal bipartite graphs. Czechoslovak Math. J. 46, 577–583 (1997)
3. Balogh, J., Bolobás, B., Weinreich, D.: Measures on monotone properties of graphs. Disc. Appl. Math. 116, 17–36 (2002)
4. Blair, J., Heggernes, P., Telle, J.A.: A practical algorithm for making filled graphs minimal. Theoretical Computer Science 250, 125–141 (2001)
5. Bodlaender, H.L., Koster, A.M.C.A.: Safe separators for treewidth. Discrete Math. 306, 337–350 (2006)
6. Bouchitté, V., Todinca, I.: Treewidth and minimum fill-in: Grouping the minimal separators. SIAM J. Comput. 31, 212–232 (2001)
7. Burzyn, P., Bonomo, F., Durán, G.: NP-completeness results for edge modification problems. Disc. Appl. Math. 154, 1824–1844 (2006)
8. Chvátal, V., Hammer, P.L.: Set-packing and threshold graphs. Univ. Waterloo Res. Report, CORR 73–21 (1973)
9. Dearing, P.M., Shier, D.R., Warner, D.D.: Maximal chordal subgraphs. Disc. Appl. Math. 20, 181–190 (1988)

10. Djidjev, H.: A linear algorithm for finding a maximal planar subgraph. *SIAM J. Disc. Math.* 20, 444–462 (2006)
11. Fomin, F.V., Kratsch, D., Todinca, I.: Exact (exponential) algorithms for treewidth and minimum fill-in. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) *ICALP 2004*. LNCS, vol. 3142, pp. 568–580. Springer, Heidelberg (2004)
12. Földes, S., Hammer, P.L.: Split graphs. *Congressus Numer.* 19, 311–315 (1977)
13. Garey, M., Johnson, D., Stockmeyer, L.: Some simplified NP-complete graph problems. *Theoretical Computer Science* 1, 237–267 (1976)
14. Golumbic, M.C.: *Algorithmic Graph Theory and Perfect Graphs*, 2nd edn. *Annals of Discrete Mathematics*, vol. 57. Elsevier, Amsterdam (2004)
15. Heggernes, P., Mancini, F.: Minimal split completions of graphs. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) *LATIN 2006*. LNCS, vol. 3887, pp. 592–604. Springer, Heidelberg (2006)
16. Heggernes, P., Mancini, F.: A completely dynamic algorithm for split graphs. *Reports in Informatics* 334, University of Bergen, Norway (2006)
17. Heggernes, P., Papadopoulos, C.: Single-edge monotonic sequences of graphs and linear-time algorithms for minimal completions and deletions. *Reports in Informatics* 345, University of Bergen, Norway (2007)
18. Heggernes, P., Suchan, K., Todinca, I., Villanger, Y.: Characterizing minimal interval completions: Towards better understanding of profile and pathwidth. In: Thomas, W., Weil, P. (eds.) *STACS 2007*. LNCS, vol. 4393, pp. 2007–2024. Springer, Heidelberg (2007)
19. Heggernes, P., Telle, J.A., Villanger, Y.: Computing minimal triangulations in time $O(n^\alpha \log n) = o(n^{2.376})$. *SIAM J. Disc. Math.* 19, 900–913 (2005)
20. Hsu, W.-L.: A linear time algorithm for finding a maximal planar subgraph based on PC-trees. In: Wang, L. (ed.) *COCOON 2005*. LNCS, vol. 3595, pp. 787–797. Springer, Heidelberg (2005)
21. Kaplan, H., Shamir, R., Tarjan, R.E.: Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM J. Comput.* 28(5), 1906–1922 (1999)
22. Kashiwabara, T., Fujisawa, T.: An NP-complete problem on interval graphs. *IEEE Symp. of Circuits and Systems*, pp. 82–83. IEEE Computer Society Press, Los Alamitos (1979)
23. Mahadev, N., Peled, U.: Threshold graphs and related topics. *Annals of Discrete Mathematics* 56. North Holland, Amsterdam (1995)
24. Meister, D.: Recognition and computation of minimal triangulations for AT-free claw-free and co-comparability graphs. *Disc. Appl. Math.* 146, 193–218 (2005)
25. Natanzon, A., Shamir, R., Sharan, R.: Complexity classification of some edge modification problems. *Disc. Appl. Math.* 113, 109–128 (2001)
26. Peng, S.-L., Chen, C.-K.: On the interval completion of chordal graphs. *Disc. Appl. Math.* 154, 1003–1010 (2006)
27. Rapaport, I., Suchan, K., Todinca, I.: Minimal proper interval completions. In: Fomin, F.V. (ed.) *WG 2006*. LNCS, vol. 4271, pp. 217–228. Springer, Heidelberg (2006)
28. Rose, D., Tarjan, R.E., Lueker, G.: Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.* 5, 266–283 (1976)
29. Yannakakis, M.: Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.* 2, 77–79 (1981)
30. Yannakakis, M.: Node deletion problems on bipartite graphs. *SIAM J. Comput.* 10, 310–327 (1981)

On the Hardness of Optimization in Power Law Graphs^{*}

Alessandro Ferrante¹, Gopal Pandurangan², and Kihong Park²

¹ Dipartimento di Informatica ed Applicazioni “R.M. Capocelli”, University of Salerno, Via Ponte don Melillo - 84084 Fisciano (SA), Italy

ferrante@dia.unisa.it

² Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA

{gopal,park}@cs.purdue.edu

Abstract. Our motivation for this work is the remarkable discovery that many large-scale real-world graphs ranging from Internet and World Wide Web to social and biological networks exhibit a power-law distribution: the number of nodes y_i of a given degree i is proportional to $i^{-\beta}$ where $\beta > 0$ is a constant that depends on the application domain. There is practical evidence that combinatorial optimization in power-law graphs is easier than in general graphs, prompting the basic theoretical question: Is combinatorial optimization in power-law graphs easy? Does the answer depend on the power-law exponent β ? Our main result is the proof that many classical NP-hard graph-theoretic optimization problems remain NP-hard on power law graphs for certain values of β . In particular, we show that some classical problems, such as CLIQUE and COLORING, remains NP-hard for all $\beta \geq 1$. Moreover, we show that all the problems that satisfy the so-called “optimal substructure property” remains NP-hard for all $\beta > 0$. This includes classical problems such as MIN VERTEX-COVER, MAX INDEPENDENT-SET, and MIN DOMINATING-SET. Our proofs involve designing efficient algorithms for constructing graphs with prescribed degree sequences that are tractable with respect to various optimization problems.

1 Overview and Results

The elegant theory of NP-hardness serves as an important cornerstone in understanding the difficulty of solving various combinatorial optimization problems in graphs. A natural and relevant question is whether such hardness results on combinatorial problems are applicable to “real-world” graphs since such graphs possess certain well-defined special properties which may very well render them tractable. Our motivation for this work is the remarkable discovery that many large-scale real-world graphs ranging from Internet and World Wide Web to social and biological networks exhibit a power-law distribution. In such networks, the number of nodes y_i of a given degree i is proportional to $i^{-\beta}$ where $\beta > 0$

^{*} Work partially supported by funds for research from MIUR ex 60% 2005.

is a constant that depends on the application domain. Power-law degree distribution has been observed in the Internet ($\beta = 2.1$), World Wide Web ($\beta = 2.1$), social networks (movie actors graph with $\beta = 2.3$, citation graph with $\beta = 3$), and biological networks (protein domains with $\beta = 1.6$, protein-protein interaction graphs with $\beta = 2.5$). In most real-world graphs, β ranges between 1 and 4 (see [3] for a comprehensive list). Thus, power-law graphs have emerged as a partial answer to the perennial search for representative real-world graphs in combinatorial optimization.

There is practical evidence that combinatorial optimization in real-world power law graphs is easier than in general graphs. For example, experiments in Internet measurement graphs (power law with $\beta = 2.1$) show that a simple greedy algorithm that exploits the power law property yields a very good approximation to the MINIMUM VERTEX COVER problem (much better than random graphs with no power law) [11][12]. Gkantsidis, Mihail, and Saberi [8] argue that the performance of the Internet suggests that multi-commodity flow can be routed more efficiently (i.e., with near-optimal congestion) in Internet graphs than in general graphs. Eubank et al. [6] show that in power-law social networks, a simple and natural greedy algorithm that again exploits the power-law property (choose enough high-degree vertices) gives a $1 + o(1)$ approximation to the DOMINATING SET problem. There is also similar practical evidence that optimization in power-law biological networks is easier [9]. All these results on disparate problems on various real-world graphs motivate a coherent and systematic algorithmic theory of optimization in power law graphs (and in general, graphs with prescribed degree sequences).

In this work, we study the following theoretical questions: What are the implications of power-law degree distribution to the algorithmic complexity of NP-hard optimization problems? Can the power-law degree distribution property alone be sufficient to design polynomial-time algorithms for NP-hard problems on power-law graphs? And does the answer depend on the exponent β ?

A number of power law graph models have been proposed in the last few years to capture and/or explain the empirically observed power-law degree distribution in real-world graphs. They can be classified into two types. The first takes a power-law degree sequence and generates graph instances with this distribution. The second type arises from attempts to explain the power-law starting from basic assumptions about a growth evolution. Both approaches are well motivated and there is a large literature on both (e.g., [4][12]). Following Aiello, Chung, and Lu [1][2], we adopt the first approach, and use the following model for (undirected) power-law graphs (henceforth called ACL model): the number of vertices y_i with degree i is roughly given¹ by $y_i = e^\alpha / i^\beta$, where e^α is a normalization constant (so that the total number of vertices sum to the size of the graph, thus α determines the size). While the above model is potentially less accurate than the detailed modeling approach of the second type, it has the advantage of being robust and general [1]: the structural properties that are true in this model will be true for all graphs with the given degree sequence.

¹ Our model is defined precisely in Section 2.

Investigating the complexity of problems in power law graphs (in particular, the ACL model) involves an important subtlety. The ACL model allows graphs with self-loops and multi-edges. However, many real-world networks, such as Internet domain graphs, are simple undirected power-law graphs. Thus, we restrict ourselves to simple undirected power-law graphs (no multi-edges or self-loops). In this paper we study the complexity of many classical graph problems in the ACL model. In particular, we first show that problems such as COLORING and CLIQUE remains NP-hard in simple power-law graphs of the the ACL model for all $\beta \geq 1$, and then we show that all the graph problems that satisfy an “optimal substructure” property (such as MINIMUM VERTEX COVER, MAXIMUM INDEPENDENT SET and MINIMUM DOMINATING SET) remain NP-hard on simple power law graphs of the ACL model for all $\beta > 0$. This property essentially states that the optimal solution for a problem on given graph is the union of the optimal (sub-)solutions on its maximal connected components. A main ingredient in our proof is a technical lemma that guarantees that any arbitrary graph can be “embedded” in a suitably large (but polynomial in the size of the given graph) graph that conforms to the prescribed power-law degree sequence. This lemma may be of independent interest and can have other applications as well e.g., in showing hardness of *approximation* of problems in power-law graphs. Another contribution is constructions of graphs with prescribed degree sequences that admit polynomial time algorithms. These constructions are useful in showing the NP-hardness of certain optimization problems that do not satisfy the optimal substructure property. In particular, we will use them to show the NP-hardness of CLIQUE and COLORING for all $\beta \geq 1$.

Our results show that the worst-case complexity of power law graphs is hard with respect to many important graph optimization problems. However, experimental evidence shows that optimization is considerably easier in real-world power-law graphs. This suggests that real-world graphs are not “worst-case” instances of power-law graphs, but rather typical instances which may be well modeled by power law *random graph* models (e.g., [6,3,4,8,10]). Combinatorial optimization is generally easier in random graphs and hence from an optimization perspective this somewhat justifies using power law random graphs to model real-world power law graphs. We believe that further investigation, both in the modeling of real-world graphs and in the optimization complexity of real-world graphs and their models, is needed to gain a better understanding of this important issue.

2 Notations and Definitions

In this section we introduce some notations and definitions that we will use throughout the paper. For all $x, y \in \mathbb{N}$ with $x \leq y$, we will use $[x, y]$ to denote $\{x, x + 1, \dots, y\}$ and $[x]$ to denote $[1, x]$.

Given a graph, we will refer to two types of sequence of integers: y -degree sequence and d -degree sequence. The first type lists the number of vertices with a certain degree (i.e., the degree distribution) and the latter lists the degrees of

the vertices in non-increasing order (i.e., the degree sequence of the graph in non-increasing order). More formally, we can define the y -degree sequence as follows. Given a graph $G = (V, E)$ with maximum degree m , the y -degree sequence is the sequence $Y^G = \langle y_1^G, \dots, y_m^G \rangle$ where $y_i = |\{v \in V : \text{degree}(v) = i\}|$, $i \in [m]$. Given a graph of n vertices, the d -degree sequence will be denoted by $D^G = \langle d_1^G, \dots, d_n^G \rangle$, where d_i^G 's are the vertex degrees in non-increasing order. When the referred graph is clear from the context, we will use only Y and D to denote the y - and d -degree sequence respectively. (We note that we don't allow vertices with zero degree (i.e., singletons) in G . This is not really a issue, because we will deal with problems in which singletons can be treated separately from the rest of the graph to obtain a (optimum) solution to the problem with "minor effects" on the running time.)

Given a sequence of integers $S = \langle s_1, \dots, s_m \rangle$, we define the following operator that expands S in a new non increasing sequence of integers.

Definition 1 (Expansion). *Let $S = \langle s_1, \dots, s_n \rangle$ be a sequence of integers and $j \in [n]$. Then we define*

$$\text{EXP}(S) = \langle \overbrace{n, \dots, n}^{s_n}, \dots, \overbrace{1, \dots, 1}^{s_1} \rangle.$$

Note that the expansion operation converts a y -degree sequence into a d -sequence. In the rest of the paper, given two degree sequences $S = \langle s_1, \dots, s_n \rangle$ and $T = \langle t_1, \dots, t_m \rangle$ with $n \geq m$, we will denote $S - T = \langle x_1, \dots, x_n \rangle$ with $x_i = s_i - t_i$ if $i \in [m]$ and $x_i = s_i$ otherwise.

The ACL model of power-law graphs introduced in [11] have a particular kind of y -degree sequence which we henceforth call (β, α) -degree sequence and is defined as follows.

Definition 2 ((β, α)-degree sequence). *Given $\alpha, \beta \in \mathbb{R}^+$, the y -degree sequence of a graph $G = (V, E)$ is a (β, α) -degree sequence (denoted by $Y^{(\beta, \alpha)} = \langle y_1^{(\beta, \alpha)}, \dots, y_m^{(\beta, \alpha)} \rangle$) if $m = \lfloor e^{\alpha/\beta} \rfloor$ and, for $i \in [m]$*

$$y_i = \begin{cases} \lfloor \frac{e^\alpha}{i^\beta} \rfloor & \text{if } i > 1 \text{ or } \sum_{k=1}^m \lfloor \frac{e^\alpha}{k^\beta} \rfloor \text{ is even} \\ \lfloor \frac{e^\alpha}{i^\beta} \rfloor + 1 & \text{otherwise.} \end{cases}$$

In the rest of the paper, given a sequence of integers $S = \langle s_1, \dots, s_k \rangle$, we will define $\text{tot}(S) = \sum_{i=1}^k s_i$ and $w(S) = \sum_{i=1}^k i s_i$. Note that if S is the y -degree sequence of a graph, then $w(S)$ is the total degree of the graph, whereas if S is the d -degree sequence of a graph, then $\text{tot}(S)$ is the total degree of the graph.

Our aim is to study the NP-hardness of graph-theoretic optimization problems when they are restricted to ACL power-law graphs with a fixed β , in particular, simple graphs belonging to this class. (Of course, showing hardness results for this class implies hardness for arbitrary power law graphs as well.) Formally, we define such graphs as:

Definition 3 (β -graph). *Given $\beta \in \mathbb{R}^+$, a graph $G = (V, E)$ is a β -graph if it is simple and there exists $\alpha \in \mathbb{R}^+$ such that the y -degree sequence of G is a (β, α) -degree sequence.*

3 NP-Hardness of CLIQUE AND COLORING

In this section we introduce a general technique to prove the NP-hardness of some optimization problems. The main idea of the proof is the following. Given an arbitrary graph G , it is possible to construct a simple graph G_1 which contains G as a set of maximal connected components. Let $G_2 = G_1 \setminus G$ be the remaining graph. Obviously, G_2 is simple and if we can show that we can efficiently (i.e., in polynomial time) compute the optimal solution in G_2 then this essentially gives us the result. However, it is a priori not obvious how to design an efficient algorithm given a particular problem. The key idea we will use here is that we have the choice of constructing G_1 (and hence G_2) and thus we can construct the graph in such a way that it admits an efficient algorithm. If we construct the graph in a careful way, it will be possible to design a polynomial time algorithm that finds the optimal.

Below we illustrate this idea by showing the NP-completeness of certain problems, including CLIQUE AND COLORING, in β -graphs for $\beta \geq 1$. Our idea here is to make G_2 to be a *simple bipartite* graph. Since bipartite graphs are 2-colorable and have a maximum clique of size 2, this immediately gives the reduction. Obviously, the main difficulty is in constructing the bipartite graph. We first need the following definitions.

Definition 4 (Contiguous Sequence). A sequence $D = \langle d_1, \dots, d_n \rangle$ with maximum value m is contiguous if $y_i^D > 0$ for all $i \in [m]$, where $y_i^D = |\{j \in [n] \text{ s.t. } d_j = i\}|$.

Definition 5 (Bipartite-Eligible Sequence). A sequence $D = \langle d_1, \dots, d_n \rangle$ with maximum value m is bipartite-eligible if it is contiguous and $m \leq \lfloor n/2 \rfloor$.

Given a simple graph $G = (V, E)$, for every vertex $u \in V$ we will denote $NEIG(u) = \{v \in V \setminus \{u\} \text{ s.t. } (u, v) \in E\}$.

Lemma 1. Let $D = \langle d_1, \dots, d_n \rangle$ be a sequence. If D is non increasing and bipartite-eligible and $tot(D)$ is even, then it is possible to construct in time $O(n^2)$ a simple bipartite graph $G = (V, E)$ such that $D^G = D$.

Proof. First note that since D is non increasing and bipartite-eligible, $d_1 \leq \lfloor n/2 \rfloor$. We build the graph iteratively by adding some edges to certain vertices. Define the *residual* degree of a vertex as its final degree minus its “current” degree. Initially all the vertices have degree 0. To build the graph we use the following algorithm:

1. let $d(s_i)$ and $d(t_i)$ be the *residual degree* of the i -th vertex of S and T ;
2. $E \leftarrow \emptyset$; $S \leftarrow \emptyset$; $T \leftarrow \emptyset$; $tot(S) \leftarrow 0$; $tot(T) \leftarrow 0$; $k \leftarrow |S|$; $l \leftarrow |T|$;
3. **while** $i \leq n$ **do**
 - (a) **while** $i \leq n$ **and** $tot(S) \leq tot(T)$ **do**
 - i. $S \leftarrow S \cup \{u \mid u \notin S\}$; $k \leftarrow k + 1$; $d(s_k) \leftarrow d_i$; $tot(S) \leftarrow tot(S) + d_i$;
 - (b) **while** $i \leq n$ **and** $tot(T) \leq tot(S)$ **do**
 - i. $T \leftarrow T \cup \{v \mid v \notin T\}$; $l \leftarrow l + 1$; $d(t_l) \leftarrow d_i$; $tot(T) \leftarrow tot(T) + d_i$;

4. **while** $tot(S) > 0$ **do**
 - (a) **SORT** S and T separately in non increasing order of the residual degree;
 - (b) **for** $i \leftarrow 1$ **to** $d(s_1)$ **do**
 - i. $E \leftarrow E \cup \{(s_1, t_i)\}$; $d(s_1) \leftarrow d(s_1) - 1$; $d(t_i) \leftarrow d(t_i) - 1$; $tot(S) \leftarrow tot(S) - 1$; $tot(T) \leftarrow tot(T) - 1$;
 - (c) **for** $i \leftarrow 2$ **to** $d(t_1) + 1$ **do**
 - i. $E \leftarrow E \cup \{(t_1, s_i)\}$; $d(t_1) \leftarrow d(t_1) - 1$; $d(s_i) \leftarrow d(s_i) - 1$; $tot(T) \leftarrow tot(T) - 1$; $tot(S) \leftarrow tot(S) - 1$;
5. **return** $G = (S \cup T, E)$;

Note that the entire loop 3) requires $O(n^2)$ time to be completed. Moreover, in every iteration of the loop 4), at least one vertex is completed and will be no longer considered in the algorithm. Therefore, the loop 4) is completed in $O(n^2)$ time and the algorithm has complexity $O(n^2)$.

Now we prove that the algorithm correctly works. We first introduce some notations. The residual degree of the set S (T respectively) after the *SORT* instruction of the round i is denoted by $R_i(S)$ ($R_i(T)$ respectively). The number of vertices with positive residual degree (*non full* vertices) in S (T) is denoted by $N_i(S)$ ($N_i(T)$). The set S is s_1^i, \dots, s_h^i and the set T is t_1^i, \dots, t_k^i .

The proof is by induction on the round i . More exactly, we prove the following invariant: *After the SORT instruction we have: (i) $R_i(S) = R_i(T)$ and (ii) $N_i(T) \geq d(s_1^i)$ and $N_i(S) \geq d(t_1^i)$.*

It is easy to see that if this invariant holds, then the algorithm correctly builds a bipartite graph. We start proving the base ($i = 1$) by showing that the above two conditions hold.

1. Let $tot_j(S)$ and $tot_j(T)$ be the total degree of the sets S and T after the insertion of the j -th vertex. We first show that $|tot_j(S) - tot_j(T)| \leq d_{j+1}$ for all $j \in [2, n - 1]$. This is obvious for $j = 2$ since the sequence is non increasing and contiguous. Let us suppose that this is true until $j - 1$ and let us show it for j .

Without loss of generality, let us suppose that the j -th vertex is assigned to T . Then this implies that $tot_{j-1}(S) \geq tot_{j-1}(T)$ and by induction $tot_{j-1}(S) - tot_{j-1}(T) \leq d_j$ and, therefore, $tot_j(S) - tot_j(T) \leq 0$.

Now we can complete the proof of the bases. w.l.o.g. let us suppose that the last one vertex is assigned to T . Then we have $R_1(S) \geq R_1(T) - 1$. But from the preceding proof we also know that $R_1(S) \leq R_1(T)$ and, from the fact that the last one vertex has degree 1 and that the total degree of D is even, we have the claim.

2. Since the degree sequence is contiguous and after the insertion we have $tot(S) = tot(T)$, it is easy to see that after the insertion we have $-1 \leq |S| - |T| \leq 1$. From this and from the hypothesis $d_1 \leq \lfloor n/2 \rfloor$ the claim follows.

Let us suppose that the invariant is true until $i - 1$ and let us prove it for i .

1. We have $R_i(S) = R_{i-1}(S) - d(s_1^{i-1}) - (d(t_1^{i-1}) - 1) = R_{i-1}(T) - d(t_1^{i-1}) - (d(s_1^{i-1}) - 1) = R_i(T)$ as claimed.

2. The case $d(s_1^i) = 0$ is trivial, therefore let us suppose that $d(s_1^i) \geq 1$. If $d(s_2^{i-1}) = 1$, then $d(s_1^i) = 1$ since the degrees in S are non increasing. Moreover, from item (1) we have $R_i(T) = R_i(S) \geq 1$ and this completes this case.

If $d(s_2^{i-1}) > 1$, then we have two cases. If $d(t_2^{i-1}) = 1$, from item (1) and the fact that $d(t_j^i) \leq 1$ for all j we simply have the claim. On the other hand, if $d(s_2^{i-1}) > 1$, we have $N_i(T) = N_{i-1}(T) \geq d(s_1^{i-1}) \geq d(s_1^i)$. \square

The following lemma shows that for $\beta \geq 1$ it is possible to embed a simple graph G in a polynomial-size β -graph G_1 such that G is a set of maximal connected components of G_1 and $G_2 = G_1 \setminus G$ is bipartite-eligible.

Lemma 2. *Let $G = (V, E)$ be a simple graph with n_1 vertices and $\beta \geq 1$. Let $\alpha_0 = \max\{4\beta, \beta \ln n_1 + \ln(n_1 + 1)\}$. Then, for all $\alpha \geq \alpha_0$ the sequence $D = \text{EXP}(Y^{(\beta, \alpha)} - Y^G)$ is contiguous and bipartite-eligible.*

Proof. Let n_2 be the number of elements in D and $\alpha \geq \alpha_0$. We have

$$n_2 \geq \sum_{i=1}^{\lfloor e^{\alpha/\beta} \rfloor} \left\lfloor \frac{e^\alpha}{i^\beta} \right\rfloor - n_1 > e^\alpha \sum_{i=1}^{\lfloor e^{\alpha/\beta} \rfloor} \frac{1}{i^\beta} - \lfloor e^{\alpha/\beta} \rfloor - n_1 \geq e^\alpha \int_{i=1}^{\lfloor e^{\alpha/\beta} \rfloor + 1} \frac{1}{i^\beta} - \lfloor e^{\alpha/\beta} \rfloor - n_1.$$

If $\beta = 1$, then we have

$$n_2 \geq \alpha e^\alpha - e^\alpha - n_1 \geq 4e^\alpha - 2e^\alpha + 1 \geq 2m + 1.$$

and if $\beta > 1$ we have

$$n_2 \geq \frac{e^\alpha}{\beta - 1} - e^{\alpha/\beta} - n_1 \geq 4e^{\alpha/\beta} - 2e^{\alpha/\beta} + 1 \geq 2m + 1.$$

Moreover $y_{n_1}^{(\beta, \alpha)} \geq \left\lfloor \frac{e^\alpha}{n_1^\beta} \right\rfloor > \frac{e^\alpha}{n_1^\beta} - 1 \geq \frac{n_1^{\beta+1} + n_1^\beta}{n_1^\beta} - 1 = n_1$, that is $\text{EXP}(Y)$ is contiguous. Therefore, $\text{EXP}(Y)$ is bipartite-eligible and this completes the proof of this lemma. \square

We finally show the NP-completeness of certain problems in β -graphs with $\beta \geq 1$. The following definition is useful to introduce the class of problems we analyze in what follows.

Definition 6 (c-Oracle). *Let P be an optimization problem and $c > 0$ a constant. A c -oracle for the problem P is a polynomial-time algorithm $A_c^P(I)$ which takes in input an instance I of P and correctly returns an optimum solution for P given that on the instance I the problem has an optimum solution with size at most c .*

The following theorem shows the NP-completeness of a particular class of decision problems defined using the c -oracle in β -graphs with $\beta \geq 1$.

Theorem 1. *Let $\beta \geq 1$. Let P be a graph decision problem such that its optimization version obeys the following properties:*

1. $OPT(G) = \max_{1 \leq i \leq k} OPT(C_i)$ (where C_i are the maximal connected components of G),
2. exists a constant $c > 0$ such that for all bipartite simple graphs H it holds $|OPT(H)| \leq c$ and
3. it admits a c -oracle.

If P is NP-complete in general graphs, then it is NP-complete in β -graphs too.

Proof. From Lemmas [2](#) and [1](#), it is possible to construct, in time $poly(|G|)$, a β -graph G_1 embedding G such that $|G_1| = poly(|G|)$, G is a set of maximal connected components of G_1 and $G_2 = G_1 \setminus G$ is a simple bipartite graph. Since $OPT(G_1) = \max_k \{OPT(C_k)\}$, $|OPT(G_2)| \leq c$ and the optimization version of P admits a c -oracle, it is easy to see that P can be reduced in polynomial time to β -P (where β -P is P restricted to β -graphs). \square

Since CLIQUE and COLORING satisfy all conditions of Theorem [1](#) with $c = 2$, we easily obtain the following corollary.

Corollary 1. *CLIQUE, and COLORING are NP-Complete in β -graphs for all $\beta \geq 1$.*

4 Hardness of Optimization Problems with Optimal Substructure

We show that if an optimization problem is NP-hard on (simple) general graphs (i.e., computing a solution in polynomial time is hard) and it satisfies the following “optimal substructure” property, then it is NP-hard on β -graphs also. We state this property as follows. Let P be an optimization problem which takes a graph as input. For *every* input G , the following should be true: every optimum solution of P on G should contain an optimum solution of P on each of G 's maximal connected components. To illustrate with an example, it is easy to see that MINIMUM VERTEX COVER problem satisfies this property: an optimal vertex cover on any graph G should contain within it an optimal vertex cover of its maximal connected components. On the other hand, MINIMUM COLORING does not satisfy the above property, since the optimal coloring of a graph need not contain an optimal coloring of all its maximal connected components. We first need some definitions. We say that a sequence D is *graphic* if there exists a simple graph G such that $D^G = D$.

Definition 7 (Eligible Sequence). *A sequence of integers $S = \langle s_1, \dots, s_n \rangle$ is eligible if $s_1 \geq \dots \geq s_n$ and, for all $k \in [n]$, $f_S(k) \geq 0$, where*

$$f_S(k) = k(k-1) + \sum_{i=k+1}^n \min\{k, s_i\} - \sum_{i=1}^k s_i.$$

The following result due to Havel and Hakimi ([5](#)) gives a straightforward algorithm to construct a simple graph from a graphic degree sequence.

Lemma 3 ([5]). *A sequence of integers $D = \langle d_1, \dots, d_n \rangle$ is graphic if and only if it is non-increasing, and the sequence of values $D' = \langle d_2 - 1, d_3 - 1, \dots, d_{d_1+1} - 1, d_{d_1+2}, \dots, d_n \rangle$ when sorted in non-increasing order is graphic.*

In the next technical lemma (whose complete proof appears in the full version [7]), we introduce a new sufficient condition for a sequence of integers to be eligible.

Lemma 4. *Let $Y^{(1)}$ and $Y^{(2)}$ be two degree sequences with m_1 and m_2 elements respectively such that (i) $y_j^{(1)} \leq y_j^{(2)}$ for all $j \in [m_1]$, and (ii) $D^{(1)} = \text{EXP}(Y^{(1)})$ and $D^{(2)} = \text{EXP}(Y^{(2)})$ are contiguous. If $D^{(1)}$ is eligible then $D^{(2)}$ is eligible.*

Proof sketch. Let $Y^{(1)}$ and $Y^{(2)}$ be two degree sequences with m_1 and m_2 elements respectively such that (i) $y_j^{(1)} \leq y_j^{(2)}$ for all $j \in [m_1]$, and (ii) $D^{(1)} = \text{EXP}(Y^{(1)})$ and $D^{(2)} = \text{EXP}(Y^{(2)})$ are contiguous.

We have to show that “if $D^{(1)}$ is eligible then $D^{(2)}$ is eligible”. Let us note that the transformation from the degree sequence $Y^{(1)}$ to the degree sequence $Y^{(2)}$ (and hence from $D^{(1)}$ to $D^{(2)}$) can be seen as a sequence of rounds of the following type: in every step a vertex with degree d is transformed into a vertex with degree $(d + 1)$ and the global sequence is rearranged with respect to the relation $y_j^{(1)} \leq y_j^{(2)}$ for all $j \in [m_1]$. In other words, to transform $Y^{(1)}$ to $Y^{(2)}$ (and hence $D^{(1)}$ to $D^{(2)}$) we can execute the following simple algorithm²:

1. $S^{(0)} \leftarrow D^{(1)}$; $i \leftarrow 0$
2. **while** $S^{(i)} \neq D^{(2)}$ **do**
 - (a) **for** $j \leftarrow m_2$ **downto** 2 **do**
 - i. **if** $|\{x \in S^{(i+1)} \text{ s.t. } x = j\}| < y_j^{(2)}$ **and** $|\{x \in S^{(i+1)} \text{ s.t. } x = j - 1\}| > 0$ **then**
 - A. $k \leftarrow \min\{x \in |S^{(i+1)} \text{ s.t. } s_x^{(i+1)} = j - 1\}$
 - B. $s_k^{(i+1)} \leftarrow s_k^{(i+1)} + 1$
 - (b) $S^{(i+1)} \leftarrow S^{(i+1)} + x$
 - (c) $i \leftarrow i + 1$

Let $n_2 = |D^{(2)}|$. From definition of eligibility, $D^{(2)}$ is eligible if $f_{D^{(2)}}(k) > 0$ for all $k \in [n_2]$. It can be showed that at the end of each iteration of the while loop of the previous algorithms, if $f_{S^{(i-1)}}(k) \geq 0$ for all $i \in [n^{(i-1)}]$ then $f_{S^{(i)}}(k) \geq 0$ for all $i \in [n^{(i)}]$, where $n^{(i)} = |S^{(i)}|$. Since $f_{S^{(0)}}(k) \geq 0$ for $k \in n^{(0)}$, this completes the proof of this lemma. □

The previous lemma is useful to show the following key lemma (*Embedding Lemma*) that shows that it is possible to quickly construct a β -graph with a certain property.

² In the rest of the paper, given a sequence $S = \langle s_1, \dots, s_n \rangle$ and an integer x we will use the notation $S + x = \langle s_1, \dots, s_n, x \rangle$ to denote the concatenation of S with the integer x .

Lemma 5 (Embedding Lemma). *Let $G = (V, E)$ be a simple undirected graph and $\beta \in \mathbb{R}^+$. Then there exists a simple undirected graph $G_1 = (V_1, E_1)$ such that G is a set of maximal connected components of G_1 , $|V_1| = \text{poly}(|V|)$ and G_1 is a β -graph. Furthermore, given G , we can construct G_1 in time polynomial in the size of G .*

Proof. Let $n_1 = |V|$. From Lemma 3, we have only to show that there exist $\alpha_0 = O(\ln n_1)$ such that for all $\alpha \geq \alpha_0$, the degree sequence $D = \text{EXP}(Y) = Y^{(\beta, \alpha)} - Y^G$ is graphic, that is, from Lemma 3 such that D is eligible. For $\beta \geq 1$ the proof directly comes from Lemmas 2 and 1. Let us complete the proof for $0 < \beta < 1$.

Note that, $y_i^{(1, \alpha)} \leq y_i^{(\beta, \alpha)}$ and $\lfloor e^{\alpha/\beta} \rfloor \geq \lfloor e^\alpha \rfloor$ for $0 < \beta < 1$ and $i \in \lfloor e^\alpha \rfloor$ and, from Lemma 2, $\text{EXP}(Y^{(1, \alpha)} - Y^G)$ is contiguous for $\alpha \geq \max\{4, \ln n_1 + \ln(n_1 + 1)\}$.

Therefore, from Lemma 4, the sequence $\text{EXP}(Y^{(\beta, \alpha)} - Y^G)$ is eligible for $0 < \beta < 1$ and $\alpha \geq \max\{4, \ln n_1 + \ln(n_1 + 1)\}$ and this completes the proof of this lemma. □

Now we are ready to show the main theorem of this section.

Theorem 2. *Let P be an optimization problem on graphs with the optimal substructure property. If P is NP-hard on (simple) general graphs, then it is also NP-hard on β -graphs for all $\beta > 0$.*

Proof. We show that we can reduce the problem of computing an optimal solution on general graphs to computing an optimal solution on β -graphs and this reduction takes polynomial time. Let $G = (V, E)$ be a simple undirected graph. Lemma 5 says that we can construct (in time polynomial in the size of G) a simple undirected graph $G_1 = (V_1, E_1)$ such that G is a set of maximal connected components of G_1 , and G_1 is a β -graph with $|V_1| = \text{poly}(|V|)$. Since P has the optimal substructure property and G is a set of maximal connected components of G_1 , this implies that an optimum solution for the graph G can be computed easily from an optimal solution for G_1 . □

5 Concluding Remarks and Open Problems

We showed a general technique for establishing NP-hardness and NP-completeness of a large class of problems in power-law graphs. Our technique of “embedding” any arbitrary (given) graph into a polynomial-sized power-law graph is quite general and can have other applications, e.g., in showing hardness of *approximation* in power-law graphs (which is the next important question, now that we have established hardness). On the positive side, one may investigate approximation algorithms that exploit the power law property to get better approximation ratios compared to general graphs. Another interesting and relevant direction is to investigate the hardness or easiness of non-trivial restrictions of the ACL model.

We conclude by mentioning some open problems that follow directly from our work. We showed NP-hardness of CLIQUE and COLORING only for power

law graphs with $\beta \geq 1$. We believe that a different construction might show that these problems are NP-Complete for all $\beta > 0$. It will also be interesting to investigate the complexity of node- and edge-deletion problems, that is a general and important class of problems defined in [13]. We finally note that our technique does not directly imply hardness in *connected* power-law graphs. We conjecture that our techniques can be extended to show these results.

References

1. Aiello, W., Chung, F.R.K., Lu, L.: A Random Graph Model for Massive Graphs. In: Proceedings of STOC 2000, pp. 171–180. ACM Press, New York (2000)
2. Aiello, W., Chung, F.R.K., Lu, L.: A random graph model for power law graphs. In *Experimental Mathematics* 10, 53–66 (2000)
3. Barabasi, A.: Emergence of Scaling in Complex Networks. In: Bornholdt, S., Schuster, H. (eds.) *Handbook of Graphs and Networks*, Wiley, Chichester (2003)
4. Bollobas, B., Riordan, O.: *Mathematical Results on Scale-free Random Graphs*. In: Bornholdt, S., Schuster, H. (eds.) *Handbook of Graphs and Networks* (2003)
5. Bondy, J.A., Murty, U.S.R.: *Graph Theory with Applications*. North-Holland, Amsterdam (1976)
6. Eubank, S., Kumar, V.S.A., Marathe, M.V., Srinivasan, A., Wang, N.: Structural and Algorithmic Aspects of Massive Social Networks. In: Proceedings of 15th ACM-SIAM Symposium on Discrete Algorithms (SODA 2004), pp. 711–720. ACM Press, New York (2004)
7. Ferrante, A., Pandurangan, G., Park, K.: On the Hardness of Optimization in Power-Law Graphs, <http://www.cs.purdue.edu/homes/gopal/papers-by-date.html>
8. Gkantsidis, C., Mihail, M., Saberi, A.: Throughput and Congestion in Power-Law Graphs. In: Proceedings of SIGMETRICS 2003, pp. 148–159. ACM Press, New York (2003)
9. Koyuturk, M., Grama, A., Szpankowski, W.: Assessing significance of connectivity and conservation in protein interaction networks. In: Apostolico, A., Guerra, C., Istrail, S., Pevzner, P., Waterman, M. (eds.) RECOMB 2006. LNCS (LNBI), vol. 3909, pp. 45–49. Springer, Heidelberg (2006)
10. Mihail, M., Papadimitriou, C., Saberi, A.: On Certain Connectivity Properties of the Internet Topology. In: Proc. of FOCS 2003, pp. 28–35. IEEE Computer Society Press, Los Alamitos (2003)
11. Park, K., Lee, H.: On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets. In: Proceedings of SIGCOMM 2001, pp. 15–26. ACM Press, New York (2001)
12. Park, K.: The Internet as a complex system. In: Park, K., Willinger, W. (eds.) *The Internet as a Large-Scale Complex System*. Santa Fe Institute Studies on the Sciences of Complexity, Oxford University Press, Oxford (2005)
13. Yannakakis, M.: Node- and Edge-Deletion NP-Complete Problems. In: Proceedings of STOC 1978. SIAM 1978, San Diego, California, pp. 253–264 (1978)

Can a Graph Have Distinct Regular Partitions?

Noga Alon^{1,*}, Asaf Shapira², and Uri Stav³

¹ Schools of Mathematics and Computer Science, Raymond and Beverly Sackler
Faculty of Exact Sciences, Tel Aviv University, Tel Aviv, Israel

`nogaa@tau.ac.il`

² Microsoft Research, Redmond, WA

`asafico@microsoft.com`

³ School of Computer Science, Raymond and Beverly Sackler Faculty of Exact
Sciences, Tel Aviv University, Tel Aviv, Israel

`uristav@tau.ac.il`

Abstract. The regularity lemma of Szemerédi gives a concise approximate description of a graph via a so called regular-partition of its vertex set. In this paper we address the following problem: can a graph have two “distinct” regular partitions? It turns out that (as observed by several researchers) for the standard notion of a regular partition, one can construct a graph that has very distinct regular partitions. On the other hand we show that for the stronger notion of a regular partition that has been recently studied, all such regular partitions of the same graph must be very “similar”.

En route, we also give a short argument for deriving a recent variant of the regularity lemma obtained independently by Rödl and Schacht ([11]) and Lovász and Szegedy ([9], [10]), from a previously known variant of the regularity lemma due to Alon et al. [2]. The proof also provides a deterministic polynomial time algorithm for finding such partitions.

1 Introduction

We start with some of the basic definitions of regularity and state the regularity lemmas that we refer to in this paper. For a comprehensive survey on the regularity lemma the reader is referred to [7]. For a set of vertices $A \subseteq V$, we denote by $E(A)$ the set of edges of the graph induced by A in G , and by $e(A)$ the size of $E(A)$. Similarly, if $A \subseteq V$ and $B \subseteq V$ are two vertex sets, then $E(A, B)$ stands for the set of edges of G connecting vertices in A and B , and $e(A, B)$ denotes the number of ordered pairs (a, b) such that $a \in A$, $b \in B$ and ab is an edge of G . Note that if A and B are disjoint this is simply the number of edges of G that connect a vertex of A with a vertex of B , that is $e(A, B) = |E(A, B)|$. The *edge density* of the pair (A, B) is defined as $d(A, B) = e(A, B)/|A||B|$. When several graphs on the same set of vertices are involved, we write $d_G(A, B)$ to specify the graph to which we refer.

* Research supported in part by a grant from the Israel Science Foundation, by the Hermann Minkowski Minerva Center for Geometry at Tel Aviv University, and by a USA-Israeli BSF grant.

Definition 1 (ϵ -Regular Pair). A pair (A, B) is ϵ -regular, if for any two subsets $A' \subseteq A$ and $B' \subseteq B$, satisfying $|A'| \geq \epsilon|A|$ and $|B'| \geq \epsilon|B|$, the inequality $|d(A', B') - d(A, B)| \leq \epsilon$ holds.

A partition $\mathcal{A} = \{V_i : 1 \leq i \leq k\}$ of the vertex set of a graph is called an *equipartition* if $|V_i|$ and $|V_j|$ differ by no more than 1 for all $1 \leq i < j \leq k$ (so in particular each V_i has one of two possible sizes). For the sake of brevity, we will henceforth use the term partition to denote an equipartition. We call the number of sets in a partition (k above) the *order* of the partition.

Definition 2 (ϵ -Regular partition). A partition $\mathcal{V} = \{V_i : 1 \leq i \leq k\}$ of $V(G)$ for which all but at most $\epsilon \binom{k}{2}$ of the pairs (V_i, V_j) are ϵ -regular is called an ϵ -regular partition of $V(G)$.

The Regularity Lemma of Szemerédi can be formulated as follows.

Lemma 1 ([13]). For every m and $\epsilon > 0$ there exists an integer $T = T_{\square}(m, \epsilon)$ with the following property: Any graph G on $n \geq T$ vertices, has an ϵ -regular partition $\mathcal{V} = \{V_i : 1 \leq i \leq k\}$ with $m \leq k \leq T$.

The main drawback of the regularity-lemma is that the bounds on the integer T , and hence on the order of \mathcal{V} , have an enormous dependency on $1/\epsilon$. The current bounds are towers of exponents of height $O(1/\epsilon^5)$. This means that the regularity measure (ϵ in Lemma 1) is very large compared to the inverse of the order of the partition (k in Lemma 1). In some cases, however, we would like the regularity measure between the pairs to have some (strong) relation to the order of the partition. This leads to the following definition.

Definition 3 (f -Regular partition). For a function $f : N \mapsto (0, 1)$, a partition $\mathcal{V} = \{V_i : 1 \leq i \leq k\}$ of $V(G)$ is said to be f -regular if all pairs (V_i, V_j) , $1 \leq i < j \leq k$, are $f(k)$ -regular.

Note that as opposed to Definition 2, in the above definition, the order of the partition and the regularity measure between the sets of the partition go “hand in hand” via the function f . One can (more or less) rephrase Lemma 1 as saying that every graph has a $(\log^*(k))^{-1/5}$ -regular partition \square . Furthermore, Gowers [6] showed that this is close to being tight. Therefore, one cannot guarantee that a general graph has an f -regular partition for a function f approaching zero faster than roughly $1/\log^*(k)$. One should thus look for certain variants of this notion and still be able to show that any graph has a similar partition.

A step in this direction was first taken by Alon, Fischer, Krivelevich and Szegedy [2] who proved a stronger variant of the regularity lemma. See Lemma 2 below for the precise statement. The following is yet another variant of the regularity lemma that was recently proved independently by Rödl and Schacht [11] and by Lovász [9] (implicitly following a result of Lovász and Szegedy in [10]).

¹ This is not accurate because Definition 3 requires *all* pairs to be $f(k)$ -regular, while Lemma 1 guarantees that only *most* pairs are regular.

This lemma does not guarantee that for any f we can find an f -regular partition of any given graph. Rather, it shows that any graph is “close” to a graph that has an f -regular partition.

Theorem 1 ([11], [9]). *For every $m, \epsilon > 0$ and non-increasing function $f : N \mapsto (0, 1)$, there is an integer $T = T_{\square}(f, \epsilon, m)$ so that given a graph G with at least T vertices, one can add-to/remove-from G at most ϵn^2 edges and thus get a graph G' that has an f -regular partition of order k , where $m \leq k \leq T$.*

Our first result in this paper is a new short proof of the above theorem. The proof is a simple application of the variant of the regularity lemma of [2] mentioned above. Basing the proof on this method provides both explicit bounds and a polynomial time algorithm for finding the partition and the necessary modifications. Section 2 consists of the proof of Theorem 1 and in Section 3 we describe a deterministic polynomial time algorithm for finding a regular partition and a set of modifications that are guaranteed by this theorem.

We now turn to the second result of this paper. In many cases, one applies the regularity lemma on a graph G , to get an ϵ -regular partition $\mathcal{V} = \{V_i : 1 \leq i \leq k\}$ and then defines a weighted complete graph on k vertices $\{1, \dots, k\}$, in which the weight of the edge connecting vertices (i, j) is $d(V_i, V_j)$. This relatively small weighted graph, sometimes called the *regularity-graph* of G , carries a lot of information on G . For example, it can be used to approximately count the number of copies of any fixed small graph in G , and to approximate the size of the maximum-cut of G . A natural question, which was suggested to us by Madhu Sudan [12], is how different can two regularity-graphs of the same graph be. We turn to define what it means for two regularity graphs, or equivalently for two regular partitions, to be ϵ -isomorphic.

Definition 4 (ϵ -Isomorphic). *We say that two partitions $\mathcal{U} = \{U_i : 1 \leq i \leq k\}$ and $\mathcal{V} = \{V_i : 1 \leq i \leq k\}$ of a graph G are ϵ -isomorphic if there is a permutation $\sigma : [k] \mapsto [k]$, such that for all but at most $\epsilon \binom{k}{2}$ pairs $1 \leq i < j \leq k$, we have $|d(U_i, U_j) - d(V_{\sigma(i)}, V_{\sigma(j)})| \leq \epsilon$.*

We first show that if one considers the standard notion of an ϵ -regular partitions (as in Definition 2), then ϵ -regular partitions of the same graph are not necessarily similar. In fact, as the following theorem shows, even $f(k)$ -regular partitions of the same graph, where $f(k) = 1/k^\delta$, are not necessarily similar. A variant of this theorem has been proved by Lovász [9].

Theorem 2. *Let $f(k) = 1/k^{1/4}$. For infinitely many k , and for every $n > n_{\square}(k)$ there is a graph $G = (V, E)$ on n vertices with two f -regular partitions of order k that are not $\frac{1}{4}$ -isomorphic.*

The proof of Theorem 2 provides explicit examples. We note that an inexplicit probabilistic proof shows that the assertion of the theorem holds even for $f(k) = \Theta(\frac{\log^{1/3} k}{k^{1/3}})$. See Section 4 for more details.

Using the terminology of Definition 2, the above theorem and its proof can be restated as saying that for any (small) $\epsilon > 0$ and all large enough $n > n_0(\epsilon)$,

there exists an n vertex graph that has two ϵ -regular partitions of order ϵ^{-4} , that are not $\frac{1}{4}$ -similar. Therefore, ϵ -regular partitions of the same graph may be very far from isomorphic.

Recall now that Theorem 1 guarantees that for any function f , any graph can be slightly modified in a way that the new graph admits an f -regular partition. As the following theorem shows, whenever $f(k) < 1/k^2$ all the regular partitions of the new graph must be close to isomorphic.

Theorem 3. *Let $f(k)$ be any function satisfying $f(k) \leq \min\{1/k^2, \frac{1}{2}\epsilon\}$, and suppose \mathcal{U} and \mathcal{V} are two f -regular partitions of some graph G . Then \mathcal{U} and \mathcal{V} are ϵ -isomorphic.*

This theorem illustrates the power of f -regular partitions, showing that (for $f(k) < 1/k^2$) they enjoy properties that do not hold for usual regular partitions. Observe that the above results imply that when, e.g., $f(k) > 1/k^{\frac{1}{4}}$, then two f -regular partitions of the same graph are not necessarily similar, whereas whenever $f(k) < 1/k^2$ they are. It may be interesting to find a tight threshold for f that guarantees ϵ -isomorphism between f -regular partitions of the same graph. It should also be interesting to find a similar threshold assuring that partitions of two close graphs are similar.

2 Proof of Theorem 1

In this section we show how to derive Theorem 1 from a variant of the regularity lemma due to Alon et al. [2]. Before we get to the proof we observe the following three simple facts. First, a standard probabilistic argument shows that for every δ and η , and for every large enough $n > n_0(\delta)$ there exists a δ -regular pair (A, B) with $|A| = |B| = n$ and $d(A, B) = \eta$. [2] The additional two facts we need are given in the following two claims, where we use the notation $x = y \pm \epsilon$ to denote the fact that $y - \epsilon \leq x \leq y + \epsilon$.

Claim 2.1. *Let δ and γ be fixed positive reals and let $n > n_0(\delta, \gamma)$ be a large enough integer. Suppose (A, B) is a δ -regular pair satisfying $d(A, B) = \eta \pm \gamma$ and $|A| = |B| = n$. Then, one can add or remove at most $2\gamma n^2$ edges from (A, B) and thus turn it into a 3δ -regular pair satisfying $d(A, B) = \eta \pm \delta$.*

Proof. Let us assume that $d(A, B) = \eta + \gamma$. The general case where $\eta - \gamma \leq d(A, B) \leq \eta + \gamma$ is similar. Suppose we delete each of the edges connecting A and B with probability $\frac{\gamma}{\eta + \gamma}$. Clearly the expected value of $d(A, B)$ after these modifications is η and assuming n is large enough, we get from a standard application of Chernoff's bound that the probability that the new density deviates from η by more than δ is at most $\frac{1}{4}$. Also, the expected number of edges removed is γn^2 and again, if n is large enough, the probability that we removed more than

² Here and throughout the rest of the paper, we say that $d(A, B) = \eta$ if $|e(A, B) - \eta|A||B| \leq 1$. This avoids rounding problems arising from the fact that $\eta|A||B|$ may be non-integral.

$2\gamma n^2$ edges is at most $\frac{1}{4}$. Consider now two subsets $A' \subseteq A$ and $B' \subseteq B$ each of size δn . As (A, B) was initially δ -regular we initially had $d(A', B') = (\eta + \gamma) \pm \delta$. As each edge is removed with probability $\frac{\gamma}{\eta + \gamma}$ the expected value of $d(A', B')$ after these modifications is $\eta \pm \frac{\delta\eta}{\eta + \gamma} = \eta \pm \delta$. By Chernoff's bound we get that for large enough n , for every such pair (A', B') the probability that $d(A', B')$ deviates from $\eta \pm \delta$ by more than δ is bounded by 2^{-4n} . As there are less than 2^{2n} choices for (A', B') we get that with probability at least $\frac{3}{4}$ all pairs (A', B') have density $\eta \pm 2\delta$. To recap, we get that with probability at least $\frac{1}{4}$ we made at most $2\gamma n^2$ modifications, $d(A, B) = \eta \pm \delta$ and $d(A', B') = \eta \pm 2\delta$, implying that (A, B) is 3δ -regular. \square

Claim 2.2. *Let (A, B) be a pair of vertex sets with $|A| = |B| = n$. Suppose A and B are partitioned into subsets A_1, \dots, A_l and B_1, \dots, B_l such that all pairs (A_i, B_j) are $\frac{1}{4}\delta^2$ -regular and satisfy $d(A_i, B_j) = d(A, B) \pm \frac{1}{4}\delta$. Then (A, B) is δ -regular.*

Proof. Consider two subsets $A' \subseteq A$ and $B' \subseteq B$ of size δn each, and set $A'_i = A' \cap A_i$ and $B'_i = B' \cap B_i$. The number of pairs $(a \in A', b \in B')$, where $a \in A'_i$, $b \in B'_j$, and either $|B'_j| < \frac{1}{4}\delta^2|B_j|$ or $|A'_i| < \frac{1}{4}\delta^2|A_i|$ is bounded by $\frac{1}{2}\delta^3 n^2$. Therefore, the possible contribution of such pairs to $d(A', B')$ is bounded by $\frac{1}{2}\delta$.

Consider now the pairs (A'_i, B'_j) satisfying $|B'_j| \geq \frac{1}{4}\delta^2|B_j|$ and $|A'_i| \geq \frac{1}{4}\delta^2|A_i|$. As (A_i, B_j) is $\frac{1}{4}\delta^2$ -regular we have $d(A'_i, B'_j) = d(A_i, B_j) \pm \frac{1}{4}\delta$. As $d(A_i, B_j) = d(A, B) \pm \frac{1}{4}\delta$ we conclude that $d(A'_i, B'_j) = d(A, B) \pm \frac{1}{2}\delta$. As the pairs discussed in the preceding paragraph can change $d(A', B')$ by at most $\frac{1}{2}\delta$, we conclude that $d(A', B') = d(A, B) \pm \delta$, as needed. \square

The following is the strengthened version of the regularity lemma, due to Alon et al. [2], from which we will deduce Theorem 1. \square

Lemma 2 ([2]). *For every integer m and function $f : N \mapsto (0, 1)$ there exists an integer $T = T_{[2]}(m, f)$ with the following property: If G is a graph with $n \geq T$ vertices, then there exists a partition $\mathcal{A} = \{V_i : 1 \leq i \leq k\}$ and a refinement $\mathcal{B} = \{V_{i,j} : 1 \leq i \leq k, 1 \leq j \leq l\}$ of \mathcal{A} that satisfy:*

1. $|\mathcal{A}| = k \geq m$ but $|\mathcal{B}| = kl \leq T$.
2. For all $1 \leq i < i' \leq k$, for all $1 \leq j, j' \leq l$ but at most $f(k)l^2$ of them, the pair $(V_{i,j}, V_{i',j'})$ is $f(k)$ -regular.
3. All $1 \leq i < i' \leq k$ but at most $f(0)\binom{k}{2}$ of them are such that for all $1 \leq j, j' \leq l$ but at most $f(0)l^2$ of them, $|d(V_i, V_{i'}) - d(V_{i,j}, V_{i',j'})| < f(0)$ holds.

Proof of Theorem 1. Given a graph G , an integer m , a real ϵ and some function $f : N \mapsto (0, 1)$ as an input to Theorem 1, let us apply Lemma 2 with the function $f'(k) = \min\{f^2(k)/12, \epsilon/8\}$ and with $m' = m$. By Lemma 2, if G has more than $T = T_{[2]}(m', f')$ vertices, then G has two partitions $\mathcal{A} = \{V_i : 1 \leq i \leq k\}$ and $\mathcal{B} = \{V_{i,j} : 1 \leq i \leq k, 1 \leq j \leq l\}$ satisfying the three assertions of the lemma.

We claim that we can make less than ϵn^2 modifications in a way that all pairs (V_i, V_j) will become $f(k)$ -regular.

We start by considering the pairs $(V_{i,j}, V_{i',j'})$, with $i < i'$, which are not $f'(k)$ -regular. Every such pair is simply replaced by an $f'(k)$ -regular bipartite graph of density $d(V_{i,j}, V_{i',j'})$. Such a pair exists by the discussion at the beginning of this section. The number of edge modifications needed for each such pair is at most $(n/kl)^2$ and by the second assertion of Lemma 2 we get that the total number of modifications we make at this stage over all pairs (V_i, V_j) is bounded by $\binom{k}{2} \cdot f'(k)l^2 \cdot (n/kl)^2 \leq \frac{\epsilon}{8}n^2$.

We now consider the pairs $(V_i, V_{i'})$ that do not satisfy the third assertion of Lemma 2, that is, those for which there are more than $f'(0)l^2$ pairs $1 \leq j, j' \leq l$ satisfying $|d(V_i, V_{i'}) - d(V_{i,j}, V_{i',j'})| \geq f'(0)$. For every such pair $(V_i, V_{i'})$ we simply remove all edges connecting V_i and $V_{i'}$. As by the third assertion there are at most $f'(0)\binom{k}{2} < \frac{\epsilon}{8}k^2$ such pairs, the total number of edge modifications we make is bounded by $\frac{\epsilon}{8}n^2$.

We finally consider the pairs $(V_i, V_{i'})$ that satisfy the third assertion of Lemma 2. Let us denote $d = d(V_i, V_{i'})$. We start with pairs $(V_{i,j}, V_{i',j'})$ satisfying $|d - d(V_{i,j}, V_{i',j'})| \geq f'(0)$. Each such pair is replaced with an $f'(k)$ -regular pair of density d . As there are at most $f'(0)l^2 \leq \frac{\epsilon}{8}l^2$ such pairs in each pair (V_i, V_j) , the total number of modifications made in the whole graph due to such pairs is bounded by $\frac{\epsilon}{8}n^2$. Let us now consider the pairs $(V_{i,j}, V_{i',j'})$ satisfying $|d - d(V_{i,j}, V_{i',j'})| \leq f'(0)$. If $d(V_{i,j}, V_{i',j'}) = d \pm f'(k)$ we do nothing. Otherwise, we apply Claim 2.1 on $(V_{i,j}, V_{i',j'})$ with $\eta = d$, $\gamma = |d - d(V_{i,j}, V_{i',j'})|$ and $\delta = f'(k)$. Note that here we are guaranteed to have $\gamma \leq f'(0) \leq \frac{1}{8}\epsilon$. Claim 2.1 guarantees that we can make at most $2\gamma(n/kl)^2 \leq \frac{1}{4}\epsilon(n/kl)^2$ modifications and thus turn $(V_{i,j}, V_{i',j'})$ into a $3f'(k)$ -regular pair with density $d \pm f'(k)$. The total number of modifications over the entire graph is bounded by $\frac{\epsilon}{4}n^2$.

To conclude, the overall number of modifications we have made in the above stages is less than ϵn^2 , as needed. Moreover, at this stage all the pairs $(V_{i,j}, V_{i',j'})$ satisfy $|d(V_{i,j}, V_{i',j'}) - d(V_i, V_{i'})| \leq f'(k) \leq \frac{1}{4}f(k)^2$ and they are all $\frac{1}{4}f^2(k)$ -regular. Therefore, by Claim 2.2 all pairs (V_i, V_j) are $f(k)$ -regular, as needed. □

3 Deterministic Algorithmic Version of Theorem 1

As mentioned before, we show that it is also possible to obtain an algorithmic version of Theorem 1. Here is a rough sketch, following the proof of Theorem 1 step by step. As described in 2, one can obtain the partition of Lemma 2 in polynomial time. In order to find the modifications that make it f -regular, the random graphs can be replaced by appropriate pseudo-random bipartite graphs. The last ingredient we need is an algorithm for finding the modifications to a bipartite graph (A, B) that are guaranteed by Claim 2.1. The algorithm we describe here combines the use of conditional probabilities (see, e.g., 3) with a certain local condition that ensures regularity. We first describe such a condition.

Given a bipartite graph on a pair of vertex sets (A, B) we denote by $d_{C_4}(A, B)$ the density of four-cycles in (A, B) , namely, the number of copies of C_4 divided by $\binom{|A|}{2}\binom{|B|}{2}$. A pair (A, B) is said to be ϵ -quad-regular if $d_{C_4}(A, B) = d^4(A, B) \pm \epsilon$. This local condition indeed ensures ϵ -regularity, as detailed in the following Lemma. The proof of the lemma appears in [5] and is based on the results of [1].

Lemma 3 ([5]). *Let (A, B) be a bipartite graph on A and B where $|A| = |B| = n$ and $\delta > 0$. Then:*

1. *If (A, B) is $\frac{1}{4}\delta^{10}$ -quad-regular then it is δ -regular.*
2. *If (A, B) is δ -regular then it is 8δ -quad-regular.*

We shall design a deterministic algorithm for the following slightly weaker version of Claim 2.1.

Claim 3.1. *There is a deterministic polynomial time algorithm that given a $\frac{1}{200}\delta^{20}$ -regular pair (A, B) with n vertices in each part (with n large enough) and $d(A, B) = \eta \pm \gamma$, modifies up to $2\gamma n^2$ edges and thus turns the bipartite graph into a 2δ -regular pair with edge density $d'(A, B) = \eta \pm \delta$.*

Note that the polynomial loss in the regularity measure with respect to Claim 2.1 can be evened by modifying the definition of f' in the proof of Theorem 1 so that $f'(k) = \min\{f^2(k)/8, \epsilon^{20}/2000\}$. Hence Claim 3.1 indeed implies an algorithm for finding the modifications and partition guaranteed by Theorem 1.

Proof of Claim 3.1: Assume $d(A, B) = \eta + \gamma$ and $\gamma > \delta$. The case $d(A, B) = \eta - \gamma$ can be treated similarly.

Consider an arbitrary ordering of the edges of (A, B) and a random process in which each edge is deleted independently with probability $\frac{\gamma}{\eta + \gamma}$. We first consider this setting and later show that a sequence of deterministic choices of the deletions can be applied so that the resulting graph satisfies the desired properties.

Define the indicator random variable X_i , $1 \leq i \leq t = \eta n^2$, for the event of not deleting the i 'th edge. Denote the number of four cycles in (A, B) by $s = d_{C_4}(A, B)\binom{n}{2}^2$, and arbitrarily index them by $1, \dots, s$. For every C_4 in (A, B) define the indicator Y_i , $1 \leq i \leq s$, for the event of its survival (i.e., none of its edges being deleted). Also let $X = \sum_{i=1}^t X_i$ and $Y = \sum_{i=1}^s Y_i$ which account for the numbers of edges and four-cycles (respectively) at the end of this process. Now define the following conditional expectations for $i = 0, 1, \dots, t$.

$$f_i(x_1, \dots, x_i) = \mathbb{E} \left[n^4(X - \eta n^2)^2 + (Y - \eta^4 \binom{n}{2}^2)^2 \mid X_1 = x_1, \dots, X_i = x_i \right] \quad (1)$$

where the expectation in (1) is taken over a uniform independent choice of X_{i+1}, \dots, X_t .

We first obtain an upper bound on f_0 . Since $X \sim B((\eta + \gamma)n^2, \frac{\eta}{\eta + \gamma})$, hence $\mathbb{E}[(X - \eta n^2)^2] = V(X) = O(n^2)$ and thus the first term in the expression for f_0 is $O(n^6)$. The expectation of the second term is

$$\mathbb{E}[(Y - \eta^4 \binom{n}{2}^2)^2] = \mathbb{E}[Y^2] - 2\mathbb{E}[Y]\eta^4 \binom{n}{2}^2 + \eta^8 \binom{n}{2}^4$$

For the linear term we have $\mathbb{E}[Y] = \sum_{i=1}^s \mathbb{E}[Y_i] = s(\frac{\eta}{\eta+\gamma})^4$. As for the quadratic term, for any pair $1 \leq i < j \leq s$ of four-cycles which share no common edge, the corresponding Y_i and Y_j are independent and hence $\mathbb{E}[Y_i Y_j] = (\frac{\eta}{\eta+\gamma})^8$. There are only $O(n^6)$ non-disjoint pairs of C_4 s, thus $\mathbb{E}[Y^2] = \mathbb{E}[\sum_{1 \leq i, j \leq s} Y_i Y_j] = s^2(\frac{\eta}{\eta+\gamma})^8 \pm O(n^6)$. By Lemma 3, $d_{C_4}(A, B) = (\eta + \gamma)^4 \pm \frac{1}{25}\delta^{20}$ and so $s = ((\eta + \gamma)^4 \pm \frac{1}{25}\delta^{20})\binom{n}{2}$. Therefore, we conclude that

$$\begin{aligned} \mathbb{E}[(Y - \eta^4 \binom{n}{2})^2] &= s^2(\frac{\eta}{\eta+\gamma})^8 \pm O(n^6) - 2s(\frac{\eta}{\eta+\gamma})^4 \eta^4 \binom{n}{2}^2 + \eta^8 \binom{n}{2}^4 \\ &\leq \frac{1}{5}\delta^{20} \binom{n}{2}^4 + O(n^6) \end{aligned}$$

This implies that altogether, for a large enough n , $f_0 \leq \frac{1}{4}\delta^{20} \binom{n}{2}^4$.

However, each $f_i(x_1, \dots, x_i)$ is a convex combination of $f_{i+1}(x_1, \dots, x_i, 0)$ and $f_{i+1}(x_1, \dots, x_i, 1)$. Thus, there is some choice of a value x_{i+1} for X_{i+1} such that $f_{i+1}(x_1, \dots, x_{i+1}) \leq f_i(x_1, \dots, x_i)$. Therefore, choosing an x_{i+1} that minimizes f_{i+1} sequentially for $i = 0, \dots, t-1$ results in an assignment of (x_1, \dots, x_t) such that $f_t(x_1, \dots, x_t) \leq f_0 \leq \frac{1}{4}\delta^{20} \binom{n}{2}^4$. In order to apply this process, one needs to be able to efficiently compute f_i . But this is straightforward, since for any partial assignment of values to the X_i s, the mutual distribution of any pair Y_i, Y_j can be calculated in time $O(1)$. Therefore, since there are at most $O(n^8)$ pairs of four-cycles, computing the expected value of the sum in (1) requires $O(n^8)$ operations. Repeating this for each edge accumulates to $O(n^{10})$. 3

To complete the proof of the claim, we only need to show that the modifications we obtained above, namely such that (x_1, \dots, x_t) satisfy $f_t(x_1, \dots, x_t) \leq \frac{1}{4}\delta^{20} \binom{n}{2}^4$, are guaranteed to satisfy the conditions of the claim. Indeed, in this case, each of the two addends which sum up to f_t is bounded by $\frac{1}{4}\delta^{20} \binom{n}{2}^4$. By the first addend, the new edge density d' is $d' = \eta \pm \frac{1}{2}\delta^{10}$. Thus, with much room to spare, the conditions on the edge density and the number of modifications are fulfilled. Note that it also follows that $d'^4 = \eta^4 \pm 3\delta^{10}$ (for, e.g., $\delta < \frac{1}{4}$), and the second addend implies that the new four-cycles density is $\eta^4 \pm \frac{1}{2}\delta^{10} = d'^4 \pm 4\delta^{10}$. By Lemma 3 the pair is now $4^{1/5}\delta$ -regular, and hence the modified graph attains all the desired properties. □

Remark 1. Another possible proof of Claim 3.1 can be obtained by using an appropriate 8-wise independent space for finding (x_1, \dots, x_t) such that f_t attains at most its expected value.

4 Isomorphism of Regular Partitions

In this section we prove Theorems 2 and 3. In order to simplify the presentation, we omit all floor and ceiling signs whenever these are not crucial. We

³ Note that each edge effects only at most $O(n^6)$ pairs of four cycles, thus the complexity can easily be reduced to $O(n^8)$, and in fact can be further reduced by a more careful implementation.

start with the proof of Theorem 2. The basic ingredient of the construction is a pseudo-random graph which satisfies the following conditions.

Lemma 4. *Let k be a square of a prime power, then there exists a graph $F = (V, E)$ on $|V| = k$ vertices such that*

1. F is $\lfloor k/2 \rfloor$ -regular, and hence $d(V, V) = \frac{\lfloor k/2 \rfloor}{k}$
2. For any pair of vertex sets A and B , if $|A| \geq k^{\frac{3}{4}}$ and $|B| \geq k^{\frac{3}{4}}$, then $d(A, B) = d(V, V) \pm k^{-\frac{1}{4}}$

Proof: We use some known pseudo-random graphs as follows, see the survey [8] for further definitions and details. An (n, d, λ) -graph is a d -regular graph on n vertices all of whose eigenvalues, except the first one, are at most λ in their absolute values. It is well known that if λ is much smaller than d , then such graphs have strong pseudo-random properties. In particular, (see, e.g., [3], Chapter 9), in this case for any two sets of vertices A and B of G : $d(A, B) = \frac{d}{n} \pm \lambda(|A||B|)^{-\frac{1}{2}}$. Thus, it is easy to verify that a $(k, \lfloor \frac{k}{2} \rfloor, \sqrt{k})$ -graph would satisfy the assertions of the lemma.

There are many known explicit constructions of (n, d, λ) -graphs. Specifically, we use the graph constructed by Delsarte and Goethals and by Turyn (see [8]). In this graph the vertex set $V(G)$ consists of all elements of the two dimensional vector space over $GF(q)$, where q is a prime power, so G has $k = q^2$ vertices. To define the edges of G we fix a set L of $\frac{q+1}{2}$ lines through the origin. Two vertices x and y of the graph G are adjacent if $x - y$ is parallel to a line in L . It is easy to check that this graph is $\frac{(q+1)(q-1)}{2} = \frac{q^2-1}{2}$ -regular. Moreover, because it is a strongly regular graph, one can compute its eigenvalues precisely and show that besides the first one they all are either $-\frac{q+1}{2}$ or $\frac{q-1}{2}$. Therefore, indeed, we obtain an $(k, \lfloor \frac{k}{2} \rfloor, \lambda)$ -graph with $\lambda < \sqrt{k}$ as necessary. □

Proof of Theorem 2: We construct our example as follows. Pick a graph F on k vertices $V(F) = \{1, \dots, k\}$ which satisfies the conditions of Lemma 4. Suppose $n \geq k^2$. The graph on n vertices G will be an $\frac{n}{k}$ blow-up of F : every vertex of F is replaced by an independent set of size $\frac{n}{k}$, and each edge is replaced by a complete bipartite graph connecting the corresponding independent sets. Every non-edge corresponds to an empty bipartite graph between the parts. Let $\mathcal{U} = \{U_i : 1 \leq i \leq k\}$ be the partition of $V(G)$ where U_i is an independent set which corresponds to the vertex i in F . It follows from the construction that for any $1 \leq i < j \leq k$ the edge density of (U_i, U_j) is either 0 or 1, and (U_i, U_j) is ϵ -regular for any $\epsilon > 0$. The second partition \mathcal{V} is generated by arbitrarily splitting every U_i into k equal-sized sets $W_{i,t}$, $1 \leq t \leq k$, and setting $V_t = \bigcup_{i=1}^k W_{i,t}$. Note that for any $1 \leq i < j \leq k$ the edge density $d_G(V_i, V_j)$ is exactly $d_F(V(F), V(F))$. Yet by Lemma 4, $d_F(V(F), V(F)) = \frac{2e(F)}{k^2} = \frac{\lfloor k/2 \rfloor}{k}$, which for $k \geq 2$ is strictly between $\frac{1}{4}$ and $\frac{3}{4}$. Hence \mathcal{U} and \mathcal{V} are not $\frac{1}{4}$ -similar, as $|d(U_i, U_j) - d(V_i, V_{j'})| > \frac{1}{4}$ for all pairs $i < j$ and $i' < j'$.

Thus, we complete the proof of the theorem by showing that all pairs (V_i, V_j) are $k^{-\frac{1}{4}}$ -regular. Suppose, towards a contradiction and without loss of generality, that there are subsets $A \subseteq V_1$ and $B \subseteq V_2$ such that $|A| \geq k^{-\frac{1}{4}}|V_1|$, $|B| \geq k^{-\frac{1}{4}}|V_2|$ and $|d(A, B) - d(V_1, V_2)| > k^{-\frac{1}{4}}$.

For any $1 \leq i \leq k$ we denote $A_i = A \cap W_{i,1}$ and $B_i = B \cap W_{i,2}$. For any vertex $x \in A$, let the *fractional degree* of x with respect to B be defined by $d_B(x) = e(\{x\}, B)/|B|$. Note that $d(A, B) = \frac{1}{|A|} \sum_{x \in A} d_B(x)$ and that if x_1 and x_2 come from the same $W_{i,1}$, then $d_B(x_1) = d_B(x_2)$. Therefore, $d(A, B)$ is a convex combination

$$d(A, B) = \sum_{i=1}^k \frac{|A_i|}{|A|} d_B(x \in W_{i,1})$$

of (at most) k possible fractional degrees of vertices in A , which come from different sets $W_{i,1}$.

First assume that $d(A, B) > d(V_1, V_2) + k^{-\frac{1}{4}}$. We sort the vertices of A by their fractional degrees with respect to B , and consider a subset \hat{A} of V_1 which consists of the union of the $k^{\frac{3}{4}}$ sets $W_{i,1}$ which have the highest fractional degrees with respect to B . Since $|A| > k^{-\frac{1}{4}}|V_1| = |\hat{A}|$, it follows that $d(\hat{A}, B) \geq d(A, B)$. Similarly, by considering the fractional degrees of the vertices of B with respect to the new subset \hat{A} , we may obtain a subset \hat{B} of V_2 such that $d(\hat{A}, \hat{B}) \geq d(\hat{A}, B) \geq d(A, B) > d(V_1, V_2) + k^{-\frac{1}{4}}$. It also follows that both \hat{A} and \hat{B} are unions of sets $W_{i,1}$ and $W_{i,2}$ respectively. Thus, the edge density $d(\hat{A}, \hat{B})$ is exactly the edge density of the corresponding vertex sets in F (both of size $k^{\frac{3}{4}}$). By Lemma 4, we get that $d(\hat{A}, \hat{B}) \leq d_F(V(F), V(F)) + k^{-\frac{1}{4}} = d_G(V_1, V_2) + k^{-\frac{1}{4}}$, which leads to a contradiction and completes the proof of Theorem 2. The case where $d(A, B) < d(V_1, V_2) - k^{-\frac{1}{4}}$ can be treated similarly. \square

Remark 2. By using the random graph $G(k, \frac{1}{2})$ one could establish an inexplicit probabilistic proof for an analog of Lemma 4. The proof applies standard Chernoff bounds on the number of edges between *any* pair of *small* vertex sets. This extends the result for any $k > 2$ and with a stronger regularity constraint. Repeating the proof of Theorem 2 with such a graph F implies that Theorem 2 holds even for $f(k) = \Theta(\frac{\log^{1/3} k}{k^{1/3}})$.

We conclude this section with the proof of Theorem 3.

Proof of Theorem 3: Consider two f -regular partitions $\mathcal{U} = \{U_i : 1 \leq i \leq k\}$ and $\mathcal{V} = \{V_i : 1 \leq i \leq k\}$ of order k . Let $W_{i,j}$ denote $V_i \cap U_j$. Consider a matrix A where $A_{i,j} = \frac{|W_{i,j}|}{|V_i|}$ is the fraction of vertices of V_i in U_j , and note that A is doubly stochastic, that is, the sum of entries in each column and row is precisely 1. A well known (and easy) theorem of Birkhoff [4] guarantees that A is a convex combination of (less than) k^2 permutation matrices. In other words, there are k^2 permutations $\sigma_1, \dots, \sigma_{k^2}$ of the elements $\{1, \dots, k\}$, and k^2 reals $0 \leq \lambda_1, \dots, \lambda_{k^2} \leq 1$ such that $\sum_t \lambda_t = 1$ and $A = \sum_t \lambda_t A_{\sigma_t}$, where A_{σ} is the

permutation matrix corresponding to σ . Let λ_p be the largest of these k^2 coefficients. Clearly $\lambda_p \geq 1/k^2$, and observe that as A is a convex combinations of the matrices A_{σ_i} , this means that for every $1 \leq i \leq k$ we have $|W_{i,\sigma_p(i)}| \geq \frac{1}{k^2}|V_i|$ and similarly $|W_{i,\sigma_p(i)}| \geq \frac{1}{k^2}|U_{\sigma_p(i)}|$. As both \mathcal{V} and \mathcal{U} are assumed to be $f(k)$ -regular and $f(k) \leq \min\{1/k^2, \epsilon/2\}$, this guarantees that for all $1 \leq i < j \leq k$ we have

$$|d(V_i, V_j) - d(U_{\sigma_p(i)}, U_{\sigma_p(j)})| \leq$$

$$|d(V_i, V_j) - d(W_{i,\sigma_p(i)}, W_{j,\sigma_p(j)})| + |d(W_{i,\sigma_p(i)}, W_{j,\sigma_p(j)}) - d(U_{\sigma_p(i)}, U_{\sigma_p(j)})| \leq \epsilon,$$

completing the proof. \square

Acknowledgments. We would like to thank Madhu Sudan for a conversation that initiated this study, and Laci Lovász for fruitful discussions.

References

1. Alon, N., Duke, R.A., Lefmann, H., Rödl, V., Yuster, R.: The algorithmic aspects of the Regularity Lemma. In: Proc. 33rd IEEE FOCS, Pittsburgh, pp. 473–481. IEEE, Los Alamitos (1992) Also: J. of Algorithms 16, 80–109 (1994)
2. Alon, N., Fischer, E., Krivelevich, M., Szegedy, M.: Efficient testing of large graphs. In: Proc. of the 40 IEEE FOCS, pp. 656–666. IEEE, Los Alamitos (1999) Also: Combinatorica 20, 451–476 (2000)
3. Alon, N., Spencer, J.H.: The Probabilistic Method, 2nd edn. Wiley, New York (2000)
4. Birkhoff, G.: Three observations on linear algebra. Univ. Nac. Tucumán. Rev. Ser. A 5, 147–151 (1946)
5. Fischer, E., Matsliach, A., Shapira, A.: A hypergraph approach for finding small regular partitions. (preprint 2007)
6. Gowers, T.: Lower bounds of tower type for Szemerédi’s uniformity lemma. GAFA 7, 322–337 (1997)
7. Komlós, J., Simonovits, M.: Szemerédi’s Regularity Lemma and its applications in graph theory. In: Miklós, D., Sós, V.T., Szönyi, T. (eds.) Combinatorics, Paul Erdős is Eighty, Budapest. János Bolyai Math. Soc., vol. II, pp. 295–352 (1996)
8. Krivelevich, M., Sudakov, B.: Pseudo-random graphs. In: Györi, E., Katona, G.O.H., Lovász, L. (eds.) More sets, graphs and numbers. Bolyai Society Mathematical Studies, vol. 15, pp. 199–262.
9. Lovász, L.: Private communication (2006)
10. Lovász, L., Szegedy, B.: Szemerédi’s lemma for the analyst, GAFA (to appear)
11. Rödl, V., Schacht, M.: Regular partitions of hypergraphs, Combinatorics, Probability and Computing (to appear)
12. Sudan, M.: Private communication (2005)
13. Szemerédi, E.: Regular partitions of graphs. In: Bermond, J.C., Fournier, J.C., Las Vergnas, M., Sotteau, D. (eds.) Proc. Colloque Inter. CNRS, pp. 399–401 (1978)

Algorithms for Core Stability, Core Largeness, Exactness, and Extendability of Flow Games^{*}

Qizhi Fang¹, Rudolf Fleischer², Jian Li^{2,**}, and Xiaoxun Sun³

¹ Department of Mathematics, Ocean University of China, Qingdao, China
qfang@ouc.edu.cn

² Department of Computer Science and Engineering
Shanghai Key Laboratory of Intelligent Information Processing
Fudan University, Shanghai, China
rudolf,lijian83@fudan.edu.cn

³ Department of Mathematics and Computing, University of Southern Queensland
w0072830@mail.connect.usq.edu.au

Abstract. In this paper, we give linear time algorithms to decide core stability, core largeness, exactness, and extendability of flow games on uniform networks (all edge capacities are 1). We show that a uniform flow game has a stable core if and only if the network is a balanced DAG (for all non-terminal vertices, indegree equals outdegree), which can be decided in linear time. Then we show that uniform flow games are exact, extendable, and have a large core if and only if the network is a balanced directed series-parallel graph, which again can be decided in linear time.

1 Introduction

In 1944, von Neumann and Morgenstern [19] introduced the concept of *stable sets* to analyse bargaining situations in cooperative games. In 1968, Lucas [13] described a ten-person game without a stable set. Deng and Papadimitriou [5] pointed out that deciding the existence of a stable set for a given cooperative game is not known to be computable. Jain and Vohra [8] recently showed that it is decidable whether a balanced game has a stable core. When the game is convex, the core is always a stable set. In general, however, the core and the stable set are not related. Hence the question arises: when do the core and the stable set coincide, and how can we decide core stability?

Several sufficient conditions for core stability have been discussed in the literature. Sharkey [14] introduced the concepts of subconvexity of a game and core largeness. He showed that convexity implies subconvexity which implies core largeness which implies core stability. In an unpublished paper, Kikuta and Shapley [12] studied another concept, later called extendability of the game by

* The work described in this paper was supported by NCET, NSFC (No. 10371114s and No. 70571040/G0105) and partially supported by NSFC (No. 60573025).

** The work was partially done when this author was visiting The Hong Kong University of Science and Technology.

Gellekom *et al.* [18], and proved that it is necessary for core largeness and sufficient for core stability.

However, only few results are known about core stability and related concepts for concrete cooperative games. Solymosi and Raghavan [15] studied these concepts for assignment games, and Bietenhader and Okamoto [1] studied them for minimum coloring games on perfect graphs.

In this paper we study flow games, introduced by Kalai and Zemel [10,11], which arise from the profit distribution problem related to the maximum flow in networks. We give the first linear time algorithms to decide core stability, core largeness, exactness, and extendability of uniform flow games (all edge capacities are 1). We obtain these efficient algorithms by characterizing structural properties of those networks that have flow games with the desired properties. These characterizations might be useful in other contexts.

We show that a uniform flow game has a stable core if and only if the network is a balanced DAG (for all non-terminal vertices, indegree equals outdegree). This can easily be tested in linear time, so we get a linear time algorithm to decide core stability, which improves a previous algorithm with runtime $O(|V|^2 \cdot |E|^2)$ [16]. We also show that uniform flow games are exact, extendable, and have a large core if and only if the network is a balanced directed series-parallel graph. Again, this can be tested in linear time [17], so we also get a linear time algorithm to decide exactness, extendability, and core largeness of uniform flow games. In [16], Sun *et al.* established the equivalence of these three properties and proved them to be equivalent to a certain structural property of the network (see Section 2.2.4) but left it as an open problem to design an efficient algorithm to decide this property. Note that core largeness, exactness, and extendability of a flow game all imply core stability (because flow games are totally balanced).

This paper is organized as follows. In Section 2 we define cooperative games and review some results on the core of flow games and its stability. In Section 3 we characterize those networks that have uniform flow games with these properties, leading to linear time algorithms to decide them. We conclude with some open problems in Section 4.

2 Definitions

2.1 Graphs

A *flow network* is a directed graph $G = (V, E; \omega)$, where V is the vertex set, E is the edge set, and $\omega : E \rightarrow \mathbb{R}^+$ is the edge capacity function. Let s (the *source*) and t (the *sink*) be two distinct vertices of G . W.l.o.g., we assume that every edge in E lies on some simple (s, t) -path. G is a *uniform flow network* if all edge capacities are equal. By scaling the capacities we can w.l.o.g. assume that the edge capacities are equal to one in this case.

If S and T partition the vertex set into two parts such that $s \in S$ and $t \in T$, then the set C of edges going from a node in S to a node in T is an (s, t) -cut. The *capacity* of C is the sum of its edge capacities. C is a *minimal* (s, t) -cut if no proper subset of C is an (s, t) -cut. C is a *minimum* (s, t) -cut if it has smallest

capacity among all (s, t) -cuts. We say a uniform network G is *cut-normal* if every minimal (s, t) -cut is already a minimum (s, t) -cut.

A directed graph is a *DAG* (*directed acyclic graph*) if it does not contain a directed cycle. A *2-terminal DAG* is a DAG with two vertices s and t such that the indegree of s and the outdegree of t are zero and every other vertex appears in at least one simple (s, t) -path. A 2-terminal DAG is *balanced* if the indegree of every vertex other than s and t equals its outdegree.

A *2-terminal directed series-parallel graph* (*2-DSPG*) is a directed graph with two distinguished vertices (*terminals*) s and t that is obtained inductively as follows. A basic 2-DSPG consists of two terminals s and t , connected by a directed edge (s, t) . If G_1 and G_2 are 2-DSPGs with terminals s_i and t_i , $i = 1, 2$, then we can combine them in series by identifying t_1 with s_2 to obtain a 2-DSPG with terminals s_1 and t_2 , or in parallel by identifying s_1 with s_2 and t_1 with t_2 to obtain a 2-DSPG with terminals the combined vertex s_1/s_2 and the combined vertex t_1/t_2 .

2.2 Cooperative Games

A *cooperative (profit) game* $\Gamma = (N, v)$ consists of a player set $N = \{1, 2, \dots, n\}$ and a profit function $v : 2^N \rightarrow \mathbb{R}$ with $v(\emptyset) = 0$. A *coalition* S is a non-empty subset of N , and $v(S)$ represents the profit that can be achieved by the players in S without help of other players. The central problem in a cooperative game is how to allocate the total profit $v(N)$ among the individual players in a ‘fair’ way. An *allocation* is a vector $x \in \mathbb{R}^n$ with $x(N) = v(N)$, where $x(S) = \sum_{i \in S} x_i$ for any $S \subseteq N$.

Different requirements for fairness, stability and rationality lead to different optimal allocations which are generally called *solution concepts*. The core is an important solution concept.

An allocation $x \in \mathbb{R}^n$ is called an *imputation* if $x_i \geq v(\{i\})$ for all players $i \in N$. Every player is happy in this case because he gets at least as much as he could expect from the profit function of the game. We denote by $X(\Gamma)$ the set of imputations of Γ .

The *core* $C(\Gamma)$ of Γ is the set of imputations where no coalition S has an incentive to split off from the grand coalition N and go their own way. Formally, $C(\Gamma) = \{x \in \mathbb{R}^n \mid x(N) = v(N) \text{ and } x(S) \geq v(S) \text{ for all } S \subseteq N\}$. A game is *balanced* if its core is not empty.

For a subset $S \subseteq N$, the *induced subgame* (S, v_S) on S has profit function $v_S(T) = v(T)$ for each $T \subseteq S$. A cooperative game Γ is called *totally balanced* if all its subgames are balanced, i.e., all its subgames have non-empty cores.

2.3 Core Stability, Core Largeness, Extendability, and Exactness

In their classical work on game theory, von Neumann and Morgenstern [19] introduced the stable set which is very useful for the analysis of bargaining situations. Suppose that x and y are imputations. We say that x *dominates* y if there is a coalition S such that $x(S) \leq v(S)$ and $x_i > y_i$ for each $i \in S$. A

set \mathcal{F} of imputations is *stable* if no two imputations in \mathcal{F} dominate each other, and any imputation not in \mathcal{F} is dominated by at least one imputation in \mathcal{F} . In particular, the core of a game is stable if for any non-core imputation y there is a core imputation x dominating y , i.e., $x(S) = v(S)$ and $x_i > y_i$ for each $i \in S$.

There are three other concepts closely related to the core stability. Let $\Gamma = (N, v)$ be an n -player cooperative game. The core of Γ is *large* if for every $y \in \mathbb{R}^n$ satisfying $y(S) \geq v(S)$, for all $S \subseteq N$, there exists a core imputation x such that $x \leq y$. Γ is *extendable* if for every $S \subseteq N$ and every core imputation y of the subgame (S, v_S) there exists a core imputation $x \in C(\Gamma)$ such that $x_i = y_i$ for all $i \in S$. Γ is called *exact* if for every $S \subset N$ there exists $x \in C(\Gamma)$ such that $x(S) = v(S)$.

Kikuta and Shapley [12] showed that a balanced game with a large core is extendable, and an extendable balanced game has a stable core. Sharkey [14] showed that a totally balanced game with a large core is exact. Biswas *et al.* [2] pointed out that extendability also implies exactness. Note that flow games are totally balanced.

2.4 Flow Games

Flow games were introduced by Kalai and Zemel [10,11]. Consider a flow network $G = (V, E; \omega)$. In the corresponding *flow game* $\Gamma = (E, v)$ on G each player controls one edge. The profit $v(S)$ of a coalition $S \subseteq E$ is the value of a maximum (s, t) -flow that only uses edges controlled by S . The flow game is *uniform* if the underlying network is uniform.

In this paper, we focus on uniform flow games. These games belong to the class of packing and covering games introduced by Deng *et al.* [4]. Kalai and Zemel [11] showed that flow games are totally balanced. Fang *et al.* [7] proved that the problem of testing membership in the core of a flow game is *co-NP*-complete. Deng *et al.* [4] showed that the core of a uniform flow game is exactly the convex hull of the indicator vectors of the minimum (s, t) -cuts of the network, which can be computed in polynomial time.

An edge $e \in E$ is called a *dummy edge* if $v(E \setminus \{e\}) = v(E)$, i.e., removal of e does not change the value of the maximum (s, t) -flow. Sun *et al.* [16] showed that a uniform flow game has a stable core if and only if the network does not contain dummy edges. Based on this structural property they designed an $O(|V|^2 \cdot |E|^2)$ time algorithm to decide core stability of uniform flow games. In Section 3 we will see that we can recognize graphs without dummy edges much faster. Note that dummy edges also play a role in the efficient computation of the nucleolus of flow games [3].

Sun *et al.* [16] also showed that for uniform flow games the concepts of exactness, extendability, and core largeness are equivalent, and that they are equivalent to the property of the network that every (s, t) -cut contains a minimum (s, t) -cut. It is easy to see that this is equivalent to being cut-normal.

Lemma 1. *A 2-terminal DAG with terminals s and t is cut-normal if and only if every (s, t) -cut contains a minimum (s, t) -cut. \square*

In next section we show that the cut-normal uniform flow networks are exactly the balanced directed serial-parallel graphs with two terminals. This immediately implies a linear time algorithm to decide whether a uniform flow game is exact, extendable, and has a large core.

3 Efficient Algorithms

3.1 Core Stability

Let $G = (V, E)$ be a uniform flow network with source s and sink t . In this section we will give a linear time algorithm to decide core stability of the flow game on G .

Theorem 2. *G contains no dummy edge if and only if G is a balanced DAG.*

Proof. If: Suppose G is a balanced DAG. Let f be a maximum integer (s, t) -flow (which is also a maximum (s, t) -flow). Then, f pushes either zero or one unit of flow along each edge. Consider the subgraph G' of G of all edges $e \in E$ with $f(e) = 0$. Since G is balanced and f satisfies the flow conservation property in every vertex except s and t , G' is also balanced. As a subgraph of G it is also acyclical. Thus, if G' is not empty, there must be a simple (s, t) -path in G' . But then we can push one more unit of flow along this path, contradicting the maximality of f . Thus, G' is empty, i.e., G contains no dummy edge.

Only if: The flow conservation property and the fact that all edges have capacity one imply that for each maximum flow f at least one edge incident to an unbalanced vertex is not used by f . Thus, every unbalanced graph contains dummy edges.

Suppose G contains a directed cycle C . Since we may w.l.o.g. assume that a maximum (s, t) -flow f does not have flow flowing around in cycles, at least one edge of C will not be used by f , i.e., it is a dummy edge. □

Corollary 3. *We can decide core stability of uniform flow games in linear time.*

Proof. Sun *et al.* [16] have shown that a uniform flow game has a stable core if and only if the network does not contain dummy edges. □

3.2 Extendability, Exactness, and Core Largeness

In this section we will give a linear time algorithm to decide exactness, extendability, and core largeness of uniform flow games. Note that these properties imply core stability. In view of Theorem 2 we can therefore w.l.o.g. assume in this subsection that flow networks are balanced DAGs.

Two graphs are *homeomorphic* if they can be made isomorphic by inserting new vertices of degree two into edges, i.e., substituting directed edges by directed paths (which does not change the topology of the graph). Let H denote the graph shown on the left side of Fig. 1.

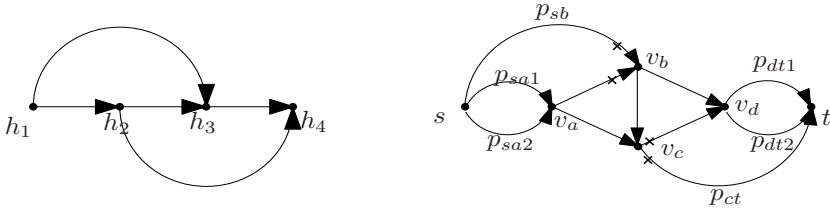


Fig. 1. Left: the graph H ; Right: the graph G_3

Theorem 4. [6,9,17] A 2-terminal DAG is a 2-DSPG if and only if it does not contain a subgraph homeomorphic to H . \square

We now give a characterization of balanced 2-DSPGs.

Theorem 5. A balanced 2-terminal DAG is cut-normal if and only if it is a balanced 2-DSPG.

Proof. If: It is easy to see that balanced 2-DSPGs can be generated inductively as 2-DSPGs with the additional constraint that a combination in series step (where we identify the two terminals t_1 and s_2) requires the indegree of t_1 being equal to the outdegree of s_2 .

We now prove the claim by induction on $|E|$. If $|E| = 1$, the graph is (s, t) -normal. Suppose the statement holds for all balanced 2-DSPGs with fewer than $|E|$ edges. If G was generated from $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ by combination in parallel, then any minimal (s, t) -cut C in G consists of a minimal (s_1, t_1) -cut $C \cap E_1$ in G_1 and a minimal (s_2, t_2) -cut $C \cap E_2$ in G_2 . By inductive hypothesis, these are minimum cuts. Thus, C is a minimum (s, t) -cut in G .

If G was generated from G_1 and G_2 by combination in series, then a minimal (s, t) -cut C in G is either a minimal (and thus minimum) (s_1, t_1) -cut in G_1 or a minimal (and thus minimum) (s_2, t_2) -cut in G_2 . Since the indegree of t_1 equals the outdegree of s_2 and cutting all edges incident to t_1 (s_2) defines a minimal (s_1, t_1) -cut in G_1 (minimal (s_2, t_2) -cut in G_2), a minimum (s_1, t_1) -cut in G_1 has the same capacity as a minimum (s_2, t_2) -cut in G_2 . Thus, C is a minimum (s, t) -cut in G .

Only if: Let $G = (V, E)$ be a balanced 2-terminal DAG with terminals s and t . Let $V = \{s = v_1, v_2, \dots, v_n = t\}$ be a topological ordering of G .

Let k denote the outdegree of s . If G is not a 2-DSPG, then we can construct a minimal cut of size larger than k , contradicting the assumption that G is cut-normal. By Theorem 4, G contains a subgraph homeomorphic to H which we denote by $H(a, b, c, d, P_{ab}, P_{bc}, P_{cd}, P_{ac}, P_{bd})$, where v_a is the node corresponding to h_1 , v_b the node corresponding to h_2 , v_c the node corresponding to h_3 , v_d the node corresponding to h_4 , and the vertex-disjoint (except at their endpoints) paths $P_{ab}, P_{bc}, P_{cd}, P_{ac}$ and P_{bd} correspond to the edges $(h_1, h_2), (h_2, h_3), (h_3, h_4), (h_1, h_3),$ and (h_2, h_4) in H , respectively. Among all subgraphs homeomorphic to H we choose one, G_H , with largest b .

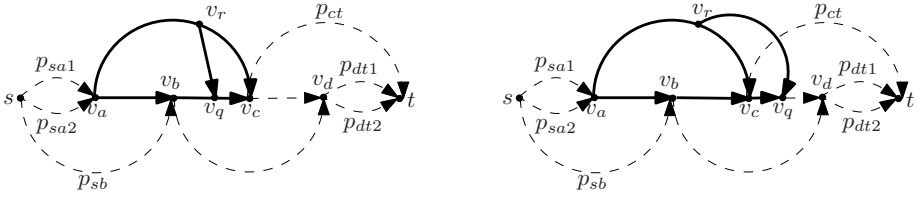


Fig. 2. Cases (1) and (2) in the proof of Theorem 5

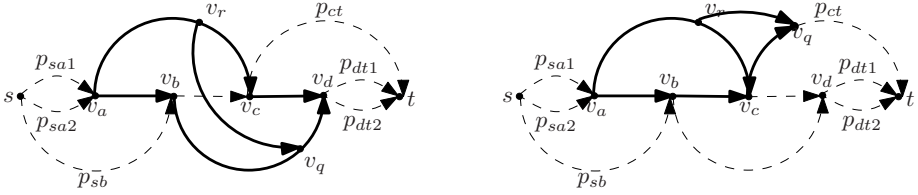


Fig. 3. Cases (3) and (4) in the proof of Theorem 5

G_H is not balanced, but since G is balanced we can find in G six pairwise edge-disjoint paths (they are also edge-disjoint with G_H) P_{sa1} , P_{sa2} , P_{sb} , P_{ct} , P_{dt1} , and P_{dt2} that we can add to G_H to obtain a balanced graph G_3 (see Fig. 1, right side). Note that G_3 is the union of three edge-disjoint (s, t) -paths $P_{sa1} + P_{ab} + P_{bd} + P_{dt1}$, $P_{sa2} + P_{ac} + P_{cd} + P_{dt2}$, and $P_{sb} + P_{bc} + P_{ct}$ (we use $P_1 + P_2$ to denote the concatenation of paths P_1 and P_2).

Consider the (s, t) -cut C_b in G induced by the partition of V into $\{v_1, \dots, v_{b-1}\}$ and $\{v_b, \dots, v_n\}$. Since G is a DAG and all vertices lie on some (s, t) -path, this is a minimal cut. If $|C_b| > k$, we have a minimal cut that is not a minimum cut. So assume $|C_b| = k$. We partition the edges of C_b into two classes, C_{b1} and C_{b2} . An edge e belongs to C_{b1} if every (s, t) -path containing e passes through v_c , otherwise it belongs to C_{b2} .

We will now show that C_{b1} is not empty. Specifically, we show it contains the edge $e = P_{ac} \cap C_b$. Assume there is an (s, t) -path P_e containing e but not v_c . Let v_r be the largest node in $P_{ac} \cap P_e$. Let v_q be the first node corresponding to a node in G_3 that we encounter on P_e when starting to walk at v_r (such a node exists because P_e ends at $t \in G_3$). Let P_{r_q} denote the path from v_r to v_q .

Since $e \in C_b$, $r \geq b$. Actually, $r > b$ because v_b is not on P_{ac} . And $r < c$ because $v_r \in P_{ac}$ and $v_c \notin P_e$. But then there exists another $H(a, r, v_c, \dots)$ in G , a contradiction because $r > b$. To see this, we distinguish five cases, depending on the location of v_q (see Fig. 2-4). Let $P_{xy}(i, j)$ denote the subpath of P_{xy} from v_i to v_j , where $x, y \in \{a, b, c, d\}$.

1. $v_q \in P_{bc}$: $H(a, r, q, c, P_{ac}(a, r), P_{r_q}, P_{bc}(q, c), P_{ab} + P_{bc}(b, q), P_{ac}(r, c))$.
2. $v_q \in P_{cd}$: $H(a, r, c, q, P_{ac}(a, r), P_{ac}(r, c), P_{cd}(c, q), P_{ab} + P_{bc}, P_{r_q})$.
3. $v_q \in P_{bd}$: $H(a, r, q, d, P_{ac}(a, r), P_{r_q}, P_{bd}(q, d), P_{ab} + P_{bd}(b, q), P_{ac}(r, c) + P_{cd})$.
4. $v_q \in P_{ct}$: $H(a, r, c, q, P_{ac}(a, r), P_{ac}(r, c), P_{ct}(c, q), P_{ab} + P_{bc}, P_{r_q})$.
5. $v_q \in P_{dt1}$ or $v_q \in P_{dt2}$: $H(a, r, d, q, P_{ac}(a, r), P_{ac}(r, c) + P_{cd}, P_{dq}, P_{ab} + P_{bd}, P_{r_q})$.

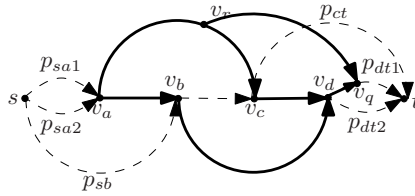


Fig. 4. Case (5) in the proof of Theorem 5

Next, we show that $|C_{b1}| < c_{out}$, where c_{out} denotes the outdegree of v_c in G . Let U be a maximal set of edge-disjoint (s, t) -paths containing v_c that includes the path $P_{sa1} + P_{ab} + P_{bc} + P_{cd} + P_{dt1}$. Note that $|U| \leq c_{out}$. Clearly, C_{b1} is a subset of $U \cap C_b$ by the definition of C_{b1} . Since the last edge of P_{ab} belongs to $U \cap C_b$ but not to C_{b1} , it is even a proper subset, proving the claim.

Let C be the set C_{b2} plus all edges outgoing from v_c . C is an (s, t) -cut because every (s, t) -path must either contain an edge in C_{b2} or the node v_c . C is a minimal (s, t) -cut because every outgoing edge of v_c is necessary because C_{b1} is not empty, and every edge in C_{b2} is necessary by definition. The size of C is $|C| = |C_{b2}| + c_{out} > |C_{b2}| + |C_{b1}| = |C_b| = k$. Thus, C is not a minimum (s, t) -cut, a contradiction. Therefore, the assumption that G is not a 2-DSPG must be wrong. \square

Corollary 6. *We can test in linear time whether a balanced uniform flow network is cut-normal. Consequently, we can decide exactness, extendability, and core largeness of uniform flow games in linear time.*

Proof. We can test in linear time whether a DAG is balanced and whether it is a 2-DSPG [17]. By Theorem 5 this is equivalent to being cut-normal. By Lemma 1, a uniform flow network is cut-normal if and only if every (s, t) -cut contains a minimum (s, t) -cut. Sun *et al.* [16] have shown that this is equivalent to the flow game being exact, extendable, and having a large core. \square

4 Open Problems

In this paper, we gave structural characterizations of exact, extendable, large-core and stable-core uniform flow games that can be tested in linear time. Currently, little is known about core stability of flow games on networks with arbitrary capacities. Although it is $co\mathcal{NP}$ -complete to decide whether an imputation belongs to the core this does not rule out the possibility that core stability can be decided efficiently. We leave it as an open problem.

Acknowledgements

We would like to thank Mordecai Golin for his helpful discussions.

References

1. Bietenhader, T., Okamoto, Y.: Core stability of minimum coloring games. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 389–401. Springer, Heidelberg (2004)
2. Biswas, A.K., Parthasarathy, T., Potters, J.A.M., Voorneveld, M.: Large cores and exactness. *Game and Economic Behavior* 28, 1–12 (1999)
3. Deng, X., Fang, Q., Sun, X.: Finding nucleolus of flow game. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06), pp. 124–131. ACM Press, New York (2006)
4. Deng, X., Ibaraki, T., Nagamochi, H.: Algorithmic aspects of the core of combinatorial optimization games. *Mathematics of Operations Research* 24, 751–766 (1999)
5. Deng, X., Papadimitriou, C.H.: On the complexity of cooperative solution concepts. *Mathematics of Operations Research* 19, 257–266 (1994)
6. Duffin, R.: Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications* 10, 303–318 (1965)
7. Fang, Q., Zhu, S., Cai, M., Deng, X.: Membership for core of LP games and other games. In: Wang, J. (ed.) COCOON 2001. LNCS, vol. 2108, pp. 247–256. Springer, Heidelberg (2001)
8. Jain, K., Vohra, R.V.: On stability of the core. Manuscript (2006), <http://www.kellogg.northwestern.edu/faculty/vohra/ftp/newcore.pdf>
9. Jakoby, A., Liśkiewicz, M., Reischuk, R.: Space efficient algorithms for directed series-parallel graphs. *Journal of Algorithms* 60(2), 85–114 (2006)
10. Kalai, E., Zemel, E.: Totally balanced games and games of flow. *Mathematics of Operations Research* 7, 476–478 (1982)
11. Kalai, E., Zemel, E.: Generalized network problems yielding totally balanced games. *Operations Research* 30, 998–1008 (1982)
12. Kikuta, K., Shapley, L.S.: Core stability in n -person games. Manuscript (1986)
13. Lucas, W.F.: A game with no solution. *Bulletin of the American Mathematical Society* 74, 237–239 (1968)
14. Sharkey, W.W.: Cooperative games with large cores. *International Journal of Game Theory* 11, 175–182 (1982)
15. Solymosi, T., Raghavan, T.E.S.: Assignment games with stable cores. *International Journal of Game Theory* 30, 177–185 (2001)
16. Sun, X., Fang, Q.: Core Stability of Flow Games. In: Akiyama, J., Chen, W.Y.C., Kano, M., Li, X., Yu, Q. (eds.) CJCDGCGT 2005. LNCS, vol. 4381, pp. 189–199. Springer, Heidelberg (2007)
17. Valdes, J., Tarjan, R.E., Lawler, E.L.: The recognition of series-parallel digraphs. In: Proceedings of the 11th Annual ACM Symposium on Theory of Computing (STOC'79), pp. 1–12. ACM Press, New York (1979)
18. van Gellekom, J.R.G., Potters, J.A.M., Reijnierse, J.H.: Prosperity properties of TU-games. *International Journal of Game Theory* 28, 211–227 (1999)
19. von Neumann, J., Morgenstern, O.: *Theory of Games and Economic Behaviour*. Princeton University Press, Princeton (1944)

Computing Symmetric Boolean Functions by Circuits with Few Exact Threshold Gates

Kristoffer Arnsfelt Hansen

Department of Computer Science
University of Aarhus*
arnsfelt@daimi.au.dk

Abstract. We consider constant depth circuits augmented with few exact threshold gates with arbitrary weights. We prove strong (up to exponential) size lower bounds for such circuits computing symmetric Boolean functions. Our lower bound is expressed in terms of a natural parameter, the *balance*, of symmetric functions. Furthermore, in the quasi-polynomial size setting our results provides an *exact* characterization of the class of symmetric functions in terms of their balance.

1 Introduction

A central program of circuit complexity is that aimed towards strong size lower bounds for larger and larger classes of constant depth circuits. Such lower bounds have been obtained for constant depth circuits consisting of AND and OR gates (\mathbf{AC}^0 circuits) [1,2,3,4]. If we however allow gates performing modular counting or a majority vote, we obtain classes of circuits (\mathbf{ACC}^0 and \mathbf{TC}^0) for which no strong size lower bounds are known. The present work belongs to two distinct lines of research both aimed at increasing the understanding of these classes thereby approaching such lower bounds.

The first of these lines of research considers \mathbf{AC}^0 circuits augmented with just few additional more powerful gates. A series of papers [5,6,7,8] considers circuits with few MAJ gates, resulting in the result that such circuits with $n^{o(1)}$ MAJ gates require size $2^{n^{\Omega(1)}}$ to compute the MOD_m function for any m . More recently have lower bounds been obtained for circuits with few modular counting gates and symmetric gates [9,10,11].

The second line of research concerns classifying which symmetric Boolean functions a class of circuits can compute. This naturally relies heavily on strong lower bounds for the class of circuits in question, but provides much more information than just a lower bound for a particular Boolean function. These results are usually stated in terms of properties of the *value vector* of a symmetric Boolean function, being composed of the outputs of the function for inputs of the $n+1$ different weights. Fagin et al. [12] prove that the symmetric functions computed by \mathbf{AC}^0 circuits of polynomial size are precisely those whose value vector

* Currently at The University of Chicago, supported by a Villum Kann Rasmussen Post.Doc. fellowship.

is constant except within both ends of poly-logarithmic length. Interestingly, the class remains the same when allowing quasi-polynomial size circuits. Zhang et al. [13] prove that the symmetric functions computed by quasi-polynomial size $\text{THR} \circ \text{AC}^0$ circuits (AC^0 circuits with an arbitrary threshold gate at the output) are precisely those whose value vector has only a poly-logarithmic number of *sign changes* (positions in the value vector where there is a change of values). Lu [14] prove that the symmetric functions computed by quasi-polynomial size AC^0 circuits augmented with MOD_q gates, where $q = p^k$ is a prime power, are precisely those whose value vector is *periodic* with poly-logarithmic period $p^{t(n)} = \log^{O(1)} n$, except within both ends of poly-logarithmic length.

Beigel [7] showed that a quasi-polynomial size AC^0 circuit augmented with a poly-logarithmic number of MAJ gates can be converted into a quasi-polynomial size $\text{MAJ} \circ \text{AC}^0$ circuit computing the same function, which implies that the characterization of Zhang et al. extends to this class of circuits.

A natural next step would be to consider circuits with few threshold gates, however here no strong lower bounds are known for computing simple functions such as symmetric Boolean functions. (For more complicated functions, some lower bounds can be obtained by the results of [9,10] combined with a multi-party protocol for evaluating threshold gates due to Nisan [15]). In this paper we instead consider circuits with few exact threshold gates computing symmetric Boolean functions. This class of circuits is simpler than the class of circuits with few threshold gates but seems to be incomparable to the class of circuits with few majority gates.

We state our lower bounds in terms of a notion of *balance* $b(f)$ of a symmetric Boolean function f , defined as the minimum number of 0's or 1's in the value vector of f . Our main result is the following general lower bound statement.

Theorem 1. *There is a constant $c > 0$ such that any depth h circuit containing s exact threshold gates and computing a symmetric function f must have size at least*

$$\frac{1}{n2^{s+1}} 2^{\frac{1}{14} \left(c \frac{b(f)}{s \log \left(\frac{n}{b(f)} \right)} \right)^{\frac{1}{h}}}$$

A notable special case to single out is quasi-polynomial size circuits. In this case we obtain a complete characterization of the symmetric functions computable.

Theorem 2. *A symmetric Boolean function can be computed by a constant depth AC^0 circuit of size $2^{\log^{O(1)} n}$ containing $\log^{O(1)} n$ exact threshold gates if and only if it has balance $\log^{O(1)} n$.*

2 Preliminaries

2.1 Constant Depth Circuits

We consider circuits built from families of unbounded fanin gates. Inputs are allowed to be Boolean variables and their negations as well as the constants 0 and 1. In addition to AND, OR and NOT gates we consider the following

variants of *threshold* functions. Let x_1, \dots, x_n be n Boolean inputs. The majority function, MAJ, is 1 if and only if $\sum_{i=1}^n x_i \geq \frac{n}{2}$. Similarly, the *exact* majority function, EMAJ, is 1 if and only if $\sum_{i=1}^n x_i = \frac{n}{2}$. Introducing weights we get a much richer class of functions. Let $w \in \mathbf{R}^n$ and let t be any real number. The threshold function with weights w and threshold t , $\text{THR}_{w,t}$ is 1 if and only if $\sum_{i=1}^n w_i x_i \geq t$. Similarly, the *exact* threshold function with weights w and threshold t , $\text{ETHR}_{w,t}$ is 1 if and only if $\sum_{i=1}^n w_i x_i = t$. For both these functions it is easy to see that we can in fact assume that the weights and threshold are integer valued, although this is not needed for our results.

Additionally, for a positive integer m , let MOD_m be the function that outputs 1 if and only if $\sum_{i=1}^n x_i \not\equiv 0 \pmod{m}$.

Let **AND** and **OR** denote the families of unbounded fanin AND and OR gates. Let **MAJ**, **ETHR**, **THR** denote the families of MAJ, $\text{ETHR}_{w,t}$ and $\text{THR}_{w,t}$ gates, for arbitrary w and t . If G is a family of boolean gates and \mathcal{C} is a family of circuits we let $G \circ \mathcal{C}$ denote the class of circuits consisting of a G gate taking circuits from \mathcal{C} as inputs.

\mathbf{AC}^0 is the class of functions computed by constant depth circuits built from AND, OR and NOT gates. We define the larger class \mathbf{TC}^0 as the class of functions computed by constant depth circuits built from AND, OR and MAJ gates. When there is no special restrictions on the number of threshold gates, we can exchange MAJ gates in this definition with either of the other three variants of threshold gates defined above without changing the class of functions computed.

2.2 The Switching Lemma

A *restriction* on a set V of boolean variables is a map $\rho : V \rightarrow \{0, 1, \star\}$. It acts on a boolean function $f : V \rightarrow \{0, 1\}$, creating a new boolean function f_ρ on the set of variables for which $\rho(x) = \star$, obtained by substituting $\rho(x)$ for $x \in V$ whenever $\rho(x) \neq \star$. Let R_n^l denote the set of all restriction ρ leaving l of n variables free.

A decision tree is a binary tree, where the internal nodes are labeled by variables and leafs are labeled by either 0 or 1. On a given input x , its value is the value of the leaf reached by starting at the root, and at any internal node labeled by x_i proceeding to the left child if $x_i = 0$ and to the right child otherwise. We will use the following version of Håstad's Switching Lemma due to Beame [16].

Lemma 1. *Let f be a DNF formula in n variables with terms of length at most r . Let $l = pn$ and pick ρ uniformly at random from R_n^l . Then the probability that f_ρ does not have a decision tree of depth d is less than $(7pr)^d$.*

The advantage of using this version of the switching lemma is that it directly gives us a decision tree. If we convert a decision tree into a DNF, we in fact obtain a *disjoint* DNF, i.e. a DNF where all terms are *mutually contradictory*. We can view it as a sum of terms, instead as an OR of AND's. This view allows us to absorb the sum into a threshold gate. We elaborate on this after the next proposition.

Using the switching lemma several times, allows one to obtain decision trees for every gate in a given \mathbf{AC}^0 circuit.

Proposition 1. *Let C be an \mathbf{AC}^0 circuit of depth h and size S . Let d be any positive integer, and choose a restriction $\rho \in R_n^{n_h}$, where $n_h = \frac{n}{14(14d)^{h-1}}$. Then with probability greater than $1 - S2^{-d}$, is every function computed by any gate of C computed by a decision tree of depth less than d , after applying the restriction ρ .*

Proof. First, whenever an input to a NOT gate is computed by a decision tree of depth less than d , the function computed by the NOT gate is also computed by a decision tree of depth less than d , by simply negating the constants at the leaves of the decision tree. Henceforth we will only consider OR and AND gates.

Define, for $0 \leq i \leq h - 1$, $n_{i+1} = \frac{n}{14(14d)^i}$. We choose the restriction $\rho \in R_n^{n_h}$ at random as a composition of randomly chosen restrictions ρ_1, \dots, ρ_h , where $\rho_1 \in R_n^{n_1}$ and $\rho_{i+1} \in R_{n_i}^{n_{i+1}}$ for $1 \leq i \leq h - 1$. Consider an AND or OR gate at level 1 of C . We can consider the inputs of the gate to be terms or clauses of size 1. This means that we can consider the gate to be a CNF with clauses of size 1 or a DNF with terms of size 1. Then by Lemma [□](#) after applying ρ_1 , the probability that the function computed by the gate is not computed by a decision tree of depth less than d is at most $(7\frac{n_1}{n})^d = 2^{-d}$.

Now assume that after having applied the restrictions ρ_1, \dots, ρ_i , every function computed by a gate at level i is also computed by a decision tree of depth at most d . Consider now a gate at level $i + 1$. If the gate is an OR gate we rewrite every input to the gate as DNF's with terms of size at most d . The OR gates of these can then be merged with the OR gate at level $i + 1$, thus obtaining a DNF with terms of size at most d computing the same function. Similarly, if the gate is an AND gate we rewrite every input to the gate as CNF's with clauses of size at most t . The AND gates of these can then be merged with the AND gate at level $i + 1$, thus obtaining a CNF with clauses of size at most d for the same function. Then by using Lemma [□](#) again, the probability that the function computed by the gate at level $i + 1$ is not computed by a decision tree of depth less than d is less than $(7\frac{n_{i+1}}{n_i}d)^d = 2^{-d}$.

To conclude, since the probability of error at a single gate is less than 2^{-d} , the total probability of error is less than $S2^{-d}$ and the result follows.

Consider now an $\mathbf{ETHR} \circ \mathbf{AC}^0$ circuit C . After successful use of Proposition [□](#) we have a restriction ρ such that after applying ρ , every function computed at an input to the output gate of C is computed by a decision tree of depth at most d , and hence by disjoint DNF's with terms of size at most d . Assume that the output gate evaluates whether $\sum_{i=1}^m w_i x_i = t$, and let t_j^i be the terms of the i 'th DNF. Then we have that the output of C_ρ evaluates whether $\sum_{i=1}^m w_i (\sum_j (\prod_{x_k \in t_j^i} x_k \cdot \prod_{\neg x_k \in t_j^i} (1 - x_k))) = t$. This means we have a real polynomial $P(x)$ of degree at most d such $P(x) \neq 0$ if and only if $C_\rho(x) = 1$ for all x .

2.3 Representation by Polynomials

Let P be a real polynomial on n variables and f a Boolean function on n variables. We say that P is a *weak equality* representation f , if P is not identically 0 and whenever $P(x) \neq 0$ for $x \in \{0, 1\}^n$ we must have $f(x) = 1$. (Thus our

definition is a real valued analogue of the notion of weak representation over \mathbf{Z}_m as defined by Green [17].

Let f be a symmetric Boolean function on n variables. Since f then only depends on the *weight* of its input, i.e. $\sum_{i=1}^n x_i$, we will identify f with a function $f : \{0, \dots, n\} \rightarrow \{0, 1\}$. By $v(f)$ we denote the string $f(0)f(1) \cdots f(n) \in \{0, 1\}^{n+1}$, called the *value vector* of f .

For a string $s \in \{0, 1\}^*$, let $n_0(s)$ denote the number of 0's in a , and likewise let $n_1(s)$ denote the number of 1's in s . Let further $n_0(f) = n_0(v(f))$ and $n_1(f) = n_1(v(f))$. Define the *balance* $b(f)$ of f by $b(f) = \min(n_0(f), n_1(f))$.

Using the symmetrization technique of Minsky and Papert [18] we can prove lower bounds on the degree of a polynomial representing a symmetric Boolean function, in terms of the number of 0's of its value vector.

Proposition 2. *Let P be a polynomial, not identically 0, that is a weak equality representation of a symmetric Boolean function f on n variables. Then the degree of P must be at least $\frac{n_0(f)}{2}$.*

Proof. Let d be the degree of P . Define

$$Q(x) = \sum_{\sigma \in S_n} (P(x_{\sigma(1)}, \dots, x_{\sigma(n)}))^2 .$$

Observe that Q is a symmetric polynomial of degree $2d$ that is also a weak equality representation of f . We thus have a polynomial H of degree $2d$ such that $H(\sum_{i=1}^n x_i) = Q(x)$ for all $x \in \{0, 1\}^n$. This polynomial H is not identically 0, and if $H(w) \neq 0$ we must have $f(w) = 1$. Since H must have at least $n_0(f)$ roots it then follows that $2d \geq n_0(f)$.

For our purposes we will for a given function f need lower bounds for polynomials that represents *either* f or its negation $\neg f$. We get precisely such a lower bound from Proposition 2 in terms of the balance of f .

Corollary 1. *Let f be any symmetric Boolean function. If P is a polynomial that is either a weak equality representation of f or $\neg f$, the degree of P must be at least $\frac{b(f)}{2}$.*

Proof. For representing f , Proposition 2 implies that the degree of P must be at least $\frac{n_0(f)}{2}$, and similarly for representing $\neg f$ the degree of P must be at least $\frac{n_0(\neg f)}{2}$. Since $n_1(f) = n_0(\neg f)$ the result follows.

3 Circuit Lower Bounds

By use of the switching lemma we will be able to obtain lower bounds of the size of circuits with few exact threshold gates computing a Boolean function f depending on the degree required for a polynomial to represent either f_ρ or $\neg f_\rho$, where ρ is a random restriction. If f_ρ is a symmetric function we can use Corollary 1 to express the lower bound in terms of the balance of f_ρ . To express

the lower bound solely in terms of a symmetric f , we will carefully choose a restriction f' of f , for which we can relate the balance of f'_ρ to that of f , for a suitable random restriction ρ .

By an *interval* of the value vector of a symmetric Boolean function f we mean a contiguous substring. We will denote such an interval by $[a, b]$, where a and b are the endpoints of the interval. When applying a restriction ρ to a symmetric Boolean function f , the function f_ρ obtained is also a symmetric Boolean function, and its value vector is an interval of the value vector of f . Conversely, every interval of the value vector of f is the value vector of a restriction of f . Let $f_I = f_{[a,b]}$ be the Boolean function whose value vector is the interval $I = [a, b]$ of the value vector of f .

With these definitions in place, we can outline the properties we desire of f' . We would like to have $f' = f_I$ for an interval $I = [a, b]$, such that a smaller interval I' in the middle of I has as large a balance as possible. This will mean that we can ensure that $f_{I'}$ is a further restriction of f'_ρ and thus will the balance of f'_ρ be at least as large as the balance of $f_{I'}$.

This is precisely the approach that Zhang et al. [13] took using *sign changes* instead of balance to prove their lower bound. However, while our approach here is similar, we have to be more careful. The reason is as follows. If we divide an interval containing a certain number of sign changes into two parts then one of the parts must contain at least half of the sign changes. For balance, the same thing clearly does not hold. As a simple example consider an interval where the first half is all 0 and the other half is all 1. Although this interval has maximal balance, each of the two halves have balance 0. What is true instead, is that there is *some* subinterval of half the length with half the balance. We will establish that we can find the function f' we desire in the following series of lemmas.

Lemma 2. *Let $s \in \{0, 1\}^*$. Suppose that $n_0(s[a, a + l]) \geq n_1(s[a, a + l])$ and $n_1(s[c, c + l]) \geq n_0(s[c, c + l])$ for $a \leq c$. Then there exists b such that $a \leq b \leq c$ and $n_0(s[b, b + l]) \leq n_1(s[b, b + l]) \leq n_0(s[b, b + l]) + 1$.*

Proof. Define $f : \{a, \dots, c\} \rightarrow \mathbf{Z}$ by $f(d) = n_1(s[d, d+l]) - n_0(s[d, d+l])$. Observe that $f(b + 1) - f(b) \in \{-2, 0, 2\}$. Since by assumption $f(a) \leq 0$ and $f(c) \geq 0$ there exists $a \leq b \leq c$ such that $0 \leq f(b) \leq 1$, which implies $n_0(s[b, b + l]) \leq n_1(s[b, b + l]) \leq n_0(s[b, b + l]) + 1$.

Lemma 3. *Let f be a symmetric Boolean function on n variables. Let $n' = 2^k - 1$, where $2^k < n + 1 \leq 2^{k+1}$. Then there is a restriction f' of f on n' variables such that $b(f') \geq \frac{b(f)-1}{2}$.*

Proof. Assume without loss of generality that $n_0(f) \leq n_1(f)$. We will then find an interval I of length at most 2^k of the value vector of f such that $n_1(f_I) \geq n_0(f_I) \geq \frac{n_0(f)-1}{2}$. This interval can then be extended to an interval I' of length 2^k such that $\min(n_0(f_{I'}), n_1(f_{I'})) \geq n_0(f_I)$. Thus for $f' = f_{I'}$ on n' variables we have $b(f') \geq \frac{b(f)-1}{2}$.

Consider the two intervals $I_1 = [0, \lfloor \frac{n+1}{2} \rfloor - 1]$ and $I_2 = [\lfloor \frac{n+1}{2} \rfloor, 2\lfloor \frac{n+1}{2} \rfloor - 1]$. Note that each of these are of length $\lfloor \frac{n+1}{2} \rfloor \leq 2^k$. Since at most one index is

left out, we must have either $n_0(f_{I_1}) \geq \frac{n_0(f)-1}{2}$ or $n_0(f_{I_2}) \geq \frac{n_0(f)-1}{2}$. Assume without loss of generality that $n_0(f_{I_1}) \geq \frac{n_0(f)-1}{2}$. If also $n_1(f_{I_1}) \geq n_0(f_{I_1})$ we may choose $I = I_1$. Thus assume $n_0(f_{I_1}) > n_1(f_{I_1})$. Now we must have $n_1(f_{I_2}) \geq n_0(f_{I_2})$, as otherwise we would have $n_0(f) \geq n_0(f_{I_1}) + n_0(f_{I_2}) \geq n_1(f_{I_1}) + n_1(f_{I_2}) + 2 \geq n_1(f) + 1$. Then using Lemma 2 we can find an interval I_3 of length $\lfloor \frac{n+1}{2} \rfloor \leq 2^k$ such that $n_0(f_{I_3}) \leq n_1(f_{I_3}) \leq n_0(f_{I_3}) + 1$. We then have $n_0(f) \leq \lfloor \frac{n+1}{2} \rfloor = n_0(f_{I_3}) + n_1(f_{I_3}) \leq 2n_0(f_{I_3}) + 1$, and we may choose $I = I_3$.

Lemma 4. *Let $s \in \{0, 1\}^{2^k}$. If $n_0(s) \geq n_1(s)$ then there is a substring s' of s of length 2^{k-1} such that $n_0(s') \geq n_1(s')$. Analogously if $n_1(s) \geq n_0(s)$ then there is a substring s' of s of length 2^{k-1} such that $n_1(s') \geq n_0(s')$.*

Proof. We prove the case $n_0(s) \geq n_1(s)$. We then have $n_0(s[0, 2^{k-1} - 1]) + n_0(s[2^{k-1}, 2^k - 1]) \geq n_1(s[0, 2^{k-1} - 1]) + n_1(s[2^{k-1}, 2^k - 1])$ and thus we must have $n_0(s[0, 2^{k-1} - 1]) \geq n_1(s[0, 2^{k-1} - 1])$ or $n_0(s[2^{k-1}, 2^k - 1]) \geq n_1(s[2^{k-1}, 2^k - 1])$. We can then let $s' = s[0, 2^{k-1} - 1]$ or $s' = s[2^{k-1}, 2^k - 1]$.

Lemma 5. *Let f be a symmetric Boolean function on n variables, where $n = 2^k - 1$. Let $2^h \leq b(f) < 2^{h+1}$, and assume that $h > 2$. Then there exist an interval I of length $2^{k'}$, where $k' \geq h - 2$, such that I is a subinterval of the interval $[2^{k'}, n - 2^{k'}]$ and $b(f_I) \geq \frac{b(f)}{4 \log_2(\frac{8(n+1)}{b(f)})}$.*

Proof. Let $b = 2^{h-2}$, and assume without loss of generality that $n_0(f_{[b, n-b]}) \leq n_1(f_{[b, n-b]})$. Consider the intervals $I_i^L = [b2^i, b2^{i+1} - 1]$ and $I_i^R = [n - b2^{i+1} + 1, n - b2^i]$ for $i = 0, \dots, k - h$. Note that these $2(k - h + 1)$ intervals precisely cover the interval $[b, n - b]$.

Suppose first that $n_1(I_i^L) \geq n_0(I_i^L)$ and $n_1(I_i^R) \geq n_0(I_i^R)$ for all i . Then we may select I among the I_i^L and I_i^R intervals obtaining $n_1(f_I) \geq n_0(f_I) \geq \frac{n_0(f_{[b, n-b]})}{2(k-h+1)} \geq \frac{b(f)-2b}{2(k-h+1)} \geq \frac{b(f)}{4 \log_2(\frac{4(n+1)}{b(f)})}$.

Otherwise we must have two intervals I_1 and I_2 chosen from the I_i^L and I_i^R intervals, such that $n_0(f_{I_1}) \geq n_1(f_{I_1})$ and $n_1(f_{I_2}) \leq n_0(f_{I_2})$, (By assumption $n_0(f_{[b, n-b]}) \leq n_1(f_{[b, n-b]})$, so we can not have that $n_1(I_i^L) < n_0(I_i^L)$ and $n_1(I_i^R) < n_0(I_i^R)$ for all i). Suppose I_1 is of length 2^{k_1} and I_2 is of length 2^{k_2} , and let $k = \min(k_1, k_2)$. Note then that I_1 and I_2 are subintervals of the interval $[2^k, n - 2^k]$.

Using Lemma 4 we may find subintervals I'_1 of I_1 and I'_2 of I_2 of length k , that are also subintervals of the interval $[2^k, n - 2^k]$, such that $n_0(f_{I'_1}) \geq n_1(f_{I'_1})$ and $n_1(f_{I'_2}) \leq n_0(f_{I'_2})$.

Then using Lemma 2 we can find an interval I of length $2^k \geq b$, which is a subinterval of the interval $[2^k, n - 2^k]$, such that $n_0(f_I) \leq n_1(f_I) \leq n_0(f_{I'_1}) + 1$, which means in fact, $n_0(f_I) = n_1(f_I)$. Thus we have $b(f_I) \geq \frac{b}{2} > \frac{b(f)}{16} =$

$$\frac{b(f)}{4 \log_2\left(\frac{8(n+1)}{\frac{n+1}{2}}\right)} \geq \frac{b(f)}{4 \log_2\left(\frac{8(n+1)}{b(f)}\right)}.$$

Corollary 2. *Let f be a symmetric Boolean function on n variables, where $n = 2^k - 1$. Assume that $b(f) \geq 8$ and let m be such that $2^m \leq \frac{b(f)}{8}$. Then there exist an interval I' of length $2^{k'}$, for some $k' > m$, such that the interval I'' of length $2^{k'-m}$ in the middle of I' satisfies $b(f_{I''}) \geq \frac{b(f)}{2^{m+2} \log_2\left(\frac{8(n+1)}{b(f)}\right)}$.*

Proof. Let I be the interval given by Lemma 5 and divide it into interval I_1, \dots, I_{2^m} , each of length $2^{k'-m}$. We will find a subinterval I'' of I such that $b(f_{I''}) \geq \frac{b(f_I)}{2^m}$. Then since I is a subinterval of $[2^{k'}, n - 2^{k'}]$, we can let I' be the interval of length $2^{k'}$ with I'' in the middle. Assume without loss of generality that $n_0(f_I) \leq n_1(f_I)$. If $n_0(f_{I_i}) \leq n_1(f_{I_i})$ for all i , then for some j we have $n_1(f_{I_j}) \geq n_0(f_{I_j}) \geq \frac{n_0(f_I)}{2^m}$, and we can let $I'' = I_j$. Otherwise, we have $n_0(f_{I_i}) \leq n_1(f_{I_i})$ and $n_1(f_{I_j}) > n_0(f_{I_j})$, for some i and j . By Lemma 2 we then have a subinterval I'' of I such that $n_0(f_{I''}) = n_1(f_{I''})$, and thus $b(f_{I''}) \geq \frac{b(f_I)}{2^m}$.

By combining Lemma 3 and Corollary 2 we finally obtain the following proposition we will use in the proof of our main lower bound.

Proposition 3. *Let f be a symmetric Boolean function on n variables. Assume that m is an integer such that $2^m \leq \frac{b(f)}{16}$. Then there exist an interval I of length 2^k for some $k > m$ such that the interval I' of length 2^m in the middle of I satisfies $b(f_{I'}) \geq \frac{b(f)-1}{2^{m+3} \log_2\left(\frac{16n}{b(f)-1}\right)}$.*

With this proposition in place we can give the proof of Theorem 1.

Proof. (Theorem 1) Assume that C is a depth h circuit containing s exact threshold gates g_1, \dots, g_s computing f . Assume there is no path from the output of g_j to an input of g_i if $i < j$. Define $b = \frac{b(f)-1}{8 \log_2\left(\frac{16n}{b(f)-1}\right)}$ and $d = \frac{1}{14} \left(\frac{b}{4(s+1)}\right)^{\frac{1}{h}}$. Assume the size of C is $S = 2^{d-(s+1)-\log_2 n}$. For $\alpha \in \{0, 1\}^s$ let C_i^α be the **ETHR** \circ **AC**⁰ subcircuit of C with g_i as output, where every g_j for $j < i$ is replaced by the constant α_j . Similarly let C^α be the **AC**⁰ circuit obtained from C when every g_i is replaced by α_i .

Now let m be an integer such that $2^m \geq 14(14t)^{h-1} > 2^{m-1}$. Using Proposition 3 we have a restriction of f to a Boolean function f' on $n' = 2^k - 1$ variables such the subinterval I' of length 2^{k-m} in the middle of the value vector of f' satisfies $b(f_{I'}) \geq \frac{b}{2^m}$.

Pick a random restriction $\rho \in R_{n'}^{n_h}$, where $n_h = \frac{n'}{14(14d)^{h-1}}$ and apply it simultaneously to the circuits C^α and the circuits obtained from C_i^α by removing the output gate. Using Proposition 1 we then have that after applying ρ , except with probability at most $2^{s+1}S2^{-d} = \frac{1}{n}$ is every function computed by any gate of these circuits computed by a decision tree of depth at most d . By the discussion following Proposition 1 we obtain real polynomials P_i^α and Q^α of degree less than d , such that for all x , $C_{i,\rho}(x) = 1$ if and only if $P_i^\alpha(x) = 0$ and $C_\rho^\alpha(x) = Q^\alpha(x)$. Pick a maximal set G of the **ETHR** gates that are 0 at the same time for some input x to C_ρ , and define α such that $\alpha_i = 1$ if and only if $g_i \in G$.

The probability that the number of variables assigned 0 and 1 by ρ differ by at most 1 is at least $\frac{1}{n'-n_d} \binom{n'-n_d}{\frac{n'-n_d}{2}}$. Using Stirling's approximation this is $\Omega\left((n'-n_d)^{-\frac{1}{2}}\right) = \Omega(n^{-\frac{1}{2}})$. Thus for sufficiently large n we can assume that ρ gives the polynomials as above and the number of variables assigned 0 and 1 by ρ differ by at most 1. We then have that I' is a subinterval of the interval defined by ρ .

Now, if there exists x such that all gates in G evaluate to 0 and at the same time $C_\rho(x) = 1$, then the polynomial $Q^\alpha(x) \prod_{g_i \in G} P_i^\alpha(x)$ is a weak equality representation of f'_ρ . Otherwise the polynomial $\prod_{g_i \in G} P_i^\alpha(x)$ is a weak equality representation of $\neg f'_\rho$. The correctness of this claim follows from the maximality of the set G .

These polynomials are of degree less than $(s + 1)t$ and since $b(f'_\rho) \geq \frac{b}{2^m}$, we must have $(s + 1)t \geq \frac{b}{2^{m+1}}$ using Corollary [11](#).

However we have

$$\begin{aligned} t(s + 1) &= \frac{1}{14} \left(\frac{b}{4(s + 1)} \right)^{\frac{1}{h}} (s + 1) \\ &= \frac{b}{4 \cdot 14 \left(\frac{b}{4(s+1)} \right)^{\frac{h-1}{h}}} = \frac{b}{4 \cdot 14(14d)^{h-1}} < \frac{b}{2^{m+1}} \end{aligned}$$

thus contracting the existence of C .

For *concrete* symmetric Boolean function slightly better lower bounds can be obtained by avoiding the use of Proposition [3](#). For example for the MAJ function the subinterval in the middle satisfies the requirements of the proof. For the MOD₂ function things are even better, since *any* subinterval satisfies the requirements. We just state the bounds since the proof is similar to the proof of Theorem [11](#).

Theorem 3. *Let C be a depth h \mathbf{AC}^0 circuit containing s exact threshold gates. If C computes the MAJ function or the MOD₂ function the size of C must be at least*

$$\frac{1}{n2^{s+1}} 2^{\frac{1}{14} \left(\frac{n}{4(s+1)} \right)^{\frac{1}{h}}} \quad \text{and} \quad \frac{1}{2^{s+1}} 2^{\frac{1}{14} \left(\frac{n}{4(s+1)} \right)^{\frac{1}{h}}}$$

respectively.

Turning to the proof of Theorem [2](#), one half of theorem follows readily from Theorem [11](#). The other half follows from the following proposition.

Proposition 4. *Any symmetric function f can be computed by a depth 3 \mathbf{AC}^0 circuit of size $b(f) + 2$ augmented with $b(f)$ exact threshold gates.*

Proof. If $b(f) = n_1(f)$ we can compute f by a $\mathbf{OR} \circ \mathbf{ETHR}$ circuit where there is an exact threshold gate corresponding to every 1 in the value vector of f . If $b(f) = n_0(f)$ we construct the circuit as before for $\neg f$ and then add a negation gate at the output.

4 Conclusion

We have obtained strong lower bounds for circuits with few exact threshold gates with arbitrary weights computing symmetric Boolean functions. With our results we have essentially reached the best we could hope for with our current techniques: the approach taken in this paper does not provide the possibility to prove lower bounds with a superlinear amount exact threshold gates. Furthermore we would obviously also need to consider non-symmetric functions since every symmetric function can be computed with a linear number of exact threshold gates.

However, we find that further exploration of the power of exact threshold in constant depth circuits could prove to be very fruitful. In general, while circuits with threshold have been extensively studied, especially depth 2 and 3 circuits, very little research have considered circuits with (weighted) exact threshold gates, even though such a study is likely to provide insight into circuits with threshold gates. Many lower bounds are known for various classes of circuits with a (weighted or unweighted) threshold gate at the output. Interestingly, most of these lower bounds holds equally well with an exact threshold gate at the output, since the lower bounds actually holds with a Boolean function at the output that is determined by sign of a very small degree (e.g constant) polynomial. This allows one to simulate an exact threshold gate, since $p(x) = 0$ if and only if $(p(x))^2 \leq 0$.

For a concrete question, proving lower bounds for depth 2 threshold circuits is well known as being a notoriously difficult problem. Is the same true for depth 2 exact threshold circuits?

References

1. Furst, M., Saxe, J.B., Sipser, M.: Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory* 17(1), 13–27 (1984)
2. Ajtai, M.: Σ_1^1 -formulae on finite structures. *Annals of Pure and Applied Logic* 24, 1–48 (1983)
3. Håstad, J.: *Computational limitations of small-depth circuits*. MIT Press, Cambridge (1987)
4. Yao, A.C.C.: Separating the polynomial-time hierarchy by oracles. In: *Proceedings 26th Annual Symposium on Foundations of Computer Science*, pp. 1–10. IEEE Computer Society Press, Los Alamitos (1985)
5. Aspnes, J., Beigel, R., Furst, M.L., Rudich, S.: The expressive power of voting polynomials. *Combinatorica* 14(2), 135–148 (1994)
6. Beigel, R., Reingold, N., Spielman, D.A.: PP is closed under intersection. *Journal of Computer and System Sciences* 50(2), 191–202 (1995)
7. Beigel, R.: When do extra majority gates help? Polylog(n) majority gates are equivalent to one. *Computational Complexity* 4(4), 314–324 (1994)
8. Barrington, D.A.M., Straubing, H.: Complex polynomials and circuit lower bounds for modular counting. *Computational Complexity* 4(4), 325–338 (1994)

9. Chattopadhyay, A., Hansen, K.A.: Lower bounds for circuits with few modular and symmetric gates. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 994–1005. Springer, Heidelberg (2005)
10. Viola, E.: Pseudorandom bits for constant depth circuits with few arbitrary symmetric gates. In: Proceedings of the 20th Annual IEEE Conference on Computational Complexity, pp. 198–209. IEEE Computer Society Press, Los Alamitos (2005)
11. Hansen, K.A.: Lower bounds for circuits with few modular gates using exponential sums. Technical Report 79, Electronic Colloquium on Computational Complexity (2006)
12. Fagin, R., Klawe, M.M., Pippenger, N.J., Stockmeyer, L.: Bounded-depth, polynomial-size circuits for symmetric functions. *Theoretical Computer Science* 36(2–3), 239–250 (1985)
13. Zhang, Z.L., Barrington, D.A.M., Tarui, J.: Computing symmetric functions with AND/OR circuits and a single MAJORITY gate. In: Enjalbert, P., Wagner, K.W., Finkel, A. (eds.) STACS 93. LNCS, vol. 665, pp. 535–544. Springer, Heidelberg (1993)
14. Lu, C.J.: An exact characterization of symmetric functions in $qAC^0[2]$. *Theoretical Computer Science* 261(2), 297–303 (2001)
15. Nisan, N.: The communication complexity of threshold gates. In: Miklós, D., Szönyi, T., S., V.T. (eds.) *Combinatorics, Paul Erdős is Eighty*. Bolyai Society. *Mathematical Studies* 1, vol. 1, pp. 301–315 (1993)
16. Beame, P.: A switching lemma primer. Technical Report UW-CSE-95-07-01, Department of Computer Science and Engineering, University of Washington (1994), Available online at www.cs.washington.edu/homes/beame
17. Green, F.: A complex-number fourier technique for lower bounds on the mod- m degree. *Computational Complexity* 9(1), 16–38 (2000)
18. Minsky, M., Papert, S.: *Perceptrons - An Introduction to Computational Geometry*. MIT Press, Cambridge (1969)

On the Complexity of Finding an Unknown Cut Via Vertex Queries

Peyman Afshani, Ehsan Chiniforooshan, Reza Dorrigiv, Arash Farzan,
Mehdi Mirzazadeh, Narges Simjour, and Hamid Zarrabi-Zadeh

School of Computer Science, University of Waterloo
Waterloo, Ontario, N2L 3G1, Canada
{pafshani,echinifo,rdorrigiv,afarzan,
mmirzaza,nsimjour,hzarrabi}@cs.uwaterloo.ca

Abstract. We investigate the problem of finding an unknown cut through querying vertices of a graph G . Our complexity measure is the number of submitted queries. To avoid some worst cases, we make a few assumptions which allow us to obtain an algorithm with the worst case query complexity of $O(k) + 2k \log \frac{n}{k}$ in which k is the number of vertices adjacent to cut-edges. We also provide a matching lowerbound and then prove if G is a tree our algorithm can asymptotically achieve the information theoretic lowerbound on the query complexity. Finally, we show it is possible to remove our extra assumptions but achieve an approximate solution.

1 Introduction

Consider a graph G together with a partition of its set of vertices, $V(G)$, into two sets, A and B . Here, we study the problem of finding the sets A and B by only asking queries about the vertices of G . In other words, the algorithm has only access to the graph G and an oracle which given a vertex v will tell the algorithm whether $v \in A$ or $v \in B$. Although we study this problem from a theoretical point of view, we can establish connections to the existing concepts and problems studied in machine learning.

In the standard learning problems, the learner is given a collection of labeled data items, which is called the *training data*. The learner is required to find a “hypothesis”, using the training data and thus predict the labels of all (or most of) the data items, even those not seen by the learner algorithm. In this context, labeling the data points is considered to be an expensive operation. Thus, reducing the size of the training data is one of the important objectives. Semi-supervised learning attempts to accomplish this by using additional information about the whole data set. Recently, new models for semi-supervised learning have emerged which use spectral or graph techniques. We can name the work of Blum et al. [1], in which they built a graph and proposed several strategies to weigh the edges of the graph and proved finding a minimum cut corresponds to several of the previously employed learning algorithms based on Random Markov Fields. This is supported by the fact that it is common to restrict the set of possible labels

to $\{+, -\}$ [7] which implies a cut in graph G . We need to mention that spectral clustering techniques (for instance [9,8]) which are closely related to cuts have also been used in context of learning (for instance see [3,5]). We refer the reader to a line of papers in this area [11,10,4,2].

Other concepts related to the problem studied here are *Query learning* and *active learning*. Basically, under these assumptions the learner algorithm is allowed to interactively ask for the label of any data item. Thus, we can claim the problem studied in this paper has strong connection to the existing topics in machine learning; in short, our problem can be described as actively learning an unknown cut in a graph G .

In the next section, we define the problem precisely and obtain a simple lower bound on the number of queries. Then, in Section 3 we develop an algorithm that can solve a stronger version of our problem. We prove that the number of queries needed by our algorithm matches the lower bound. We discuss the problem for trees as a special family of graphs in Section 4. Finally, we relax the balancedness assumption and develop an ϵ -approximation algorithm instead of an exact algorithm in Section 5.

2 Preliminaries

Given a graph G , here we choose to represent the cut using a *labeling* $l : V(G) \rightarrow \{+, -\}$ which is the assignment of $+$ or $-$ to the vertices of G . Our goal is to design an algorithm which through querying labels of vertices can detect *all* the cut-edges. Clearly, the challenge is to minimize the number of queries or otherwise n queries can trivially solve the problem. Thus, we measure the complexity of the problem by the number of submitted queries and the parameters involved are the number of vertices, n , number of edges in the cut, k , and number of vertices adjacent to cut-edges, k' .

Notice that if graph G is a k -regular graph and all the vertices except one vertex v are labeled $+$, then it is easy to see that the algorithm must perform n queries in the worst case to find the single vertex v . This (the unbalanced cut having undesirable properties) is a common phenomenon which also appears in the spectral and clustering techniques. For instance, the definitions of normalized cut and ratio cut both factor in a form of balancedness condition. Here, we require a different form of balancedness: suppose we remove all the cut-edges in the graph. We call a labeling of a graph α -*balanced*, if each connected component in the new graph has at least αn vertices. Now we formally define our first problem:

Definition 1 (Problem A). *Suppose a graph G with an unknown α -balanced labeling of cut-size k is given. Use the structure of G together with the value of α to find this labeling using a small number of queries.*

Now, we show how to reduce the above problem to a seemingly easier problem by a probabilistic reduction. For a cut \mathcal{C} , define $G \setminus \mathcal{C}$ as the graph constructed from G by deleting all cut-edges of \mathcal{C} . Given a graph G , a *hint-set* is a set S of vertices of G such that S has at least one vertex from each connected component of $G \setminus \mathcal{C}$.

Definition 2 (Problem B). *Given an input graph G and a hint-set for an unknown labeling of G , find the labeling using a small number of queries.*

To show that Problem A can be reduced to Problem B, we select $\frac{c \log n}{\alpha}$ vertices uniformly at random and query the labels of the selected vertices. Since a connected component C contains at least αn vertices, the probability that a randomly selected vertex is not in C is at most $1 - \alpha$. Hence, the probability that all selected vertices fall outside C is at most

$$(1 - \alpha)^{\frac{c \log n}{\alpha}} \leq ((1 - \alpha)^{\frac{1}{\alpha}})^{c \log n} \leq \left(\frac{1}{n}\right)^c$$

The number of connected components is α^{-1} thus, with high probability we have obtained a hint-set and reduced the Problem A to Problem B.

In addition to this probabilistic reduction, a non-probabilistic one, though with an exponential running time, can also be proposed for the situation in which an upper bound k on the number of cut-edges of the labeling is given. As follows from a theorem proved by Kleinberg [6], for any graph G , parameter α , and integer $k < \frac{\alpha^2}{20} |V(G)|$, any subset of size $O(\frac{1}{\alpha} \log \frac{2}{\alpha})$, with probability at least $\frac{1}{2}$, is a hint-set for all α -balanced cuts of G with at most k cut-edges. So, we may examine each subset of vertices of G against every α -balanced cut of G with at most k cut-edges to find a subset that is a hint-set for all possible input labellings; then we will have an equivalent instance of problem B.

2.1 The Lower Bound

The balancedness condition prevents the problem from having a huge and trivial lowerbound. The idea behind our lowerbound is to construct a large number of balanced cuts on a fixed hint-set S .

Lemma 1. *Treating α as a constant, any algorithm that solves Problem A or B needs $k \log \frac{n}{k} - O(k)$ queries in the worst case, where n is the size of the graph and k is the cut-size of the labeling.*

Proof. Let G consist of k paths, each of length n/k , connected to each other through their endpoints as depicted in Figure 1. Suppose that the vertices in the

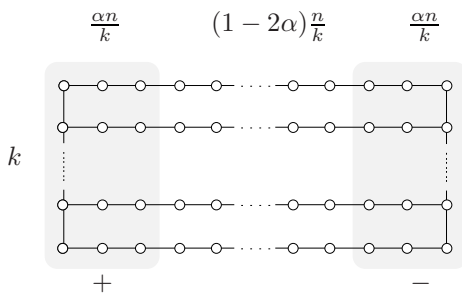


Fig. 1. Construction of the lowerbound

first (respectively, the last) α portion of each path are labeled with $+$ (respectively, with $-$). The cut we are looking for is formed by k edges from the middle $(1 - 2\alpha)$ portion of the paths, one from each path. There are $\left((1 - 2\alpha)\left(\frac{n}{k}\right)\right)^k$ choices for selecting these edges. Thus, any algorithm for finding the cut in this graph needs $\log\left(\left(1 - 2\alpha\right)\left(\frac{n}{k}\right)\right)^k = k \log \frac{n}{k} - O(k)$ queries in the worst case. \square

3 An Optimal Algorithm

First we examine a very special case in which the graph G is a path and the cut is a single edge. This special case will come handy in the solution for the general case.

3.1 Algorithm for Paths with Cut-Size One

In this case the cut essentially is just one edge and all vertices to one side of it are all labeled $+$ and the vertices on the other side are all labeled $-$. The solution is direct and is similar to the binary search algorithm. We start from both ends and query the labels of the endpoints. These will have opposite labels. Then we query the label of the midpoint and depending on the answer our search will be confined to one side of the path. We continue this binary search which eventually will find the cut using $O(\log n)$ queries.

Notice this binary search approach can still be used to find a cut-edge provided we start from two vertices with opposite labels.

3.2 Algorithm for Balanced Cuts

In this section, we develop an algorithm which matches the lowerbound proved in Section 2.1. We also focus on Problem B and assume a hint set $S = \{v_1, \dots, v_c\}$ is given. First, the algorithm uses c queries to find out labels of vertices in S . It then computes a sequence $G_0 = G, G_1, \dots, G_k = G \setminus \mathcal{C}$ of subgraphs of G where, for $1 \leq i \leq k, G_i = G_{i-1} - e_i$ for a cut-edge e_i of \mathcal{C} . To find a cut-edge e_i in G_{i-1} , it selects vertices u and v of S that are in the same connected component of G_i but have different labels. Then a “binary search” on a path between u and v in G_i is used to find a cut-edge. If computed naively, this path can have $\Omega(n)$ nodes, causing the algorithm to perform too many queries ($\Omega(\log n)$ in the worst case) at each step. Thus this naive approach will only result in the bound $O(k \log n)$. Although this bound of $O(k \log n)$ seems efficient, it still does not match the lowerbound in the previous section. To obtain a matching upperbound, we need one last ingredient: “domination sets” of G_i .

Definition 3. *In a connected graph G , a set of vertices R is an r -domination set, if the distance of any vertex in G to the set R is at most r . We use $f(r)$ to denote the size of an r -domination set with the least number of vertices among all r -domination sets of G .*

The next two lemma shows how domination sets are used to find a path of length at most $2r + 1$ connecting two vertices with different labels in G_{i-1} .

Lemma 2. *Given a graph G and an r -domination set R of G , for every two vertices u and v of the same connected component of G , one can construct a walk W from u to v such that every $2r + 1$ consecutive vertices in W contain at least one vertex from R .*

Proof. Let $P = (u = p_1, p_2, \dots, p_l = v)$ be an arbitrary path between u and v in G . Since R is an r -domination, for each vertex $p_i \in P$ there is a vertex $r_i \in R$ such that there is a path P_i of length at most r connecting p_i to r_i . We define P'_i to be the reverse of P_i , for all $1 \leq i \leq l$, and W to be the walk $(P_1, P'_1, P_2, P'_2, \dots, P_l, P'_l)$. Intuitively, the walk W starts walking along the vertices in P and at each vertex p_i , it first goes to r_i and then returns to p_i . So, each segment of size $2r + 1$ of W contains at least one vertex from R . □

Lemma 3. *Suppose l is a labeling of a graph G , R is an r -domination set in G , and u_1 and u_2 are vertices with $l(u_1) \neq l(u_2)$. There are vertices v_1 and v_2 of distance at most $2r + 1$ in $R \cup \{u_1, u_2\}$ such that $l(v_1) \neq l(v_2)$.*

Proof. Consider the walk W from u_1 to u_2 described in Lemma 2 and let r_1, r_2, \dots, r_p be the vertices of R appearing in W in order from u_1 to u_2 . Also, define $r_0 = u_1$ and $r_{p+1} = u_2$. By Lemma 2, there are at most $2r$ vertices in W between r_i and r_{i+1} , exclusive, for $0 \leq i \leq p$. Since $l(r_0) \neq l(r_{p+1})$, there is an $0 \leq i \leq p$ such that $l(r_i) \neq l(r_{i+1})$. The correctness of the lemma follows by setting $v_1 = r_i$ and $v_2 = r_{i+1}$. □

Our algorithm will use the above two lemmas to iteratively find and extract the cut edges. Since at each step of the algorithm we will remove a cut edge, we must be able to update the r -domination set under edge deletions. Fortunately, this can be done trivially as the next lemma shows. We omit the prove since the proof is mostly intuitive.

Lemma 4. *If R is an r -domination set in G and $uv \in E$, then $R' = R \cup \{u, v\}$ is an r -domination set in $G' = G - uv$.*

The following property of r -domination sets allow the algorithm to asymptotically achieve the query complexity of the existing lower bound.

Lemma 5. *In any n -vertex connected graph, $f(r) \leq 2n/r$.*

Proof. This upper bound can be achieved by a naive greedy algorithm. Start with one vertex as the initial domination set R and progressively add vertices to R in steps. In each step, find a vertex with distance more than r to R and add it to R (if there is no such vertex, we are finished). We claim $|R| \leq 2n/r$.

We define the d -neighborhood of a vertex u , denoted by $N_u(d)$, as the set of all vertices within distance d of u . If v and w are in the greedily-selected r -domination set R , then $N_v(r/2)$ and $N_w(r/2)$ have an empty intersection. Hence, there are $|R|$ $\frac{r}{2}$ -neighborhoods formed around vertices of R that are pairwise disjoint. Since, there are at least $r/2$ vertices (including v) in $N_v(r/2)$, for any vertex v , $r|R|/2$ is a lower bound on the number of vertices n . This immediately implies that $|R| \leq 2n/r$. □

Algorithm FINDCUTEDGES(graph G , hint-set S , integers r and κ)

1. Find an r -domination set R for G using the greedy approach
 2. Query labels of vertices in S and R
 3. Set $C = \emptyset$
 4. **while** there are u_1, u_2 in S in the same component of G with $l(u_1) \neq l(u_2)$ **do**
 5. Find a path P between two vertices v_1 and v_2 in $R \cup \{u_1, u_2\}$ with length at most $2r + 1$ such that $l(v_1) \neq l(v_2)$
 6. Use binary search to find an edge $e = wx$ of P such that $l(w) \neq l(x)$
 7. Set $C = C \cup e$, $G = G - e$, and $R = R \cup \{w, x\}$
 8. **if** $|C| > \kappa$ **then** Fail;
 9. **Return** C
-

Fig. 2. Algorithm for finding the cut-edges of an unknown labeling of a graph

The greedy algorithm above seems naive as it can be far from achieving the optimal value of $f(r)$. However we prove, in the next lemma, that the size of its output can be upper bounded by the function f in some way.

Lemma 6. *Suppose, on an input graph G , the greedy algorithm, as described in Lemma 5, comes up with an r -domination set of size $\text{Greedy}(r)$. Then,*

$$f(r) \leq \text{Greedy}(r) \leq f(r/2).$$

Proof. It is obvious that $f(r) \leq \text{Greedy}(r)$, as $f(r)$ is the minimum size of a domination set and $\text{Greedy}(r)$ is the size one such set. To prove $\text{Greedy}(r) \leq f(r/2)$, it is sufficient to consider the $\frac{r}{2}$ -neighborhoods formed around the selected vertices in Lemma 5. For a set to be an eligible $\frac{r}{2}$ -domination set, it has to include at least one vertex from each of these $\frac{r}{2}$ -neighborhoods; otherwise the center would be at a distance more than $r/2$ from all the vertices in the domination set. Since these neighborhoods are pairwise disjoint (see the proof of Lemma 5), the size of any eligible $\frac{r}{2}$ -domination set must be at least the number of $\frac{r}{2}$ -neighborhoods which is exactly $\text{Greedy}(r)$. Hence, $f(r/2) \geq \text{Greedy}(r)$. □

Consider the algorithm shown in Figure 2. It accepts additional parameters r and κ where κ is enforced to be an upper-bound on the cut-size. The algorithm starts by constructing an r -domination set R of G . Then, it performs $|R| + |S|$ queries to find out the labels of the vertices in R and in S . The algorithm runs in at most κ steps and in the i -th step it constructs the graph G_i . We use G_i , R_i , and C_i , respectively, to denote values of variables G , R , and C at the beginning of the i -th iteration of the while loop ($i \geq 0$).

At the beginning of the iteration i , we have a partially computed cut C_i , a graph $G_i = G \setminus C_i$, and an r -domination set R_i for G_i . Also, the algorithm knows the labels of vertices in S and R_i . Next, it uses Lemma 3 to find a path P of length at most $2r + 1$ between two vertices with different labels. Hence, the algorithm uses at most $\lceil \log(2r + 1) \rceil$ queries to find a new cut-edge e in G_i . The edge e is removed from the graph and the r -domination set is updated based on

Lemma 4 If $\kappa \geq k$, removing all edges of \mathcal{C} one by one in this way, the algorithm finds the solution using $|S| + Greedy(r) + \kappa \log r$ queries. The algorithm fails, when $\kappa < k$, before submitting more than $|S| + Greedy(r) + \kappa \log r$ queries.

Notice that we can compute $Greedy(r)$ for different values of r in advance and choose the value which minimizes the query complexity. Then, the number of queries will be at most $|S| + O(\kappa) + \min_r \{Greedy(r) + \kappa \log r\}$ which is at most

$$|S| + O(\kappa) + \min_r \{f(r/2) + \kappa \log r\} = |S| + O(\kappa) + \min_r \{f(r) + \kappa \log r\}$$

according to Lemma 6

Let U be the set of endpoints of all cut-edges. We can further reduce the number of queries by noticing that every edge between a vertex u with positive label and a vertex v with negative label must be a cut-edge. Once we remove all such trivial cut-edges, the next step of the algorithm will find a new vertex $v \in U$. This implies we can bound the number of steps by $k' = |U|$. The final theorem is as follows:

Theorem 1. *For a graph G , a labeling l , and an integer κ , one can use $|S| + O(\kappa) + \min_r \{f(r) + \kappa \log r\}$ queries to discover if $\kappa < k'$ and to solve the problem B when $\kappa \geq k'$, where S is a hint-set for l .*

According to Lemma 5, $f(2n/\kappa) \leq \kappa$. Thus, if we run the algorithm of Theorem 1 with parameters $\kappa = 1, \kappa = 2, \kappa = 4, \dots$ and $r = 2n/\kappa$, until κ becomes as large as k' , we get the following theorem.

Corollary 1. *For a graph G and a labeling l , the problem B can be solved using $|S| + O(k') + 2k' \log \frac{n}{k'}$ queries, where k' is the number of vertices adjacent to cut-edges and S is a hint-set for l .*

Finally, we must mention that our probabilistic reduction of problem A to problem B used $\frac{c \log n}{\alpha}$ queries which is always asymptotically smaller than the above query complexity and thus we can safely omit this term.

4 The Tightness of the Bounds

The result of the Corollary 1 is tight with respect to parameters n and k' . However, for specific graphs better bounds might be possible. Note that given a graph G and a hit-set S one available lowerbound is the logarithm of the number of cuts for which S is the hint-set. Clearly, this number depends entirely on the structure of graph G . Same can be said about the result of Theorem 1. For instance, if G is a full binary tree then we have $f(r) = \theta(\frac{n}{2^r})$ which means the query complexity of Theorem 1 is in fact $\kappa \log \log \frac{n}{\kappa} + O(\kappa)$ with a matching asymptotic lowerbound for this particular tree. In this section we generalize this observation for all trees. In other words, we prove if G is a tree then the result of Theorem 1 is asymptotically tight by providing a matching lowerbound which entirely depends on the structure of G and thus throughout this section we always assume G is a tree.

We use the output S of the greedy algorithm of Lemma 5 to construct many of labellings for G , all with the same hint-set S . Consider an arbitrary r and the r -domination set S returned by greedy algorithm. This special r -domination set has the property that for every $u, v \in S$, $dist(u, v) \geq r + 1$. We call any r -domination set with this property a *distributed r -domination set*. For a vertex $v \in S$ let N_v be the $\lceil \frac{r}{2} \rceil$ -neighborhood of v . If $u \neq v$, then the edge-sets of the graphs induced by N_u and N_v do not intersect, that is, $E(G[N_u]) \cap E(G[N_v]) = \emptyset$. Suppose v_0, \dots, v_{m-1} is an ordering of the vertices of S . We have assumed G is a tree and thus there is a unique path connecting v_i to v_j . Define P_{ij} to be the portion of this path which falls inside N_{v_i} . We have $|P_{ij}| \geq 1 + \lceil r/2 \rceil$, for every $0 \leq i < j \leq m - 1$. The fact that $E(G[N_{v_{i_1}}]) \cap E(G[N_{v_{i_2}}]) = \emptyset$ implies that the edge sets of the paths $P_{i_1 j_1}$ and $P_{i_2 j_2}$ do not intersect, for $i_1 \neq i_2$. This results in the following lemma.

Lemma 7. *If S is a distributed r -domination set of size m in a tree G and $k < m$ be an arbitrary integer then, there are at least $\lceil \frac{r}{2} \rceil^k$ cuts each having k cut-edges such that S is a hint-set for every one of them.*

Proof. Consider the notation above and the following algorithm:

```

Set  $C = \emptyset$ 
while  $|C| < k$  do
    - Choose the lexicographically smallest pair  $(i, j)$ ,  $0 \leq i < j \leq m - 1$ 
      such that  $v_i$  and  $v_j$  are in the same component of  $G \setminus C$ .
    - Nondeterministically select an edge of  $P_{ij}$  and add it to  $C$ 
    
```

Since $k \leq m - 1$, the selection of the pair (i, j) in each execution of the body of the while loop is feasible. As each path P_{ij} has $\lceil \frac{r}{2} \rceil$ edges and the body of the while loop is executed k times, there are $\lceil \frac{r}{2} \rceil^k$ possibilities, in overall, for nondeterministic choices of the algorithm. Moreover, each time that an edge is deleted from $G \setminus C$ and a connected component of $G \setminus C$ is split into two, each new connected component still includes a vertex of S (either v_i or v_j). Therefore, S is a hint-set for the cut specified by any set C generated by this algorithm.

It remains to show that all the $\lceil \frac{r}{2} \rceil^k$ sets C generated by the algorithm are distinct. Consider two different executions \mathcal{E}_1 and \mathcal{E}_2 of the algorithm and consider the first point that the algorithm makes different decisions in \mathcal{E}_1 and in \mathcal{E}_2 . Suppose \mathcal{E}_1 chooses an edge e_1 of P_{ij} while \mathcal{E}_2 selects a different edge e_2 of P_{ij} . Without loss of generality, assume e_1 appears before e_2 in P_{ij} . The edge e_2 is in $\lceil \frac{r}{2} \rceil$ -neighborhood of v_i and after adding e_1 to C in \mathcal{E}_1 , e_2 is not in the same component as v_i anymore; so \mathcal{E}_1 will never add e_2 to C . Thus, the value of C in \mathcal{E}_2 will be different from the value of C in \mathcal{E}_1 . So, each execution of the algorithm generates a distinct set of edges. □

Next theorem uses this set of cuts to give a lowerbound.

Theorem 2. *For every tree G with n vertices and integer $k > 0$, there is an integer r such that any algorithm solving problem B must perform $f(r) + k \log r - O(k)$ queries.*

Proof. Due to the term $-O(k)$, we can assume $k \leq \frac{n}{2}$. Define m_t as the maximum size of any distributed t -domination set. The size of the maximum independent set of a tree is at least $\frac{n}{2}$ which implies $m_1 \geq \frac{n}{2}$. As r increases, m_r must decrease, and in particular $m_n = 1$. We know $k \leq \lceil \frac{n}{2} \rceil$ and thus there is an integer $t > 1$ such that $f(t + 1) \leq m_{t+1} \leq k < m_t$. According to Lemma 7 any algorithm solving problem B must perform $k \log \frac{t}{2}$ queries and a simple calculation shows that $f(t + 1) + k \log(t + 1) - O(k) \leq k \log t$ which means $f(r) + k \log r - O(k)$ is a lowerbound for the number of queries for $r = t + 1$. \square

5 Relaxing the Balancedness Assumption

In this section we show how to remove the balancedness assumption at the expense of obtaining an approximation algorithm. Thus, we present an ε -approximation algorithm that performs $k \ln(3/\varepsilon) + k \log(n/k) + O(k)$ queries on a given graph with n vertices and cut-size k and reports a labeling which with high probability has at most εn vertices mislabeled.

The algorithm is fundamentally same as before. We select a random set of vertices uniformly as the hint-set and perform Algorithm 2 just once for $\kappa = k$ and $r = n/k$. Nevertheless, the probabilistic analysis presented in Section 2 holds no more; the connected components here can be arbitrarily small so there could be components which do not contain any vertex from the hint set.

We call the components that have a vertex representative in the sample as the *represented* components and the rest as the *unrepresented*. Firstly, we claim that if we select $(k + 1) \ln(\varepsilon/3)$ sample vertices, with high probability, the number of vertices in an unrepresented component is at most εn . The argument is probabilistic; we compute the expected number of vertices in unrepresented components. We denote by n_1, \dots, n_t the sizes of connected components C_1, \dots, C_t in the initial graph respectively. Using basic probability arguments, the probability that the component C_i is unrepresented is at most $(1 - \frac{n_i}{n})^{(k+1) \ln(\varepsilon/3)}$. Therefore, the expected total number of vertices in unrepresented components is

$$\mathbf{E} = \sum_{i=1}^t n_i \left(1 - \frac{n_i}{n}\right)^{(k+1) \ln(\varepsilon/3)} \leq n \sum_{i=1}^t \frac{n_i}{n} e^{-\frac{n_i}{n} (k+1) \ln(\varepsilon/3)}.$$

As the function xe^{-cx} is convex for any fixed c , the maximum happens when $\frac{n_i}{n}$ are equal and thus $\mathbf{E} \leq n \left(t \frac{1}{t} e^{-\frac{1}{t} (k+1) \ln(\varepsilon/3)}\right) \leq n\varepsilon/3$. The last inequality is due to the fact that k cannot be less than $t - 1$. By using Markov inequality, one can assert that the probability that there are more than εn vertices in unreported components is less than $1/3$.

Secondly, we observe that by a run of Algorithm 2, labels of vertices of represented components are reported correctly. This fact follows from the correctness of the algorithm, and the observation that it never reports an edge as a cut-edge if it is not indeed a cut-edge. The algorithm might miss some cut-edges in contrast to the previous setting, as some components are unrepresented, and so the labeling of such components can be reported arbitrarily.

Combining the two latter facts, we conclude that the total number of vertices whose labels are misreported is less than εn for an arbitrary small $\varepsilon < 1$.

6 Conclusion

In this paper we studied the query learning problem, while we assumed both labeled and unlabeled data were available to us and we had a similarity graph constructed on data items. The problem was discussed in the following two settings: We studied the case where we are given a hint set of vertices in which there exists at least one vertex from each connected component of the same label. We gave the lower bound $k \log n/k - O(k)$ on the number of queries required for this case. We provided an algorithm that finds the optimal hypothesis using at most $O(k \log n/k)$ queries, which matched our lower bound for general graphs. Hence, our proposed algorithm is optimal in terms of the number of queries.

In the second setting, we showed that if the labeling is α -balanced, i.e. each connected component of the vertices with the same label has at least α fraction of all vertices, we could find a hint set with high probability using $\frac{c \log n}{\alpha}$ random queries. Note that our previous lower bound works for the α -balanced graphs, too. This gives an evidence of the optimality of our algorithm for general graphs in the latter setting.

Although our algorithm for the first setting was proved to be optimal for general graphs, we may have different lower bounds for special families of graphs. In Section 4, we investigated the problem for trees and proved for tree our algorithm is asymptotically optimal. One natural question is whether it is possible to extend this result to other classes of graphs or not.

Finally, we considered the problem for non-balance cuts. We developed an ε -approximation algorithm for this case. However, we assumed that the number of cut-edges is given to the algorithm. Thus, the following problem remained open:

Open Problem: *Suppose that G is a graph, l is a labeling of G with k cut-edges, and $0 < \varepsilon \leq 1$ is a real number. Does there exist an ε -approximation algorithm that runs in polynomial time and submits at most $O(\text{poly}(1/\varepsilon)k \log n/k)$ queries without knowing k in advance?*

References

1. Blum, A., Chawla, S.: Learning from labeled and unlabeled data using graph mincuts. In: Proceedings of the Eighteenth International Conference on Machine Learning, pp. 19–26. Morgan Kaufmann Publishers, San Francisco (2001)
2. Blum, A., Lafferty, J., Rwebangira, M.R., Reddy, R.: Semi-supervised learning using randomized mincuts. In: Proceedings of the twenty-first international conference on Machine learning, p. 13. ACM Press, New York (2004)
3. Joachims, T.: Transductive learning via spectral graph partitioning. In: Twentieth International Conference on Machine Learning (2003)

4. Joachims, T.: Transductive learning via spectral graph partitioning. In: Proceedings of the International Conference on Machine Learning, pp. 290–297 (2003)
5. Kamvar, S., Klein, D., Manning, C.: Spectral learning. In: International Joint Conference On Artificial Intelligence (2003)
6. Kleinberg, J.: Detecting a network failure. In: Proceedings of the Forty-First Annual Symposium on Foundations of Computer Science, p. 231. IEEE Computer Society Press, Los Alamitos (2000)
7. Mitchell, T.: Machine Learning. McGraw-Hill, New York (1997)
8. Ng, A., Jordan, M., Weiss, Y.: On spectral clustering: Analysis and an algorithm. In: Advances in Neural Information Processing Systems (2001)
9. Shi, J., Malik, J.: Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* 22(8), 888–905 (2000)
10. Zhu, X., Ghahramani, Z., Lafferty, J.: Semi-supervised learning using Gaussian fields and harmonic functions. In: Proceedings of the Twentieth International Conference on Machine Learning, pp. 912–919 (2003)

“Resistant” Polynomials and Stronger Lower Bounds for Depth-Three Arithmetical Formulas^{*}

Maurice J. Jansen and Kenneth W. Regan^{**}

Dept. of CSE, University at Buffalo (SUNY), 201 Bell Hall, Buffalo, NY 14260-2000
Tel.: (716) 645-3180 x114; Fax: 645-3464
regan@cse.buffalo.edu.

Abstract. We derive quadratic lower bounds on the $*$ -complexity of sum-of-products-of-sums ($\Sigma\Pi\Sigma$) formulas for classes of polynomials f that have too few partial derivatives for the techniques of Shpilka and Wigderson [10,9]. This involves a notion of “resistance” which connotes full-degree behavior of f under any projection to an affine space of sufficiently high dimension. They also show stronger lower bounds over the reals than the complex numbers or over arbitrary fields. Separately, by applying a special form of the Baur-Strassen Derivative Lemma tailored to $\Sigma\Pi\Sigma$ formulas, we obtain sharper bounds on $+$, $*$ -complexity than those shown for $*$ -complexity by Shpilka and Wigderson [10], most notably for the lowest-degree cases of the polynomials they consider.

Keywords: Computational complexity, arithmetical circuits, lower bounds, constant depth formulas, partial derivatives.

1 Introduction

In contrast to the presence of exponential size lower bounds on constant-depth Boolean circuits for majority and related functions [3,13,7], and depth-3 arithmetical circuits over finite fields [5,6], Shpilka and Wigderson [10] observed that *over fields of characteristic zero* (which are infinite), super-quadratic lower bounds are not known even for constant-depth *formulas*. Indeed they are unknown for unbounded fan-in, depth 3 formulas that are sums of products of affine linear functions, which they call $\Sigma\Pi\Sigma$ formulas. These formulas have notable *upper-bound power* because they can carry out forms of Lagrange interpolation. As they ascribed to M. Ben-Or, $\Sigma\Pi\Sigma$ formulas can compute the elementary symmetric polynomials S_n^k (defined as the sum of all degree- k monomials in n variables, and analogous to majority and threshold- k Boolean functions) in size $O(n^2)$ independent of k . Thus $\Sigma\Pi\Sigma$ formulas present a substantial challenge for lower bounds, as well as being a nice small-scale model to study.

Shpilka and Wigderson defined the *multiplicative size* of an arithmetical (circuit or) formula ϕ to be the total fan-in to multiplication gates. We denote this

^{*} Part of this work by both authors was supported by NSF Grant CCR-9821040.

^{**} Corresponding author.

by $\ell^*(\phi)$, and write $\ell(\phi)$ for the total fan-in to all gates, i.e. $+$ gates as well. The best known lower bound for general arithmetical circuits has remained for thirty years the $\Omega(n \log n)$ lower bound on ℓ^* by the “Degree Method” of Strassen [11] (see also [12]). However, this comes nowhere near the exponential lower bounds conjectured by Valiant [12] for the permanent and expected by many for other NP-hard arithmetical functions. For polynomials f of total degree $n^{O(1)}$, the method is not even capable of $\Omega(n^{1+\epsilon})$ circuit lower bounds, not for any $\epsilon > 0$. Hence it is notable that [10] achieved better lower bounds on $\ell_3^*(f)$, where the subscript-3 refers to $\Sigma\Pi\Sigma$ formulas. These were $\Omega(n^2)$ for $f = S_n^k$ when $k = \Theta(n)$, $n^{2-\epsilon_k}$ for S_n^k with small values of k , and $\Omega(N^2/\text{polylog}(N))$ for the determinant, with $N = n^2$. However, $\Omega(n^2)$ is the best this can do for $\Sigma\Pi\Sigma$ formulas. Shpilka [9] got past this only in some further-restricted cases, and also considered a depth-2 model consisting of an arbitrary symmetric function of sums. This barrier provides another reason to study the $\Sigma\Pi\Sigma$ model, in order to understand the obstacles and what might be needed to surpass them.

The techniques in [8,10,9] all depend on the set of d th-order partial derivatives of f being large. This condition fails for functions such as $f(x_1, \dots, x_n) = x_1^n + \dots + x_n^n$, which has only n d th-order partials for any d . We refine the analysis to show the sufficiency of f behaving like a degree- r polynomial on any affine subspace A of sufficiently high dimension (for this f , $r = n$ and any affine line suffices). Our technical condition is that for every polynomial g of total degree at most $r - 1$ and every such A , there exists a d -th order partial of $f - g$ that is non-constant on A . This enables us to prove an absolutely sharp n^2 bound on $\ell_3^*(f)$ for this f computed over the real or rational numbers, and a lower bound of $n^2/2$ over any field of characteristic zero. Note the absence of “ O , Ω ” notation. We prove similar tight bounds for sums of powered monomial blocks, powers of inner-products, and functions depending on ℓ_p -norm distance from the origin, and also replicate the bounds of [10,9] for symmetric polynomials. Even in the last case, we give an example where our simple existential condition may work deeper than the main question highlighted in [9] on the maximum dimension of subspaces A on which S_n^k vanishes.

In Section 5 we prove lower bounds on $+, *$ complexity $\ell_3(f)$ that are significantly higher (but still sub-quadratic) than those given for $\ell_3^*(f)$ in [10] when the degree r of f is small. This is done intuitively by exploiting a closed-form application of the Baur-Strassen “Derivative Lemma” [1] to $\Sigma\Pi\Sigma$ formulas, showing that f and all of its n first partial derivatives can be computed with only a constant-factor increase in ℓ and ℓ^* over $\Sigma\Pi\Sigma$ formulas for f .

2 Preliminaries

A $\Sigma\Pi\Sigma$ -formula is an arithmetic formula consisting of four consecutive layers: a layer of inputs, next a layer of addition gates, then a layer of multiplication gates, and finally the output sum gate. The gates have unbounded fan-in from the previous layer (only), and individual wires may carry arbitrary constants from the underlying field. Given a $\Sigma\Pi\Sigma$ -formula we can write $p = \sum_{i=1}^s M_i$,

where $M_i = \prod_{j=1}^{d_i} l_{i,j}$, and $l_{i,j} = c_{i,j,1}x_1 + c_{i,j,2}x_2 + \dots + c_{i,j,n}x_n + c_{i,j,0}$. Here d_i is the in-degree of the i th multiplication gate, and $c_{i,j,k}$ is nonzero iff there is a wire from x_k to the addition gate computing $l_{i,j}$.

Let $X = (x_1, \dots, x_n)$ be an n -tuple of variables. For any affine linear subspace $A \subset F^n$, we can always find a set of variables $B \subset X$, and affine linear forms l_b in the variables $X \setminus B$, for each $b \in B$, such that A is the set of solutions of $\{x_b = l_b : b \in B\}$. This representation is not unique. The set B is called a *base* of A . The size $|B|$ always equals the co-dimension of A . In the following, we always assume some base B of A to be fixed. Any of our numerical “progress measures” used to prove lower bounds will not depend on the choice of a base.

Following Shpilka and Wigderson [10], for polynomial $f \in F[x_1, \dots, x_n]$, the *restriction of f to A* is defined to be the polynomial obtained by substitution of l_b for variable x_b for each $b \in B$, and is denoted by $f|_A$. For a set of polynomials W , define $W|_A = \{f|_A \mid f \in W\}$. For a linear form $l = c_1x_1 + \dots + c_nx_n + c_0$, we denote $l^h = c_1x_1 + \dots + c_nx_n$. For a set S of linear forms, $S^h = \{l^h : l \in S\}$.

3 Resistance of Polynomials

We state our new definition in the weakest and simplest form that suffices for the lower bounds, although the functions in our applications all meet the stronger condition of Lemma 1 below.

Definition 1. *A polynomial f in variables x_1, x_2, \dots, x_n is (d, r, k) -resistant if for any polynomial $g(x_1, x_2, \dots, x_n)$ of degree at most $r - 1$, for any affine linear subspace A of co-dimension k , there exists a d th order partial derivative of $f - g$ that is non-constant on A .*

For a multiset X of size d with elements taken from $\{x_1, x_2, \dots, x_n\}$, we will use the notation $\frac{\partial^d f}{\partial X}$ to indicate the d th-order derivative with respect to the variables in X . As our applications all have $r = \deg(f)$, we call f simply (d, k) -resistant in this case. Then the case $d = 0$ says that f itself has full degree on any affine A of co-dimension k , and in most cases corresponds to the non-vanishing condition in [10]. We separate our notion from [10] in applications and notably in the important case of the elementary symmetric polynomials in Section 4.4 below.

The conclusion of Definition 1 is not equivalent to saying that some $(d + 1)$ st-order partial of $f - g$ is non-vanishing on A , because the restriction of this partial on A need not be the same as a first-partial of the restriction of the d th-order partial to A . Moreover, (d, k) -resistance need not imply $(d - 1, k)$ -resistance, even for $d, k = 1$: consider $f(x, y) = xy$ and A defined by $x = 0$.

Theorem 1. *Suppose $f(x_1, x_2, \dots, x_n)$ is (d, r, k) -resistant, then*

$$\ell_3^*(f) \geq r \frac{k + 1}{d + 1}.$$

Proof. Consider a $\Sigma\Pi\Sigma$ -formula that computes f . Remove all multiplication gates that have degree at most $r - 1$. Doing so we obtain a $\Sigma\Pi\Sigma$ formula \mathcal{F} computing $f - g$, where g is some polynomial of degree at most $r - 1$. Say \mathcal{F} has s multiplication gates. Write: $f - g = \sum_{i=1}^s M_i$, where $M_i = \prod_{j=1}^{d_i} l_{i,j}$ and $l_{i,j} = c_{i,j,1}x_1 + c_{i,j,2}x_2 + \dots + c_{i,j,n}x_n + c_{i,j,0}$. The degree of each multiplication gate in \mathcal{F} is at least r , i.e. $d_i \geq r$, for each $1 \leq i \leq s$. Now select a set S of input linear forms using the following algorithm:

```

S = ∅
for i = 1 to s do
  repeat d + 1 times:
    if (∃j ∈ {1, 2, ..., d_i}): S^h ∪ {l_{i,j}^h} is a set of independent vectors
      then S = S ∪ {l_{i,j}}
    
```

Let A be the set of common zeroes of the linear forms in S . Since S^h is an independent set, A is affine linear of co-dimension $|S| \leq (d + 1)s$.

We claim that if at a multiplication gate M_i we picked strictly fewer than $d + 1$ linear forms, then any linear form that was not picked is constant on A . Namely, each linear form l that was not picked had l^h already in the span of S^h , for the set S built up so far. Hence we can write $l = c + l^h = c + \sum_{g \in S} c_g g^h$, for certain scalars c_g . Since each g^h is constant on A , we conclude l is constant on A . This settles the claim, and yields that for each multiplication gate either

1. $(d + 1)$ input linear forms vanish on A , or
2. fewer than $(d + 1)$ linear forms vanish on A , with all others constant on A .

For each multiset X of size d with elements from $\{x_1, x_2, \dots, x_n\}$, the d th order partial derivative $\partial^d(f - g)/\partial X$ is in the linear span of the set

$$\left\{ \prod_{\substack{j=1 \\ j \notin J}}^{d_i} l_{ij} : 1 \leq i \leq s, J \subseteq \{1, 2, \dots, d_i\}, |J| = d \right\}$$

This follows from the sum and product rules for derivatives and the fact that a first order derivative of an individual linear form l_{ij} is a constant. Consider $1 \leq i \leq s$ and $J \subseteq \{1, 2, \dots, d_i\}$ with $|J| = d$. If item 1. holds for the multiplication gate M_i , then $\prod_{\substack{j=1 \\ j \notin J}}^{d_i} l_{ij}$ vanishes on A , since there must be one l_{ij} that vanishes on A that was not selected, given that $|J| = d$. If item 2 holds for M_i , then this product is constant on A .

Hence, we conclude that $\partial^d(f - g)/\partial X$ is constant on A . Since f is (d, r, k) -resistant, we must have that the co-dimension of A is at least $k + 1$. Hence $(d + 1)s \geq k + 1$. Since each gate in \mathcal{F} is of degree at least r , we obtain $\ell_3^*(\mathcal{F}) \geq r \frac{k+1}{d+1}$. Since \mathcal{F} was obtained by removing zero or more multiplication gates from a $\Sigma\Pi\Sigma$ -formula computing f , we have proven the statement of the theorem. \square

To prove lower bounds on resistance, we supply the following lemma:

Lemma 1. *Over fields of characteristic zero, for any $d \leq r$, $k > 0$, and any polynomial $f(x_1, x_2, \dots, x_n)$, if for every affine linear subspace A of co-dimension k , there exists some d th order partial derivative of f such that*

$$\deg \left(\left(\frac{\partial^d f}{\partial X} \right) \Big|_A \right) \geq r - d + 1, \quad \text{then } f \text{ is } (d, r + 1, k)\text{-resistant.}$$

Proof. Assume for every affine linear subspace A of co-dimension k , there exists some d th order partial derivative derivative of f such that

$$\deg \left(\left(\frac{\partial^d f}{\partial X} \right) \Big|_A \right) \geq r - d + 1.$$

Let g be an arbitrary polynomial of degree r . Then

$$\left(\frac{\partial^d f - g}{\partial X} \right) \Big|_A = \left(\frac{\partial^d f}{\partial X} - \frac{\partial^d g}{\partial X} \right) \Big|_A = \left(\frac{\partial^d f}{\partial X} \right) \Big|_A - \left(\frac{\partial^d g}{\partial X} \right) \Big|_A.$$

The term $\left(\frac{\partial^d f}{\partial X} \right) \Big|_A$ has degree at least $r - d + 1$, whereas the term $\left(\frac{\partial^d g}{\partial X} \right) \Big|_A$ can have degree at most $r - d$. Hence $\deg \left(\left(\frac{\partial^d f - g}{\partial X} \right) \Big|_A \right) \geq r - d + 1 \geq 1$. Since over fields of characteristic zero, syntactically different polynomials define different mappings, we conclude $\frac{\partial^d f - g}{\partial X}$ must be non-constant on A . □

The main difference between Lemma 1 and the original Definition 1 appears to be the order of quantifying the polynomial “ g ” of degree $r - 1$ out front in the former, whereas analogous considerations in the lemma universally quantify it later (making a stronger condition). We have not found a neat way to exploit this difference in any prominent application, however.

4 Applications

4.1 Sum of n th Powers Polynomial

Consider $f = \sum_{i=1}^n x_i^n$. By repeated squaring for each x_i^n , one obtains $\Sigma\Pi$ circuits (not formulas) of size $O(n \log n)$. All arithmetical circuits require size $\Omega(n \log n)$ for f [1]. The expression for f yields a $\Sigma\Pi\Sigma$ formula ϕ with n multiplication gates of degree n , with n^2 wires in the top linear layer fanning in to them. This works over any field, but makes $\ell(\phi) = \ell^*(\phi) = n^2$. We prove that this is close to optimal.

Theorem 2. *Over fields of characteristic zero, any $\Sigma\Pi\Sigma$ -formula for $f = \sum_{i=1}^n x_i^n$ has multiplicative size at least $n^2/2$.*

Proof. By Theorem [1](#) it suffices to show f is $(1, n - 1)$ -resistant. Let g be an arbitrary polynomial of degree $n - 1$. Letting g_1, \dots, g_n denote the first order partial derivatives of g , we get that the i th partial derivative of $f - g$ equals $nx_i^{n-1} - g_i(x_1, \dots, x_n)$. Note that the g_i 's are of total degree at most $n - 2$.

We claim there is no affine linear subspace of dimension greater than zero on which all $\partial f / \partial x_i$ are constant. Consider an arbitrary affine line $x_i = c_i + d_i t$ parameterized by a variable t , where c_i and d_i are constants for all $i \in [n]$, and with at least one d_i nonzero. Then $\frac{\partial(f-g)}{\partial x_i}$ restricted to the line is given by $n(c_i + d_i t)^{n-1} - h_i(t)$, for some univariate polynomials $h_i(t)$ of degree $\leq n - 2$. Since there must exist *some* i such that d_i is nonzero, we know some partial derivative restricted to the affine line is parameterized by a univariate polynomial of degree $n - 1$, and thus, given that the field is of characteristic zero, is not constant for all t . □

In case the underlying field is the real numbers \mathbf{R} and n is even, we can improve the above result to prove an absolutely tight n^2 lower bound.

Theorem 3. *Over the real numbers, for even n , any $\Sigma\Pi\Sigma$ -formula for $f = \sum_{i=1}^n x_i^n$ has multiplicative size at least n^2 .*

Proof. Since f is symmetric we can assume without loss of generality that A has the base representation $x_{k+1} = l_1(x_1, \dots, x_k), \dots, x_n = l_{n-k}(x_1, \dots, x_k)$. Then

$$f|_A = x_1^n + \dots + x_k^n + l_1^n + \dots + l_{n-k}^n.$$

Hence $f|_A$ must include the term x_1^n , since each l_j^n has a non-negative coefficient for the term x_1^n and n is even. Thus via Lemma [1](#) we conclude that over the real numbers f is $(0, n - 1)$ -resistant. Hence, by Theorem [1](#) we get that $\ell_3^*(f) \geq \deg(f) \frac{n}{1} = n^2$. □

Let us note that $f = \sum_{i=1}^n x_i^n$ is an example of a polynomial that has few d -th partial derivatives, namely only n of them regardless of d . This renders the partial derivatives technique of Shpilka and Wigderson [\[10\]](#)—which we will describe and extend in the next section—not directly applicable.

4.2 Blocks of Powers Polynomials

Let the underlying field have characteristic zero, and suppose $n = m^2$ for some m . Consider the “ m blocks of m powers” polynomial $f = \sum_{i=1}^m \prod_{j=(i-1)m+1}^{im} x_j^m$. The straightforward $\Sigma\Pi\Sigma$ -formula for f , that computes each term/block using a multiplication gate of degree n , is of multiplicative size $n^{3/2}$. We will show this is tight.

Proposition 1. *The blocks of powers polynomial f is $(0, m - 1)$ -resistant.*

Proof. Consider an affine linear space of co-dimension $m - 1$. For any base B of A , restriction to A consists of substitution of the $m - 1$ variables in B by linear forms in the remaining variables X/B . This means there is at least one term/block

$B_i := \prod_{j=(i-1)m+1}^{im} x_j^m$ of f whose variables are disjoint from B . This block B_i remains the same under restriction to A . Also, for every other term/block there is at least one variable that is not assigned to. As a consequence, B_i cannot be canceled against terms resulting from restriction to A of other blocks. Hence $\deg(f|_A) = \deg(f)$. Hence by Lemma [1](#) we have that f is $(0, m - 1)$ -resistant. \square

Corollary 1. *For the blocks of powers polynomial f , $\ell_3^*(f) \geq nm = n^{3/2}$.*

Alternatively, one can observe that by substitution of a variable y_i for each variable appearing in the i th block one obtains from a $\Sigma\Pi\Sigma$ -formula \mathcal{F} for f a formula for $f' = \sum_{i=1}^m y_i^n$ of the same size as \mathcal{F} . Theorem [2](#) generalizes to show that $\ell_3^*(f') \geq \frac{1}{2}n^{3/2}$, which implies $\ell_3^*(f) \geq \frac{1}{2}n^{3/2}$.

4.3 Polynomials Depending on Distance to the Origin

Over the real numbers, $d_2(x) = x_1^2 + x_2^2 + \dots + x_n^2$ is the square of the Euclidean distance of the point x to the origin. Polynomials f of the form $q(d_2(x))$ where q is a single-variable polynomial can be readily seen to have high resistance. Only the leading term of q matters. For example, consider $f = (x_1^2 + x_2^2 + \dots + x_n^2)^m$. On any affine line L in \mathbf{R}^n , $\deg(f|_L) = 2m$. Therefore, by Lemma [1](#), over the reals, f is $(0, n - 1)$ -resistant. Hence by Theorem [1](#) we get that

Proposition 2. *Over the real numbers, $\ell_3^*((x_1^2 + x_2^2 + \dots + x_n^2)^m) \geq 2mn$.*

Observe that by reduction this means that the “ m th-power of an inner product polynomial”, defined by $g = (x_1y_1 + x_2y_2 + \dots + x_ny_n)^m$, must also have $\Sigma\Pi\Sigma$ -size at least $2mn$ over the reals numbers. Results for l_p norms, $p \neq 2$, are similar.

4.4 The Case of Symmetric Polynomials

The special case of $(0, k)$ -resistance is implicitly given by Shpilka [\[9\]](#), at least insofar as the sufficient condition of Lemma [1](#) is used for the special case $d = 0$ in which no derivatives are taken. For the elementary symmetric polynomial S_n^r of degree $r \geq 2$ in n variables, Theorem 4.3 of [\[9\]](#) implies (via Lemma [1](#)) that S_n^r is $(0, n - \frac{n+r}{2})$ -resistant. Shpilka proves for $r \geq 2$, $\ell_3^*(S_n^r) = \Omega(r(n - r))$, which can be verified using Theorem [1](#): $\ell_3^*(S_n^r) \geq (r + 1)(n - \frac{n+r}{2}) = \Omega(r(n - r))$.

The symmetric polynomials S_n^k collectively have the “telescoping” property that every d th-order partial is (zero or) the symmetric polynomial S_{n-d}^{k-d} on an $(n - d)$ -subset of the variables. Shpilka [\[9\]](#) devolves the analysis into the question, “What is the maximum dimension of a linear subspace of \mathbf{C}^n on which S_n^r vanishes?” In Shpilka’s answer, divisibility properties of r come into play as is witnessed by Theorem 5.9 of [\[9\]](#). To give an example case of this theorem, one can check that S_9^2 vanishes on the 3-dimensional linear space given by

$$\{(x_1, \omega x_1, \omega^2 x_1, x_2, \omega x_2, \omega^2 x_2, x_3, \omega x_3, \omega^2 x_3) : x_1, x_2, x_3 \in \mathbf{C}\},$$

where ω can be selected to be either primitive 3rd root of unity. Let

$$\rho_0(f) = \max\{k : \text{for any linear } A \text{ of codim. } k, f|_A \neq 0\}.$$

Shpilka proved for $r > n/2$, that $\rho_0(S_n^r) = n - r$, and for $r \geq 2$, that $\frac{n-r}{2} < \rho_0(S_n^r) \leq n - r$. For S_9^2 we see via divisibility properties of d that the value for ρ_0 can get less than the optimum value, although the $\frac{n-r}{2}$ lower bound suffices for obtaining the above mentioned $\ell_3^*(S_n^r) = \Omega(r(n-r))$ lower bound. We have some indication from computer runs using the polynomial algebra package *Singular* [4] that the “unruly” behavior seen for ρ_0 because of divisibility properties for $r \leq n/2$ can be made to go away by considering the following notion:

$$\rho_1(f) = \max \left\{ k : \text{for any linear } A \text{ of codim. } k, \text{ there exists } i, \left(\frac{\partial f}{\partial x_i} \right)_{|_A} \neq 0 \right\}$$

One can still see from the fact that S_n^r is homogeneous and using Lemma [1] and Theorem [1] that $\ell_3^*(S_n^r) \geq \frac{r \cdot (\rho_1(S_n^r) + 1)}{2}$. Establishing the exact value of $\rho_1(S_n^r)$, which we conjecture to be $n + 1 - r$ at least over the rationals, seems at least to simplify obtaining the $\ell_3(S_n^r) = \Omega(d(n - d))$ lower bound. In the full version we prove that for $r \geq 2$, $\rho_1(S_{n+1}^{r+1}) \geq \rho_0(S_n^{r-1})$.

For another example, S_6^3 is made to vanish at dimension 3 not by any subspace that zeroes out 3 co-ordinates but rather by $A = \{(u, -u, w, -w, y, -y) : u, w, y \in \mathbf{C}\}$. Now add a new variable t in defining $f = S_7^4$. The notable fact is that f 1-resists the dimension-3 subspace A' obtained by adjoining $t = 0$ to the equations for A , upon existentially choosing to derive by a variable other than t , such as u . All terms of $\partial f / \partial u$ that include t vanish, leaving 10 terms in the variables v, w, x, y, z . Of these, 4 pairs cancel under the equations $x = -w, z = -y$, but the leftover $vwx + vyz$ part equates to $uw^2 + uy^2$, which not only doesn't cancel but also dominates any contribution from the lower-degree g . Gröbner basis runs using *Singular* imply that S_7^4 is (1, 4)-resistant over \mathbf{C} as well as the rationals and reals, though we have not yet made this a consequence of a general resistance theorem for all S_n^r .

Hence our (1, k)-resistance analysis for S_7^4 is not impacted by the achieved upper bound of 3 represented by A . Admittedly the symmetric polynomials f have $O(n^2)$ upper bounds on $\ell_3(f)$, so our distinction in this case does not directly help surmount the quadratic barrier. But it does show promise of making progress in our algebraic understanding of polynomials in general.

5 Bounds for +,*-Complexity

The partial derivatives technique used by Shpilka and Wigderson [10] ignores the wires of the formula present in the first layer. In the following we show how to account for them. As a result we get a sharpening of several lower bounds, though not on ℓ_3^* but on total formula size. We employ the concepts and lemmas from [10]. For $f \in F[x_1, \dots, x_n]$, let $\partial_d(f)$ be the set of all d th order formal partial derivatives of f w.r.t. variables from $\{x_1, \dots, x_n\}$. For a set of polynomials $A = \{f_1, \dots, f_t\}$ $\text{span}(A) = \{\sum_{i=1}^t c_i f_i \mid c_i \in F\}$. Write $\text{dim}[A]$ as shorthand for $\text{dim}[\text{span}(A)]$. Note $\text{span}(f_1, \dots, f_t)_{|_A} = \text{span}(f_{1|_A}, \dots, f_{t|_A})$, and that $\text{dim}[W_{|_A}] \leq \text{dim}[W]$. The basic inequality from [10] then becomes:

Proposition 3. $\dim[\partial_d(c_1f_1 + c_2f_2)|_A] \leq \dim[\partial_d(f_1)|_A] + \dim[\partial_d(f_2)|_A]$.

We refine two main results in [10] *-complexity into results with tighter bounds but for +, *-complexity. In each case we compare old and new versions.

Theorem 4 ([10]). *Let $f \in F[x_1, \dots, x_n]$. Suppose for integers d, D, κ it holds that for every affine subspace A of co-dimension κ , $\dim(\partial_d(f)|_A) > D$. Then $\ell_3^*(f) \geq \min(\frac{\kappa^2}{d}, \frac{D}{\binom{\kappa+d}{d}})$;*

Theorem 5 (new). *Let $f \in F[x_1, \dots, x_n]$. Suppose for integers d, D, κ it holds that for every affine subspace A of co-dimension κ , $\sum_{i=1}^n \dim[\partial_d(\frac{\partial f}{\partial x_i})|_A] > D$. Then $\ell_3(f) \geq \min(\frac{\kappa^2}{d+2}, \frac{D}{\binom{\kappa+d}{d}})$.*

Proof. Consider a minimum-size $\Sigma\Pi\Sigma$ -formula for f with multiplication gates M_1, \dots, M_s . We have that $f = \sum_{i=1}^s M_i$, where for $1 \leq i \leq s$, $M_i = \prod_{j=1}^{d_i} l_{i,j}$ and $l_{i,j} = c_{i,j,1}x_1 + c_{i,j,2}x_2 + \dots + c_{i,j,n}x_n + c_{i,j,0}$, for certain constants $c_{i,j,k} \in F$. Computing the partial derivative of f w.r.t. variable x_k we get

$$\frac{\partial f}{\partial x_k} = \sum_{i=1}^s \sum_{j=1}^{d_i} c_{i,j,k} \frac{M_i}{l_{i,j}}. \tag{1}$$

Let $S = \{i : \dim[M_i^h] \geq \kappa\}$. If $|S| \geq \frac{\kappa}{d+2}$, then $\ell_3(f) \geq \frac{\kappa^2}{d+2}$. Suppose $|S| < \frac{\kappa}{d+2}$. If $S = \emptyset$, then let A be an arbitrary affine subspace of co-dimension κ . Otherwise, construct an affine space A as follows. Since $|S|(d+2) < \kappa$, and since for each $j \in S$, $\dim[M_j^h] \geq \kappa$, it is possible to pick $d+2$ input linear forms $l_{j,1}, \dots, l_{j,d+2}$ of each multiplication gate M_j with $j \in S$, such that $\{l_{j,1}^h, \dots, l_{j,d+2}^h | j \in S\}$ is a set of $|S|(d+2) < \kappa$ independent homogeneous linear forms. Define

$$A = \{x : l_{i,j}(x) = 0, \text{ for any } i \in S, j \in [d+2]\}.$$

We have that the co-dimension of A is at most κ . W.l.o.g. assume the co-dimension of A equals κ . For each $i \in S$, $d+2$ linear forms of M_i vanish on A . This implies that $\dim[\partial_d(\frac{M_i}{l_{i,j}})|_A] = 0$, for any $i \in S$. For any $i \notin S$, by Proposition 2.3 in [10], $\dim[\partial_d(\frac{M_i}{l_{i,j}})|_A] < \binom{\kappa+d}{d}$. Let $D_k = \dim[\partial_d(\frac{\partial f}{\partial x_k})|_A]$. By Proposition 3 and equation (1),

$$D_k \leq \sum_{i \notin S} \sum_{\substack{j \\ c_{i,j,k} \neq 0}} \dim \left[\partial_d \left(\frac{M_i}{l_{i,j}} \right) \Big|_A \right].$$

Hence there must be at least $\frac{D_k}{\binom{\kappa+d}{d}}$ terms on the r.h.s., i.e. there are at least that many wires from x_k to gates in the first layer. Hence in total the number of wires to the first layer is at least $\sum_{i=1}^n \frac{D_i}{\binom{\kappa+d}{d}} > \frac{D}{\binom{\kappa+d}{d}}$. \square

Theorem 6 ([10]). *Let $f \in F[x_1, \dots, x_n]$. Suppose for integers d, D, κ it holds that for every affine subspace A of co-dimension κ , $\dim(\partial_d(f|_A)) > D$. Then for every $m \geq 2$, $\ell_3^*(f) \geq \min(\kappa m, \frac{D}{\binom{m}{d}})$.*

Theorem 7 (new). *Let $f \in F[x_1, \dots, x_n]$. Suppose for integers d, D, κ with $d \geq 1$, it holds that for every affine subspace A of co-dimension κ , $\sum_{i=1}^n \dim[\partial_d(\frac{\partial f}{\partial x_i}|_A)] > D$. Then for every $m \geq 2$, $\ell_3(f) \geq \min(\frac{1}{2}\kappa m, \frac{D}{\binom{m-1}{d}})$.*

The proof of Theorem 7 is analogous to above and appears in the full version.

In [10] it was proved that for $d \leq \log n$, $\ell_3^*(S_n^{2d}) = \Omega(n^{\frac{2d}{d+2}})$. Note that for $d = 2$, this lower bound is only $\Omega(n)$. We can apply Theorem 5 to prove the following stronger lower bound on the total formula size of S_n^{2d} . In particular for $d = 2$, we get an $\Omega(n^{\frac{4}{3}})$ bound.

Theorem 8. *For $1 \leq d \leq \log n$, $\ell_3(S_n^{2d}) = \Omega(n^{\frac{2d}{d+1}})$.*

Proof. For any affine subspace A of co-dimension κ and $d \geq 2$ we have that

$$\sum_{i=1}^n \dim \left[\partial_{d-1} \left(\frac{\partial S_n^{2d}}{\partial x_i} \right) \Big|_A \right] \geq \dim[\partial_d(S_n^{2d})|_A] \geq \binom{n-\kappa}{d}.$$

The latter inequality follows from Lemma 4.4 in [10]. Applying Theorem 5 we get that

$$\ell_3(S_n^{2d}) \geq \min \left(\frac{\kappa^2}{d+1}, \frac{\binom{n-\kappa}{d}}{\binom{\kappa+d-1}{d-1}} \right) = \min \left(\frac{\kappa^2}{d+1}, \frac{\binom{n-\kappa}{d} \kappa + d}{\binom{\kappa+d}{d}} \right). \tag{2}$$

Set $\kappa = \frac{1}{9}n^{\frac{d}{d+1}}$. Then we have that

$$\frac{\binom{n-\kappa}{d} \kappa + d}{\binom{\kappa+d}{d}} \geq \left(\frac{n-\kappa}{\kappa+d} \right)^d \frac{\kappa+d}{d} \geq \left(\frac{8/9n}{2/9n^{\frac{d}{d+1}}} \right)^d \frac{\kappa+d}{d} = 4^d n^{\frac{d}{d+1}} \frac{\kappa+d}{d} \geq \frac{4^d}{9^d} n^{\frac{2d}{d+1}} \geq n^{\frac{2d}{d+1}}.$$

Hence (2) is at least $\min(n^{\frac{2d}{81(d+1)}}, n^{\frac{2d}{d+1}}) = \Omega(n^{\frac{2d}{d+1}})$. □

Corollary 2. $\ell_3(S_n^4) = \Omega(n^{4/3})$.

Shpilka and Wigderson defined the “product-of-inner-products” polynomial over $2d$ variable sets of size n (superscript indicate different variables, each variable has degree one) by $PIP_n^d = \prod_{i=1}^d \sum_{j=1}^n x_j^i y_j^i$.

Theorem 9. *For any constant $d > 0$, $\ell_3(PIP_n^d) = \Omega(n^{\frac{2d}{d+1}})$.*

Proof. Let $f = PIP_n^d$. Essentially we have that $\frac{\partial f}{\partial x_j^i} = y_j^i PIP_n^{d-1}$, where the PIP_n^{d-1} must be chosen on the appropriate variable set. Let A be an arbitrary affine linear subspace of co-dimension κ . Then

$$\begin{aligned} \sum_{i=1}^d \sum_{j=1}^n \dim \left[\partial_{d-1} \left(\frac{\partial f}{\partial x_j^i} \Big|_A \right) \right] &= \sum_{i=1}^d \sum_{j=1}^n \dim[\partial_{d-1}(y_j^i PIP_n^{d-1}|_A)] \\ &\geq (dn - \kappa) \dim[\partial_{d-1}(PIP_n^{d-1}|_A)] \end{aligned}$$

The last inequality follows because at least $dn - \kappa$ of the y -variables are not assigned to with the restriction to A . From Lemma 4.9 in [10] one gets

$$\dim[\partial_{d-1}(PIP_n^{d-1}|_A) \geq n^{d-1} - 2^{2d-1}\kappa n^{d-2}.$$

Using Theorem 7 we get

$$\ell_3(f) \geq \min\left(\frac{\kappa^2}{2}, \frac{(dn - \kappa)(n^{d-1} - 2^{2d-1}\kappa n^{d-2})}{\binom{\kappa-1}{d-1}}\right).$$

Taking $\kappa = n^{\frac{d}{d+1}}$, one gets for constant d that $\ell_3(PIP_n^d) = \Omega(n^{\frac{2d}{d+1}})$. \square

For comparison, in [10] one gets $\ell_3^*(PIP_n^d) = \Omega(n^{\frac{2d}{d+2}})$.

6 Conclusion

We have taken some further steps after Shpilka and Wigderson [10,9], obtaining absolutely tight (rather than asymptotically so) multiplicative size lower bounds for some natural functions, and obtaining somewhat improved bounds on $+$, $*$ -size for low-degree symmetric and product-of-inner-product polynomials. However, these may if anything enhance the feeling from [10,9] that the concepts being employed may go no further than quadratic for lower bounds. One cannot after all say that a function $f(x_1, \dots, x_n)$ is non-vanishing on an affine-linear space of co-dimension more than n . The quest then is for a mathematical invariant that scales beyond linear with the number of degree- d -or-higher multiplication gates in the formula.

Acknowledgments. We thank Avi Wigderson for comments on a very early version of this work, and referees of later versions for very helpful criticism.

References

1. Baur, W., Strassen, V.: The complexity of partial derivatives. *Theor. Comp. Sci.* 22, 317–330 (1982)
2. Bürgisser, P., Clausen, M., Shokrollahi, M.A.: *Algebraic Complexity Theory*. Springer, Heidelberg (1997)
3. Fürst, M., Saxe, J., Sipser, M.: Parity, circuits, and the polynomial-time hierarchy. *Math. Sys. Thy.* 17, 13–27 (1984)
4. Greuel, G.-M., Pfister, G., Schönemann, H.: *Singular 3.0. A Computer Algebra System for Polynomial Computations*, Centre for Computer Algebra, University of Kaiserslautern (2005), <http://www.singular.uni-kl.de>
5. Grigoriev, D., Karpinski, M.: An exponential lower bound for depth 3 arithmetic circuits. In: *Proc. 30th Annual ACM Symposium on the Theory of Computing*, pp. 577–582. ACM Press, New York (1998)
6. Grigoriev, D., Razborov, A.A.: Exponential lower bounds for depth 3 algebraic circuits in algebras of functions over finite fields. *Applicable Algebra in Engineering, Communication, and Computing* 10, 465–487 (1998) (preliminary version FOCS 1998)

7. Håstad, J.: Almost optimal lower bounds for small-depth circuits. In: Micali, S. (ed.) *Randomness and Computation*. *Advances in Computing Research*, vol. 5, pp. 143–170. JAI Press, Greenwich, CT (1989)
8. Nisan, N., Wigderson, A.: Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity* 6, 217–234 (1996)
9. Shpilka, A.: Affine projections of symmetric polynomials. *J. Comp. Sys. Sci.* 65, 639–659 (2002)
10. Shpilka, A., Wigderson, A.: Depth-3 arithmetic formulae over fields of characteristic zero. *Computational Complexity* 10, 1–27 (2001)
11. Strassen, V.: *Berechnung und Programm II*. *Acta Informatica* 2, 64–79 (1973)
12. Valiant, L.: The complexity of computing the permanent. *Theor. Comp. Sci.* 8, 189–201 (1979)
13. Yao, A.: Separating the polynomial-time hierarchy by oracles. In: *Proc. 26th Annual IEEE Symposium on Foundations of Computer Science*, pp. 1–10. IEEE Computer Society Press, Los Alamitos (1985)

An Improved Algorithm for Tree Edit Distance Incorporating Structural Linearity*

Shihyen Chen and Kaizhong Zhang

Department of Computer Science,
The University of Western Ontario, London, Ontario, Canada, N6A 5B7
{schen,kzhang}@csd.uwo.ca

Abstract. An ordered labeled tree is a tree in which the nodes are labeled and the left-to-right order among siblings is significant. The edit distance between two ordered labeled trees is the minimum cost of transforming one tree into the other by a sequence of edit operations. Among the best known tree edit distance algorithms, the majority can be categorized in terms of a framework named cover strategy. In this paper, we investigate how certain locally linear features may be utilized to improve the time complexity for computing the tree edit distance. We define structural linearity and present a method incorporating linearity which can work with existing cover-strategy based tree algorithms. We show that by this method the time complexity for an input of size n becomes $\mathcal{O}(n^2 + \phi(\mathcal{A}, \tilde{n}))$ where $\phi(\mathcal{A}, \tilde{n})$ is the time complexity of any cover-strategy algorithm \mathcal{A} applied to an input size \tilde{n} , with $\tilde{n} \leq n$, and the magnitude of \tilde{n} is reversely related to the degree of linearity. This result is an improvement of previous results when $\tilde{n} < n$ and would be useful for situations in which \tilde{n} is in general substantially smaller than n , such as RNA secondary structure comparisons in computational biology.

Keywords: Tree edit distance, dynamic programming, RNA secondary structure comparison.

1 Introduction

An ordered labeled tree is a tree in which the nodes are labeled and the left-to-right order among siblings is significant. Trees can represent many phenomena, such as grammar parses, image descriptions and structured texts, to name a couple. In many applications where trees are useful representations of objects, the need for comparing trees frequently arises.

The tree edit distance metric was introduced by Tai [7] as a generalization of the string edit distance problem [9]. Given two trees T_1 and T_2 , the tree edit distance between T_1 and T_2 is the minimum cost of transforming one tree into the other, with the sibling and ancestor orders preserved, by a sequence of edit operations on the nodes (relabeling, insertion and deletion) as shown in Figure 1.

* Research supported partially by the Natural Sciences and Engineering Research Council of Canada under Grant No. OGP0046373 and a grant from MITACS, a Network of Centres of Excellence for the Mathematical Sciences.

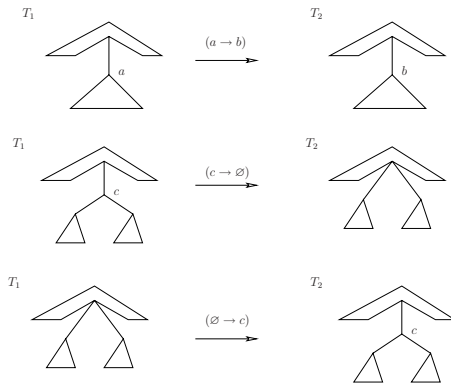


Fig. 1. Tree edit operations. From top to bottom: relabeling, deletion and insertion.

Among the known algorithms of comparable results such as in [12,4,10], the majority [2,4,10] can be categorized in terms of a generalized framework by the name of *cover strategy* [3] which prescribes the direction by which a dynamic program builds up the solution. Briefly, a tree is decomposed into a set of disjoint paths which, in this paper, we refer to as *special paths*. Each special path is associated with a subtree such that the special path coincides with a path of the subtree which runs from the root to a leaf. The dynamic program proceeds in a bottom-up order with respect to the special paths such that for any node i on a special path, the subtrees hanging off the portion of the special path no higher than i have been processed before the node i is reached. When there are subtrees hanging off on both sides of a special path, the decision as to which side takes precedence is referred to as *strategy*. In the Zhang-Shasha algorithm [10], the special paths are chosen to be the leftmost paths. In Klein’s algorithm [4] as well as that of Demaine *et al.* [2], the special paths are chosen such that every node on a special path is the root of a largest subtree over its sibling subtrees. These special paths are referred to as *heavy paths* [6]. Examples of leftmost paths and heavy paths are shown in Figure 2.

In these algorithms, no consideration is given to any structural characteristics which may exist in the tree. In this paper, we investigate the possibility of utilizing certain linear features within the trees to speed up the computation of the tree edit distance. We show that by embedding a procedure in any cover-



Fig. 2. Left: Decomposition of a tree into leftmost paths (in bold). Right: Decomposition of a tree into heavy paths (in bold).

strategy algorithm \mathcal{A} , the resulting time complexity is $\mathcal{O}(n^2 + \phi(\mathcal{A}, \tilde{n}))$ where n is the original input size, $\phi(\mathcal{A}, \tilde{n})$ is the time complexity of algorithm \mathcal{A} applied to an input size \tilde{n} , with $\tilde{n} \leq n$, and the magnitude of \tilde{n} is reversely related to the degree of linearity. This result would be useful for applications in which \tilde{n} is in general substantially smaller than n .

The rest of the paper is organized as follows. In Section 2, we define structural linearity and give a new representation of trees based on reduction of the tree size due to the linearity. In Section 3, we show the algorithmic aspects as a result of incorporating the linearity and the implications on the time complexity. In Section 4, we describe one suitable application of our result. We give concluding remarks in Section 5.

2 Preliminaries

2.1 Notations

Given a tree T , we denote by $t[i]$ the i th node in the left-to-right post-order numbering. The index of the leftmost leaf of the subtree rooted at $t[i]$ is denoted by $l(i)$. We denote by $F[i, j]$ the ordered sub-forest of T induced by the nodes indexed i to j inclusive. The subtree rooted at $t[i]$ in T is denoted by $T[i]$, i.e., $T[i] = F[l(i), i]$. The sub-forest induced by removing $t[i]$ from $T[i]$ is denoted by $F[i]$, i.e., $F[i] = F[l(i), i - 1]$. When referring to the children of a specific node, we adopt a subscript notation in accordance with the left-to-right sibling order. For example, the children of $t[i]$, from left to right, may be denoted by $(t[i_1], t[i_2], \dots, t[i_k])$.

2.2 Linearity

Definition 1 (V-Component). *Given a tree T with left-to-right post-order and pre-order numberings, a path π of T is a v-component (i.e., vertically linear component) if all of the following conditions hold.*

- Both the post-order and the pre-order numberings along this path form a sequence of continuous indices.
- No other path containing π satisfies the above condition.

Definition 2 (V-Reduction). *The v-reduction on a tree is to replace every v-component in the tree by a single node.*

Definition 3 (H-Component). *Given a tree T and another tree \tilde{T} obtained by a v-reduction on T , any set of connected components of T corresponding to a set of leaves \tilde{L} in \tilde{T} form an h-component (i.e., horizontally linear component) if all of the following conditions hold.*

- $|\tilde{L}| \geq 2$.
- All the leaves in \tilde{L} share the same parent.
- A left-to-right post-order or pre-order numbering on \tilde{T} produces a sequence of continuous indices for \tilde{L} .
- No other set of leaves containing \tilde{L} satisfy the above conditions.

Definition 4 (H-Reduction). *The h-reduction on a tree is to replace every h-component in the tree by a single node.*

A tree possesses vertical (horizontal) linearity if it contains any v-component (h-component). A tree is v-reduced if it is obtained by a v-reduction only. A tree is vh-reduced if it is obtained by a v-reduction followed by an h-reduction.

Note that a reduced tree is just a compact representation of the original tree. The edit distance of two reduced trees is the same as the edit distance of the original trees. In the case when a tree does not possess any linearity as defined above, the reduced tree is the same as the original tree.

In Figure 3, we give an example showing the v-components and h-components of a tree and the corresponding reduced trees. Note that an h-component can also contain v-components.

Each node in a v-reduced tree corresponds to either a v-component or a single node in the corresponding full tree. Given a v-reduced tree \tilde{T} , we define

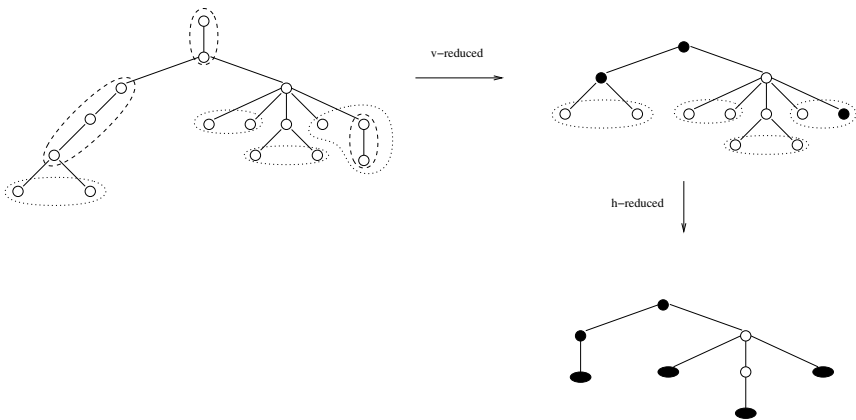


Fig. 3. The v-components (in dashed enclosures) and the h-components (in dotted enclosures). Also shown are the reduced trees as a result of reduction. The parts of the original tree affected by reduction are represented by black nodes in the reduced tree.

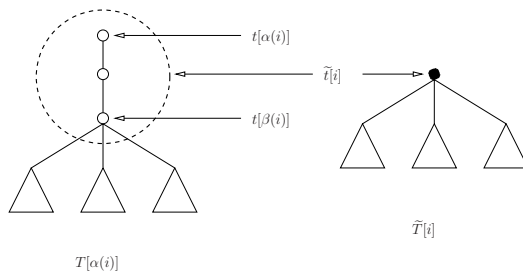


Fig. 4. A partial view of the mapping of nodes between a tree (left) and its v-reduced tree (right)

two functions $\alpha()$ and $\beta()$ which respectively map a node $\tilde{t}[i]$ to the highest indexed node $t[\alpha(i)]$ and the lowest indexed node $t[\beta(i)]$ of the corresponding v -component in the full tree T . In the special case when $\tilde{t}[i]$ corresponds to a single node in T , $t[\alpha(i)] = t[\beta(i)]$. An example of this mapping is given in Figure 4. When \tilde{T} is h -reduced to yield \hat{T} , $\alpha()$ and $\beta()$ apply in the same way for the mapping from \hat{T} to \tilde{T} .

3 Algorithm

In this section, we show how to incorporate vertical linearity in the Zhang-Shasha algorithm. We will also show that the method can be incorporated in all the cover-strategy algorithms.

Due to space limitation, we shall not discuss the incorporation of horizontal linearity in this paper. The method involves adapting techniques from matrix searching and will be given elsewhere in the future.

3.1 Incorporating Vertical Linearity

We denote by $d(,)$ the edit distance. The following lemmas incorporate vertical linearity in the Zhang-Shasha algorithm.

Lemma 1

1. $d(\emptyset, \emptyset) = 0$.
2. $\forall i \in \tilde{T}_1, \forall i' \in [l(i), i], d(\tilde{F}_1[l(i), i'], \emptyset) = d(\tilde{F}_1[l(i), i' - 1], \emptyset) + d(\tilde{t}_1[i'], \emptyset)$.
3. $\forall j \in \tilde{T}_2, \forall j' \in [l(j), j], d(\emptyset, \tilde{F}_2[l(j), j']) = d(\emptyset, \tilde{F}_2[l(j), j' - 1]) + d(\emptyset, \tilde{t}_2[j'])$.

Proof Case 1 requires no edit operation. In case 2 and case 3, the distances correspond to the costs of deleting and inserting the nodes in $\tilde{F}_1[l(i), i']$ and $\tilde{F}_2[l(j), j']$, respectively. □

Lemma 2. $\forall (i, j) \in (\tilde{T}_1, \tilde{T}_2), \forall i' \in [l(i), i]$ and $\forall j' \in [l(j), j]$,
 if $l(i') = l(i)$ and $l(j') = l(j)$,

$$d(\tilde{F}_1[l(i), i'], \tilde{F}_2[l(j), j']) = d(\tilde{T}_1[i'], \tilde{T}_2[j']) ;$$

otherwise,

$$d(\tilde{F}_1[l(i), i'], \tilde{F}_2[l(j), j']) = \min \left\{ \begin{array}{l} d(\tilde{F}_1[l(i), i' - 1], \tilde{F}_2[l(j), j']) + d(\tilde{t}_1[i'], \emptyset), \\ d(\tilde{F}_1[l(i), i'], \tilde{F}_2[l(j), j' - 1]) + d(\emptyset, \tilde{t}_2[j']), \\ d(\tilde{F}_1[l(i), l(i') - 1], \tilde{F}_2[l(j), l(j') - 1]) \\ + d(\tilde{T}_1[i'], \tilde{T}_2[j']) \end{array} \right\} .$$

Proof. The condition “ $l(i') = l(i)$ and $l(j') = l(j)$ ” implies that the two forests are simply two trees and the equality clearly holds. We now consider the other condition in which “ $l(i') \neq l(i)$ or $l(j') \neq l(j)$ ”. If $t_1[\alpha(i')] = t_1[\beta(i')]$ and

$t_2[\alpha(j')] = t_2[\beta(j')]$, the formula holds as a known result. Otherwise, at least one of $t_1[i']$ and $t_2[j']$ corresponds to a v-component in $(T_1[\alpha(i)], T_2[\alpha(j)])$. Consider the connected components in $(T_1[\alpha(i)], T_2[\alpha(j)])$ corresponding to $(t_1[i'], \tilde{t}_2[j'])$. There are two cases to consider: either (1) there is no occurrence of node-to-node match between the connected components; or (2) there is at least one occurrence of node-to-node match between the connected components. In case 1, one of the components must be entirely deleted which implies that either $\tilde{t}_1[i']$ must be deleted or $\tilde{t}_2[j']$ must be inserted. In case 2, in order to preserve the ancestor-descendent relationship $\tilde{T}_1[i']$ and $\tilde{T}_2[j']$ must be matched. \square

Note. In Lemma 2 for the condition “ $l(i') \neq l(i)$ or $l(j') \neq l(j)$ ” the value of $d(\tilde{T}_1[i'], \tilde{T}_2[j'])$ would already be available if implemented in a bottom-up order, since it involves a subproblem of $d(\tilde{F}_1[l(i), i'], \tilde{F}_2[l(j), j'])$ and would have been computed. For the condition “ $l(i') = l(i)$ and $l(j') = l(j)$ ”, however, we encounter the problem involving $(\tilde{T}_1[i'], \tilde{T}_2[j'])$ for the first time and must compute its value.

We show how to compute $d(\tilde{T}_1[i'], \tilde{T}_2[j'])$ in the following lemmas.

Lemma 3. $\forall u \in [\beta(i'), \alpha(i')]$,

$$d(T_1[u], F_2[\beta(j')]) = \min \left\{ \begin{array}{l} d(F_1[u], F_2[\beta(j')]) + d(t_1[u], \emptyset), \\ \min_{j'_1 \leq q \leq j'_k} \{d(T_1[u], T_2[\alpha(q)]) - d(\emptyset, T_2[\alpha(q)])\} \\ + d(\emptyset, F_2[\beta(j')]) \end{array} \right\} .$$

Proof. This is the edit distance between the tree $T_1[u]$ and the forest $F_2[\beta(j')]$. There are two cases. In the first case, $t_1[u]$ is constrained to be deleted and the remaining substructure $F_1[u]$ is matched to $F_2[\beta(j')]$. In the second case, $t_1[u]$ is constrained to be matched to a node somewhere in $F_2[\beta(j')]$. This is equivalent to stating that $T_1[u]$ is constrained to be matched to a subtree in $F_2[\beta(j')]$. The question thus becomes finding a subtree in $F_2[\beta(j')]$ to be matched to $T_1[u]$ so as to minimize the distance between $T_1[u]$ and $F_2[\beta(j')]$ under such constraint. This can be done by considering the set of all combinations in which exactly one tree in $F_2[\beta(j')]$ is matched to $T_1[u]$ while the remainder of $F_2[\beta(j')]$ is deleted. The minimum in this set is the edit distance for the second case. \square

Lemma 4. $\forall v \in [\beta(j'), \alpha(j')]$,

$$d(F_1[\beta(i')], T_2[v]) = \min \left\{ \begin{array}{l} d(F_1[\beta(i')], F_2[v]) + d(\emptyset, t_2[v]), \\ \min_{i'_1 \leq p \leq i'_k} \{d(T_1[\alpha(p)], T_2[v]) - d(T_1[\alpha(p)], \emptyset)\} \\ + d(F_1[\beta(i')], \emptyset) \end{array} \right\} .$$

Proof. This is symmetric to that of Lemma 3. \square

Lemma 5. $\forall u \in [\beta(i'), \alpha(i')]$ and $\forall v \in [\beta(j'), \alpha(j')]$,

$$d(T_1[u], T_2[v]) = \min \left\{ \begin{array}{l} d(F_1[u], T_2[v]) + d(t_1[u], \emptyset), \\ d(T_1[u], F_2[v]) + d(\emptyset, t_2[v]), \\ d(F_1[u], F_2[v]) + d(t_1[u], t_2[v]) \end{array} \right\} .$$

Proof. This is a known result for the tree-to-tree edit distance. \square

Note. In the computation for every $d(\tilde{T}_1[i'], \tilde{T}_2[j'])$, we save the values for $d(T_1[u], T_2[\alpha(j')]) \forall u \in [\beta(i'), \alpha(i')]$ and $d(T_1[\alpha(i')], T_2[v]) \forall v \in [\beta(j'), \alpha(j')]$. This ensures that when $d(T_1[u], F_2[\beta(j')])$ in Lemma 3 and $d(F_1[\beta(i')], T_2[v])$ in Lemma 4 are evaluated in a bottom-up order the values of the terms involving $d(T_1[u], T_2[\alpha(j')])$ and $d(T_1[\alpha(i')], T_2[v])$ would be available.

Lemma 6. $d(\tilde{T}_1[i'], \tilde{T}_2[j']) = d(T_1[\alpha(i')], T_2[\alpha(j')])$.

Proof. The result follows from the tree definitions. □

3.2 The New Algorithm

For every node i of tree T , we designate a child of i , if any, to be its *special child*, denoted by $sc(i)$. Note that in the Zhang-Shasha algorithm $sc(i)$ is the leftmost child of i whereas in a different cover-strategy the choice of $sc(i)$ may be different. Denote by $p(i)$ the parent of i . We define a set of nodes, called *key roots*, for tree T as follows.

$$keyroots(T) = \{k \mid k = root(T) \text{ or } k \neq sc(p(k))\} \text{ .}$$

This is a generalized version of the *LR-keyroots* used in [10] and is suitable for any known decomposition strategy as in [2,4,10]. Referring to Figure 2, in every special path the highest numbered node in a left-to-right post-order is a key root.

We now give the new algorithm in Algorithms 1 and 2. Algorithm 1 contains the main loop which repeatedly calls Algorithm 2 to compute $d(\tilde{T}_1[i], \tilde{T}_2[j])$ where (i, j) are key roots in $(\tilde{T}_1, \tilde{T}_2)$.

Theorem 1. *The new algorithm correctly computes $d(T_1, T_2)$.*

Proof. The correctness of all the computed values in Algorithm 2 follows from the lemmas. By Lemma 6, $d(\tilde{T}_1, \tilde{T}_2) = d(T_1, T_2)$ when (i', j') are set to be the roots of $(\tilde{T}_1, \tilde{T}_2)$. Since these roots are key roots, $d(T_1, T_2)$ is always computed by Algorithm 1. □

Algorithm 1. Computing $d(\tilde{T}_1, \tilde{T}_2)$

Input: $(\tilde{T}_1, \tilde{T}_2)$

Output: $d(\tilde{T}_1[i], \tilde{T}_2[j])$, where $1 \leq i \leq |\tilde{T}_1|$ and $1 \leq j \leq |\tilde{T}_2|$

- 1: Sort $keyroots(\tilde{T}_1)$ and $keyroots(\tilde{T}_2)$ in increasing order into arrays K_1 and K_2 , respectively
- 2: **for** $i' \leftarrow 1, \dots, |keyroots(\tilde{T}_1)|$ **do**
- 3: **for** $j' \leftarrow 1, \dots, |keyroots(\tilde{T}_2)|$ **do**
- 4: $i \leftarrow K_1[i']$
- 5: $j \leftarrow K_2[j']$
- 6: Compute $d(\tilde{T}_1[i], \tilde{T}_2[j])$ by Algorithm 2
- 7: **end for**
- 8: **end for**

Algorithm 2. Computing $d(\tilde{T}_1[i], \tilde{T}_2[j])$

```

1:  $d(\emptyset, \emptyset) \leftarrow 0$ 
2: for  $i' \leftarrow l(i), \dots, i$  do
3:    $d(\tilde{F}_1[l(i), i'], \emptyset) \leftarrow d(\tilde{F}_1[l(i), i' - 1], \emptyset) + d(\tilde{t}_1[i'], \emptyset)$ 
4: end for
5: for  $j' \leftarrow l(j), \dots, j$  do
6:    $d(\emptyset, \tilde{F}_2[l(j), j']) \leftarrow d(\emptyset, \tilde{F}_2[l(j), j' - 1]) + d(\emptyset, \tilde{t}_2[j'])$ 
7: end for
8: for  $i' \leftarrow l(i), \dots, i$  do
9:   for  $j' \leftarrow l(j), \dots, j$  do
10:    if  $l(i') = l(i)$  and  $l(j') = l(j)$  then
11:      for  $u \leftarrow \beta(i'), \dots, \alpha(i')$  do
12:         $d(T_1[u], F_2[\beta(j')]) \leftarrow$ 
13:           $\min \left\{ \begin{array}{l} d(F_1[u], F_2[\beta(j')]) + d(t_1[u], \emptyset), \\ \min_{j'_1 \leq q \leq j'_1} \{d(T_1[u], T_2[\alpha(q)]) - d(\emptyset, T_2[\alpha(q)])\} \\ + d(\emptyset, F_2[\beta(j')]) \end{array} \right\}$ 
14:      end for
15:      for  $v \leftarrow \beta(j'), \dots, \alpha(j')$  do
16:         $d(F_1[\beta(i')], T_2[v]) \leftarrow$ 
17:           $\min \left\{ \begin{array}{l} d(F_1[\beta(i')], F_2[v]) + d(\emptyset, t_2[v]), \\ \min_{i'_1 \leq p \leq i'_1} \{d(T_1[\alpha(p)], T_2[v]) - d(T_1[\alpha(p)], \emptyset)\} \\ + d(F_1[\beta(i')], \emptyset) \end{array} \right\}$ 
18:      end for
19:      for  $u \leftarrow \beta(i'), \dots, \alpha(i')$  do
20:        for  $v \leftarrow \beta(j'), \dots, \alpha(j')$  do
21:           $d(T_1[u], T_2[v]) \leftarrow \min \left\{ \begin{array}{l} d(F_1[u], T_2[v]) + d(t_1[u], \emptyset), \\ d(T_1[u], F_2[v]) + d(\emptyset, t_2[v]), \\ d(F_1[u], F_2[v]) + d(t_1[u], t_2[v]) \end{array} \right\}$ 
22:        end for
23:      end for
24:       $d(\tilde{T}_1[i'], \tilde{T}_2[j']) \leftarrow d(T_1[\alpha(i')], T_2[\alpha(j')])$ 
25:    else
26:       $d(\tilde{F}_1[l(i), i'], \tilde{F}_2[l(j), j']) \leftarrow$ 
27:         $\min \left\{ \begin{array}{l} d(\tilde{F}_1[l(i), i' - 1], \tilde{F}_2[l(j), j']) + d(\tilde{t}_1[i'], \emptyset), \\ d(\tilde{F}_1[l(i), i'], \tilde{F}_2[l(j), j' - 1]) + d(\emptyset, \tilde{t}_2[j']), \\ d(\tilde{F}_1[l(i), l(i') - 1], \tilde{F}_2[l(j), l(j') - 1]) + d(\tilde{T}_1[i'], \tilde{T}_2[j']) \end{array} \right\}$ 

```

Theorem 2. *The new algorithm runs in $\mathcal{O}(n^2 + \tilde{n}^4)$ time and $\mathcal{O}(n^2)$ space, where n is the original input size and $\tilde{n} \leq n$.*

Proof. We first consider the time complexity. The v-reduced trees can be built in linear time in a preprocess. Identifying and sorting the key roots can be done in linear time. As well, all the values associated with insertion or deletion of a subtree or a sub-forest, as appearing in Lemma 3 and Lemma 4, can be

obtained beforehand in linear time during a tree traversal. The Zhang-Shasha algorithm has a worst-case running time of $\mathcal{O}(\tilde{n}^4)$ for an input size \tilde{n} . Referring to Algorithm 2, we consider the block from line 11 to 21 which concerns computation involving the (i', j') pairs on leftmost paths, based on Lemmas 3, 4 and 5. This is the part of the algorithm that structurally differs from its counterpart in the Zhang-Shasha algorithm. For each such (i', j') pair, this part takes $\mathcal{O}((\alpha(i') - \beta(i') + 1) \times (j'_1 - j'_1 + 1) + (\alpha(j') - \beta(j') + 1) \times (i'_k - i'_1 + 1) + (\alpha(i') - \beta(i') + 1) \times (\alpha(j') - \beta(j') + 1))$. All the subproblems of (T_1, T_2) associated with these (i', j') pairs are disjoint. Summing over all these pairs, we can bound the complexity by $\mathcal{O}(n^2)$. Hence, the overall time complexity is $\mathcal{O}(n^2 + \tilde{n}^4)$.

We now consider the space complexity. We use three different arrays: the full-tree array, the reduced-tree array and the reduced-forest array. The reduced-forest array is a temporary array and its values can be rewritten during the computation of the reduced-forest distances. The other two arrays are permanent arrays for storing tree distances. The space for the reduced-tree array and the reduced-forest array is bounded by $\mathcal{O}(\tilde{n}^2)$. The space for the full-tree array is bounded by $\mathcal{O}(n^2)$. Hence, the space complexity is $\mathcal{O}(n^2)$. \square

Theorem 3. *Given (T_1, T_2) of maximum size n , the edit distance $d(T_1, T_2)$ can be computed in $\mathcal{O}(n^2 + \phi(\mathcal{A}, \tilde{n}))$ time where $\phi(\mathcal{A}, \tilde{n})$ is the time complexity of any cover-strategy algorithm \mathcal{A} applied to an input size \tilde{n} , with $\tilde{n} \leq n$.*

Proof. Since a v-component is consisted of a simple path, there is only one way a dynamic program can recurse along this path regardless which strategy is used. Hence, Lemmas 3 to 6 are valid for all cover strategies. Lemmas 1 and 2, after a proper adjustment of the subtree orderings in each forest to adapt to the given strategy, are also valid. The theorem is implied from Theorems 1 and 2 when the lemmas are properly embedded in any cover-strategy algorithm. \square

4 Application

We describe one application which would benefit from our result, namely RNA secondary structure comparison. RNA is a molecule consisted of a single strand of nucleotides (abbreviated as A, C, G and U) which folds back onto itself by means of hydrogen bonding between distant complementary nucleotides, giving rise to a so-called secondary structure. The secondary structure of an RNA molecule can be topologically represented by a tree. An example is depicted in Figure 5. In this representation, an internal node represents a pair of complementary nucleotides interacting via hydrogen bonding. When a number of such pairs are stacked up, they form a local structure called *stem*, which corresponds to a v-component in the tree representation. The secondary structure plays an important role in the functions of RNA [5]. Therefore, comparing the secondary structures of RNA molecules can help understand their comparative functions. One way to compare two trees is to compute the edit distance between them.

To gain an impression, we list the size reductions for a set of selected tRNA molecules in Table 1. $|T|$ is the size of the original tree. $|\tilde{T}|$ is the size of the

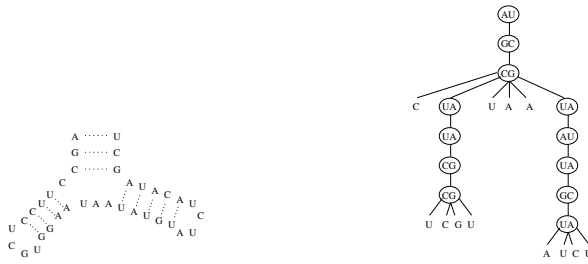


Fig. 5. Left: Secondary folding of RNA. Dotted lines represent hydrogen bonds. Right: The corresponding tree representation.

Table 1. Reduction of tree sizes for selected tRNA molecules [8]

Name	$ T $	$ \tilde{T} $	Reduction (%)
Athal-chr4.trna25	52	35	33%
cb25.fpc2454.trna15	52	40	23%
CHROMOSOME_I.trna38	51	38	25%
chr1.trna1190	51	34	33%
Acinetobacter_sp_AD1.trna45	55	38	31%
Aquifex_aeolicus.trna21	55	38	31%
Azoarcus_sp_EbN1.trna58	55	38	31%
Bacillus_subtilis.trna63	52	35	33%
Aeropyrum_pernix_K1.trna3	56	39	30%
Sulfolobus_tokodaii.trna25	53	36	32%

v-reduced tree. The last column shows the size reductions in percentage. On average, we observe a size reduction by nearly one third of the original size, which roughly translates into a one-half decrease of running time for the known cover-strategy algorithms.

5 Conclusions

We presented a new method for computing tree edit distance by incorporating structural linearity. This method can work with any existing cover-strategy based algorithm to yield a time complexity of $\mathcal{O}(n^2 + \phi(\mathcal{A}, \tilde{n}))$ where n is the original input size and $\phi(\mathcal{A}, \tilde{n})$ assumes the same form of the time complexity of the given algorithm \mathcal{A} when it is applied to an input size \tilde{n} , with $\tilde{n} \leq n$. The magnitude of \tilde{n} is reversely related to the degree of linearity.

This result would be useful when \tilde{n} is in general substantially smaller than n . Therefore, incorporating our technique in any existing cover-strategy algorithm may yield an improved performance for such situations. One application which can readily benefit from this improvement is RNA secondary structure comparisons in computational biology.

References

1. Chen, W.: New algorithm for ordered tree-to-tree correction problem. *Journal of Algorithms* 40(2), 135–158 (2001)
2. Demaine, E.D., Mozes, S., Rossman, B., Weimann, O.: An optimal decomposition algorithm for tree edit distance. In: *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (To appear)*
3. Dulucq, S., Touzet, H.: Decomposition algorithms for the tree edit distance problem. *Journal of Discrete Algorithms* 3, 448–471 (2005)
4. Klein, P.N.: Computing the edit-distance between unrooted ordered trees. In: *Proceedings of the 6th European Symposium on Algorithms(ESA)*, pp. 91–102 (1998)
5. Moore, P.B.: Structural motifs in RNA. *Annual review of biochemistry* 68, 287–300 (1999)
6. Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. *Journal of Computer and System Sciences* 26, 362–391 (1983)
7. Tai, K.: The tree-to-tree correction problem. *Journal of the Association for Computing Machinery (JACM)* 26(3), 422–433 (1979)
8. Genomic tRNA Database. <http://lowelab.ucsc.edu/gtrnadb/>
9. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. *Journal of the ACM* 21(1), 168–173 (1974)
10. Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing* 18(6), 1245–1262 (1989)

Approximation Algorithms for Reconstructing the Duplication History of Tandem Repeats

Lusheng Wang^{1,*}, Zhanyong Wang¹, and Zhizhong Chen²

¹ Department of Computer Science, City University of Hong Kong, Hong Kong
cswangl@cityu.edu.hk, zhyong@cs.cityu.edu.hk

² Department of Mathematical Sciences, Tokyo Denki University,
Hatoyama Saitama, 350-0394, Japan
cswangl@cityu.edu.hk

Abstract. Tandem repeated regions are closely related to some genetic diseases in human beings. Once a region containing pseudo-periodic repeats is found, it is interesting to study the history of creating the repeats. It is important to reveal the relationship between repeats and genetic diseases. The duplication model has been proposed to describe the history [3,10,11]. We design a polynomial time approximation scheme (PTAS) for the case where the size of the duplication box is 1. Our PTAS is faster than the best known algorithm in [11]. For example, to reach ratio-1.5, our algorithm takes $O(n^5)$ time while the algorithm in [11] takes $O(n^{11})$ time. We also design a ratio-2 approximation algorithm for the case where the size of the duplication box is at most 2. This is the first approximation algorithm with guaranteed ratio for this case.

1 Introduction

The genomes of many species are dominated by short sequences repeated consecutively. It is estimated that over 10% of the human genome, the totality of human genetic information, consists of repeated sequences. About 10-25% of all known proteins have some form of repeated structures ranging from simple homopolymers to multiple duplications of entire globular domains. In some other species, repeated regions can even dominate the whole genome. For example, in the Kangaroo rat (*Dipomys ordii*) more than half of the genome consists of three patterns of repeated regions: AAG (2.4 billion repetitions), TTAGG (2.2 billion repetitions) and ACACAGCGGG (1.2 billion repetitions) [9]. Recent studies show that tandem repeats are closely related with human diseases, including neurodegenerative disorders such as fragile X syndrome, Huntington's disease and spinocerebellar ataxia, and some cancers [12]. These tandem repeats may occur in protein coding regions of genes or non-coding regions. Since the initial discovery of tandem repeats [12], many theories on the biological mechanisms that create and extend tandem repeats have been proposed, e.g., slipped-strand

* Lusheng Wang is supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 1196/03E].

mis-pairing, unequal sister-chromatid exchange and unequal genetic recombination during meiosis. (See [8] for details.) The exact mechanisms responsible for tandem repeat expansion are still controversial. Thus, the study of repeated regions in biological sequences has attracted lots of attentions [4,8,11,10,7,6,5].

The Duplication Model

The model for the duplication history of tandem repeated sequences was proposed by Fitch in 1977 [3] and re-proposed by Tang et al. [10] and Jaitly et al. [11]. The model captures both evolutionary history and the observed order of sequences on a chromosome. Let $S = s_1s_2 \dots s_n$ be an observed string containing n segment s_i for $i = 1, 2, \dots, n$, where each s_i has exactly m letters. Let $r_i r_{i+1} \dots r_{i+k-1}$ be k consecutive segments in an ancestor string of S in the evolutionary history. A duplication event generates $2k$ consecutive segments $lc(r_i)lc(r_{i+1}) \dots lc(r_{i+k-1})rc(r_i)rc(r_{i+1}) \dots rc(r_{i+k-1})$ by (approximately) copying the k segments $r_i r_{i+1} \dots r_{i+k-1}$ twice, where both $lc(r_{i+j})$ and $rc(r_{i+j})$ are approximate copies of r_{i+j} . See Figure 1. Assume that the n segments s_1, s_2, \dots, s_n were formed from a locus by tandem duplications. Then, the locus had grown from a single copy through a series of duplications. A duplication replaces a stretch of DNA containing several repeats with two identical and adjacent copies of itself. If the stretch contains k repeats, the duplication is called a k -duplication.

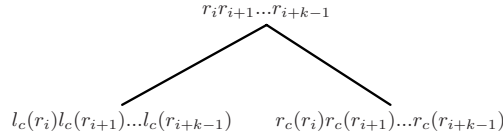


Fig. 1. A duplication event generates consecutive segments $lc(r_i) lc(r_{i+1}) \dots lc(r_{i+k-1})rc(r_i) rc(r_{i+1}) \dots rc(r_{i+k-1})$ by (approximately) copying the k segments $r_i r_{i+1} \dots r_{i+k}$ twice, where both $lc(r_{i+j})$ and $rc(r_{i+j})$ are approximate copies of r_{i+j}

A duplication model M for tandem repeated sequences is a directed graph that contains nodes, edges and blocks. A node in M represents a repeated segment. A directed edge (u, v) from u to v indicates that v is a child of u . A node s is an ancestor of a node t if there is a directed path from s to t . A node that has no outgoing edges is called a leaf; it is labeled with a given segment s_i . A non-leaf node is called an internal node; it has a left child and a right child. The root, which has only outgoing edges, represents the original copy at the locus. A block in M represents a duplication event. Each internal node appears in a unique block; no node is an ancestor of another in a block. If the block corresponds to a k -duplication, then it contains k nodes v_1, v_2, \dots, v_k from left to right. Let $lc(v_i)$ and $rc(v_i)$ be the left and right child of v_i . Then, $lc(v_1), lc(v_2), \dots, lc(v_k), rc(v_1), rc(v_2), \dots, rc(v_k)$ are placed from left to right in the model. Hence, for any i and $j, 1 \leq i < j \leq k$, the edges $(v_i, rc(v_i))$ and $(v_j, lc(v_j))$ cross each other. However, no other edge crosses in the model. For

simplicity, we will only draw a box for a block representing a q -duplication event for $q \geq 2$. Figure 2 gives an example. We also refer q as the size of the duplication box. The cost on each edge in the duplication model is the hamming distance between the two segments associated with the two ends of the edge. The cost of the duplication model is the total cost of all edges in the model.

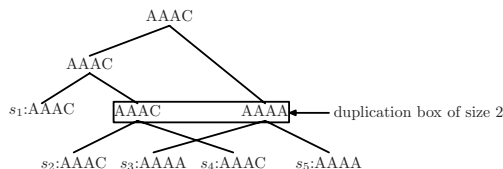


Fig. 2. A duplication model for $S = s_1s_2s_3s_4s_5$, where $s_1 = AAAC$, $s_2 = AAAC$, $s_3 = AAAA$, $s_4 = AAAC$, and $s_5 = AAAA$

Reconstructing the Duplication History: Given $S = s_1, s_2, \dots, s_n$, find a duplication model with the smallest cost.

In this paper, we only consider the cases, where the size of the duplication box is 1 or at most 2. When the size of the duplication box is 1, we design a polynomial time approximation scheme (PTAS). For ratio $1 + \frac{2^t}{t2^t - 2q}$, the running time is $O(n^{k+1}(f(k) + g(k)))$, where $k = 2^t - q$, and $f(k)$ and $g(k)$ are constant for any constant k . Our PTAS is faster than the best known algorithm in [11]. For example, to reach ratio-1.5, our algorithm takes $O(n^5)$ time while the algorithm in [11] takes $O(n^{11})$ time. We also design a ratio-2 approximation algorithm for the case where the size of the duplication box is at most 2. This is the first approximation algorithm with guaranteed ratio for this case.

2 The PTAS When the Size of the Duplication Box is 1

A *full phylogeny* is a phylogeny in which all internal nodes are associated with sequences. Any sequence in $\{s_1, s_2, \dots, s_n\}$ is called a *leaf sequence*. A sequence of length m is a *non-leaf sequence* if it is not in $\{s_1, s_2, \dots, s_n\}$. A *real full k -phylogeny* is a full phylogeny with k leaves, where each internal node except possibly the root is assigned a non-leaf sequence.

Consider a full phylogeny T , where some of the internal nodes are assigned some leaf sequences. The full phylogeny T can be decomposed into a set of small real full phylogenies. The roots of those (except the one on top) small real full phylogenies are leaf sequences and they are connected via identical leaf sequences. (See Figure 3.)

A phylogeny for n terminals is called a k -size phylogeny if the sizes of all its real full phylogenies are *at most* k . The basic idea of the approximation algorithm is to assign leaf sequences to some internal nodes so that the whole phylogeny is decomposed into a set of small phylogenies and we do local optimization for each small phylogeny.

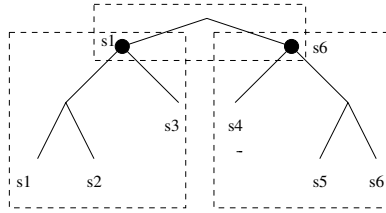


Fig. 3. A full phylogeny is decomposed into a set of three small real full phylogenies. Each rectangle contains a small real full phylogeny.

Let T be a binary tree and P a path in T . We use $c(T)$ and $c(P)$ to represent their costs. Let T_{opt} be the optimal phylogeny for s_1, s_2, \dots, s_n .

2.1 The Ratio for $k = 2^t$

Now, we want to show that there exists a 2^t -size phylogeny T for s_1, s_2, \dots, s_n such that the cost $c(T)$ is at most $(1 + \frac{1}{t})c(T_{opt})$. In order to prove the theorem, we need some definitions and a preliminary lemma. Let T_{opt} be the optimal phylogeny for s_1, s_2, \dots, s_n . Starting from the root, a *counter-clockwise walk* along the outside of the optimal tree T_{opt} travels through all the edges twice, one in each direction, and comes back to the root. The cost of the clockwise walk is twice of the optimal, i.e., $2c(T_{opt})$.

Consider an internal node v in T_{opt} . We use $l(v)$ and $r(v)$ to represent the left most and right most decedent leaves of v in T_{opt} , respectively. Let $in(T)$ be the set of all internal nodes other than the root T .

For each internal node $u \in in(T_{opt})$, there is a *connecting path* $P(u)$ such that it is from $r(u)$ to u in the counter-clockwise walk if u is the left child of its parent, or it is from u to $l(u)$ if u is the right child of its parent. (See Figure 4.)

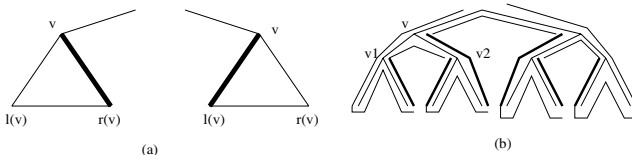


Fig. 4. (a) The connecting paths of v . (b) The walk and connecting paths.

The total length of all connecting paths in the clockwise walk is at most $c(T_{opt})$. Formally, we have

Lemma 1

$$\sum_{u \in in(T_{opt})} c(P(u)) \leq c(T_{opt}).$$

Proof. The total length of the clockwise is $2c(T_{opt})$. Let v_1 and v_2 be the two children of v . The path

$$P : r(v_1), \dots, v_1, v, v_2, \dots, l(v_2)$$

in the clockwise walk of T_{opt} connects the two subtrees $T_{opt}(v_1)$ and $T_{opt}(v_2)$. Deleting the two edges (v_1, v) and (v, v_2) , we have the two connecting paths $P(v_1)$ and $P(v_2)$, where path $P(v_1)$ in the clockwise walk is from $r(v_1)$ to v_1 and $P(v_2)$ is from v_2 to $l(v_2)$. Note that, for each node v in T_{opt} , there are two edges (v, v_1) and (v, v_2) in T_{opt} .

$$\sum_{v \in in(T_{opt}) \cup r} d(v, v_1) + d(v, v_2) = c(T_{opt}). \tag{1}$$

By \square , the total length of all the deleted edges in the clockwise walk is $c(T_{opt})$. Therefore, the total length of the connecting paths is at most $2c(T_{opt}) - c(T_{opt}) \leq c(T_{opt})$. \square

Theorem 1. *There exists a 2^t -size phylogeny T for s_1, s_2, \dots, s_n such that the cost $c(T)$ is at most $(1 + \frac{1}{t})c(T_{opt})$.*

Proof. Let T_{opt} be an optimal tree. Let V_i be a set containing all nodes at level i in T_{opt} . We partition the nodes of T_{opt} into t groups G_0, G_1, \dots, G_{t-1} , where

$$G_i = \bigcup_{j \equiv i \pmod t} V_j.$$

For any node v in G_i ($i = 0, 1, \dots, t - 1$), define $T_{opt}(v, t)$ to be a subtree of T_{opt} rooted at v containing $t + 1$ levels of nodes. If v is at level k , then all the leaves in $T_{opt}(v, t)$ are at level $k + t$. Given a subtree $T_{opt}(v, t)$, we can obtain a full 2^t -phylogeny $ST_{opt}(v, t)$, where 2^t is the number of leaves in $T_{opt}(v, t)$, as follows: (1) We replace the sequence associated with every leaf u in $T_{opt}(v, t)$ with the sequence that is on the leaf $r(u)$ if u is the left child of its parent, or on the leaf $l(u)$ if u is the right child of its parent. (Note that u is an internal node in T_{opt} and $l(u)$ and $r(u)$ are leaves in T_{opt} .) (2) any other nodes including v in $ST_{opt}(v, t)$ are assigned the sequence as in T_{opt} .

Let r be the root of T_{opt} . The set of all subtrees $ST_{opt}(r, i)$ and $ST_{opt}(v, t)$ for $v \in G_i$, forms a 2^t -size full phylogeny $ST_{opt}[i, t]$ for s_1, s_2, \dots, s_n . Here each internal node $u \in G_i$ appears as a leaf of some $ST_{opt}(v, t)$ once, and appears as the root of the subtree $ST_{opt}(u, t)$ once. The full 2^t -phylogenies $ST_{opt}(v, t)$ for $v \in G_i$ are connected via the nodes associated with common sequences s_q 's. Figure 5 gives an example for $t = 2$. The left to right linear order among the leaves in T_{opt} is still kept in $ST_{opt}[i, t]$.

Consider the cost $c(ST_{opt}[i, t])$ of $ST_{opt}[i, t]$.

$$c(ST_{opt}[i, t]) \leq c(T_{opt}) + \sum_{v \in G_{i-r}} c(P(v)),$$

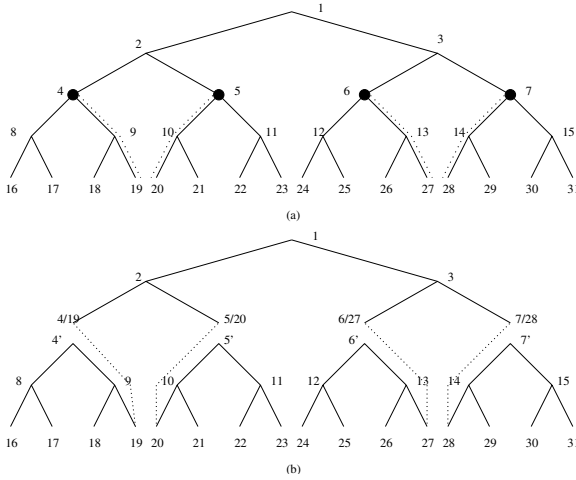


Fig. 5. (a) The tree T_{opt} . The dark nodes are in G_0 . (b) The tree $ST_{opt}(i, t)$ for $t = 2$ and $i = 0$. The connection of subtrees $ST_{opt}(v, t)$ is via those nodes sharing common sequences. For example, the subtree $ST_{opt}(4, 2)$ is connected with the subtree $ST_{opt}(1, 2)$ with the sequence on node 19.

where the extra cost $\sum_{v \in G_{i-r}} c(P(v))$ are the costs for those connecting paths $P(v)$ for $v \in G_i$. The total cost of the t trees $ST(0, t), ST(2, t), \dots, ST(t-1, t)$ is

$$\sum_{i=0}^{t-1} c(ST_{opt}(i, t)) \leq t \cdot c(T_{opt}) + \sum_{i=0}^{t-1} \sum_{v \in G_{i-r}} c(P(v)) \leq (t+1) \cdot c(T_{opt}). \quad (2)$$

From (4), the cost of t trees is at most $(t+1) \cdot c(T_{opt})$. Thus, we know that there exists a $ST_{opt}(i, t)$ whose cost $c(ST_{opt}(i, t))$ is at most $\frac{t+1}{t} c(T_{opt}) = (1 + \frac{1}{t}) c(T_{opt})$. This completes the proof of Theorem 1. \square

The Algorithm

Now, we describe our algorithm. Consider a fixed integer t . Let $D(i, j, k)$ be the cost of a minimum cost k -tree $T(i, j, k)$ for segment s_i, s_{i+1}, \dots, s_j , where $i \leq j$, such that (1) the leaves are labeled, from left to right, by s_i, s_{i+1}, \dots, s_j ; and (2) the top of the tree is a k -tree.

Consider the computation of $D(i, j, k)$. The top of the tree is a k -tree that divides the consecutive segments s_i, s_{i+1}, \dots, s_j , denoted as (i, j) , into k parts, $(k_1, k_2 - 1), (k_2, k_3 - 1), \dots, (k_{k-1}, k_k - 1), (k_k, k_{k+1})$, where $k_1 = i$ and $k_{k+1} = j$.

Consider the k -tree with topology T_{top} at the top of $T(i, j, t)$. The k leaves of T_{top} are $s_{i_1}, s_{i_2}, \dots, s_{i_k}$, where s_{i_j} is either k_j of $k_{j+1} - 1$ depending on the topology T_{top} , i.e., if leaf j is the left child of its parent in T_{top} , then $i_j = k_{j+1} - 1$; if leaf j is the right child of its parent, then $i_j = k_j$. Let $c(s_{i_1}, s_{i_2}, \dots, s_{i_k}, T_{top})$ be the cost the k -tree at the top with topology T_{top} such that s_{i_j} is assigned to the j -th leaf of T_{top} .

$$D(i, j, k) = c(s_{i_1}, s_{i_2}, \dots, s_{i_k}, T_{top}) + \sum_{j=1}^k D(k_j, k_{j+1} - 1, k). \tag{3}$$

In fact, equation (3) is for the ideal case, where the top topology T_{top} is a tree F whose leaves are all at the same level and the number of leaves is k . We have to consider the degenerated cases, where the top topology T_{top} becomes a tree obtained by deleting some subtrees of F . In that case, we have to have a formula corresponding to (3), where if a subtree in F is deleted, we have to merge some consecutive regions $(k_j, k_{j+1} - 1), \dots, (k_{jj}, k_{jj+1} - 1)$ corresponding to the leaves of the subtree into one region $(s_j, s_{jj+1} - 1)$. To compute $D(i, j, k)$, we have to consider all the degenerated case. The number of all degenerated cases is upper bounded by $2^{2^t - 1}$ since there are $2^t - 1$ internal nodes in F and each node may become a leaf in the degenerated topology.

It follows that $D(1, n, 2^t)$ is at most $(1 + \frac{1}{t})$ times of the optimum.

Computing $D(i, j, k)$ needs to know $k - 1$ breaking points in (i, j) . Thus, for each $D(i, j, k)$ the time required is $O(n^{k-1}(f(k) + g(k)))$, where $f(k)$ is the total number of possible degenerated cases upper bounded by 2^{k-1} and $g(k)$ is the time required for computing $c(s_{i_1}, s_{i_2}, \dots, s_{i_k}, Top)$. Since there are $O(n^2)$ $D(i, j, t)$'s for a fixed t , the total time required for the algorithm is $O(n^{k+1}(f(k) + g(k)))$.

Theorem 2. *The algorithm is a PTAS with ratio $1 + \frac{1}{t}$ and runs in $O(n^{2^t+1}(f(2^t) + g(2^t)))$ time.*

For a ratio-1.5 algorithm, the running time of our approach is $O(n^5(f(2) + g(2)))$, where the old algorithm in [11] requires $O(n^{11} + n^5g(r))$.

2.2 The Ratio for k -Size Phylogenies

For any $k > 1$, k can be written as $k = 2^t - q$ for some $t \geq 2$ and $0 \leq q \leq 2^t - 1$.

Theorem 3. *The algorithm is a PTAS with ratio $1 + \frac{2^t}{t2^t - 2q}$ and runs in $O(n^{k+1}(f(k) + g(k)))$ time.*

To reach ratio 1.67, we needs $O(n^4)$ time, while the algorithm in [11] needs $O(n^7)$ time.

3 A Ratio-2 Algorithm When the Size of the Duplication Box ≤ 2

In this section, we consider the case where each block in a duplication model has size at most 2. We give a ratio-2 algorithm for this case.

Let T be a minimum cost tree fitting a duplication model for segments s_1, s_2, \dots, s_n . Consider the counter-clockwise walk of T starting from s_1 and ending with s_n . In this walk, if we ignore all the internal nodes and directly connect the leaves (segments) according to the order in the walk, we get a *spanning*

path for T . Obviously, if the size of all the duplication boxes is 1, then the spanning path is $s_1, s_2, \dots, s_i, s_{i+1}, \dots, s_n$. However, if there are size-2 duplication boxes, the order will be different. An arbitrary order of the n segments may not admit a duplication model and thus may not be a spanning path. Zhang et. al. gives an algorithm to test if a given order of the segments can fit a duplication model [4].

Let T_{opt} be the optimal (minimum cost) tree for all possible duplication models. SP_{opt} is the spanning path for T_{opt} .

Lemma 2. *The cost of the spanning path SP_{opt} is at most twice of that for T_{opt} .*

From Lemma 2, if we can compute SP_{opt} , then we get a ratio-2 algorithm. However, SP_{opt} is defined from the optimal solution T_{opt} and is hard to compute. Instead, we will compute a spanning path with minimum cost among all possible duplication models for $S = s_1 s_2 \dots s_n$. Note that if the size of all duplication boxes is 1, then for any range $[i, j]$, the set of segments s_k 's with $k \in [i, j]$ forms one sub-path of the spanning path. The key idea for computing a spanning path with minimum cost is based on the observation that the spanning path for a duplication model can be decomposed into a set of ranges $[i, j]$ such that for each range $[i, j]$, all the segments s_k 's with $k \in [i, j]$ form at most two sub-paths of the spanning path.

Decomposition of the Spanning Path

We give a decomposition of the spanning path that will be used for the computation of the optimal spanning path. Let T be the tree for a duplication model. In the decomposition, each internal node in T corresponds to a sub-path. A sub-path is *complete* if the sub-path contains all the segments s_k with $k \in [i, j]$, where i is the smallest index and j is the largest index in the sub-path. Two sub-paths form a *complete pair* if the two sub-paths contain all the segments s_k with $k \in [i, j]$, where i is the smallest sub-index, j is the largest sub-index in the two sub-paths and one path is from s_i to $s_{j'}$ and the other path is from $s_{i'}$ to s_j with $i < i' < j' < j$.

Let us consider the duplication boxes at the bottom of T . For a size-1 box at the bottom, it corresponds to a sub-path (actually an edge) s_i and s_{i+1} . For a size-2 box at the bottom, it corresponds to a complete pair (actually two edges), (s_i, s_{i+2}) and (s_{i+1}, s_{i+3}) . For simplicity, we use $[s_i, s_j]$ to represent a complete sub-path from s_i to s_j and use $[s_i, s_j, s_k, s_l]$ to indicate a complete pair containing a sub-path from s_i to s_k and a sub-path from s_j to s_l .

Lemma 3. *Each internal node in a size-1 box of the duplication model corresponds to a complete path.*

Lemma 4. *Each size-2 box in the duplication model corresponds to a complete pair of paths.*

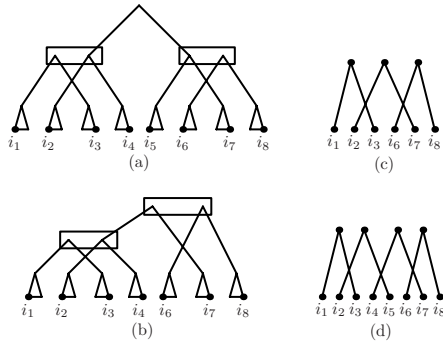


Fig. 6. Cases (a) to (d)

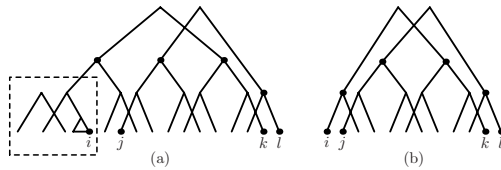


Fig. 7. Cases (a) There are more than four nodes that are co-related. (b) We ‘cut’ at node i . The component shown in the dashed part in (a) and the component in (b) share the common node i .

Let $E[i, j]$ be the minimum cost for all complete paths $[s_i, s_j]$. Let $D[i, i_2, i_3, j]$ denote the minimum cost for all complete pairs of paths $[s_i, s_{i_2}, s_{i_3}, s_j]$. Let $d(s_i, s_j)$ be the distance between the two segments s_i and s_j .

Lemma 5

$$E[i, i] = 0; E[i, i + 1] = d(s_i, s_{i+1}). \tag{4}$$

$$E[i, j] = \min\left\{ \min_{i < i' \leq j} d(s_{i'}, s_{i'+1}) + E[i, i'] + E[i' + 1, j], \right. \\ \left. \min_{i \leq i_2 < i_3 \leq j} D[i, i_2, i_3, j] + d(s_{i_2}, s_{i_3}) \right\}. \tag{5}$$

Two size-2 boxes could be connected via a node and form a component containing three co-related nodes as shown in Figure 6 (a) and (b). The resulting configuration is shown in Figure 6 (c), where we use $[s_{i_1}, s_{i_2}, s_{i_3}, s_{i_6}, s_{i_7}, s_{i_8}]$ to denote the component and $F[i_1, i_2, i_3, i_6, i_7, i_8]$ to denote its cost. This case can be further extended to have four co-related nodes as shown in Figure 6 (d). We use $[s_{i_1}, s_{i_2}, s_{i_3}, s_{i_4}, s_{i_5}, s_{i_6}, s_{i_7}, s_{i_8}]$ to denote the component and $G[i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8]$ to denote its cost. There is no need to consider further extension of Figure 6 (d) since a size-2 duplication event contains at most four nodes. If there are more than four nodes that are co-related, we can ‘cut’ the component such that it has four co-related nodes. (See Figure 7) Thus, in order to get Lemma 6 there are ten cases that we have to consider as shown in Figure 8.

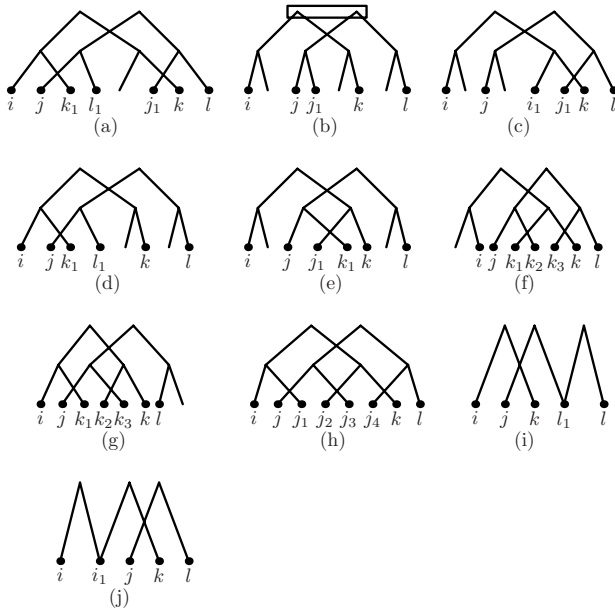


Fig. 8. Cases (a) to (j)

Besides, we can show that

$$\begin{aligned}
 F[i_1, i_2, i_3, i_6, i_7, i_8] &= \min_{i_3 < i_4 < i_6} \{ (E[i_1, i_2, i_3, i_4] + E[i_4, i_6, i_7, i_8]), \\
 &\quad (E[i_1, i_2, i_3, i_4] + E[i_4 + 1, i_6, i_7, i_8] + d(s_{i_4}, s_{i_4+1})) \} \text{ and} \\
 G[i, j, j_1, j_2, j_3, j_4, k, l] &= \min_{j_1 < l_1 < j_2} \{ (E[i, j, j_1, l_1] + F[l_1, j_2, j_3, j_4, k, l]), \\
 &\quad (E[i, j, j_1, l_1] + F[l_1 + 1, j_2, j_3, j_4, k, l] + d(s_{l_1}, s_{l_1+1})) \}.
 \end{aligned}$$

Lemma 6. $D[i, j, k, l]$ is the minimum of

$$\left\{ \begin{aligned}
 &\min_{j < k_1 < l_1 < j_1 < k} (D[i, j, k_1, l_1] + D[l_1 + 1, j_1, k, l] + d(s_{k_1}, s_{l_1+1}) \\
 &\quad + d(s_{l_1}, s_{j_1})) \\
 &\min_{j < j_1 < k} (E[i, j - 1] + E[j, j_1] + E[j_1 + 1, k] + E[k + 1, l] \\
 &\quad + d(s_{j-1}, s_{j_1+1}) + d(s_{j_1}, s_{k+1})) \\
 &\min_{j < i_1 < j_1 < k} (E[i, j - 1] + E[j, i_1 - 1] + D[i_1, j_1, k, l] + d(s_{j-1}, s_{i_1}) \\
 &\quad + d(s_{i_1-1}, s_{j_1})) \\
 &\min_{j < k_1 < l_1 < k} (D[i, j, k_1, l_1] + E[l_1 + 1, k] + E[k + 1, l] + d(s_{k_1}, s_{l_1-1}) + \\
 &\quad d(s_{l_1}, s_{k+1})) \\
 &\min_{j < j_1 < k_1 < k} (E[i, j - 1] + D[j, j_1, k_1, k] + E[k + 1, l] + d(s_{j-1}, s_{j_1}) \\
 &\quad + d(s_{k_1}, s_{k+1})) \\
 &\min_{i+1 < k_1 < k_2 < k_3 < k} F[i + 1, k_1, k_2, k_3, k, l] + d(s_i, s_{k_1}) + d(s_{k_2}, s_{k_3}), \quad (j = i + 1) \\
 &\min_{j < k_1 < k_2 < k_3 < k} F[i, j, k_1, k_2, k_3, k] + d(s_{k_1}, s_{k_2}) + d(s_{k_3}, s_l), \quad (l = k + 1) \\
 &\min_{j < j_1 < j_2 < j_3 < j_4 < k} (G[i, j, j_1, j_2, j_3, j_4, k, l] + d(s_{j_1}, s_{j_2}) + d(s_{j_3}, s_{j_4})). \\
 &\min_{k < l_1 < l} D[i, j, k, l_1] + E[l_1, l] \\
 &\min_{i < i_1 < j} E[i, i_1] + D[i_1, j, k, l]
 \end{aligned} \right.$$

References

1. Boby, T., Patch, A.-M., Aves, S.J.: TRbase: a database relating tandem repeats to disease genes for the human genome. *Bioinformatics*, Advance Access published on October 12, 2004
2. Subramanian, S., Mishra, R.K., Singh, L.: Genome-wide analysis of Bkm sequences (GATA repeats): predominant association with sex chromosomes and potential role in higher order chromatin organization and function. *Bioinformatics* 19, 681–685 (2003)
3. Fitch, W.: Phylogenies constrained by cross-over process as illustrated by human hemoglobins in a thirteen cycle, eleven amino-acid repeat in human apolipoprotein A-I. *Genetics* 86, 623C644 (1977)
4. Zhang, L., Ma, B., Wang, L., Xu, Y.: Greedy method for inferring tandem duplication history. *Bioinformatics* 19, 1497–1504 (2003)
5. Otu, H.H., Sayood, K.: A new sequence distance measure for phylogenetic tree construction. *Bioinformatics* 19, 2122–2130 (2003)
6. Macas, J., Mszros, T., Nouzov, M.: PlantSat: a specialized database for plant satellite repeats. *Bioinformatics* 18, 28–35 (2002)
7. Lillo, F., Basile, S., Mantegna, R.N.: Comparative genomics study of inverted repeats in bacteria. *Bioinformatics* 18, 971–979 (2002)
8. Benson, G., Dong, L.: Reconstructing the duplication history of a tandem repeat. In: *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology (ISMB-99)*, pp. 44–53 (1999)
9. Widegren, B., Arnason, U., Akusjarvi, G.: Characteristics of a conserved 1,579-bp highly repetitive component in the killer whale, *Ornicus orca*. *Mol. Biol. Evol.* 2, 411–419 (1985)
10. Tang, M., Waterman, M.S., Yooseph, S.: Zinc Finger Gene Clusters and Tandem Gene Duplication. *Journal of Computational Biology* 9(2), 429–446 (2002)
11. Jaitly, D., Kearney, P.E., Lin, G.-H., Ma, B.: Methods for reconstructing the history of tandem repeats and their application to the human genome. *J. Comput. Syst. Sci.* 65(3), 494–507 (2002)
12. Wyman, A.H., White, R.: A highly polymorphic locus in human DNA. *Proc. Natl. Acad. Sci.* 77, 6745–6758 (1980)

Priority Algorithms for the Subset-Sum Problem

Yuli Ye and Allan Borodin

Department of Computer Science
University of Toronto
Toronto, ON, Canada M5S 3G4
{y3ye,bor}@cs.toronto.edu

Abstract. Greedy algorithms are simple, but their relative power is not well understood. The priority framework [5] captures a key notion of “greediness” in the sense that it processes (in some locally optimal manner) one data item at a time, depending on and only on the current knowledge of the input. This algorithmic model provides a tool to assess the computational power and limitations of greedy algorithms, especially in terms of their approximability. In this paper, we study priority algorithm approximation ratios for the Subset-Sum Problem, focusing on the power of revocable decisions. We first provide a tight bound of $\alpha \approx 0.657$ for irrevocable priority algorithms. We then show that the approximation ratio of fixed order revocable priority algorithms is between $\beta \approx 0.780$ and $\gamma \approx 0.852$, and the ratio of adaptive order revocable priority algorithms is between 0.8 and $\delta \approx 0.893$.

1 Introduction

Greedy algorithms are of great interest because of their simplicity and efficiency. In many cases they produce reasonable (and sometimes optimal) solutions. Surprisingly, it is not obvious how to formalize the concept of a greedy algorithm and given such a formalism how to determine its power and limitations with regard to natural combinatorial optimization problems. Borodin, Nielson and Rackoff [5] suggested the priority model which provides a rigorous framework to analyze greedy-like algorithms. In this framework, they define fixed order and adaptive (order) priority algorithms, both of which capture a key notion of greedy algorithms in the sense that they process one data item at a time. For fixed order priority, the ordering function used to evaluate the priority of a data item is fixed before execution of the algorithm, while for adaptive priority, the ordering function can change during every iteration of the algorithm. By restricting algorithms to this framework, approximability results and limitations¹ for many problems have been obtained; for example, scheduling problems [5,18], facility

¹ We note that similar to the study of online competitive analysis, negative priority results are in some sense incomparable with hardness of approximation results as there are no explicit complexity considerations as to how a priority algorithm can choose its next item and how it decides what to do with that item. Negative results are derived from the structure of the algorithm.

location and set cover [1], job interval selection (JISP and WJISP) [12], and various graph problems [6,9]. The original priority framework specified that decisions (being made for the current input item) are irrevocable. Even within this restrictive framework, the gap between the best known algorithm and provable negative remains significant for most problems. Following [10,4], Horn [12] extended the priority framework to allow revocable acceptances when considering packing problems. That is, input items could be accepted and then later rejected, the only restriction being that a feasible solution is maintained at the end of each iteration. The revocable (decision) priority model is intuitively more powerful and almost as conceptually simple as the irrevocable model and it is perhaps surprising that it is not a more commonly used type of algorithm. Erlebach and Spieksma [10] and independently Bar-Noy et al. [4] provide a simple revocable priority approximation algorithm for the WJISP problem, and Horn [12] formalizes this model and provides an approximation upper bound² of $\approx 1/(1.17)$ for the special case of the weighted interval scheduling problem. Moore's [17] optimal "greedy algorithm" for the unweighted throughput maximization problem without release times (i.e. $1 \parallel \sum_j \bar{U}_j$ in Graham's scheduling notation) can be implemented as a fixed order revocable priority algorithm. It is not difficult to show that this problem cannot be solved optimally by an irrevocable priority algorithm.

The *Subset-Sum Problem* (SSP) is one the most fundamental NP-complete problems [1], and perhaps the simplest of its kind. Approximation algorithms for SSP have been studied extensively in the literature. The first FPTAS (for the more general knapsack problem) is due to Ibarra and Kim [13], and the best current approximation algorithm is due to Kellerer et al. [15], having time and space complexity $O(\min\{\frac{n}{\epsilon}, n + \frac{1}{\epsilon^2} \log \frac{1}{\epsilon}\})$ and $O(n + \frac{1}{\epsilon})$ respectively. Greedy-like approximation algorithms have also been studied for SSP; an algorithm called greedy but using multiple passes, has approximation ratio 0.75, see [16]. In this paper, we study priority algorithms for SSP. Although in some sense one may consider SSP to be a "solved problem", the problem still presents an interesting challenge for the study of greedy algorithms. We believe the ideas employed for SSP will be applicable to the study of simple algorithms for other (say scheduling) problems which are not well understood, such as the throughput maximization problem (with release times) and some of its more tractable subcases. In particular, can we derive priority approximation algorithms for throughput maximization when all jobs have a fixed processing time (i.e. $1 \parallel r_j, p_j = p \mid \sum_j w_j \bar{U}_j$)? (We note that Horn's [12] $1/(1.17)$ bound also applies to this problem.) Baptiste [3] optimally solves this special case of throughput maximization using a dynamic programming algorithm with time complexity $O(n^7)$. (See also Chuzhoy et al. [8] and Chrobak et al. [7] for additional throughput maximization results.)

In spite of the conceptual simplicity of the SSP problem and the priority framework, there is still a great deal of flexibility in how one can design algorithms, both in terms of the ordering and in terms of which items to accept and

² As we are considering maximization problems in this paper, all approximation ratios will be ≤ 1 so that negative results become upper bounds on the ratio.

(for the revocable model) which items to discard in order to fit in a new item. We give a tight bound of $\alpha \approx 0.657$ for irrevocable priority algorithms showing that in this case adaptive ordering does not help. For fixed order revocable algorithms, we can show that the best approximation ratio is between $\beta \approx 0.780$ and $\gamma \approx 0.852$; for adaptive revocable priority algorithms, the best approximation ratio is between 0.8 and $\delta \approx 0.893$. All omitted proofs can be found in [19].

2 Definitions and Notation

We use **bold** font letters to denote sets of data items. For a given set \mathbf{R} of data items, we use $|\mathbf{R}|$ to denote its cardinality and $\|\mathbf{R}\|$, its total weight. For a data item u , we use u to represent the singleton set $\{u\}$ and $2u$, the multi-set $\{u, u\}$; we also use u to represent the weight of u since it is the only attribute. The term u here is an overloaded term, but the meaning will become clear in the actual context. For set operations, we use \oplus to denote set addition, and use \ominus to denote set subtraction.

2.1 The Subset-Sum Problem

Given a set of n data items with positive weights and a capacity c , the *maximization version* of SSP is to find a subset such that the corresponding total weight is maximized without exceeding the capacity c . Without loss of generality, we make two assumptions. First of all, the weights are all scaled to their relative values to the capacity; hence we can use 1 instead of c for the capacity. Secondly, we assume each data item has weight $\in (0, 1]$. An instance of SSP is a set $\mathbf{I} = \{u_1, u_2, \dots, u_n\}$ of n data items, where the set \mathbf{I} is the *input set*, and u_1, u_2, \dots, u_n are the *data items*. A *feasible* solution is a subset \mathbf{B} of \mathbf{I} such that $\|\mathbf{B}\| \leq 1$. An *optimal* solution is a feasible solution with maximum weight. Let \mathcal{A} be an algorithm for SSP, we denote \mathbf{ALG} the solution achieved by \mathcal{A} and \mathbf{OPT} , the optimal solution, then the approximation ratio of \mathcal{A} on that instance is denoted by $\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|}$. The *approximation ratio* of \mathcal{A} is the infimum of the set of ratios achieved by \mathcal{A} over all instances of SSP.

2.2 Priority Model

We base our terminology and model on that of [5], and start with the class of fixed order irrevocable priority algorithms for SSP. For a given instance, a *fixed order irrevocable* priority algorithm maintains a feasible solution \mathbf{B} throughout the algorithm. The structure of the algorithm³ is as follows:

³ We formalize the allowable (fixed) orderings by saying that the algorithm specifies a total ordering on all possible input items. The items that constitute the actual input set \mathbf{I} will then inherit this ordering. That is, the priority model insists that the ordering satisfies Arrow's Independence of Irrelevant Attributes IIA Axiom [2]. For adaptive orderings the algorithm can construct a new IIA ordering based on all the items that it has already seen as well as those items it can deduce are not in the input set.

FIXED ORDER IRREVOCABLE PRIORITY

Ordering: Determine a total ordering of all possible data items

while \mathbf{I} is not empty

$next :=$ index of the data item in \mathbf{I} that comes first in the ordering

Decision: Decide whether or not to add u_{next} to \mathbf{B} , and then remove u_{next} from \mathbf{I}

end while

An *adaptive irrevocable* priority algorithm is similar to a fixed order one, but instead of looking at a data item according to some pre-determined ordering, the algorithm is allowed to reorder the remaining data items in \mathbf{I} at each iteration. This gives the algorithm an advantage since now it can take into account the information that has been revealed so far to determine which is the best data item to consider next. The structure of an adaptive irrevocable priority algorithm is described as follows:

ADAPTIVE IRREVOCABLE PRIORITY

while \mathbf{I} is not empty

Ordering: Determine a total ordering of all possible (remaining) data items

$next :=$ index of the data item in \mathbf{I} that comes first in the ordering

Decision: Decide whether or not to add u_{next} to \mathbf{B} , and then remove u_{next} from \mathbf{I}

end while

The above defined priority algorithms are “irrevocable” in the sense that once a data item is admitted to the solution it cannot be removed. We can extend our notion of “fixed order” and “adaptive” to the class of revocable priority algorithms, where revocable decisions on accepted data items are allowed. Accordingly, those algorithms are called *fixed order revocable* and *adaptive revocable* priority algorithms respectively. The extension⁴ to revocable acceptances provides additional power; for example, as shown in [14], *online irrevocable* algorithms for SSP cannot achieve any constant bound approximation ratio, while *online revocable* algorithms can achieve a tight approximation ratio of $\frac{\sqrt{5}-1}{2} \approx 0.618$.

2.3 Adversarial Strategy

We utilize an adversary in proving approximation bounds. For a given priority algorithm, we run the adversary against the algorithm in the following scheme. At the beginning of the algorithm, the adversary first presents a set of data items to the algorithm, possibly with some data items having the same weight. Furthermore, our adversary promises that the actual input is contained in this set⁵. Since weight is the only input parameter, the algorithm give the same pri-

⁴ This extension applies to priority algorithms for packing problems.

⁵ This assumption is optional. The approximation bounds clearly hold for a stronger adversary.

ority to all items having the same weight⁶. At each step, the adversary asks the algorithm to select one data item in the remaining set and make a decision on that data item. Once the algorithm makes a decision on the selected item, the adversary then has the power to remove any number of data items in the remaining set; this repeats until the remaining set is empty, which then terminates the algorithm.

For convenience, we often use a diagram to illustrate an adversarial strategy. A diagram of an adversarial strategy is an acyclic directed graph, where each node represents a possible state of the strategy, and each arc indicates a possible transition. Each state of the strategy contains two boxes. The first box indicates the current solution maintained by the algorithm, the second box indicates the remaining set of data items maintained by the adversary. A state can be either terminal or non-terminal. A state is *terminal* if and only if it is a sink, in the sense that the adversary no longer need perform any additional action; we indicate a terminal state using **bold** boxes. Each transition also contains two lines of actions. The first line indicates the action taken by the algorithm and the second line indicates the action taken by the adversary. Sometimes the algorithm may need to reject certain data items in order to accept a new one, so an action may contain multiple operations which occur at the same time; we use \oslash if there is no action. Note that to calculate a bound for the approximation ratio of an algorithm, it is sufficient to consider the approximation ratios achieved in all terminal states. We will see such diagrams in Sect. 3.

3 Priority Algorithms and Approximation Bounds

We first define four constants that will be used for our results. Let α , β , γ and δ be the real roots (respectively) of the equations $2x^3 + x^2 - 1 = 0$, $2x^2 + x - 2 = 0$, $10x^2 - 5x - 3 = 0$ and $6x^2 - 2x - 3 = 0$ between 0 and 1. The corresponding values are shown in Table 1.

Table 1. Corresponding values

name	α	β	γ	δ
value	≈ 0.657	≈ 0.780	≈ 0.852	≈ 0.893

We now give a tight bound for irrevocable priority algorithms. It is interesting that there is no approximability difference between fixed order and adaptive irrevocable priority algorithms.

Theorem 1. *There is a fixed order irrevocable priority algorithm for SSP with approximation ratio α , and every irrevocable priority algorithm for SSP has approximation ratio at most α .*

⁶ Technically we can use an item number identifier to further distinguish items, but by providing sufficiently many copies of an item the adversary can effectively achieve what the statement claims.

The case for revocable priority algorithms is more interesting. The ability to make revocable acceptances gives the algorithm a certain flexibility to “regret”. The data items admitted into the solution are never “safe” until the termination of the algorithm. Therefore, if there is enough “room”, it never hurts to accept a data item no matter how “bad” it is, as we can always reject it later at any time and with no cost. For the rest of the paper, we assume our algorithms will take advantage of this property. We start with fixed order revocable priority algorithm by giving two tight bounds for non-increasing order (i.e. items are ordered so that $u_1 \geq u_2 \dots \geq u_n$) and non-decreasing order revocable priority algorithms.

Theorem 2. *There is a non-increasing order revocable priority algorithm for SSP that has approximation ratio α , and every such algorithm has approximation ratio at most α . (Note that the simple ordering here is different from the fixed order irrevocable algorithm of Theorem 1.)*

Theorem 3. *There is a non-decreasing order revocable priority algorithm for SSP that has approximation ratio β , and every such algorithm has approximation ratio at most β .*

The improvement using non-decreasing order is perhaps counter-intuitive 1 as one might think it is more flexible to fill in with small items at the end. Next, we give a approximation bound for any fixed order revocable priority algorithm; this exhibits the first approximation gap we are unable to close. The technique used in the proof is based on a chain of possible item priorities. It turns out, in order to achieve certain approximation ratio, some data items must be placed before some other data items. This order relation is transitive and therefore, has to be acyclic.

Theorem 4. *No fixed order revocable priority algorithm of SSP can achieve approximation ratio better than γ .*

Proof. Let $u_1 = 0.2$, $u_2 = \frac{1}{2}\gamma - \frac{1}{10} \approx 0.326$, $u_3 = 0.5$, and $u_4 = 0.8$. For a data item u , we denote by $rank(u)$ its priority. There are four cases:

1. If $rank(u_4) > rank(u_3)$, then the adversarial strategy is shown in Fig. 1. If the algorithm terminates via state s_1 , then

$$\rho = \frac{\|ALG\|}{\|OPT\|} = \frac{u_3}{u_4} < \gamma.$$

If the algorithm terminates via state s_2 , then

$$\rho = \frac{\|ALG\|}{\|OPT\|} \leq \frac{u_4}{2u_3} = u_4 < \gamma.$$

⁷ As another example, in the identical machines makespan problem, it is provably advantageous to consider the largest items first.

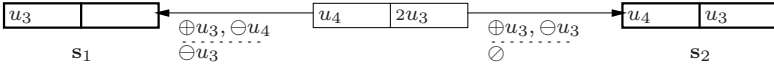


Fig. 1. Adversarial strategy for $\text{rank}(u_4) > \text{rank}(u_3)$

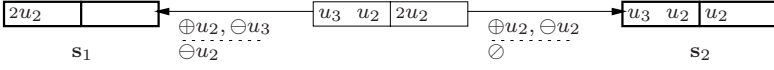


Fig. 2. Adversarial strategy for $\text{rank}(u_3) > \text{rank}(u_2)$

2. If $\text{rank}(u_3) > \text{rank}(u_2)$, then the adversarial strategy is shown in Fig. 2. If the algorithm terminates via state s_1 , then

$$\rho = \frac{\|\text{ALG}\|}{\|\text{OPT}\|} = \frac{2u_2}{u_2 + u_3} = \frac{\gamma - \frac{1}{5}}{\frac{1}{2} + \frac{1}{2}\gamma - \frac{1}{10}} = \frac{10\gamma - 2}{5\gamma + 4} < \gamma.$$

If the algorithm terminates via state s_2 , then

$$\rho = \frac{\|\text{ALG}\|}{\|\text{OPT}\|} \leq \frac{u_2 + u_3}{3u_2} = \frac{\frac{1}{2} + \frac{1}{2}\gamma - \frac{1}{10}}{\frac{3}{2}\gamma - \frac{3}{10}} = \frac{5\gamma + 4}{15\gamma - 3} < \gamma.$$

3. If $\text{rank}(u_2) > \text{rank}(u_1)$, then the adversarial strategy is shown in Fig. 3.

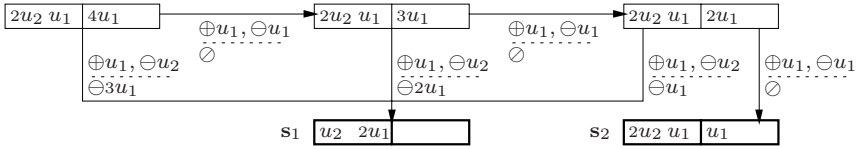


Fig. 3. Adversarial strategy for $\text{rank}(u_2) > \text{rank}(u_1)$

If the algorithm terminates via state s_1 , then

$$\rho = \frac{\|\text{ALG}\|}{\|\text{OPT}\|} \leq \frac{2u_1 + u_2}{u_1 + 2u_2} = \frac{\frac{2}{5} + \frac{1}{2}\gamma - \frac{1}{10}}{\frac{1}{5} + \gamma - \frac{1}{5}} = \frac{\frac{1}{2}\gamma + \frac{3}{10}}{\gamma} = \frac{5\gamma + 3}{10\gamma} = \gamma.$$

If the algorithm terminates via state s_2 , then

$$\rho = \frac{\|\text{ALG}\|}{\|\text{OPT}\|} \leq \frac{u_1 + 2u_2}{5u_1} = u_1 + 2u_2 = \frac{1}{5} + \gamma - \frac{1}{5} = \gamma.$$

4. If $\text{rank}(u_1) > \text{rank}(u_2) > \text{rank}(u_3) > \text{rank}(u_4)$, then the adversarial strategy is shown in Fig. 4.

If the algorithm terminates via state s_1 , then

$$\rho = \frac{\|\text{ALG}\|}{\|\text{OPT}\|} = \frac{u_1 + u_3}{u_2 + u_3} = \frac{\frac{1}{5} + \frac{1}{2}}{\frac{1}{2}\gamma - \frac{1}{10} + \frac{1}{2}} = \frac{7}{5\gamma + 4} < \gamma.$$

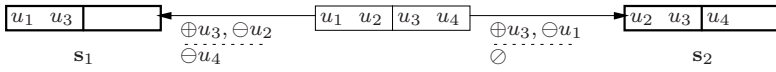


Fig. 4. Adversarial strategy for $\text{rank}(u_1) > \text{rank}(u_2) > \text{rank}(u_3) > \text{rank}(u_4)$

If the algorithm terminates via state s_2 , then

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} \leq \frac{u_2 + u_3}{u_1 + u_4} = u_2 + u_3 = \frac{1}{2}\gamma - \frac{1}{10} + \frac{1}{2} < \gamma.$$

As a conclusion, no fixed order revocable priority algorithm of SSP can achieve approximation ratio better than γ . This completes the proof. \square

Finally, we study adaptive revocable priority algorithms. This is the strongest class of algorithms studied in this paper and arguably represents the ultimate approximation power of greedy algorithms (for packing problems). We show that no such algorithm can achieve an approximation ratio better than δ , and then we develop a relatively subtle algorithm having approximation ratio 0.8 in Theorem 6, thus leaving another gap in what is provably the best approximation ratio possible.

Theorem 5. *No adaptive revocable priority algorithm of SSP can achieve approximation ratio better than δ .*

Proof. Let $u_1 = \frac{1}{3}\delta \approx 0.298$ and $u_2 = 0.5$. For a given algorithm, we utilize the following adversary strategy shown in Fig. 5. If the algorithm terminates via state s_1 or s_2 , then

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} \leq \frac{3u_1}{2u_2} = 3u_1 = \delta.$$

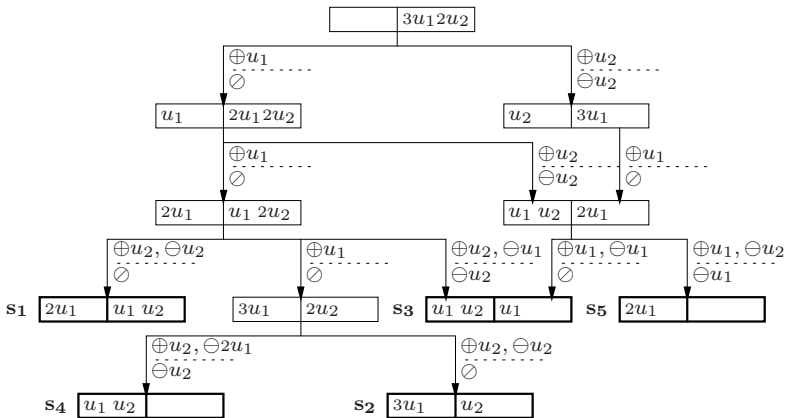


Fig. 5. Adversarial strategy for adaptive revocable priority algorithms

If the algorithm terminates via state \mathbf{s}_3 or \mathbf{s}_4 , then

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} \leq \frac{u_1 + u_2}{3u_1} = \frac{\frac{1}{3}\delta + \frac{1}{2}}{\delta} = \frac{2\delta + 3}{6\delta} = \delta.$$

If the algorithm terminates via state \mathbf{s}_5 , then

$$\rho = \frac{\|\mathbf{ALG}\|}{\|\mathbf{OPT}\|} = \frac{2u_1}{u_1 + u_2} = \frac{\frac{2}{3}\delta}{\frac{1}{3}\delta + \frac{1}{2}} = \frac{4\delta}{2\delta + 3} < \delta.$$

In all three cases, the adversary forces the algorithm to have approximation ratio no better than δ ; this completes the proof. \square

Our 0.8 approximation adaptive priority algorithm is facilitated by a few simplifying observations. First of all, since we are targeting a ratio of 0.8, the algorithm can terminate whenever it detects an item u in the remaining input such that a subset of $(\mathbf{B} \oplus u)$ has total weight ≥ 0.8 ; such an item is *adaptively* given the highest priority and we call this a *terminal condition*. In our algorithm description, it is understood that the algorithm first *adaptively* checks for such a condition, and terminates if it is satisfied. Note that the running time of checking such condition is bounded by a constant. Secondly, data items outside $(0.2, 0.8)$ are not needed for the analysis of the algorithm. That is, items in $[0.8, 1]$ trivialize the problem and items in $(0, .2]$ can be considered at the end and added to \mathbf{B} (if necessary) to achieve the desired bound. Finally, we assume the current set of accepted items \mathbf{B} operates in one of the following four modes:

1. **Queue Mode**: In this mode, accepted items are discarded in the FIFO order to accommodate the new data item u .
2. **Queue_1 Mode**: In this mode, the first accepted item is never discarded, the rest data items are discarded in the FIFO order to accommodate the new data item u .
3. **Stack Mode**: In this mode, accepted items are discarded in the FILO order to accommodate the new data item u .
4. **Optimum Mode**: In this mode, accepted items are discarded to maximize $\|\mathbf{B}\|$; the new data item u may also be discarded for this purpose.

We use \mathbf{B}_{mode} to represent the operational mode of \mathbf{B} . The algorithm can switch among these four modes during the processing of data items; we do not explicitly mention in the algorithm what data items are being discarded since it is well-defined under its operational mode.

The algorithm uses an ordering of data items which is determined by its distance to 0.3, i.e., the closer a data item to 0.3, the higher its priority is, breaking tie arbitrarily. Note that by the first observation given earlier, this ordering may be interrupted if at any point of time a terminal condition is satisfied, so this is not a fixed order priority algorithm. The algorithm is described below.

Algorithm ADAPTIVE

```

1: B :=  $\emptyset$ ;
2: if the first data item is in  $(0.2, 0.35)$  then
3:   Bmode := Queue;
4: else
5:   Bmode := Queue_1;
6: end if
7: while I contains a data item  $\in (0.2, 0.4]$  do
8:   let  $u$  be the next data item in I;
9:   I := I  $\ominus$   $u$ ;
10:  accept  $u$ ;
11: end while
12: if B contains exactly three data items and all are  $\in (0.2, 0.3]$  then
13:   Bmode := Stack;
14: else
15:   Bmode := Optimum;
16: end if
17: while I contains a data item  $\in (0.4, 0.8)$  do
18:   let  $u$  be the next data item in I;
19:   I := I  $\ominus$   $u$ ;
20:   accept  $u$ ;
21: end while

```

Theorem 6. *Algorithm ADAPTIVE achieves approximation ratio 0.8 for SSP.*

It is seemingly a small step from a 0.78 algorithm to a 0.8 algorithm, but the latter algorithm requires a substantially more refined approach and detailed analysis. The detailed analysis can be found in the full paper. The merit, we believe, in studying such a class of “simple algorithms” is that the simplicity of the structure suggests algorithmic ideas and allows a careful analysis of such algorithms. Limiting ourselves to simple algorithmic forms and exploiting the flexibility within such forms may very well give us a better understanding of the structure of a given problem and a better chance to derive new algorithms.

4 Conclusion

We analyze different types of priority algorithms for SSP leaving open two approximability gaps, one for fixed order and one for adaptive revocable priority algorithms. It is interesting that such gaps and non-trivial algorithms exist for such a simple class of algorithms and such a basic problem as SSP. We optimistically believe that surprisingly good algorithms can be designed within the revocable priority framework for problems which are currently not well understood.

References

1. Angelopoulos, S., Borodin, A.: On the power of priority algorithms for facility location and set cover. *Algorithmica* 40, 271–291 (2004)
2. Arrow, K.: *Social Choice and Individual Values*. Wiley, Chichester (1951)
3. Baptiste, P.: Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. *Journal of Scheduling* 2, 245–252 (1999)
4. Bar-Noy, A., Guha, S., Naor, J., Schieber, B.: Approximating throughput in real-time scheduling. *SIAM Journal of Computing* 31, 331–352 (2001)
5. Borodin, A., Nielsen, M., Rackoff, C.: (Incremental) priority algorithms. *Algorithmica* 37, 295–326 (2003)
6. Borodin, A., Boyar, J., Larsen, K.: Priority algorithms for graph optimization problems. In: Persiano, G., Solis-Oba, R. (eds.) *WAOA 2004*. LNCS, vol. 3351, pp. 126–139. Springer, Heidelberg (2005)
7. Chrobak, M., Durr, C., Jawor, W., Kowalik, L., Kurowski, M.: On scheduling equal length jobs to maximize throughput. To appear in *Journal of Scheduling*.
8. Chuzhoy, J., Ostrovsky, R., Rabani, Y.: Approximation algorithms for the job interval scheduling problem and related scheduling problems. In: *Proceedings of 42nd Annual IEEE Symposium of Foundations of Computer Science*, pp. 348–356. IEEE Computer Society Press, Los Alamitos (2001)
9. Davis, S., Impagliazzo, R.: Models of greedy algorithms for graph problems. In: *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 381–390. ACM Press, New York (2004)
10. Erlebach, T., Spieksma, F.: Interval selection: Applications, algorithms, and lower bounds. *Journal of Algorithms* 46, 27–53 (2003)
11. Garey, M., Johnson, D.: *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, San Francisco (1979)
12. Horn, S.: One-pass algorithms with revocable acceptances for job interval selection. Master’s thesis, University of Toronto (2004)
13. Ibarra, O., Kim, C.: Fast approximation algorithms for the knapsack and sum of subset problem. *Journal of the ACM* 22, 463–468 (1975)
14. Iwama, K., Taketomi, S.: Removable online knapsack problems. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) *ICALP 2002*. LNCS, vol. 2380, pp. 293–305. Springer, Heidelberg (2002)
15. Kellerer, H., Mansini, R., Pferschy, U., Speranza, M.: An efficient fully polynomial approximation scheme for the subset-sum problem. *Journal of Computer and System Science* 66, 349–370 (2003)
16. Martello, S., Toth, P.: *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley and Sons, Chichester (1990)
17. Moore, J.: An n -job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science* 15, 102–109 (1968)
18. Regev, O.: Priority algorithms for makespan minimization in the subset model. *Information Processing Letters* 84, 153–157 (2002)
19. Ye, Y., Borodin, A.: Priority algorithms for the subset-sum problem. Technical Report, University of Toronto (2007) <http://www.cs.toronto.edu/~bor>

Distributed Approximation Algorithms for Weighted Problems in Minor-Closed Families^{*}

A. Czygrinow¹ and M. Hańćkowiak²

¹ Department of Mathematics and Statistics
Arizona State University
Tempe, AZ 85287-1804, USA
andrzej@math.la.asu.edu

² Faculty of Mathematics and Computer Science
Adam Mickiewicz University, Poznań, Poland
mhanckow@amu.edu.pl

Abstract. We give efficient distributed approximation algorithms for weighted versions of the maximum matching problem and the minimum dominating set problem for graphs from minor-closed families. To complement these results we indicate that no efficient distributed algorithm for the minimum weight connected dominating set exists.

1 Introduction

Efficient distributed algorithms for only very few graph-theoretic problems are known. At the same time there has been much more success in designing efficient distributed algorithms in case the underlying topology of the network has additional properties. For example, many problems can be solved efficiently in constant maximum degree graphs and some problems admit rather easy distributed approximations in graphs of bounded arboricity (for example in planar graphs). In this paper, we will study distributed complexity of three fundamental problems in proper minor-closed families of graphs. We will show that the maximum-weight matching problem and the minimum-weight dominating set problem admit efficient distributed approximations but the minimum-weight connected dominating set problem does not. This extends and complements the results from [3] where distributed complexity of unweighted versions of the above problems is analyzed. Note however that algorithms for weighted problems are significantly different than the ones from [3]. In fact, even the distributed complexity of weighted and unweighted problems can be different. For example, in [3] we proved that the minimum-connected dominating set problems admits an efficient distributed approximation in connected graphs which come from minor-closed families. This is not the case for the weighted analog as we indicate in the last section of this paper. The algorithms for weighted versions of the maximum matching and the minimum dominating set problems are in turn based on a completely new and provably more powerful partitioning algorithm than the corresponding clustering procedure in [3].

^{*} This work was supported by grant N206 017 32/2452 for years 2007-2010.

1.1 Terminology and Notation

We will consider the message-passing distributed model (see Linial [10]). In this model, network is represented by an undirected graph with vertices corresponding to processors, and edges corresponding to communication links between processors. The network is synchronized and computations proceed in discrete rounds. In a single round a vertex can send and receive messages from its neighbors, and can perform some local computations. Neither the amount of local computations nor the lengths of messages are restricted in any way. We will also assume that nodes in the network have unique identifiers which are positive integers from $\{1, \dots, |G|\}$ where $|G|$ is the order of G and is globally known. Although different possible measures of efficiency of a distributed algorithm can be assumed, following [10] we call a distributed algorithm *efficient* if it runs in a poly-logarithmic (in the order of the graph) number of rounds. Consequently, if the diameter of the underlying network is poly-logarithmic then any of the above problems admits a trivial efficient solution. In this paper, we shall focus on distributed approximation algorithms for minor-closed families. All graphs are simple and in the graph-theoretic terminology we will follow [5]. A graph H is called a *minor* of G if it can be obtained from a subgraph of G by a series of edge contractions. A family \mathcal{C} is called *minor-closed* if for any graph $G \in \mathcal{C}$ every minor of G is also in \mathcal{C} . A family \mathcal{C} is proper if there is a graph G which is not in \mathcal{C} and is non-trivial if it contains a graph with at least one edge. We will always assume that our minor-closed family is both proper and non-trivial. Although the most important example of a minor-closed family is certainly the class of planar graphs, algorithmic questions for different minor-closed classes of graphs, like the family of graphs with a bounded tree-width or a bounded genus, have recently attracted attention. Let \mathcal{C} be a minor-closed family and let $\rho := \sup_{G \in \mathcal{C}} \frac{|E(G)|}{|G|}$ be the edge density of \mathcal{C} . It is known (see for example [12]) that ρ is finite if and only if \mathcal{C} is a proper minor-closed family. We will write \mathcal{C}_ρ for a minor-closed family with edge density ρ and assume that ρ is known by an algorithm. A *matching* in graph G is a subset M of edges of G with no two edges from M sharing a vertex. For an edge-weighted graph $(G, \bar{\omega})$ with $\bar{\omega} : E(G) \rightarrow R^+ \cup \{0\}$ we denote by $\beta(G)$ the maximum weight of a matching in G , that is $\beta(G) = \max_M \sum_{e \in M} \bar{\omega}(e)$. A *dominating set* in a graph G is a subset D of vertices such that for every vertex $v \notin D$ a neighbor of v is in D . For a vertex-weighted graph (G, ω) with $\omega : V(G) \rightarrow R^+ \cup \{0\}$ we denote by $\gamma(G)$ the minimum weight of a dominating set in G , that is $\gamma(G) = \min_D \sum_{v \in D} \omega(v)$. Finally, a dominating set D is called a *connected dominating set* in G if it is a dominating set and the subgraph of G induced by D is connected.

We will denote by $|G|$ the order of G , that is the number of vertices of G and by $\|G\|$ the size of G , that is the number of edges of G . As already noted we assume that each vertex v has a unique identifier $ID(v)$ and $ID : V(G) \rightarrow \{1, \dots, |G|\}$. Since our partitioning algorithm will be applied to an auxiliary graph of G and it will be important to distinguish between the range of identifiers in the auxiliary graph and the order of the graph we denote by $ID(H) = \bigcup_{v \in V(H)} \{ID(v)\}$.

1.2 Results

We give distributed approximation algorithms for the maximum-weight matching problem and for the minimum-weight dominating set problem for graphs from a minor-closed family \mathcal{C}_ρ . In the case of the maximum-weight matching problem we will give a distributed algorithm which given a positive integer d finds in an edge-weighted graph $(G, \bar{\omega})$ with $G \in \mathcal{C}_\rho$ a matching M of weight $\bar{\omega}(M) \geq \left(1 - \frac{1}{\log^d |G|}\right) \beta(G)$. The algorithm runs in a poly-logarithmic number of rounds. (Theorem 4) For the minimum-weight dominating set problem, we will prove that there is a distributed algorithm which given a positive integer d finds in vertex-weighted graph (G, ω) with $G \in \mathcal{C}_\rho$ a dominating set D such that $\omega(D) \leq \left(1 + \frac{1}{\log^d |G|}\right) \gamma(G)$. This algorithm again runs in a poly-logarithmic number of rounds. (Theorem 5) For the minimum-weight connected dominating set problem we indicate that to accomplish any finite multiplicative approximation error, $\Omega(|G|)$ rounds are needed. Both algorithms use a vertex partitioning procedure which partitions the vertex set of a graph G into sets V_1, \dots, V_l so that each $G[V_i]$ has a poly-logarithmic diameter and the weight of the border vertices is small with respect to the total weight of G (see Corollary 3 for a precise statement).

1.3 Related Work

We will briefly put our results in a more general context. The reader is directed to Elkin's survey [6] for a more comprehensive overview of distributed approximation algorithms. Let us first note that efficient distributed algorithms that find exact solutions to the above problems do not exist (even for unweighted analogs). For example, the minimum dominating set problem and the maximum matching problem when restricted to a cycle G cannot be found in $o(|G|)$ rounds ([10]). In addition, to achieve a poly-logarithmic approximation ratio for minimum dominating set at least $\max\{\Omega(\sqrt{\log |G| / \log \log |G|}), \Omega(\log \Delta / \log \log \Delta)\}$ rounds are required ([8]).

Distributed approximation algorithms for planar graphs were studied in [2] and [4]. In particular, [2] contains an efficient distributed approximation for the maximum-weight independent set problem in planar graphs and [3] contains efficient distributed algorithms for unweighted versions of the three problems in minor-closed families of graphs.

1.4 Organization

In the rest of the paper we will first discuss vertex partitioning problems in weighted graphs and give our main auxiliary procedure (Section 2). Then, in Section 3, we give our approximation algorithms.

2 Partitioning of Vertex-Weighted Graphs

We will start with fixing some general graph-theoretic terminology. For a graph G , $V(G)$ will denote the vertex set of G and $E(G)$ will denote the edge set of

G . If U, U' are two disjoint subsets of $V(G)$ then $E_G(U, U')$ denotes the set of edges with one endpoint in U , another in U' . For $v \in V(G)$, $N(v)$ denotes the set of neighbors of v in G and if $U \subseteq V(G)$ then $N_G(U) := \bigcup_{u \in U} N(u) \setminus U$. For two vertices u, u' , $dist_G(u, u')$ is the length of the shortest path between u and u' , the diameter of G , $diam_G$, is the maximum of $dist_G(u, u')$ over all pair (u, u') , and for sets U, U' , we set $dist_G(U, U') := \min_{u \in U, u' \in U'} dist_G(u, u')$. In addition for a subgraph H of G we will consider two different diameters of H . The strong diameter of H , $SDiam_G(H)$, will be defined as $diam_H$ and the weak diameter of H , $WDiam_G(H)$, will be defined as $max_{u, u' \in V(H)} dist_G(u, u')$. Clearly $WDiam_G(H) \leq SDiam_G(H)$.

Let \mathcal{C}_ρ be a minor-closed family of graphs G with the edge-density ρ , that is

$$\rho = \sup \frac{|E_G|}{|G|}$$

where the supremum is taken over all graphs from \mathcal{C}_ρ . It is known (see [12] for this and many other results) that ρ is finite if and only if \mathcal{C}_ρ is a proper minor-closed family. In addition, in the paper, we will always assume that \mathcal{C}_ρ is proper and $\rho > 0$.

Although vast majority of the paper is concerned with vertex-weighted graphs, we will start with a brief discussion that shows a connection between distributed partitioning problems for vertex-weighted and edge-weighted graphs (Section 2.1). In Section 2.2, we will give an efficient distributed partitioning algorithm for vertex-weighted graphs from \mathcal{C}_ρ . The distributed algorithm is deterministic but we assume that both $|G|$ and ρ are known to all vertices of G .

2.1 Weighted Graphs

For a graph $G \in \mathcal{C}_\rho$ we will consider two types of weight functions on G . Pair (G, ω) with $\omega : V(G) \rightarrow R^+ \cup \{0\}$ will be called *vertex-weighted graph* G and the pair $(G, \bar{\omega})$ with $\bar{\omega} : E(G) \rightarrow R^+ \cup \{0\}$ will be called *edge-weighted graph* G . We need some more notation and terminology. Let (G, ω) be a vertex-weighted graph. For a set $S \subseteq V(G)$ we define $\omega(S) := \sum_{v \in S} \omega(v)$. A vertex of S is called a *border vertex* in S if it has a neighbor in $V(G) \setminus S$. The set of all border vertices in S is denoted by $\partial(S)$ and for a partition $P = (V_1, V_2, \dots, V_l)$ of $V(G)$ we set $\partial(P) := \bigcup_{i=1}^l \partial(V_i)$. In the case of an edge-weighted graph $(G, \bar{\omega})$ we define $\bar{\partial}(S)$ to be the set of all edges with one endpoint in S and another in $V(G) \setminus S$. Then for a partition $P = (V_1, V_2, \dots, V_l)$ of $V(G)$, $\bar{\partial}(P) := \bigcup_{i=1}^l \bar{\partial}(V_i)$.

Definition 1. Let (G, ω) be a vertex-weighted graph and let $a(\cdot), b(\cdot)$ be functions to R . A partition $P = (V_1, V_2, \dots, V_l)$ of $V(G)$ is called an (a, b) -vertex-weight partition if the following two conditions are satisfied:

- For $i = 1, \dots, l$, $G[V_i]$ is connected and $WDiam_G(G[V_i]) \leq a(|G|)$.
- $\omega(\partial(P)) \leq \omega(V(G))/b(|G|)$.

Similarly we define an (a, b) -edge-weight partition of $(G, \bar{\omega})$. We will be almost exclusively interested in cases when both a and b are poly-logarithmic functions.

In [2], a distributed algorithm that finds a $(\log |G|, \log |G|)$ - edge-weight partition of an edge-weighted planar graph G is given. The algorithm runs in a poly-logarithmic number of rounds. This edge-weight partition can be used to give distributed approximation algorithms for the maximum-weight independent set problem. In addition, a similar procedure can be used to give distributed approximations for the unweighted versions of the maximum matching problem or the minimum dominating set problem in graphs G which come from a fixed minor-closed family. On the other hand, the edge-weight partition property is not strong enough to yield approximations for weighted analogs of the maximum matching problem or the minimum dominating set problem. As we will show in the next section, vertex-weight partition can be found by a distributed algorithm efficiently and can be used to design approximations for the weighted versions of the above two problems. Let us first note however the vertex-weight partition is indeed stronger than an edge-weight partition.

2.2 Partitioning Algorithm

We will now give an algorithm which finds an (a, b) -vertex-weight partition. Let us start by fixing some additional terminology. Let G be a graph from \mathcal{C}_ρ and let $\omega : V(G) \rightarrow \mathbb{R}^+ \cup \{0\}$. A small modification (change in the number of iterations) of the algorithms CLUSTERING and WISPlanar from [2] yields the following two facts.

Lemma 1. *Let \mathcal{C}_ρ be a minor-closed family of graphs. Let $G \in \mathcal{C}_\rho$ and let $(G, \bar{\omega})$ be an edge-weighted graph. There exists a distributed algorithm which given a constant $d > 1$ finds in $O(\log |G| \log^* |G|)$ rounds a (D_ρ, d) -edge-weight partition for some constant $D_\rho = D_\rho(d)$.*

Lemma 2. *Let \mathcal{C}_ρ be a minor-closed family of graphs. Let $G \in \mathcal{C}_\rho$ and let (G, ω) be a vertex-weighted graph. There exists a distributed algorithm which given a constant $d > 2\rho$ finds in $O(\log |G| \log^* |G|)$ rounds a maximal independent set I in G with*

$$\omega(I) \geq \omega(V(G))/d.$$

Our first procedure, HEAVY SUBSET, finds a subset of vertices of a large weight which induces subgraphs of small weak diameter. As the procedure is a bit technical we will divide it into two phases.

HEAVY SUBSET PHASE 1. Use the algorithm DECOMPOSITION from [3] to find a partition (V_1, \dots, V_k) of $V(G)$ with properties that each V_i is an independent set, $k = O(\log |G|)$, and for every i , if $v \in V_i$ then $|N(v) \cap \bigcup_{j>i} V_j| \leq 3\rho$. Give the orientation (u, v) (from u to v) to every edge $\{u, v\}$ with $u \in V_i$ and $v \in V_j$ whenever $i < j$ and define the weight of (u, v) by setting $\bar{\omega}(u, v) := \omega(u)$. Note that the out-degree of this directed graph is at most 3ρ . Let $d := 3 \cdot \rho / \left(1 - \frac{1}{2\rho+1}\right)$ and let D_ρ denote the constant from Lemma 1. Find a (D_ρ, d) -edge-weight partition (V_1, \dots, V_k) of $(G, \bar{\omega})$. Consider two sets of vertices: B (black) and W (white). Set initially $B := V(G)$ and $W := \emptyset$. For every vertex u ,

in parallel, if $u \in V_i$ and there is a vertex $v \in V \setminus V_i$ such that (u, v) is an arc, then change the color of u to white. We will end the phase one here. First note the following fact.

Fact 1. *After HEAVY SUBSET PHASE 1 all edges with endpoints in different V_i 's have at least one endpoint in W .*

In addition, we have the following easy lemma.

Lemma 3. *Let B be the set of black vertices in G after HEAVY SUBSET PHASE 1. We have*

$$\omega(B) \geq \frac{\omega(V(G))}{2\rho + 1}.$$

HEAVY SUBSET PHASE 2. After the execution of phase one (V_1, \dots, V_k) is a partition of $V(G)$ and every edge with endpoints in different V_i 's has at least one endpoint in W . Consequently some of the border vertices of each V_i can be white. A vertex w is called a *troubler* if $w \in W$ and for some $v_i \in V_i \cap B$ and $v_j \in V_j \cap B$ with $i \neq j$, $v_i w v_j$ is a path (of length two) in G . In other word a troubler is a white vertex which is connected by an edge with two black vertices in different V_i 's. Clearly only a border vertex can be a troubler. Recall that in phase one we gave an orientation to all edges of G . We shall now define an auxiliary hypergraph. For each troubler w , if w is in V_i and has more than one out-neighbor in $B \cap (V \setminus V_i)$ then consider the hyper-edge f_w consisting of these out-neighbors and let \mathcal{H} be the hypergraph $\mathcal{H} := (B, \bigcup\{f_w\})$. Note that as \mathcal{H} is on B , there can be many isolated vertices in \mathcal{H} . In addition $|f_w| \leq 3\rho$ for any troubler w as the out-degree is at most 3ρ .

Next task is to find a "heavy" maximal independent set I in \mathcal{H} . This is done by consider the graph G' with $V(G') := B$ and the edge set $E(G')$ obtained in the following way. Every troubler w selects two distinct vertices $u, v \in f_w$ and adds the edge $\{u, v\}$ to $E(G')$. Then every edge in G' corresponds to a path uwv in G with $w \in W$ and different paths contain different w 's. Therefore G' is a topological minor of G and so $G' \in \mathcal{C}_\rho$. Use Lemma 2 to find a maximal independent set I in G' with $\omega(I) \geq \omega(B)/(2\rho + 1) \geq \omega(V(G))/(2\rho + 1)^2$ and repaint vertices from $B \setminus I$ with the white color. Repeat the process after updating sets f_w and the hypergraph \mathcal{H} . Note that in each round of the above procedure, the size of f_w drops by at least one and so after $3\rho - 1$ rounds $|f_w| \leq 1$ for every troubler w . Consequently, the last instance of I from the loop above is an independent set in the initial hypergraph \mathcal{H} . In addition we see that I has a large weight.

Fact 2. *Set I of vertices in G is an independent set in the hypergraph \mathcal{H} and $\omega(I) \geq \omega(V(G))/(2\rho + 1)^{3\rho}$.*

Now for every troubler w with $|f_w| = 1$ let u_w denote the vertex in f_w and let $S = \{\{w, u_w\} : |f_w| = 1\}$. Consider the subgraph \tilde{G}_i of $G[V_i]$ induced by edges which have at least one endpoint in $B \cap V_i$. In addition, consider another auxiliary graph G'' : For every $i = 1, \dots, k$, contract every component U of the subgraph \tilde{G}_i to a vertex and let v_U denote the vertex obtained from set U . Put

an edge between two vertices $v_U, v_{U'}$ whenever $E_G(U, U') \cap S \neq \emptyset$. In addition, set $\omega(v_U) := \sum_{w \in B \cap U} \omega(w)$. Finally, note that the graph G'' is in \mathcal{C}_ρ and so by Lemma 2 we can find an independent set I in G'' of weight which is at least $\omega(V(G''))/(2\rho + 1)$ which by Fact 2 is at least $\omega(V(G))/(2\rho + 1)^{3\rho+1}$. Now repaint all vertices of B which are not in a set U with $v_U \in I$ with the white color. Finally consider the subgraph \tilde{G} of G induced by edges from $E(G)$ which have at least one endpoint in B and return the components of \tilde{G} . This is the end of phase two.

Lemma 4. *Let B be the set obtained in Phase 2 and let L_1, L_2, \dots, L_p be the components of \tilde{G} which are returned in the end of phase two. We have:*

- $\omega(B) \geq \omega(V(G))/(2\rho + 1)^{3\rho+1}$.
- For $i = 1, \dots, p$, $WDiam_G(L_i) \leq D_\rho + 2$.

Proof. We have already proved the first part. Recall that I denotes the independent set in G'' obtained in HEAVYSUBSET PHASE 2 and let $v_U, v_{U'} \in I$. We will first show that $dist_G(U \cap B, U' \cap B) \geq 3$. To that end assume first that $U \subset V_i$ and $U' \subset V_j$ with $i \neq j$ and suppose that there is a path of length at most two with one endpoint in $U \cap B$ and another in $U' \cap B$. Clearly the path cannot have length one as every edge from $E_G(V_i, V_j)$ has one endpoint in W (Fact 1). Consequently the path has length two and has the form $v_i w v_j$ with $w \in W$. Thus w is a troubler after all of the iterations in \mathcal{H} and either $v_i = u_w$ or $v_j = u_w$ which yields an edge between $v_U, v_{U'}$ in G'' and contradicts the fact that I is independent.

Now suppose that $U, U' \subset V_i$. Then the graphs induced by U, U' are components in \tilde{G}_i and so the distance between $U \cap B$ and $U' \cap B$ in $G[V_i]$ is at least three. In addition, if there is vertex $w \in V_j$ for $j \neq i$ with a neighbor in $U \cap B$ and $U' \cap B$ then $w \in W$ and $|f_w| \geq 2$ which is not possible. Now take a component L of \tilde{G} and let $v_U \in I$ be such that L and U intersect in a black vertex. Also let i be such that $U \subseteq V_i$. If $N_{\tilde{G}}(U \cap B) \subseteq V_i$ then the vertex set of L is a subset of V_i and so the diameter of L is at most D_ρ . Otherwise take a vertex $u \in N_{\tilde{G}}(U \cap B) \setminus V_i$. Then u is white and so u is troubler. As there are no edges in \tilde{G} with both endpoints white, u has at most two neighbors in \tilde{G} and both of them are black. If one of them is in $U' \neq U$ then $dist_G(B \cap U, B \cap U') \leq 2$. Consequently u has one neighbor in \tilde{G} from $B \cap U$. Therefore $V(L)$ is a subset of $N_G(U \cap B) \cup U \subseteq N_G(V_i) \cup V_i$. Since $WDiam_G(G[V_i]) \leq D_\rho$ we have $WDiam_G(L) \leq D_\rho + 2$. □

Now we can describe our main partitioning algorithm.

VERTEX-WEIGHT PARTITION. Given is a vertex-weighted graph (G, ω) with $G \in \mathcal{C}_\rho$ and a positive integer t which can depend on $|G|$. Iterate with i from 1 to t . In the i th iteration, invoke HEAVY SUBSET to find the set of black vertices B and the components L_1, L_2, \dots, L_p from Lemma 4. Obtain the minor G_i of G_{i-1} by contracting each of the L_1, L_2, \dots, L_p to a single vertex and set the weight of the vertex obtained from L_i to be equal to the total weight of white vertices in L_i . Let G_t be the graph obtained from G after all t iterations. For each vertex Q

in G_t consider the set V_Q of all vertices in G which have been contracted to Q in the above iterations and return the partition $P = (V_Q | Q \in V(G_t))$.

Lemma 5. *Let $P = (V_1, V_2, \dots, V_k)$ be the partition of graph $G \in \mathcal{C}_\rho$ obtained by VERTEX-WEIGHT PARTITION with given parameter t .*

(a) *Let D_ρ be the constant from Lemma 4 obtained by setting $d = 3 \cdot \rho / (1 - \frac{1}{2\rho+1})$. Then*

$$WDiam_G(G[V_i]) \leq (D_\rho + 3)^t.$$

(b) $\omega(\partial(P)) \leq \left(1 - \frac{1}{(2\rho+1)^{3\rho+1}}\right)^t \omega(V(G))$.

Proof. Let G_i denote the graph obtained after the i th iteration of VERTEX-WEIGHT PARTITION and let $G_0 := G$. To show (a), let $diam_i$ be the maximum of $WDiam_G(G[V'])$ over subsets $V' \subset V(G)$ that are contracted to single vertices in G_i . Clearly $diam_0 = 0$ and by Lemma 4 (part two) $diam_{i+1} \leq (D_\rho + 3) \cdot diam_i + D_\rho + 2$. Consequently $diam_t \leq (D_\rho + 3)^t$. To verify part (b), we consider the sequence of partitions $\{P_i\}$ where P_i is the partition of $V(G)$ obtained by creating a partition class for each vertex Q in G_i consisting of vertices that has been contracted to Q in iterations $1, \dots, i$. Let $\partial_i = \partial(P_i)$ with $\partial_0 := V(G)$. Note that after the coloring of $V(G_i)$ performed by HEAVY SUBSET only white vertices in a component L have neighbors in $V(G_i) \setminus L$. Therefore, $\omega(\partial_i)$ is smaller than or equal to the weight of white vertices in G_i . By definition of the weights in VERTEX-WEIGHT PARTITION, $\omega(V(G_{i+1}))$ is equal to the weight of white vertices in G_i and so $\omega(V(G_i)) \leq \left(1 - \frac{1}{(2\rho+1)^{3\rho+1}}\right)^{i-1} \omega(V(G))$ which in view of Lemma 4 (part one) gives

$$\omega(\partial_i) \leq \left(1 - \frac{1}{(2\rho+1)^{3\rho+1}}\right)^i \omega(V(G)).$$

□

For the next corollary, recall that $ID : V(G) \rightarrow \{1, \dots, m\}$ is a function with $ID(v)$ equal to the identifier of vertex v . Although, as mentioned in the introduction, in the original graph G , $ID(V(G))$ is assumed to be equal to $V(G)$, in applications we will partition auxiliary graphs and it will be important to distinguish between $|G|$ and the order of the auxiliary graph.

Corollary 3

- (a) *There is a distributed algorithm which given $0 < \epsilon < 1$ finds in a vertex-weighted graph (G, ω) with $G \in \mathcal{C}_\rho$ an (a, b) -vertex-weight partition $P = (V_1, \dots, V_k)$ with $b \geq 1/\epsilon$ and $a \leq D(\epsilon)$ for some constant $D(\epsilon)$. The algorithm runs in $O(\log |G| \log^* |G|)$ rounds.*
- (b) *There is a distributed algorithm which given a positive integer p finds in a vertex-weighted graph (G, ω) with $G \in \mathcal{C}_\rho$ and $ID(v) \leq m$ for every $v \in V(G)$ an (a, b) -vertex-weight partition $P = (V_1, \dots, V_k)$ with $b = \log^p m$ and $a = \text{polylog}(m)$. The algorithm runs in a pol-logarithmic (in m) number of rounds.*

3 Applications

We will now show how to use the vertex-weight partition to design distributed approximations for the minimum-weight dominating set problem and the maximum-weight matching problem.

3.1 Matchings

Let us start with the maximum-weight matching problem. Let $(G, \bar{\omega})$ be an edge-weighted graph with $G \in \mathcal{C}_\rho$. In the algorithm, we first find a subgraph of G and use it to define the vertex-weighted graph (G, ω) . Then we apply the partitioning procedure from Corollary 3. The procedure takes a positive integer d as an input.

APPROXMWM. Use the algorithm DECOMPOSITION from 3 to find a partition of $V(G)$ into k sets V_1, \dots, V_k so that each V_i is an independent set, $k = O(\log |G|)$, and for every i , if $v \in V_i$ then $|N(v) \cap \bigcup_{j>i} V_j| \leq 3\rho$. For every vertex v if $v \in V_j$ then v properly colors all edges in $E(\{v\}, \bigcup_{j>i} V_j)$ using colors from $\{1, \dots, 3\rho\}$. Let F_i be the subgraph of G induced by edges of color i . Then F_i is a forest every component of which has diameter $O(\log |G|)$. Find in each F_i a maximum weight matching N_i and let $Q := \bigcup N_i$. Now for every vertex v set $\omega(v) := \bar{\omega}(e)$ where e is an edge in Q of maximum weight which is incident to v . If no such edge exists set $\omega(v) := 0$. Use the algorithm from Corollary 3 (b) with $p = d + 1$ and $m = |G|$ to obtain a vertex-weight partition $P = (V_1, \dots, V_k)$ of (G, ω) . Find a maximum weight matching M_i in each of $(G[V_i], \bar{\omega})$ and return $\bigcup M_i$.

Theorem 4. *Let $(G, \bar{\omega})$ be an edge-weighted graph with $G \in \mathcal{C}_\rho$. There is a distributed algorithm which given a positive integer d finds a matching M in G with*

$$\bar{\omega}(M) \geq \left(1 - \frac{1}{\log^d |G|}\right) \beta(G)$$

where $\beta(G)$ is the weight of a maximum-weight matching in G . The algorithm runs in a poly-logarithmic number of rounds.

Proof. We use APPROXMWM. Note that $\bar{\omega}(Q) \leq 3\rho\beta(G)$ and so the total vertex weight of G satisfies $\omega(V(G)) \leq 3\rho\beta(G)$. Moreover we have that, for every edge $\{u, v\} \in E(G)$, $\bar{\omega}(\{u, v\}) \leq \omega(u) + \omega(v)$. Indeed if $\{u, v\} \in Q$ then this is clear. If $\{u, v\}$ is not in Q and $\{u, v\}$ is in F_i then there exist at most two edges $e_1 = \{u, w\}, e_2 = \{v, z\}$ in M_i such that $\bar{\omega}(\{u, v\}) \leq \bar{\omega}(e_1) + \bar{\omega}(e_2)$. Consequently $\bar{\omega}(\{u, v\}) \leq \omega(u) + \omega(v)$. We have

$$\omega(\partial(P)) \leq \omega(V(G)) / \log^{d+1} |G| \leq \frac{3\rho\beta(G)}{\log^{d+1} |G|} \leq \frac{\beta(G)}{\log^d |G|}. \tag{1}$$

Every matching in G contains two types of edges: edges with both endpoints in some V_i and edges that are incident to $\partial(P)$. The total weight of the latter is at most $\frac{\beta(G)}{\log^d |G|}$ by (II) and so the matching M returned by APPROXMWM satisfies

$$\beta(G) \leq \omega(M) + \frac{\beta(G)}{\log^d |G|}.$$

□

3.2 Dominating and Connected Dominating Sets

In this section we discuss dominating set problems. Due to space limitations, we will not be able to provide full details (the full version is available from authors' web sites). Let (G, ω) be a vertex-weighted graph. Recall that for any $D \subseteq V(G)$ we have $\omega(D) := \sum_{v \in D} \omega(v)$. We will denote by $\gamma(G) = \min \omega(D)$ where the minimum is taken over all dominating sets in graph G . For a vertex v , recall that $N(v)$ denotes the set of neighbors of v and $N[v] := N(v) \cup \{v\}$. Pick one vertex in $N[v]$, $s(v)$, with $\omega(s(v)) := \min_{w \in N[v]} \omega(w)$ and set $\bar{D} := \bigcup \{s(v)\}$.

Lemma 6. *Let (G, ω) be a vertex-weighted graph and let $\bar{D} := \bigcup \{s(v)\}$. Then \bar{D} is a dominating set and $\omega(\bar{D}) \leq |G|\gamma(G)$.*

Our approximation algorithm proceeds in two main phases. First we find a constant approximation of $\gamma(G)$ and next we find a more accurate approximation. For a dominating set D in G , every vertex $v \in V(G) \setminus D$ selects one vertex w in $N(v) \cap D$ and joins group U_w . Let G_D be obtained from G by contracting $U_w \cup \{w\}$ to a single vertex u_w with $\omega(u_w) := \omega(w)$. Then, clearly, $\omega(V(G_D)) = \omega(D)$. Our algorithm APPROXMWDS is given a positive integer d which will be used in the second phase of the procedure.

APPROXMWDS PHASE 1. Let $D := \bar{D}$. We iterate with i from 1 to $\log_2 |G|$. In the i th iteration, we consider (G_D, ω) and use Corollary 3 (a) with $\epsilon = 1/2$ to find a vertex-weight partition (V'_1, \dots, V'_k) of G_D . This gives a partition P of G by setting V_j to be the the union of U_w 's with $u_w \in V'_j$. In each $G[V_j]$ we find a dominating set D_i with $\omega(D_j) = \gamma(G[V_j])$ and set $D := \bigcup_{j=1}^k D_j$. Then we have the following fact.

Lemma 7. *Let (G, ω) be a vertex-weighted graph with $G \in \mathcal{C}_p$ and let D be the set obtained by APPROXMWDS PHASE 1. Then $\omega(D) \leq \gamma(G)/2$.*

APPROXMWDS PHASE 2. Let D be the dominating set obtained from APPROXMWDS PHASE 1. Consider G_D and use the algorithm from Corollary 3 (b) with $p := d + 1$ and $m = |G|$ to find a vertex-weight partition (V'_1, \dots, V'_k) of G_D . This gives a partition $P = (V_1, \dots, V_k)$ of G as in phase one and we again find an optimal solution in each of $G[V_i]$'s and return the union.

Theorem 5. *Let \mathcal{C}_ρ be a minor-closed family. There exists a distributed algorithm which given a positive integer d finds in vertex-weighted graph (G, ω) with $G \in \mathcal{C}_\rho$ a dominating set D with*

$$\omega(D) \leq \left(1 + \frac{1}{\log^d |G|}\right) \gamma(G).$$

The algorithm runs in a poly-logarithmic number of rounds.

In the case of the weighted connected dominating set problem, one can show that no non-trivial approximation factor can be obtained in $o(|G|)$ rounds even when G is a cycle.

References

1. Awerbuch, B., Goldberg, A.V., Luby, M., Plotkin, S.A.: Network Decomposition and Locality in Distributed Computation. In: Proc. 30th IEEE Symp. on Foundations of Computer Science (FOCS), pp. 364–369. IEEE Computer Society Press, Los Alamitos (1989)
2. Czygrinow, A., Hańćkowiak, M.: Distributed algorithms for weighted problems in sparse graphs. *Journal of Discrete Algorithms* 4(4), 588–607 (2006)
3. Czygrinow, A., Hańćkowiak, M.: Distributed almost exact approximations for minor-closed families. In: Azar, Y., Erlebach, T. (eds.) *ESA 2006*. LNCS, vol. 4168, pp. 244–255. Springer, Heidelberg (2006)
4. Czygrinow, A., Hańćkowiak, M., Szymańska, E.: Distributed approximation algorithms in planar graphs, 6th Conference on Algorithms and Complexity (CIAC). In: Calamoneri, T., Finocchi, I., Italiano, G.F. (eds.) *CIAC 2006*. LNCS, vol. 3998, pp. 296–307. Springer, Heidelberg (2006)
5. Diestel, R.: *Graph Theory*, 3rd edn. Springer, Heidelberg (2005)
6. Elkin, M.: An Overview of Distributed Approximation. In *ACM SIGACT News Distributed Computing Column*. 35(4,132), 40–57 (2004)
7. Kuhn, F., Wattenhofer, R.: Constant-Time Distributed Dominating Set Approximation. 22nd ACM Symposium on the Principles of Distributed Computing (PODC), pp. 25–32. ACM Press, New York (2003)
8. Kuhn, F., Moscibroda, T., Wattenhofer, R.: What Cannot Be Computed Locally! In: Proceedings of 23rd ACM Symposium on the Principles of Distributed Computing (PODC), pp. 300–309. ACM Press, New York (2004)
9. Kutten, S., Peleg, D.: Fast distributed construction of k -dominating sets and applications. In: Proceedings of the 14th ACM symposium on Principles of Distributed Computing (PODC), pp. 238–251. ACM Press, New York (1995)
10. Linial, N.: Locality in distributed graph algorithms. *SIAM Journal on Computing* 21(1), 193–201 (1992)
11. Luby, M.: A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing* 15(4), 1036–1053 (1986)
12. Nešetřil, J., de Mendez, P.O.: Colorings and homomorphisms of minor closed classes. In: Aronov, B., Basu, S., Pach, J., Sharir, M. (eds.) *Discrete and Computational Geometry, The Goodman-Pollack Festschrift*. Algorithms and Combinatorics, vol. 25, pp. 651–664. Springer, Heidelberg (2003)

A 1-Local 13/9-Competitive Algorithm for Multicoloring Hexagonal Graphs

Francis Y.L. Chin^{1,*}, Yong Zhang¹, and Hong Zhu^{2,**}

¹ Department of Computer Science, The University of Hong Kong, Hong Kong
{chin,yzhang}@cs.hku.hk

² Institute of Theoretical Computing, East China Normal University, China
hzhu@sei.ecnu.edu.cn

Abstract. In the frequency allocation problem, we are given a mobile telephone network, whose geographical coverage area is divided into cells, wherein phone calls are serviced by assigning frequencies to them so that no two calls emanating from the same or neighboring cells are assigned the same frequency. The problem is to use the frequencies efficiently, i.e., minimize the span of frequencies used. The frequency allocation problem can be regarded as a multicoloring problem on a weighted hexagonal graph. In this paper, we give a 1-local $4/3$ -competitive distributed algorithm for multicoloring a triangle-free hexagonal graph, which is a special case. Based on this result, we then propose a 1-local $13/9$ -competitive algorithm for multicoloring the (general-case) hexagonal graph, thereby improving the previous 1-local $3/2$ -competitive algorithm.

1 Introduction

Wireless communication based on Frequency Division Multiplexing (FDM) technology is widely used in the area of mobile computing today. In such FDM networks, a geographic area is divided into small cellular regions or *cells*, each containing one base station. Base stations communicate with each other via a high-speed wired network. Calls between any two clients (even within the same cell) must be established through base stations. When a call arrives, the nearest base station must allocate a frequency from the available spectrum to the call without causing any interference to other calls. In practice, when the same frequency is assigned to two different calls emanating from cells that are geographically close to each other, interference may occur which distorts the radio signals. To avoid interference, the temptation is to use many frequencies. However, spectrum is a scarce resource and efficient utilization of the available spectrum is essential for FDM networks.

The *frequency allocation problem* has been extensively studied [6,9,12,13,11,10,2,3,4]. Both the off-line and online versions of the problem have

* This research was supported by Hong Kong RGC Grant HKU-7113/07E.

** This research was supported in part by National Natural Science Fund (grant no. 60496321).

been studied. For the off-line problem on cellular networks (where cells are hexagonal regions as shown in Fig. 1 and the calls to be serviced are known *a priori*), McDiarmid and Reed [12] have shown that the problem of minimizing the span of frequencies to satisfy all the call requests is NP-hard, and 4/3-approximation algorithms were given in [12,14].

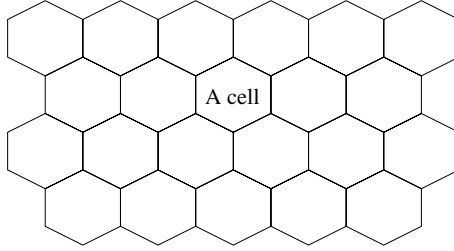


Fig. 1. Example of a cellular network (with hexagonal cells)

For the online version, there are mainly three strategies, which have been introduced: the *fixed allocation assignment* (FAA) [11], the *greedy algorithm* (Greedy) [5] and the *hybrid algorithm* [4]. FAA partitions cells into independent sets which are each assigned a separate set of frequencies. It is easy to see that FAA is 3-competitive as cellular networks are 3-colorable. Greedy assigns the minimum available number (frequency) to a new call so that the call does not interfere with calls of the same or adjacent cells. Caragiannis et al. [5] proved that the competitive ratio of Greedy for FAC is at least 17/7 and at most 2.5. Chan et al. [3] gave a tighter analysis to show that Greedy is 17/7-competitive. Furthermore, Chan et al. [4] proposed a 2-competitive hybrid algorithm, which combines the FAA and Greedy approach and which is optimal since the lower bound of this problem is also 2.

The frequency allocation problem in cellular network can be abstracted to the problem of *multicoloring a weighted hexagonal graph*, in which each vertex has a weight which specifies how many different colors have to be assigned to the vertex. Given the constraint that the same color cannot be assigned to adjacent vertices, the target is minimize the number of assigned colors.

In frequency allocation problem, the size of cellular network is very large, when handling a call request, the computation will be very complex if all information needed. In reality, each server in the cells knows its position before processing the sequence of calls; when satisfying call requests, each server only know its local information, i.e., some information within a fixed distance. In this paper, we focus on *distributed algorithms* for the multicoloring, i.e., each vertex is an independent server, which runs the algorithm to assign multicolors to the vertex based on what is known as *k-local* information. The concept of *k-local* distributed algorithms was introduced by Janssen et al [8], where an algorithm is *k-local* if the computation at a vertex depends only on the information of the neighboring vertices of at most *k* distance away (suppose each edge has unit distance). We can

also assume that in multicoloring problem, each vertex is also know its position in the graph.

In [8], Janssen et al. proved (the next lemma) that a k -local c -approximate off-line algorithm can be easily converted to a k -local c -competitive online algorithm. Thus, to design a k -local online algorithm, we need only to focus on the off-line problem.

Lemma 1. [8] *Let A be a k -local c -approximate off-line algorithm for multicoloring. Then A can be converted into a k -local c -competitive online algorithm for multicoloring.*

An interesting induced graph, called a *triangle-free hexagonal graph*, has been studied for the multicoloring problem. A graph is *triangle-free* if there are no 3-cliques in the graph, i.e., there are no three mutually-adjacent vertices with positive weights. An example of a triangle-free hexagonal graph is shown in Fig. 2.

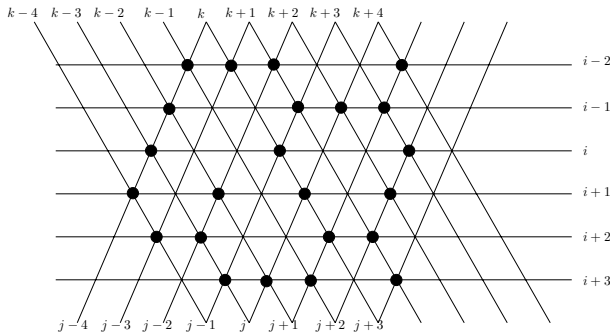


Fig. 2. An example of a triangle-free hexagonal graph, where solid circles are vertices with positive weights

The best known competitive ratios for 0-, 1-, 2- and 4-local distributed algorithms for multicoloring on (general) hexagonal graphs are 3, 3/2, 4/3 and 4/3, respectively [8,17]. It is possible to do better for triangle-free hexagonal graphs. For example, in [16], a 2-local 5/4 competitive algorithm was given, and an inductive proof for the 7/6 ratio was reported in [7].

In this paper, we first give a 1-local 4/3-competitive algorithm for multicoloring a triangle-free hexagonal graph in Section 3. Based on this result, we then propose, in Section 4, a 1-local 13/9-competitive algorithm for the multicoloring problem in hexagonal graph, which improves the previous 3/2-competitive result. Section 2 introduces some preliminary terminology to be used in this paper, and Section 5 concludes.

2 Preliminary Terminology

Suppose the hexagonal graph has been three-colored with each vertex colored with one of three *base colors*, without loss of generality, say *Red*, *Green* and

Blue. We assume a transitive order on these three base colors: namely, $Red < Green < Blue$.

We use a 3-coordinate system to represent each vertex. In particular, referring to the lines shown in Fig. 2, each vertex can be represented by coordinate (i, j, k) where i is the coordinate for the horizontal line, j for the up-sloping line and k for the down-sloping line. (In fact, two coordinates are enough to represent vertices since coordinate k is redundant, but we find it convenient to refer to three coordinates.) For example, a vertex with coordinate (i, j, k) and its six neighboring vertices, denoted by UL, L, DL, UR, R and DR, are represented as shown in Fig. 3.

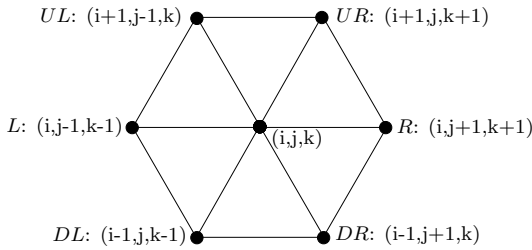


Fig. 3. Coordinates of a vertex (i, j, k) and its neighboring vertices

Next, we use the definition in [17] to define the *parity* of a vertex with respect to its various neighbors. We say that the parity of a vertex v with coordinate (i, j, k) is:

1. **odd** (alternatively, **even**) with respect to its L or R neighbor if $j \equiv 1 \pmod 2$ (correspondingly, $j \equiv 0 \pmod 2$);
2. **odd** (alternatively, **even**) with respect to its UL or DR neighbor if $i \equiv 1 \pmod 2$ (correspondingly, $i \equiv 0 \pmod 2$);
3. **odd** (alternatively, **even**) with respect to its DL or UR neighbor if $k \equiv 1 \pmod 2$ (correspondingly, $k \equiv 0 \pmod 2$).

Let w_v be the weight of vertex v , which corresponds to the number of colors needed to multicolor v . After the multicoloring assignment, each vertex v will be assigned a set F_v of colors, such that $F_v \subset Z^+$ and $|F_v| = w_v$, where, for any two adjacent vertices u and v , $F_u \cap F_v = \phi$.

3 Multicoloring in Triangle-Free Hexagonal Graphs

In this section, we shall study the problem of multicoloring a special type of hexagonal graph. Finding a good solution for this problem will lead to an algorithm for finding good solutions for general hexagonal graphs.

A graph is *triangle-free* if no three mutually-adjacent vertices have positive weights. For a given vertex u with positive weight w_u , from this definition of

triangle-free graph, only two possible configurations may exist for the structure of neighbors with positive weights, which are shown in Fig. 4. There exist a simple structure in triangle-free graph, i.e., a vertex has only one neighbor, we can regard this structure as the case in Fig. 4(b).

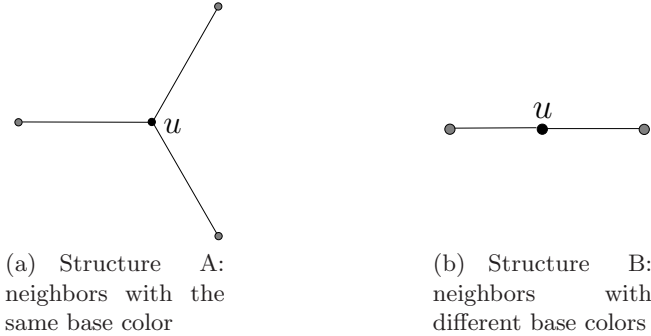


Fig. 4. Structure of neighbors with positive weight

Consider vertex u with positive weight w_u . Compute $c_u = w_u + \max\{w_v \mid v \text{ is } u\text{'s neighbor}\}$. c_u is the weight of the maximum 2-clique adjacent to u , which also gives the minimum number of colors needed for multicoloring a triangle-free hexagonal graph. From the definition of c_u , any feasible coloring of vertex u and its neighbors requires at least c_u colors.

Let $d_u = \lceil c_u/3 \rceil$. For each vertex u , we define four color sets, each of size d_u :

1. $colorset_u(Red) = \{j \in \{1, \dots, 4d_u\} \mid j = 1 \pmod{4}\}$,
2. $colorset_u(Green) = \{j \in \{1, \dots, 4d_u\} \mid j = 2 \pmod{4}\}$,
3. $colorset_u(Blue) = \{j \in \{1, \dots, 4d_u\} \mid j = 3 \pmod{4}\}$, and
4. $extraset_u = \{j \in \{1, \dots, 4d_u\} \mid j = 0 \pmod{4}\}$.

We will give a strategy to multicolor any vertex u with weight w_u by assigning w_u colors from the above four sets so that no adjacent vertices are assigned the same color. The assignment strategy assigns multicolors to u according to its base color and neighboring structure and can be described as follows.

Assume vertex u with base color X has neighboring structure A, i.e. all its neighbors have the same base color $Y \neq X$. Let the third base color be Z where $Z \neq X$ and $Z \neq Y$. In this case, our strategy would be to assign multicolors to vertex u first from $colorset_u(X)$, then $colorset_u(Z)$ and finally $colorset_u(Y)$.

On the other hand, if vertex u with base color X has neighboring structure B, then all three base colors will be used by u and its neighbors. The strategy will first assign multicolors from $colorset_u(X)$, then from the extra color set $extraset_u$ and finally from $colorset_u(Y)$ where base color $Y > Z \neq X$.

Note that in both cases, the colors in each color set may be assigned either from bottom to top or from top to bottom, depending on the base color or the parity of the vertex so as to avoid any conflicts.

THE STRATEGY

1. If vertex u has no neighbors, just assign w_u colors from 1 to w_u .
2. If vertex u with base color X has neighboring structure A (Fig. 4(a)), let Y be the base color of u 's neighbors and Z be the other third color. Assign w_u multicolors to vertex u as follows:
 - (a) Assign colors from $colorset_u(X)$ in bottom-to-top order.
 - (b) If not enough, assign colors from $colorset_u(Z)$ in bottom-to-top order if $X < Y$; top-to-bottom otherwise.
 - (c) If still not enough, assign colors from $colorset_u(Y)$ in top-to-bottom order.
3. If vertex u with base color X has neighboring structure B (Fig. 4(b)), let Y and Z be the base colors of the left neighbor and the right neighbor, respectively. Without loss of generality, assume $Y > Z$. Assign w_u multicolors to vertex u as follows:
 - (a) Assign colors from $colorset_u(X)$ in bottom-to-top order.
 - (b) If not enough, assign colors from $extraset_u$ in bottom-to-top order if u is *odd* with respect to its left or right neighbor; top-to-bottom order otherwise.
 - (c) If still not enough, assign colors from $colorset_u(Y)$ in top-to-bottom order.

Theorem 1. *The above strategy is 1-local and can solve the multicoloring problem in triangle-free hexagonal graphs with performance ratio 4/3.*

Proof. From the description of the strategy, it is clear that the colors assigned to any vertex depend only on neighboring information within distance 1, and thus, the strategy is 1-local.

To prove that the above strategy solves the multicoloring problem, we must prove that the colors assigned to any two adjacent vertices u and v are all different. As it turns out, we need to analyze the three structures shown in Fig. 5. X and Y denote the two respective different base colors of u and v . It is easy to see that different kinds of color sets of u and v have no common colors. For example, $colorset_u(Red) \cap extraset_v = \emptyset$.

From the definition of c_u , we have $c_u \geq w_u + w_v$ and, since $d_u = \lceil c_u/3 \rceil$, $c_u \leq 3d_u$.

For Case A, the strategy would assign colors to u from $colorset_u(X)$, then $colorset_u(Z)$ and then $colorset_u(Y)$, and would assign colors to v from $colorset_v(Y)$, then $colorset_v(Z)$ and then $colorset_v(X)$. There are three subcases to consider:

- (A-1) Case where u is assigned colors from $colorset_u(X)$ and v is assigned colors from $colorset_v(X)$ after exhausting all colors in $colorset_v(Y)$ and $colorset_v(Z)$. Then, the weight w_v of vertex v should be very large since all the colors in $colorset_v(Y)$ and $colorset_v(Z)$ must be used up. Since $w_u + w_v \leq c_u \leq 3d_u$, $w_u + w_v \leq c_v \leq 3d_v$, and since u and v use $colorset(X) = colorset_u(X) \cup colorset_v(X)$ from opposite directions (depending on whether $X < Y$ or otherwise), u and v will not be assigned the same color.

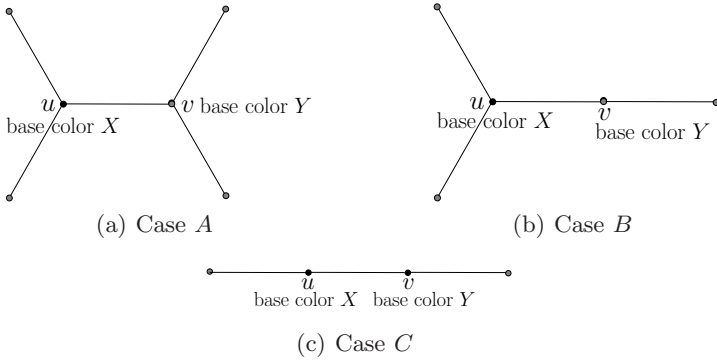


Fig. 5. The local structure of vertices u and v

- (A-2) Case where u is assigned colors from $colorset_u(Z)$ and v is assigned colors from $colorset_v(Z)$. Then, all the colors in $colorset_u(X)$ are assigned to u and all the colors in $colorset_v(Y)$ are assigned to v . Since $w_u + w_v \leq 3d_u, 3d_v$, and since u and v use $colorset(Z) = colorset_u(Z) \cup colorset_v(Z)$ from opposite directions, u and v will not be assigned the same color.
- (A-3) Case where u is assigned colors from $colorset_u(Y)$ and v is assigned colors from $colorset_v(Y)$. By similar analysis as in case (A-1), we can say that u and v will not be assigned the same color.

In Case B , the strategy would assign colors to u from $colorset_u(X)$, then $colorset_u(Z)$ and then $colorset_u(Y)$. Also, the strategy would assign colors to v from $colorset_v(Y)$, then $extraset_v$ and then $colorset_v(X)$ or $colorset_v(Z)$ depending whether $X < Z$ or otherwise. Without loss of generality, we assume $colorset_v(X)$ is used. There are two subcases to consider:

- (B-1) Case where u is assigned colors from $colorset_u(X)$ and v is assigned colors from $colorset_v(X)$ after exhausting all colors in $colorset_v(Y)$ and $extraset_v$. This means the weight w_v of v is very large. Since $w_u + w_v \leq 3d_u, 3d_v$, and since u and v use $colorset(X) = colorset_u(X) \cup colorset_v(X)$ from opposite directions, u and v will not be assigned the same color.
- (B-2) Case where u is assigned colors from $colorset_u(Y)$ and v is assigned colors from $colorset_v(Y)$. By similar analysis as in case (B-1), we can say that u and v will not be assigned the same color.

In Case C , the strategy would assign colors to u from $colorset_u(X)$, then $extraset_u$ and then $colorset_u(Y)$, and would assign colors to v from $colorset_v(Y)$, then $extraset_v$ and then $colorset_v(X)$. There are three subcases to consider:

- (C-1) Case where u is assigned colors from $colorset_u(X)$ and v is assigned colors from $colorset_v(X)$ after exhausting all colors in $colorset_v(Y)$ and $extraset_v$. This means the weight w_v of v is very large. Since

- $w_u + w_v \leq 3d_u, 3d_v$, and since u and v use $colorset(X) = colorset_u(X) \cup colorset_v(X)$ from opposite directions, u and v will not be assigned the same color.
- (C-2) Case where u is assigned colors from $extraset_u$ and v is assigned colors from $extraset_v$. This means all the colors in $colorset_u(X)$ and $colorset_v(Y)$ have been assigned to u and v , respectively. Since $w_u + w_v \leq 3d_u, 3d_v$, and since u and v use $extraset = extraset_u \cup extraset_v$ from opposite directions (the parities of u and v are different), u and v will not be assigned the same color.
 - (C-3) Case where u is assigned colors from $colorset_u(Y)$ and v is assigned colors from $colorset_v(Y)$. By similar analysis as in case (C-1), we can say that u and v will not be assigned the same color.

For the whole triangle-free hexagonal graph, the maximal weight clique (2-clique) $c = \max_u \{c_u\}$ is a lower bound on the optimal value, and our algorithm uses at most $4 \max_u \{ \lceil c_u/3 \rceil \}$ colors. Thus, the above strategy has a performance ratio of $4/3$. □

From Theorem 1, we can easily have a 1-local $4/3$ -competitive online algorithm for frequency allocation in triangle-free cellular networks.

4 Multicoloring in Hexagonal Graphs

In this section, we consider multicoloring hexagonal graphs. Our strategy works in two stages. In the first stage, each vertex assigns colors using local information on the weights of this vertex and its neighboring vertices. After the first stage, some vertices may be unsatisfied, i.e. not all of the necessary colors have been assigned, and the unsatisfied vertices, along with the edges connecting them, form a triangle-free graph. Applying the algorithm in the previous section, each vertex can be assigned colors, to satisfy all the remaining unsatisfied vertices, by using 1-local information. Combining these two stages, we have a 1-local algorithm for multicoloring hexagonal graphs.

We now describe the first stage, which is similar to the first stage in [12]. In [12], the algorithm needs to have the global information about the maximum weights of “all” 3-cliques in the graph (stage 1) so as to have an acyclic graph of the remaining unsatisfied vertices (for stage 2). As for an algorithm which is 1-local, only the maximal weights of the local 3-cliques will be available (stage 1) and a triangle-free hexagonal graph (which can be cyclic) will result (for stage 2). Consider vertex u with base color X . Let C_u be the maximal weights among the 3-cliques including u , and let $k_u = \lceil C_u/3 \rceil$. For the three base colors *Red*, *Green* and *Blue*, we define a cyclic order among them as *Red* \rightarrow *Green*, *Green* \rightarrow *Blue* and *Blue* \rightarrow *Red*. If $X \rightarrow Y$, let m_u be the maximal weight of the neighboring vertices with color Y . We define color sets: $colorset_u(Red) = \{j \in \{1, \dots, 3k_u\} \mid j \equiv 1 \pmod 3\}$, $colorset_u(Green) = \{j \in \{1, \dots, 3k_u\} \mid j \equiv 2 \pmod 3\}$ and $colorset_u(Blue) = \{j \in \{1, \dots, 3k_u\} \mid j \equiv 0 \pmod 3\}$. In the first stage, vertex u with base color X and weight w_u is assigned colors from these sets using the strategy described as follows:

1. Vertex u is assigned colors from $colorset_u(X)$ in bottom-to-top order.
2. If not enough and $m_u < k_u$, vertex u is assigned the upper $\min\{k_u - m_u, w_u - k_u\}$ colors from $colorset_u(Y)$.

After the first stage, each vertex has been assigned with some colors. The remaining graph contains only those vertices whose calls have not been totally satisfied, i.e., the number of assigned colors in vertex u is less than its weight.

Lemma 2. *The remaining graph is triangle-free, i.e., contains no 3-clique.*

Proof. If some vertex u is still unsatisfied, it must be that $w_u > \max\{k_u, 2k_u - m_u\}$. Thus, the remaining unsatisfied weight in vertex u is $w'_u = w_u - \max\{k_u, 2k_u - m_u\}$. For any three mutually-adjacent vertices (3-clique) u, v and t , since $\min\{C_u, C_v, C_t\} \geq w_u + w_v + w_t$, $\min\{k_u, k_v, k_t\} \geq \min\{w_u, w_v, w_t\}$ and at most two of $\{w'_u, w'_v, w'_t\}$ are strictly positive, at least one of the vertices (u, v and t) has all its required colors totally assigned in the first stage. Therefore, the remaining graph contains no 3-clique, i.e., is a triangle-free hexagonal graph. \square

Lemma 3. *The total weight of two neighboring vertices u and v in the remaining graph is at most $\max\{C_u, C_v\}/3$.*

Proof. For the remaining unsatisfied vertices, since $w'_u = w_u - \max\{k_u, 2k_u - m_u\}$, we have $w'_u \leq w_u - (2k_u - m_u) = w_u + m_u - 2k_u \leq C_u - 2k_u$. For any two adjacent unsatisfied vertices u and v , we can also get $w'_u + w'_v \leq w_u - k_u + w_v - k_v \leq C_v/3$. If $C_u \geq C_v$, which implies $k_u \geq k_v$, then we have $w'_u + w'_v \leq C_v - 2k_v$ as $C_v \geq w_u + w_v$. Similarly, if $C_u \leq C_v$, which implies $k_u \leq k_v$, then we have $w'_u + w'_v \leq C_u - 2k_u \leq C_u/3$. Thus, the total remaining weight of any two adjacent unsatisfied vertices is at most $\max\{C_u, C_v\}/3$. \square

From Lemma 2, the remaining graph is triangle-free, so in the second stage, we can use the algorithm in Section 3 to process the remaining unsatisfied vertices. Each vertex gets the remaining weight information from its adjacent vertices and the total number of colors used in this stage is at most $4 \max_u \lceil \frac{C_u}{9} \rceil$ (Theorem 1 and Lemma 3).

Combining these two stages, we use at most $\max_u (3k_u + 4 \lceil \frac{C_u}{9} \rceil) \leq \max_u (3 \lceil \frac{C_u}{3} \rceil + 4 \lceil \frac{C_u}{9} \rceil) \leq \frac{13}{9} C_u + 7$ colors. Since C_u is a lower bound on the optimal solution, the performance ratio for our strategy is $13/9$.

Thus, we have the following theorem.

Theorem 2. *For the multicoloring problem in hexagonal graphs, a 1-local 13/9-competitive algorithm can be achieved.*

5 Conclusion

We have given a 13/9-approximation algorithm for multicoloring hexagonal graphs. This implies a 13/9-competitive solution for the online frequency allocation problem, which involves servicing calls in each cell in a cellular network. The

distributed algorithm is practical in the sense that frequency allocation can be done based on information about its neighbors and itself only. We note that, in fact, when calls are requested or released in a cell, a constant number of frequencies might have to be reassigned so as to actually achieve the 13/9-competitive bound.

Acknowledgements. The authors thank Dr. Bethany M.Y. Chan for her efforts in making this paper more readable.

References

1. Aardal, K.I., van Hoesel, S.P.M., Koster, A.M.C.A., Mannino, C., Sassano, A.: Models and solution techniques for frequency assignment problems. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies (4OR)* 1(4), 261–317 (2003)
2. Chan, W.-T., Chin, F.Y.L., Ye, D., Zhang, Y., Zhu, H.: Frequency Allocation Problem for Linear Cellular Networks. In: Asano, T. (ed.) *ISAAC 2006*. LNCS, vol. 4288, pp. 61–70. Springer, Heidelberg (2006)
3. Chan, W.-T., Chin, F.Y.L., Ye, D., Zhang, Y., Zhu, H.: Greedy Online Frequency Allocation in Cellular Networks. *Information Processing Letters* 102, 55–61 (2007)
4. Chan, W.-T., Chin, F.Y.L., Ye, D., Zhang, Y.: Online Frequency Allocation in Cellular Networks. To appear in *Proc. of the 19th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '07)*
5. Caragiannis, I., Kaklamanis, C., Papaioannou, E.: Efficient on-line frequency allocation and call control in cellular networks. *Theory Comput. Syst.* 35(5), 521–543 (2002) A preliminary version of the paper appeared in *SPAA 2000*
6. Hale, W.: Frequency assignment: Theory and applications. *Proceedings of the IEEE* 68(12), 1497–1514 (1980)
7. Havet, F.: Channel assignment and multicoloring of the induced subgraphs of the triangular lattice. *Discrete Math.* 233, 219C231 (2001)
8. Janssen, J., Krizanc, D., Narayanan, L., Shende, S.M.: Distributed online frequency assignment in cellular networks. *J. Algorithms* 36(2), 119–151 (2000)
9. Jaumard, B., Marcotte, O., Meyer, C.: Mathematical models and exact methods for channel assignment in cellular networks. In: Sansò, B., Soriano, P. (eds.) *Telecommunications Network Planning*, pp. 239–255. Kluwer Academic Publishers, Dordrecht (1999)
10. Katzela, I., Naghshineh, M.: Channel assignment schemes for cellular mobile telecommunication systems: A comprehensive survey. *IEEE Personal Communications* 3(3), 10–31 (1996)
11. MacDonald, V.: Advanced mobile phone service: The cellular concept. *Bell Systems Technical Journal*, vol. 58(1) (1979)
12. McDiarmid, C., Reed, B.A.: Channel assignment and weighted coloring. *Networks* 36(2), 114–117 (2000)
13. Narayanan, L.: Channel assignment and graph multicoloring. In: Stojmenović, I. (ed.) *Handbook of Wireless Networks and Mobile Computing*, pp. 71–94. John Wiley & Sons, Chichester (2002)
14. Narayanan, L., Shende, S.M.: Static frequency assignment in cellular networks. *Algorithmica* 29(3), 396–409 (2001)

15. Narayanan, L., Tang, Y.: Worst-case analysis of a dynamic channel assignment strategy. *Discrete Applied Mathematics* 140(1-3), 115–141 (2004)
16. Sparl, P., Zerovnik, J.: 2-local $5/4$ -competitive algorithm for multicoloring triangle-free hexagonal graphs. *Information Processing Letters* 90, 239–246 (2004)
17. Sparl, P., Zerovnik, J.: 2-local $4/3$ -competitive algorithm for multicoloring hexagonal graphs. *J. Algorithms* 55(1), 29–41 (2005)

Improved Algorithms for Weighted and Unweighted Set Splitting Problems^{*}

Jianer Chen and Songjian Lu

Department of Computer Science
Texas A&M University
College Station, TX 77843-3112, USA
{chen,sjlu}@cs.tamu.edu

Abstract. In this paper, we study parameterized algorithms for the SET SPLITTING problem, for both weighted and unweighted versions. First, we develop a new and effective technique based on a probabilistic method that allows us to develop a simpler and more efficient (deterministic) kernelization algorithm for the unweighted SET SPLITTING problem. We then propose a randomized algorithm for the weighted SET SPLITTING problem that is based on a new subset partition technique and has its running time bounded by $O^*(2^k)$, which even significantly improves the previously known upper bound for the unweighted SET SPLITTING problem. We also show that our algorithm can be de-randomized, thus derive the first fixed parameter tractable algorithm for the weighted SET SPLITTING problem.

1 Introduction

Let X be a set. A *partition* of X is a pair of subsets (X_1, X_2) of X such that $X_1 \cup X_2 = X$ and $X_1 \cap X_2 = \emptyset$. We say that a subset S of X is *split* by the partition (X_1, X_2) of X if S intersects with both X_1 and X_2 . The SET SPLITTING problem is defined as follows: given a collection \mathcal{F} of subsets of a ground set X , construct a partition of X that maximizes the number of split subsets in \mathcal{F} .

A more generalized version of the SET SPLITTING problem is the *weighted* SET SPLITTING problem, in which each subset in the collection \mathcal{F} is associated with a weight that is a real number, and the objective is to construct a partition of the ground set that maximizes the sum of the weights of the split subsets.

The SET SPLITTING problem is an important NP-complete problem [10]. A number of well-known NP-complete problems are related to the SET SPLITTING problem, including the HITTING SET problem that is to find a small subset of the ground X that intersects all subsets in a collection \mathcal{F} , and the SET PACKING problem that is to find a large sub-collection \mathcal{F}' of the collection \mathcal{F} of subsets such that the subsets in \mathcal{F}' are all pairwise disjoint.

In terms of approximability, the SET SPLITTING problem is APX-complete [3]. Andersson and Engebretsen [2] gave an approximation algorithm for the

^{*} This work was supported in part by the National Science Foundation under the Grants CCR-0311590 and CCF-0430683.

problem that has an approximation ratio bounded by 0.724. Zhang and Ling [16] presented an improved approximation algorithm of approximation ratio 0.7499 for the problem. Better approximation algorithms can be achieved if we further restrict the number of elements in each subset in the input [11,16,17,18].

On the need of applications, such as the analysis of micro-array data, people have studied the parameterized version of the SET SPLITTING problem, by associating each input with a parameter k , which is in general small. Formally, an instance of a *parameterized weighted SET SPLITTING* problem consists of a collection \mathcal{F} of subsets of a ground set X , in which each subset has a weight, and a parameter k . The objective is to construct a partition of the ground set X that maximizes the weight sum of k split subsets in \mathcal{F} , or report that no partition of X can split k subsets in \mathcal{F} . A restricted version of the parameterized weighted SET SPLITTING problem is the *parameterized unweighted SET SPLITTING* problem in which all subsets in the collection \mathcal{F} has weight 1.

In this paper, we are mainly concerned with *fixed parameter tractable algorithms* [8] for the parameterized SET SPLITTING problems, where the algorithms run in time $f(k)n^{O(1)}$, with $f(k)$ being a function that only depends on the parameter k . In particular, for small values of the parameter k , a fixed parameter tractable algorithm for the parameterized SET SPLITTING problem may become effective in practice. Since we will be only considering parameterized versions of the SET SPLITTING problems, we will drop the word “parameterized” when we refer to the problems.

The unweighted SET SPLITTING problem has been studied in the literature. Dehne, Fellows, and Rosamond [6] were the first to study the problem and provided a fixed parameter tractable algorithm of running time $O^*(72^k)$ for the problem [1]. In the same paper, the authors also proved that the unweighted SET SPLITTING problem has a kernel of size bounded by $2k$: that is, there is a polynomial time algorithm that on a given instance (X, \mathcal{F}, k) of unweighted SET SPLITTING, produces another instance (X', \mathcal{F}', k') for the problem such that $|X'| \leq |X|$, $|\mathcal{F}'| < 2k$, $k' \leq k$, and that the set X has a partition that splits k subsets in the collection \mathcal{F} if and only if the set X' has a partition that splits k' subsets in the collection \mathcal{F}' . Later, Dehne, Fellows, Rosamond, and Shaw [7] developed an improved algorithm of running time $O^*(8^k)$ for the problem. The improved algorithm was obtained by combining the recently developed techniques *greedy localization* and *modeled crown reduction* in the study of parameterized algorithms. The current best algorithm for the unweighted SET SPLITTING problem is developed by Lokshtanov and Sloper [14], where they used Chen and Kanj’s result for MAX-SAT problem [4] and reached a time complexity of $O^*(2.65^k)$.

No fixed parameter tractable algorithms have been known for the weighted SET SPLITTING problem. In fact, none of the techniques developed previously for the unweighted SET SPLITTING problem, such as those in [6,7,14], seems to be extendable to the weighted case.

¹ Following the recent convention, by the notation $O^*(c^k)$, where $c > 1$ is a constant, we refer to a function of order $O(c^k n^{O(1)} g(k))$, where $g(k) = c^{o(k)}$.

In this paper, we develop new techniques in dealing with the SET SPLITTING problems. First, we develop a new and effective technique based on a probabilistic method that allows us to develop a (deterministic) kernelization algorithm for the unweighted SET SPLITTING problem. The new kernelization algorithm is simpler and more efficient compared to the previous algorithm given in [6]. We then propose a randomized algorithm for the weighted SET SPLITTING problem that is based on a new subset partition technique and has its running time bounded by $O^*(2^k)$. This even significantly improves the previous best upper bound $O^*(2.65^k)$ given in [14] for the (simpler) unweighted SET SPLITTING problem. We also show that, using the subset partition family proposed by Naor, Schulman, and Srinivasan [15], we can de-randomize our randomized algorithm, which gives the first fixed parameter tractable algorithm for the weighted SET SPLITTING problem.

2 A New Kernelization Algorithm for the Unweighted SET SPLITTING Problem

In this section, we focus on the unweighted SET SPLITTING problem. By a *kernelization algorithm* for unweighted SET SPLITTING, we mean a polynomial time algorithm that, on an instance (X, \mathcal{F}, k) of unweighted SET SPLITTING, produces another instance (X', \mathcal{F}', k') for the problem such that the size of the instance (X', \mathcal{F}', k') only depends on the parameter k . The instance (X', \mathcal{F}', k') will be called a *kernel* for the instance (X, \mathcal{F}, k) . Dehne, Fellows, Rosamond and Shaw [6] developed a kernelization algorithm by which the kernel (X', \mathcal{F}', k') satisfies the conditions $|\mathcal{F}'| < 2k$ and that each subset in \mathcal{F}' has at most $2k$ elements. Lokshantov and Sloper [14] used the crown decomposition and obtained a kernel that both $|\mathcal{F}'|$ and $|X'|$ are less than $2k$. We introduce a new method to find the kernel for the unweighted SET SPLITTING problem. What is interesting in our method is that we use a probabilistic method to derive a deterministic kernelization algorithm. In particular, our method is simpler, has lower time complexity, and can also get a better kernel in term of the number of subsets in \mathcal{F}' if there are many subsets that have more than two elements.

Lemma 1. *Given an instance (X, \mathcal{F}, k) of the unweighted SET SPLITTING problem, let m_1 be the number of subsets in \mathcal{F} that have only one element. If $|\mathcal{F}| - m_1 \geq 2k$, then a partition of X exists that splits at least k subsets in \mathcal{F} .*

Proof. For each subset $S \in \mathcal{F}$, if S has at least two elements, we pick any two elements from S . Let V be the set of all these elements picked from the subsets in \mathcal{F} that have more than one element. Note that for two subsets S_1 and S_2 in \mathcal{F} that have more than one element, the two elements in S_1 and the two elements in S_2 may not be disjoint.

Suppose $|V| = t$. We randomly partition V into two subsets V_l and V_r , such that $|V_l| = \lfloor t/2 \rfloor$, $|V_r| = t - |V_l|$, i.e. we randomly pick $\lfloor t/2 \rfloor$ elements of V into V_l and let the remaining $t - \lfloor t/2 \rfloor$ elements of V in V_r . Thus, for any subset S in \mathcal{F} :

$$Pr(S \text{ is split}) \begin{cases} \geq \frac{2\binom{t-2}{\lfloor t/2 \rfloor - 1}}{\binom{t}{\lfloor t/2 \rfloor}} = \frac{2\lfloor t/2 \rfloor (t - \lfloor t/2 \rfloor)}{t(t-1)} > \frac{1}{2}, & \text{if } S \text{ has more than one element} \\ = 0, & \text{otherwise.} \end{cases}$$

If we let:

$$X_S = \begin{cases} 1, & \text{if } S \text{ is split,} \\ 0, & \text{otherwise,} \end{cases}$$

then the expectation of the number of split subsets in \mathcal{F} satisfies

$$E \left(\sum_{S \in \mathcal{F}} X_S \right) \geq \frac{1}{2} (|\mathcal{F}| - m_1),$$

Therefore, if $|\mathcal{F}| - m_1 \geq 2k$, there must exist a partition of the ground set X such that the number of split subsets in \mathcal{F} is at least k . This completes the proof of the lemma. □

The following lemma shows that we can directly include subsets of at least k elements in our split subsets while we are solving the unweighted SET SPLITTING problem.

Lemma 2. *Let (X, \mathcal{F}, k) be an instance of the unweighted SET SPLITTING problem, and let S be a subset in \mathcal{F} that contains at least k elements. Then there is a partition of X that splits k subsets in \mathcal{F} if and only if there is a partition of X that splits $k - 1$ subsets in $\mathcal{F} - \{S\}$.*

Proof. Suppose that there is a partition (X_l, X_r) of X that splits k subsets in \mathcal{F} . Then it is obvious that (X_l, X_r) splits (at least) $k - 1$ subsets in $\mathcal{F} - \{S\}$.

On the other hand, suppose that there is a partition (X_l, X_r) of X that splits $k - 1$ subsets S_1, \dots, S_{k-1} in $\mathcal{F} - \{S\}$. Let $l_i, r_i \in S_i, l_i \in X_l$, and $r_i \in X_r$, for all $1 \leq i \leq k - 1$. Since S has at least k elements, there are at least two elements l and r in S such that $l \notin \{r_1, \dots, r_{k-1}\}$ and $r \notin \{l_1, \dots, l_{k-1}\}$. Therefore, if we modify the partition (X_l, X_r) to enforce l in X_l and r in X_r (note that this modification still keeps l_i in X_l and r_i in X_r for all $1 \leq i \leq k - 1$), then the new partition splits the subset S , as well as the $k - 1$ subsets S_1, \dots, S_{k-1} in $\mathcal{F} - \{S\}$. In consequence, the new partition of the ground set X splits (at least) k subsets in the collection \mathcal{F} . □

Now we are ready to state our first kernelization result.

Theorem 1. *Given an instance (X, \mathcal{F}, k) of the unweighted SET SPLITTING problem, we can construct in time $O(N + 2k^2)$, where N is the input size in terms of (X, \mathcal{F}, k) , a kernel (X', \mathcal{F}', k') such that $|\mathcal{F}'| < 2k, k' \leq k, |X'| < 2k^2$, and that each subset in \mathcal{F}' has at most $k - 1$ elements.*

Proof. We use the following procedure to find the kernel: given an instance (X, \mathcal{F}, k) of the unweighted SET SPLITTING problem, we delete each subset that has only one element, also delete each subset that has at least k elements and decrease k by 1. Let the resulting instance be (X', \mathcal{F}', k') . If $|\mathcal{F}| \geq 2k$, then

by Lemma 1 (note that \mathcal{F}' contains no subsets of one element), we know that the given instance is a “Yes” instance; otherwise we have $|\mathcal{F}'| < 2k$, $k' \leq k$. Moreover, since we have removed all subsets of at least k elements, every subset in \mathcal{F}' contains at most $k - 1$ elements. In consequence, $|X'| < 2k^2$. The time to find this kernel is obviously $O(N + 2k^2)$. □

Theorem 1 improves the time complexity of the kernelization algorithm given in [6], which takes time $O(N + n^4)$, as well as that given in [14], which takes time $O(N + n^2)$, where n is a number satisfying $|\mathcal{F}| = O(n)$ and $|X| = O(n)$.

From intuition, when we randomly partition X into $X_l \cup X_r$ such that each element in X has a probability of $1/2$ to be assigned to X_l and a probability of $1/2$ to be assigned to X_r , a big subset has more chance to be split. This is true, if many subsets in \mathcal{F} have many elements, we can obtain a better kernel, or a kernel that has fewer subsets in \mathcal{F}' .

Lemma 3. *Let (X, \mathcal{F}, k) be an instance of the unweighted SET SPLITTING problem. Suppose the number of subsets that have i elements is m_i for $1 \leq i \leq k - 1$ and the number of subsets that have at least k elements is m'_k . If $\sum_{i=2}^{k-1} \frac{2^i - 2}{2^i} m_i + m'_k \geq k$, then a partition of X exists that splits at least k subsets in \mathcal{F} .*

Proof. Let $S_1, \dots, S_{m'_k}$ be the subsets in \mathcal{F} that have at least k elements and let $\mathcal{F}_{<k} = \mathcal{F} - \{S_1, \dots, S_{m'_k}\}$.

We use a randomized process to partition X into (X_l, X_r) and let each element in X go to X_l with a probability of $1/2$ and go to X_r with a probability of $1/2$, then for any subset $S \in \mathcal{F}_{<k}$ that has i elements:

$$Pr(S \text{ is split}) = \frac{2^i - 2}{2^i}.$$

If we let:

$$X_S = \begin{cases} 1, & \text{if } S \text{ is split,} \\ 0, & \text{otherwise.} \end{cases}$$

then the expectation of the number of split subsets in $\mathcal{F}_{<k}$ satisfies

$$E \left(\sum_{S \in \mathcal{F}_{<k}} X_S \right) = \sum_{i=1}^{k-1} \sum_{|S|=i} E(S \text{ is split}) = \sum_{i=1}^{k-1} \frac{2^i - 2}{2^i} m_i.$$

So there exists a partition of X such that the number of subsets in $\mathcal{F}_{<k}$ that are split is at least $\sum_{i=1}^{k-1} \frac{2^i - 2}{2^i} m_i$. Hence if $\sum_{i=1}^{k-1} \frac{2^i - 2}{2^i} m_i \geq k - m'_k$, there must exist a partition of X such that $k - m'_k$ subsets in $\mathcal{F}_{<k}$ are split. By repeatedly using Lemma 2, we conclude that there is a partition of X that splits $k - m'_k + 1$ subsets in $\mathcal{F}_{<k} \cup \{S_1\}$; there is a partition of X that splits $k - m'_k + 2$ subsets in $\mathcal{F}_{<k} \cup \{S_1, S_2\}$; and so on. In consequence, there is a partition of X that splits k subsets in $\mathcal{F}_{<k} \cup \{S_1, \dots, S_{m'_k}\} = \mathcal{F}$. □

Using the procedure that is similar to Theorem 1, but counting the number of subsets in \mathcal{F} of different size and using the result of Lemma 3, we have the following Theorem that is stronger than Theorem 1.

Theorem 2. *Given an instance (X, \mathcal{F}, k) of SET SPLITTING problem, suppose the number of subsets in \mathcal{F} that have i elements is m_i for $1 \leq i \leq k - 1$ and the number of subsets that have at least k elements is m'_k . Then in time $O(N + 2k^2)$, where N is the input size in terms of (X, \mathcal{F}, k) , we can find a kernel (X', \mathcal{F}', k') such that $|\mathcal{F}'| < 2k - \sum_{i=3}^{k-1} \frac{2^{i-1}-2}{2^{i-1}} m_i - 2m'_k$, that $k' \leq k$, that each subset in \mathcal{F}' has at most $k - 1$ elements, and that $|X'| < 2k^2$.*

Proof. We use the procedure that is similar to Theorem 1 to find the kernel: given an instance (X, \mathcal{F}, k) of the unweighted SET SPLITTING problem, we delete each subset that has only one element, also delete each subset that has more than $k - 1$ elements and decrease k by 1. In this procedure, we also obtain m_i for $1 \leq i \leq k - 1$ and m'_k . By Lemma 3, if $\sum_{i=2}^{k-1} \frac{2^i-2}{2^i} m_i + m'_k \geq k$, we know the given instance is a “Yes” instance; otherwise $\sum_{i=2}^{k-1} \frac{2^i-2}{2^i} m_i + m'_k < k$, i.e. $|\mathcal{F}'| = \sum_{i=2}^{k-1} m_i < 2k - \sum_{i=3}^{k-1} \frac{2^{i-1}-2}{2^{i-1}} m_i - 2m'_k$. So we find a kernel (X', \mathcal{F}', k') that $|\mathcal{F}'| < 2k - \sum_{i=3}^{k-1} \frac{2^{i-1}-2}{2^{i-1}} m_i - 2m'_k$, that $k' \leq k$, that each subset in \mathcal{F}' has at most $k - 1$ elements, and that $|X'| < 4k^2$. The time needed to find the kernel is $O(N + 2k^2)$. □

3 A Randomized Algorithm for the Weighted SET SPLITTING Problem

For the unweighted SET SPLITTING problem, Lokshatanov and Sloper [14] have currently the best parameterized algorithm, whose running time is bounded by $O^*(2.65^k)$. Unfortunately, their method does not seem to be extendable to the weighted case, neither do the methods presented in [6,7] for the unweighted case. In fact, no previous work is known that gives a fixed parameter tractable algorithm for the weighted SET SPLITTING problem.

In this section, we present a randomized algorithm to solve the weighted SET SPLITTING problem. Our basic idea is that if a given instance (X, \mathcal{F}, k) of the weighted SET SPLITTING problem has a partition of the ground set X that splits k subsets in the collection \mathcal{F} , then there exists a subset X' of at most $2k$ elements in X such that a proper partition of the elements in X' can split at least k subsets in \mathcal{F} . If we use a randomized process to partition X into (X_l, X_r) and let each element in X go to X_l with a probability of $1/2$ and go to X_r with a probability of $1/2$, then the probability that the elements in X' are partitioned properly is at least $2/2^{2k}$. Thus, if we try $O(4^k)$ times, we have a good chance to find a proper partition if it exists. In fact, we can do better than this in a randomized algorithm.

Theorem 3. *The weighted SET SPLITTING problem can be solved by a randomized algorithm of running time $O(2^k N)$, where N is the input size in terms of (X, \mathcal{F}, k) .*

Proof. Let (X, \mathcal{F}, k) be an instance of the weighted SET SPLITTING problem. Suppose that there is a partition of the ground set X that splits at least k subsets

Algorithm-1 SetSplitting(X, \mathcal{F}, k)

input: A ground set X , a collection \mathcal{F} of subsets of X , and an integer k

output: A partition (X_l, X_r) of X and k subsets in \mathcal{F} that are split by

(X_l, X_r) , or report "no partition of X splits k subsets in \mathcal{F} ".

1. $Q_0 = \emptyset$;
2. **for** $i = 1$ **to** $10 \cdot 2^k$ **do**
- 2.1. randomly partition X into X_l and X_r such that each element in X has a probability $1/2$ in X_l and a probability $1/2$ in X_r ;
- 2.2. let Q be the collection of subsets in \mathcal{F} that are split by (X_l, X_r) ;
- 2.3. **if** Q contains at least k subsets **then**
delete all but the k subsets of maximum weight in Q ;
- 2.4. **if** the weight sum of subsets in Q is larger than that in Q_0 **then**
 $Q_0 = Q$;
3. return Q_0 .

Fig. 1. Randomized algorithm for WEIGHTED SET SPLITTING problem

in the collection \mathcal{F} . Let (X_l, X_r) be a partition of the ground set X and let S_1, \dots, S_k be k subsets in the collection \mathcal{F} that are split by the partition (X_l, X_r) , such that the weight sum of S_1, \dots, S_k is the maximum over all collections of k subsets in \mathcal{F} that can be split by a partition of X . More specifically, let $(l_1, r_1), \dots, (l_k, r_k)$ be k pairs of elements in the ground set X such that $l_i, r_i \in S_i$, $l_i \in X_l$, and $r_i \in X_r$ for all $1 \leq i \leq k$. Note that it is possible that $l_i = l_j$ or $r_i = r_j$ for some $i \neq j$. In consequence, each of the sets $\{l_1, \dots, l_k\}$ and $\{r_1, \dots, r_k\}$ may contain fewer than k elements.

We construct a graph $G = (V, E)$, where $V = \{l_1, l_2, \dots, l_k\} \cup \{r_1, r_2, \dots, r_k\}$ and $E = \{(l_i, r_i) \mid 1 \leq i \leq k\}$. It is obvious that G is a bipartite graph with the left vertex set $L = \{l_1, l_2, \dots, l_k\}$ and the right vertex set $R = \{r_1, r_2, \dots, r_k\}$. Suppose that the graph G has t connected components C_1, \dots, C_t , where $C_i = (V_i, E_i)$, with $n_i = |V_i|$ and $m_i = |E_i|$, for $1 \leq i \leq t$. Then $n_i \leq m_i + 1$ for $1 \leq i \leq t$ and $\sum_{i=1}^t m_i = k$. If we use a randomized process to partition X into (X_l, X_r) and let each element in X go to X_l with a probability of $1/2$ and go to X_r with a probability of $1/2$, then for each connected component C_i of the graph G , the probability that the vertex set V_i of C_i is properly partitioned, i.e., either $L \cap V_i \subseteq X_l$ and $R \cap V_i \subseteq X_r$, or $R \cap V_i \subseteq X_l$ and $L \cap V_i \subseteq X_r$, is $2/2^{m_i}$. Therefore, the total probability that the vertex set V_i for every connected component C_i is properly partitioned, i.e., that the pair (l_i, r_i) intersects with both X_l and X_r for all $1 \leq i \leq k$, is not less than

$$\frac{2}{2^{n_1}} \cdot \frac{2}{2^{n_2}} \cdots \frac{2}{2^{n_t}} \geq \frac{2}{2^{m_1+1}} \cdot \frac{2}{2^{m_2+1}} \cdots \frac{2}{2^{m_t+1}} = \frac{2^t}{2^{\sum_{i=1}^t m_i + t}} = \frac{1}{2^k}.$$

The algorithm in Figure 1 implements the above idea. By the above discussion, each random partition (X_l, X_r) constructed in step 2.1 has a probability of at least $1/2^k$ to split the k subsets S_1, \dots, S_k (recall that S_1, \dots, S_k are the k subsets in \mathcal{F} whose weight sum is the maximum over all collections of k subsets

in \mathcal{F} that are split by a partition of X). Since step 2 loops $10 \cdot 2^k$ times, with a probability of at least

$$1 - \left(1 - \frac{1}{2^k}\right)^{10 \cdot 2^k} \geq 99.99\%,$$

one partition (X_l, X_r) constructed by step 2.1 splits the k subsets S_1, \dots, S_k . For this partition (X_l, X_r) , steps 2.2-2.4 produces a collection \mathcal{Q} of k subsets in \mathcal{F} whose weight sum is the maximum over all collections of k subsets in \mathcal{F} that are split by a partition of the ground set X .

Since each execution of steps 2.1-2.4 obviously takes time $O(N)$, we conclude that the running time of the algorithm **SetSplitting** is bounded by $O(2^k N)$. \square

Obviously, the algorithm **SetSplitting** of running time $O(2^k N)$ can be directly used to solve the unweighted SET SPLITTING problem, and its running time significantly improves the previous best algorithm [14] for the unweighted SET SPLITTING problem. Moreover, the algorithm **SetSplitting** is much simpler than the one presented in [14].

4 Derandomization

For many parameterized NP-Complete problems, such as the k -PATH, 3-SET PACKING, and 3D-MATCHING problems, the solution is a subset of size $O(k)$. If we have a way to partition this subset properly, we only need to deal with several subproblems of smaller solution sizes, such as two $(\frac{k}{2})$ -path problems. For a given subset of size k , or a k -subset, we have 2^k ways of partitioning it into two subsets. If we know this k -subset, we can enumerate all possible partitions in time $O(2^k)$. But usually, we do not know this k -subset. In this section, we present a set partition method such that given any set V of size n and an integer k , we can get a partition family $\mathcal{F}(V, k)$ to partition any k -subset S of V in all 2^k partition ways. The method was described in an extended abstract by Noar, Schulman, and Srinivasan [15], with many details omitted. In this section, we provide further details and concrete constructions for the parts related to our problems, and show how these techniques can be used to derive efficient parameterized algorithms for the weighted SET SPLITTING problem.

Lemma 4 ([1]). *Suppose $n = 2^d - 1$ and $k = 2t + 1 \leq n$. Then there exists a uniform probability space Ω of size $2(n + 1)^t$ and k -wise independent random variables ξ_1, \dots, ξ_n over Ω each of which takes the values 0 and 1 with probability $1/2$.*

The Ω in lemma 4 is a $n \times 2(n + 1)^{\lfloor k/2 \rfloor}$ matrix that takes only value 0 and 1. In our case, we can see each row as an element in a set V of size n and each column as a partition of V . By the lemma 4, for each k -subset S of V and each partition of S , there are $\frac{1}{2^k} 2(n + 1)^{\lfloor k/2 \rfloor}$ columns to partition S this way. This result was used by paper [15] to prove Lemma 7.

Lemma 5. a) *Given two sets A and B and a relation $R \subset A \times B$, if $(a, b) \in R$, we say element a in set A relates to element b in set B . If each element in set A relates to at least fraction p of elements in set B , then there is an element b' in set B such that there are at least fraction p of elements in A that relate to it.*

b) *For any given n and k and $k \leq n$, if $t > ek2^k(\log n + 1)$, then $\binom{n}{k}2^k(1 - \frac{1}{2^k})^t < 1$.*

Proof. **a)** Each element in set A relates to $p|B|$ elements in set B , then there are $p|B||A|$ elements in set $R \subset A \times B$. So there is at least one element in set B that there exist $p|A|$ elements in set A that relate to it.

b) Because $\binom{n}{k}2^k(1 - \frac{1}{2^k})^t < n^k2^k(1 - \frac{1}{2^k})^t$, just let $t = ek2^k(\log n + 1)$, you can verify the inequality: $\binom{n}{k}2^k(1 - \frac{1}{2^k})^t < 1$ for $t > ek2^k(\log n + 1)$. □

Lemma 6. [9] *Given $U = \{1, 2, \dots, n\}$ with $p = n + 1$ a prime number, and given $W \subset U$ with $|W| = k$, then there exists a $k' \in U$, such that mapping $x \rightarrow (k'x \bmod p) \bmod k^2$ is one-to-one from W to $\{0, 1, 2, \dots, k^2 - 1\}$.*

Lemma [6] was proposed by Fredman, Komlos, and Szemerédi. It was used in Theorem [4] to reduce the size of ground set V from n to k^2 . Now we give the most important Lemma in the construction of partition family $\mathcal{F}(V, k)$.

Lemma 7. [15] *For any given set V of size n and $k \leq n$, there is a partition function family $\mathcal{F}(V, k)$ of V such that for any k -subset V' of V , any partition of V' can be done by a partition function in $\mathcal{F}(V, k)$. Furthermore the cardinality of $\mathcal{F}(V, k)$ is $O(k2^k \log n)$. And $\mathcal{F}(V, k)$ can be constructed deterministically in time $O(\binom{n}{k}k2^{2k}n^{\lfloor \frac{k}{2} \rfloor})$.*

Proof. For each k -subset S of V and each partition p of S , we make a pair (S, p) and let A be a set of all such pairs; thus $|A| = \binom{n}{k}2^k$. Let $B = \Omega$, where Ω is the probability space as in Lemma [4]. Any $(S, p) \in A$ and $p' \in B$, if p' agree the partition p in subset S (We say partition p' covers pair (S, p)), let $((S, p), p') \in R \subset A \times B$.

By Lemma [4], each $(S, p) \in A$, if we randomly choose $p' \in B$, the probability that p' covers (S, p) is $\frac{1}{2^k}$, i.e. there are $|B|\frac{1}{2^k}$ elements in B that cover (S, p) . So by Lemma [5]a), there is a partition p' in B such that p' covers $|A|\frac{1}{2^k}$ pairs in A . We put this p' into $\mathcal{F}(V, k)$ and delete the pairs that are covered by p' . The number of pairs remaining in A is $|A|(1 - \frac{1}{2^k})$. If we do this process again, the number of pairs remaining in A becomes $|A|(1 - \frac{1}{2^k})^2$. By Lemma [5]b), if we repeat this process $ek2^k(\log n + 1) + 1$ times, the number of pairs remaining in A is less than 1. This means that every pair in A is covered by some partitions in $\mathcal{F}(V, k)$ and the cardinality of $\mathcal{F}(V, k)$ is $O(k2^k \log n)$.

Because $|A| = \binom{n}{k}2^k$, $|B| = 2n^{\lfloor k/2 \rfloor}$, the time to construct $\mathcal{F}(V, k)$ is:

$$\sum_{i=0}^{O(k2^k \log n)} |A||B| \left(1 - \frac{1}{2^k}\right)^i < O\left(\binom{n}{k}k2^{2k}n^{\lfloor \frac{k}{2} \rfloor}\right).$$

□

Lemma 7 gave a way to construct partition family $\mathcal{F}(V, k)$ of cardinality $O(k2^k \log n)$, but the time complexity is too large. In fact, Lemma 7 is only a middle step result, the final construction is in the theorem 4.

Theorem 4. [15] *For any given set V of size n and $k \leq n$, there is a partition function family $\mathcal{F}(V, k)$ of V such that for any k -subset V' of V , any partition of V' can be done by a partition function in $\mathcal{F}(V, k)$ applying to V' . The family $\mathcal{F}(V, k)$ contains $O(n2^{k+12 \log^2 k-4 \log k})$ partitions of V , and can be constructed in time $O(n2^{k+12 \log^2 k-4 \log k})$.*

Proof. The construction of $\mathcal{F}(V, k)$ includes three steps:

First step: Mapping V to $\{0, 1, \dots, k^2 - 1\}$ by function $f_i(x) = (ix \bmod p) \bmod k^2$, where p is a prime number between n and $2n$. From number theory, this prime number p must exist. By Lemma 6 for a fixed k -subset V' of V , there is $1 \leq i < 2n$ such that every element in V' is mapped to different value in $\{0, 1, \dots, k^2 - 1\}$ by $f_i(x)$. This step has $O(n)$ branches.

Second step: Partitioning $\{0, 1, \dots, k^2 - 1\}$ into $4 \log k$ subsets $V_1, V_2, \dots, V_{4 \log k}$ such that each subset has $\frac{k}{4 \log k}$ elements of the fixed k -subset V' . This step has $O(k^{8(\log k-1)})$ branches.

Third step: Enumerating all possible combinations of $\mathcal{F}(V_i, \frac{k}{4 \log k})$ that obtained by using the method from Lemma 7. The number of branches in this step is $O((\frac{k}{4 \log k})^{4 \log k} (2 \log k)^{4 \log k} 2^k)$.

Because the time to construct all $\mathcal{F}(V_i, \frac{k}{4 \log k})$ is $O(k2^{\frac{3k}{4} + \frac{k}{2 \log k}})$, both the cardinality of $\mathcal{F}(V, k)$ and the time to construct $\mathcal{F}(v, k)$ are $O(n2^{k+12 \log^2 k-4 \log k})$. □

Using the result of Theorem 4, we derive the first fixed parameter tractable algorithm for the weighted SET SPLITTING problem, as given below.

Theorem 5. *The WEIGHTED SET SPLITTING problem can be solved determinately by an algorithm of running time $O((|\mathcal{F}|k + n)n2^{2k+12 \log^2 k})$.*

Proof. Given an instance of WEIGHTED SET SPLITTING problem, if it has at least k subsets in \mathcal{F} that can be split by a partition of X , then there exist two sets $\{l_1, l_2, \dots, l_k\}$ and $\{r_1, r_2, \dots, r_k\}$. If we can partition $\{l_1, l_2, \dots, l_k\}$ and $\{r_1, r_2, \dots, r_k\}$ into two different groups, then there are at least k subsets in \mathcal{F} that are split by this partition. Using the partition family $\mathcal{F}(V, 2k)$ that is constructed as in Theorem 4, we can do it. So we replace line 1 in Algorithm-1 by looping all partitions in this $\mathcal{F}(V, 2k)$, then we can find a partition to split at least k subsets in \mathcal{F} determinately if this partition exists. □

Finally, we remark that the partition function family $\mathcal{F}(V, k)$ can also be used to derandomize the algorithms for k -PATH, 3D-MATCHING and 3-SET PACKING problems that are based on the divide-and-conquer techniques [5, 12].

References

1. Alon, N., Babai, L., Itai, A.: A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms* 7, 567–683 (1986)
2. Andersson, G., Engebretsen, L.: Better approximation algorithms and tighter analysis for set splitting and not-all-equal sat. In: *ECCCTR: Electronic colloquium on computational complexity* (1997)
3. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Proti, M.: *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, Heidelberg (1999)
4. Chen, J., Kanj, I.: Improved Exact Algorithms for Max-Sat. *Discrete Applied Mathematics* 142, 17–27 (2004)
5. Chen, J., Lu, S., Sze, S., Zhang, F.: Improved algorithms for path, matching, and packing problems. In: *SODA*, pp. 298–307 (2007)
6. Dehne, F., Fellows, M., Rosamond, F.: An FPT Algorithm for Set Splitting. In: Bodlaender, H.L. (ed.) *WG 2003*. LNCS, vol. 2880, pp. 180–191. Springer, Heidelberg (2003)
7. Dehne, F., Fellows, M., Rosamond, F., Shaw, P.: Greedy localization, iterative compression, modeled crown reductions: new FPT techniques, and improved algorithm for set splitting, and a novel $2k$ kernelization of vertex cover. In: Downey, R.G., Fellows, M.R., Dehne, F. (eds.) *IWPEC 2004*. LNCS, vol. 3162, pp. 127–137. Springer, Heidelberg (2004)
8. Downey, R., Fellows, M.: *Parameterized Complexity*. Springer, Heidelberg (1999)
9. Fredman, M., Komlos, J., Szemerédi, E.: Storing a sparse table with $O(1)$ worst case access time. *Journal of the ACM* 31, 538–544 (1984)
10. Garey, M., Johnson, D.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco (1979)
11. Kann, V., Lagergren, J., Panconesi, A.: Approximability of maximum splitting of k -sets and some other apx-complete problems. *Information Processing Letters* 58(3), 105–110 (1996)
12. Kneis, J., Molle, D., Richter, S., Rossmanith, P.: Divide-and-color. In: Fomin, F.V. (ed.) *WG 2006*. LNCS, vol. 4271, pp. 58–67. Springer, Heidelberg (2006)
13. Liu, Y., Lu, S., Chen, J., Sze, S.: Greedy localization and color-coding: Improved Matching and Packing Algorithms. In: Bodlaender, H.L., Langston, M.A. (eds.) *IWPEC 2006*. LNCS, vol. 4169, pp. 84–95. Springer, Heidelberg (2006)
14. Lokshtanov, D., Sloper, C.: Fixed parameter set splitting, linear kernel and improved running time. *Algorithms and Complexity in Durham 2005*, King’s College Press, *Texts in Algorithmics* 4, 105–113 (2005)
15. Naor, M., Schulman, L., Srinivasan, A.: Splitters and near-optimal derandomization. In: *FOCS*, pp. 182–190 (1995)
16. Zhang, H., Ling, C.: An improved learning algorithm for augmented naive bayes. In: Cheung, D., Williams, G.J., Li, Q. (eds.) *PAKDD 2001*. LNCS (LNAI), vol. 2035, pp. 581–586. Springer, Heidelberg (2001)
17. Zwick, U.: Approximation algorithms for constraint satisfaction problems involving at most three variables per constraint. In: *SODA*, pp. 201–220 (1998)
18. Zwick, U.: Outward rotations: A tool for rounding solutions of semidefinite programming relaxation, with applications to max cut and other problem. In: *STOC*, pp. 679–687 (1999)

An $\frac{8}{5}$ -Approximation Algorithm for a Hard Variant of Stable Marriage

Robert W. Irving* and David F. Manlove*

Department of Computing Science, University of Glasgow, Glasgow G12 8QQ, UK
rwi@dcs.gla.ac.uk, davidm@dcs.gla.ac.uk

Abstract. When ties and incomplete preference lists are permitted in the Stable Marriage problem, stable matchings can have different sizes. The problem of finding a maximum cardinality stable matching in this context is known to be NP-hard, even under very severe restrictions on the number, size and position of ties. In this paper, we describe a polynomial-time $\frac{8}{5}$ -approximation algorithm for a variant in which ties are on one side only and at the end of the preference lists. The particular variant is motivated by important applications in large scale centralized matching schemes.

1 Introduction

Background

An instance of the Stable Marriage problem with Ties and Incomplete Lists (SMTI) comprises a set of n_1 men m_1, \dots, m_{n_1} and a set of n_2 women w_1, \dots, w_{n_2} . Each person has a *preference list* consisting of a subset of the members of the opposite sex, his or her *acceptable partners*, listed in order of preference, with ties, consisting of two or more persons of equal preference, permitted. If man m and woman w appear on each other's preference list then (m, w) is called an *acceptable pair*. If w precedes w' on m 's list then m is said to *prefer* w to w' , while if w and w' appear together in a tie on m 's list then m is said to be *indifferent* between w and w' .

A *matching* is a set M of acceptable pairs so that each person appears in at most one pair of M . If M is a matching and $(m, w) \in M$ we write $w = M(m)$ and $m = M(w)$, and we say that m and w are *partners* in M . A pair (m, w) is a *blocking pair* for M , or *blocks* M , if m is either unmatched in M or prefers w to $M(m)$, and simultaneously w is either unmatched in M or prefers m to $M(w)$. A matching for which there is no blocking pair is said to be *stable*.

SMTI is an extension of the classical Stable Marriage problem (SM) introduced by Gale and Shapley [2]. In the classical case, the numbers of men and women are equal, all preference lists are complete, i.e., they contain all members of the opposite sex, and ties are not permitted, i.e., all preferences are strict. Gale and Shapley proved that, for every instance of SM, there is at least one

* Supported by EPSRC research grant EP/E011993/1.

stable matching, and they described an $O(n^2)$ time algorithm to find such a matching; this has come to be known as the *Gale-Shapley algorithm*.

This algorithm is easily extended to the case in which the numbers of men and women differ and preference lists are incomplete (SMI – Stable Marriage with Incomplete lists); it has complexity $O(a)$ in this case, where a is the number of acceptable pairs [4]. In the case of SMI, not everyone need be matched in a stable matching. In general, for a given instance of SMI, there may be many stable matchings – exponentially many in extreme cases – but all stable matchings have the same size and match exactly the same sets of men and women [15,3].

The Gale-Shapley algorithm may be applied from either the men’s side or the women’s side, and in general these two applications will produce different stable matchings. When applied from the men’s side, the *man-optimal* stable matching is found; in this, every man has the best partner that he can have in any stable matching, and every woman the worst. When the algorithm is applied from the women’s side, the *woman-optimal* stable matching results, with analogous properties. Exceptionally, the man-optimal and woman-optimal stable matchings may coincide, in which case this is the unique stable matching, but in general there may be other stable matchings – possibly exponentially many – between these two extremes. However, for a given instance of SMI, all stable matchings have the same size and match exactly the same sets of men and women [15,3].

The situation for SMTI is dramatically different. Again, at least one stable matching exists for every instance, and can be found in $O(a)$ time by breaking all ties in an arbitrary way to give an instance of SMI, and applying the Gale-Shapley algorithm to that instance. However, the ways in which ties are broken can significantly affect the outcome. In particular, not all stable matchings need be of the same size, and in the most extreme case, there may be two stable matchings M and M' with $|M| = 2|M'|$. Furthermore, the problem of finding a stable matching of maximum cardinality for an instance of SMTI – problem MAX-SMTI – is NP-hard [13]. This hardness result holds even under severe restrictions, for example, if the ties are on one side only, each list contains at most one tie, and that tie, if present, is at the end of the list.

Practical Applications

The practical importance of stable matching problems arises from their application in the assignment of applicants to positions in various job markets. The many-one version of the problem has come to be known as the Hospitals/Residents problem (HR) because of its widespread application in the medical employment domain [15,9,14,11,16]. In an instance of HR, each resident has a preference list of acceptable hospitals, while each hospital has a preference list of acceptable residents together with a quota of positions. A matching M is a set of acceptable resident-hospital pairs such that each resident is in at most one pair, and each hospital is in a number of pairs that is bounded by its quota. If pair (r, h) is in M , we write $h = M(r)$ and $r \in M(h)$, so that $M(h)$ is a *set* of residents for each hospital h . A matching is stable if it admits no blocking pair, i.e., an acceptable pair (r, h) such that r is either unmatched in M or prefers h

Men's preferences	Women's preferences
$m_1 : w_1 \ w_2$	$w_1 : (m_1 \ m_2)$
$m_2 : w_1$	$w_2 : m_1$

Fig. 1. An instance of SSMTI with stable matchings of sizes 1 and 2.

to $M(r)$, and simultaneously h is either under quota or prefers r to at least one member of $M(h)$.

As in the case of SMI, all stable matchings for an instance of HR have the same size, and so-called *resident-optimal* and *hospital-optimal* stable matchings can be found by applying an extended version of the Gale-Shapley algorithm from the residents' side or the hospitals' side respectively. But if ties are allowed in the preference lists - the Hospitals/Residents problem with Ties (HRT) - then as in the case of SMTI, stable matchings can have different sizes, and it is NP-hard to find a stable matching of maximum size, even under severe restrictions on the number, size, and position of ties.

Variants of the extended Gale-Shapley algorithm are routinely used in a number of countries, including the United States [14], Canada [1] and Scotland [16], to allocate graduating medical students to hospital posts, and in a variety of other countries and contexts. In large scale matching schemes of this kind, participants, particularly large popular hospitals, may not be able to provide a genuine strict preference order over what may be a very large number of applicants, so that HRT is a more appropriate model than HR. If artificial tie-breaking is carried out, either by the participant, because a strictly ordered list is required by the matching scheme, or by the administrators of the scheme, prior to running an algorithm that requires strict preferences, then the size of the resulting stable matching is likely to be affected. Breaking ties in different ways will typically yield stable matchings of different sizes; what would be ideal would be to find a way of breaking the ties that maximizes the size of the resulting stable matching, but the NP-hardness of this problem makes this an objective that is unlikely to be feasible.

A special case of HRT arises if residents are required to strictly rank their chosen hospitals but hospitals are asked to rank only as many of their applicants as they reasonably can, and then place the remainder in a single tie at the end. For example this variant has been employed in the Scottish Foundation Allocation Scheme (SFAS). The correspondingly restricted version of SMTI, where all men's lists are strict and women's lists may contain one tie at the end, is the special case in which all quotas are equal to one. We refer to these restricted versions of SMTI and HRT as *Special SMTI/HRT* (SSMTI/SHRT), and use the terms MAX-SSMTI and MAX-SHRT for the problems of finding maximum cardinality stable matchings in these cases, which remain NP-hard problems [13].

Figure 1 shows an example of SSMTI in which there are two stable matchings, $M_1 = \{(m_1, w_2), (m_2, w_1)\}$ of size 2 and $M_2 = \{(m_1, w_1)\}$ of size 1. A tie in the preference lists is indicated by parentheses.

Related Results

It is trivial to establish that there can be at most a factor of two difference between the sizes of a minimum and maximum cardinality stable matching for an instance of SMTI, and as a consequence, breaking ties arbitrarily and applying the Gale-Shapley algorithm gives a 2-approximation algorithm for MAX-SMTI. A number of improved approximation algorithms for versions of SMTI have recently been proposed.

For the general case, Iwama et al [11] gave an algorithm with a performance guarantee of $(2 - c/\sqrt{n})$, (for the case of n men and n women), for a constant c . Very recently, Iwama et al [12] gave the first approximation algorithm for the general case with a constant performance guarantee better than 2, namely $\frac{15}{8}$. From the inapproximability point of view, Halldórsson et al showed the problem to be APX-complete [5], and gave a lower bound of $\frac{21}{19}$ on any polynomial-time approximation algorithm (assuming $P \neq NP$) [6]. This lower bound applies even to MAX-SSMTI.

As far as special cases are concerned, Halldórsson et al [6] gave a $(2/(1+L^{-2}))$ -approximation algorithm for the case where all ties are on one side, and are of length at most L – so, for example, this gives a bound of $\frac{8}{5}$ when all ties are of length 2. If ties are on both sides and restricted to be of length 2, a bound of $\frac{13}{7}$ is shown in [6]. Halldórsson et al [7] also described a randomized algorithm with an expected performance guarantee of $\frac{10}{7}$ for the same special case under the additional restriction that there is at most one tie per list.

The Contribution of This Paper

In this paper, we focus on the problem MAX-SSMTI described above, and give a polynomial-time $\frac{8}{5}$ -approximation algorithm for this case. The algorithm is relatively easy to extend to MAX-HRT, and the same $\frac{8}{5}$ performance guarantee holds in this more general setting. We also show that this performance guarantee is the best that can be proved for the algorithm by providing an example for which this bound is realised.

2 The Algorithm

In what follows, we assume that each man's preference list is strict, and each woman's preference list is strict except for a tie (of length ≥ 1) at the end. The algorithm consists of three phases. The first phase is a variant of the Gale-Shapley algorithm for the classical stable marriage problem, applied from the women's side, but with women proposing only as far as the tie (if any) in their list. This results in a provisional matching involving precisely those men who received proposals. The second phase adds to this provisional matching a maximal set of acceptable pairs from among the remaining men and women. Finally, in Phase 3 all ties are broken, favouring unmatched men over matched men, and the standard Gale-Shapley algorithm is run to completion on the resulting instance of SMI.

```

assign each person to be free;
while (some woman  $w$  is free) and ( $w$  has a non-empty list)
  and ( $w$  has an untied man  $m$  at the head of her list) {
     $w$  proposes, and becomes engaged to  $m$ ;
    for each successor  $w'$  of  $w$  on  $m$ 's list {
      if  $w'$  is engaged to  $m$ 
        break the engagement, so that  $w'$  becomes free;
      delete the pair  $(m, w')$  from the preference lists;
    }
  }

```

Fig. 2. Phase 1 of Algorithm SSMTI-APPROX

Phase 1 of Algorithm SSMTI-APPROX

The first phase of the algorithm is a variant of the Gale-Shapley algorithm for the classical stable marriage problem, applied from the women's side. During this phase, zero or more deletions are made from the preference lists – by the deletion of the pair (m, w) , we mean the removal of w from m 's list and the removal of m from that of w . Initially, everyone is free. During execution of the algorithm, a woman may alternate between being free and being engaged, but once a man becomes engaged, he remains in that state, though the identity of his partner may change over time. A free woman w who still has an untied man on her current list proposes to the first such man and becomes (at least temporarily) engaged to that man. When a man m receives a proposal from woman w , he rejects his current partner (if any), setting her free, and all pairs (m, w') such that m prefers w to w' are deleted. This phase of the algorithm is summarised in Figure 2.

When Phase 1 of the algorithm terminates on a given instance I , a woman w 's preference list must be in one of three possible states – it may be empty, it may consist of a single tie, or it may have a unique untied man m at its head. In the latter case, it is clear that m cannot be the unique man at the head of any other woman's list, and that w is the last entry in m 's list.

Lemma 1. *On termination of Phase 1 of Algorithm SSMTI-APPROX,*

- (i) *no deleted pair can belong to a stable matching;*
- (ii) *if man m is the unique man at the head of some woman's list then m is matched in every stable matching.*

Proof. (i) Suppose that (m, w) is a deleted pair that belongs to a stable matching M , and that (m, w) was the first such pair to be deleted in an execution of Phase 1 of the algorithm. This must have happened because m received a proposal from some woman w' whom he prefers to w . Woman w' is either unmatched in M or prefers m to $M(w)$, because any pair (m', w') such that w' prefers m' to m must have been previously deleted, and by our assumption, this pair cannot be in a stable matching. Hence (m, w') blocks M , a contradiction.

$V = Y_1 \cup Q_1$;
 $E = \{(m, w) \in Y_1 \times Q_1 : (m, w) \text{ is a Phase 1 acceptable pair}\}$;
 construct the bipartite graph $G = (V, E)$;
 $K =$ a maximum cardinality matching in G ;
 for each pair $(m, w) \in K$
 promote m from the tie to the head of w 's list;
 re-activate the proposal sequence of Phase 1;

Fig. 3. Phase 2 of Algorithm SSMTI-APPROX

(ii) Suppose that m is the unique man at the head of woman w 's list, and that M is a stable matching in which m is unmatched. Then, by part (i), w is either unmatched in M or prefers m to $M(w)$, so that (m, w) blocks M , a contradiction. \square

We refer to the men who appear untied at the head of some woman's list after Phase 1 of the algorithm as the *Phase 1 X-men* and the other men as the *Phase 1 Y-men*, and we denote these sets by X_1 and Y_1 respectively. Likewise, the women who have an untied man at the head of their list are the *Phase 1 P-women* and the others are the *Phase 1 Q-women*, denoted by P_1 and Q_1 . So the engaged pairs at the end of Phase 1 constitute a perfect matching between X_1 and P_1 , and the essence of Lemma 1(ii) is that each member of X_1 is matched in every stable matching. We call the preference lists that remain after Phase 1 the *Phase 1 lists*, and if man m and woman w are in each other's Phase 1 lists, we say that (m, w) is a *Phase 1 acceptable pair*.

Phase 2 of Algorithm SSMTI-APPROX

In Phase 2 of the algorithm, we seek to increase the number of men who are guaranteed to be matched. To this end, we find a maximum cardinality matching K of Y_1 to Q_1 , where a pair (m, w) can be in this matching only if $m \in Y_1$, $w \in Q_1$, and (m, w) is a Phase 1 acceptable pair. For each pair (m, w) in K , we break the tie in w 's Phase-1 list by promoting m to the head of that list (and leaving the rest of the tie intact). We then re-activate the proposal sequence of Phase 1, which will lead to a single proposal corresponding to each pair in K , and which may result in some further deletions from the preference lists, but no rejections and no other proposals. This produces an instance I' of SSMTI that is a refinement of the original instance I – or more properly, a refinement of the variant of I that results from application of Phase 1; clearly any matching that is stable for I' is also stable for I , but not necessarily vice-versa. Phase 2 of the algorithm is summarised in Figure 3.

Lemma 2. *Every man who is untied at the head of some woman's list on termination of Phase 2 of Algorithm SSMTI-APPROX is matched in every stable matching for I' .*

Proof. The proof is completely analogous to that of Lemma 1(ii). □

Note that, while Lemma 2 can be expected, in many cases, to give a stronger lower bound on the size of a stable matching than is given by Lemma 1, this need not be the case. It is perfectly possible that the Phase 1 Q -women have only Phase 1 X -men in their preference lists, and that, as a consequence, K is the empty matching. However, we now extend the set of X -men and P -women to include those who became engaged during Phase 2. Henceforth, we use the term X -men to refer to those men who appear untied at the head of some woman's list, and the P -women are the women who have an X -man at the head of their list, after Phase 2 of the algorithm. Let x be the number of X -men and P -women, and suppose that these sets are $X = \{m_1, \dots, m_x\}$ and $P = \{w_1, \dots, w_x\}$ respectively. We also define $Y = \{m_{x+1}, \dots, m_{n_1}\}$ and $Q = \{w_{x+1}, \dots, w_{n_2}\}$, and refer to these sets as the Y -men and Q -women respectively.

Lemma 3. *Let A be a matching that is stable for I' , and let M be a maximum cardinality stable matching for I . Then*

- (i) *A Y -man who is matched in M must be matched in M with a P -woman.*
- (ii) $|M| \leq |A| + x$;
- (iii) $|M| \leq 2x$.

Proof. (i) Suppose that m is a Y -man and that $(m, w) \in M$. Then if w were a Q -woman, she must be a Phase 1 Q -woman who failed to be matched during Phase 2, and therefore the matching found in Phase 2 could have been extended by adding the pair (m, w) , contradicting its maximality.

(ii) By Lemma 2, all of the X -men are matched in A . So the only men who can be matched in M but not in A are Y -men. By (i), such a man must be matched in M with a P -woman. The inequality follows, as there are just x P -women.

(iii) Men matched in M are either X -men, and there are x of these, or Y -men matched with P -women (by (i)), and there are x of the latter, hence at most $2x$ such men in total. □

Phase 3 of Algorithm SSMTI-APPROX

Phase 3 of the algorithm involves completely breaking the remaining ties and then applying to the resulting instance of SMI the standard Gale-Shapley algorithm (or at least the extended version of that algorithm that deletes redundant entries from the preference lists - see 4). The algorithm may be applied from either the men's or women's side; as is well known, the size of the resulting matching will be the same in each case. Tie-breaking is carried out according to just one restriction, namely, for each tie, the Y -men are given priority over the X -men. In other words, each tie is resolved by listing the Y -men that it contains, in arbitrary order, followed by the X -men that it contains, again in arbitrary order. It is immediate that the algorithm produces a matching that is stable for the original instance of SMTI. For an instance I of SMTI, we denote by I'' an instance of SMI obtained by application of Phases 1 and 2 of Algorithm SSMTI-APPROX, followed by tie-breaking according to this rule. Again

```

for each woman  $w$ 
    break the tie (if any) in  $w$ 's list, placing the  $Y$ -men ahead of the  $X$ -men;
/* Now apply the standard Gale-Shapley algorithm */
assign each person to be free;
while (some man  $m$  is free) and ( $m$  has a non-empty list) {
     $w =$  the first woman on  $m$ 's list;
     $m$  proposes, and becomes engaged to  $w$ ;
    for each successor  $m'$  of  $m$  on  $w$ 's list {
        if  $m'$  is engaged to  $w$ 
            break the engagement, so that  $m'$  becomes free;
        delete the pair  $(m', w)$  from the preference lists;
    }
}
return the set  $A$  of engaged pairs;

```

Fig. 4. Phase 3 of Algorithm SSMTI-APPROX

it is immediate that a matching that is stable for I'' is also stable for I . Phase 3 of the algorithm is summarised in Figure 4.

The Performance Guarantee

Let A be a matching produced by application of Algorithm SSMTI-APPROX, and let M be a maximum cardinality stable matching for the original instance I of SSMTI. As previously established, all of the X -men are matched in A . Suppose that exactly r of the Y -men, say m_{x+1}, \dots, m_{x+r} , are matched in M but not in A . Let us call these men the *extra men* (for M), and their partners in M the *extra women*.

Lemma 4. (i) *Each extra woman is matched in A .*
(ii) *An extra woman is either matched in A to a Y -man or strictly prefers her A -partner to her M -partner.*

Proof. Let w be an extra woman, and let m be her partner in M . Recall that M is stable for the original instance I , while A is stable for the refined instance I'' (of SMI), and therefore also for the instances I' and I (of SSMTI).

(i) By definition, m is an extra man and therefore is not matched in A , so that if w is not matched in A it is immediate that the pair (m, w) blocks A .

(ii) Let a be w 's A -partner. If w strictly prefers m to a then, since m is unmatched in A , the pair (m, w) blocks A in I , a contradiction. If m and a are tied in w 's list, and a is an X -man then, when that tie was broken to form I'' , m , being a Y -man, must have preceded a in the resulting strict preference list. Hence, again since m is unmatched in A , the pair (m, w) blocks A in I'' , a contradiction. \square

We partition M 's extra men into two sets U and V ; those in U have an M -partner who is matched in A to an X -man, and those in V have an M -partner who is matched in A to a Y -man. Suppose, without loss of generality, that

$U = \{m_{x+1}, \dots, m_{x+s}\}$ and $V = \{m_{x+s+1}, \dots, m_{x+r}\}$, i.e., $|U| = s$, $|V| = r - s$. Let $M(U)$ denote the set of women who are matched in M to a man in U . Suppose that, among the men who are matched in A with women in $M(U)$, exactly t ($\leq s$) are unmatched in M . (These are all X -men, by definition of U , but some X -men – those who became so during Phase 2 of the algorithm, need not be matched in M .)

Our next lemma gives us certain inequalities involving the sizes of matchings M and A that will enable us to establish the claimed performance guarantee for Algorithm SSMTI-APPROX.

Lemma 5. (i) $|M| \leq |A| + r - t$.

(ii) $|A| \geq x + r - s$.

(iii) $|A| \geq r + s - t$.

Proof. (i) All the X -men are matched in A , but at least t of them are not matched in M , and the Y -men who are matched in M but not in A number exactly r .

(ii) Consider the set V . Each woman w who is the partner in M of a man in V is matched in A to a Y -man, and there are $r - s$ such women w . This gives us $r - s$ of the Y -men who are matched in A , and together with all x of the X -men who, by Lemma 4(ii), are all matched in A , we have a total of $x + r - s$ distinct men who are matched in A .

(iii) Consider the set U , and suppose that (m_{x+j}, w_{i_j}) is in M for $j = 1, \dots, s$. By definition of U , each w_{i_j} has an X -man as her partner in A ; without loss of generality, suppose that (m_j, w_{i_j}) is in A , for $j = 1, \dots, s$. By Lemma 4, w_{i_j} strictly prefers her A -partner m_j to her M -partner m_{x+j} . Each m_j is an X -man and $s - t$ is the number of these men who are matched in M ; suppose, without loss of generality, that (m_j, w_{k_j}) is in M , for $j = 1, \dots, s - t$. Then none of these w_{k_j} can be an extra woman, for the M -partners of the latter are Y -men. Also, each of the men m_j prefers w_{k_j} to w_{i_j} , for otherwise (m_j, w_{i_j}) would block M . Furthermore, each w_{k_j} must be matched in A . For if not, the pair (m_j, w_{k_j}) would block A . It follows that we have a total of $r + s - t$ women who must be matched in A , namely the r extra women, by Lemma 4, and the $s - t$ women $w_{k_1}, \dots, w_{k_{s-t}}$. □

We are now in a position to establish our main theorem.

Theorem 1. For a given instance of SSMTI, let M be a maximum cardinality stable matching and let A be a stable matching returned by Algorithm SSMTI-APPROX. Then $|M| \leq 8|A|/5$.

Proof. By Lemma 5 we have $|A| \geq \max(x + r - s, r + s - t) \geq \frac{1}{2}((x + r - s) + (r + s - t)) = x/2 + r - t/2$. So, by Lemma 3(iii), $|A| \geq |M|/4 + r - t/2$. Hence, by Lemma 5(i), $|A| \geq |M|/4 + |M| - |A| + t - t/2$, and so $2|A| \geq 5|M|/4 + t/2$, from which the claimed bound follows. □

Complexity of the Algorithm

The worst-case complexity of Algorithm SSMTI-APPROX is dominated by the maximum cardinality matching step in Phase 2. Using the Hopcroft-Karp algorithm [8], this can be achieved in $O(\sqrt{na})$ time, where n is the total number of men and women, and a is the sum of the lengths of the preference lists. However, there is a variant of the algorithm that achieves the same performance guarantee but with $O(a)$ complexity. This is obtained by observing that, in Phase 2, it suffices to find a *maximal* matching – i.e., a matching that cannot be extended to a larger matching by adding further pairs – rather than a maximum cardinality matching, of men in Y_1 to women in Q_1 . The only place in the subsequent argument where the relevant property of this matching is needed is in the proof of Lemma 3(i), and it is indeed merely maximality that is required. A maximal matching can be found in $O(a)$ time, and all other parts of the algorithm are merely variants of the Gale-Shapley algorithm. It is not hard to show that these can also be implemented to run in $O(a)$ time (see [4]).

Tightness of the Approximation Guarantee

This is the tightest bound that can be established for Algorithm SSMTI-APPROX. Figure 5 shows an example where the ratio of $|M|$ to $|A|$ is $\frac{8}{5}$. The matching $M = \{(m_1, w_5), (m_2, w_6), (m_3, w_7), (m_4, w_8), (m_5, w_1), (m_6, w_2), (m_7, w_3), (m_8, w_4)\}$ is a maximum cardinality stable matching of size 8, whereas if ties are broken simply by removing the parentheses, the algorithm returns matching $A = \{(m_1, w_2), (m_2, w_3), (m_3, w_5), (m_4, w_6), (m_8, w_1)\}$, of size 5. By duplicating this pattern, we can obtain arbitrarily large instances realising the $\frac{8}{5}$ ratio.

Men's preferences	Women's preferences
$m_1 : w_5 \ w_2$	$w_1 : m_3 \ (m_8 \ m_5)$
$m_2 : w_6 \ w_3$	$w_2 : m_1 \ m_6$
$m_3 : w_5 \ w_7 \ w_8 \ w_1$	$w_3 : m_2 \ m_7$
$m_4 : w_6 \ w_8 \ w_7 \ w_4$	$w_4 : m_4 \ m_8$
$m_5 : w_1$	$w_5 : (m_3 \ m_1)$
$m_6 : w_2$	$w_6 : (m_4 \ m_2)$
$m_7 : w_3$	$w_7 : (m_4 \ m_3)$
$m_8 : w_1 \ w_4$	$w_8 : (m_3 \ m_4)$

Fig. 5. An instance of SSMTI with ratio $\frac{8}{5}$

Extension to Special HRT

In view of the fact that our study was motivated by practical applications of the HRT problem, it is important to note that we can obtain exactly the same $\frac{8}{5}$ performance guarantee for an analogous algorithm for the special case of HRT in which each hospital's preference list has a tie of length ≥ 1 at the end. Full details of the extended algorithm and a correctness proof can be found in [10].

3 Summary and Open Problems

We have described a polynomial-time approximation algorithm with a performance guarantee of $\frac{8}{5}$ for a maximum cardinality stable matching in NP-hard variants of the Stable Marriage and Hospitals/Residents problems that are of significant practical interest. We have also shown that this performance guarantee is the best that can be proved for the algorithm.

The most obvious open question to pursue is whether this or a similar approach can yield useful performance guarantees for more general versions of SMTI and HRT, for example when there can be a single tie at the end of the lists on both sides, or when the lists on one side can contain arbitrary ties.

References

1. (Canadian Resident Matching Service) <http://www.carms.ca/jsp/main.jsp>
2. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. *American Mathematical Monthly* 69, 9–15 (1962)
3. Gale, D., Sotomayor, M.: Some remarks on the stable matching problem. *Discrete Applied Mathematics* 11, 223–232 (1985)
4. Gusfield, D., Irving, R.W.: *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, Cambridge (1989)
5. Halldórsson, M., Irving, R.W., Iwama, K., Manlove, D.F., Miyazaki, S., Morita, Y., Scott, S.: Approximability results for stable marriage problems with ties. *Theoretical Computer Science* 306(1-3), 431–447 (2003)
6. Halldórsson, M., Iwama, K., Miyazaki, S., Yanagisawa, H.: Improved approximation of the stable marriage problem. In: Di Battista, G., Zwick, U. (eds.) *ESA 2003*. LNCS, vol. 2832, pp. 266–277. Springer, Heidelberg (2003)
7. Halldórsson, M.M., Iwama, K., Miyazaki, S., Yanagisawa, H.: Randomized approximation of the stable marriage problem. *Theoretical Computer Science* 325(3), 439–465 (2004)
8. Hopcroft, J.E., Karp, R.M.: A $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing* 2, 225–231 (1973)
9. Irving, R.W.: Matching medical students to pairs of hospitals: a new variation on a well-known theme. In: Bilardi, G., Pietracaprina, A., Italiano, G.F., Pucci, G. (eds.) *ESA 1998*. LNCS, vol. 1461, pp. 381–392. Springer, Heidelberg (1998)
10. Irving, R.W., Manlove, D.F.: $8/5$ -approximation algorithms for hard variants of the stable marriage and hospitals/residents problems. Technical Report TR-2007-232, University of Glasgow, Department of Computing Science (February 2007)
11. Iwama, K., Miyazaki, S., Yamauchi, N.: A $\left(2 - c\frac{1}{\sqrt{n}}\right)$ approximation algorithm for the stable marriage problem. In: Deng, X., Du, D.-Z. (eds.) *ISAAC 2005*. LNCS, vol. 3827, pp. 902–914. Springer, Heidelberg (2005)
12. Iwama, K., Miyazaki, S., Yamauchi, N.: A 1.875-approximation algorithm for the stable marriage problem. In: *Proceedings of SODA 2007*, pp. 288–297 (2007)
13. Manlove, D.F., Irving, R.W., Iwama, K., Miyazaki, S., Morita, Y.: Hard variants of stable marriage. *Theoretical Computer Science* 276(1-2), 261–279 (2002)
14. (National Resident Matching Program) http://www.nrmp.org/about_nrmp/
15. Roth, A.E.: The evolution of the labor market for medical interns and residents: a case study in game theory. *Journal of Political Economy* 92(6), 991–1016 (1984)
16. (Scottish Foundation Allocation Scheme) <http://www.nes.scot.nhs.uk/sfas/>

Approximation Algorithms for the Black and White Traveling Salesman Problem

Binay Bhattacharya^{1,*}, Yuzhuang Hu², and Alexander Kononov³

¹ School of Computing Science, Simon Fraser University, Burnaby, Canada, V5A 1S6
{binay,yhu1}@cs.sfu.ca

² Laboratory “Mathematical Models of Decision Making”, Sobolev Institute of Mathematics, Acad. Koptyug Avenue, 630090 Novosibirsk, Russia
alvenko@math.nsc.ru

Abstract. The black and white traveling salesman problem (BWTSP) is to find the minimum cost hamiltonian tour of an undirected complete graph G , containing black and white vertices, subject to two restrictions: the number of white vertices, and the cost of the subtour between two consecutive black vertices are bounded. This paper focuses on designing approximation algorithms for the BWTSP in a graph satisfying the triangle inequality. We show that approximating the tour which satisfies the length constraint is NP-hard. We then show that the BWTSP can be approximated with tour cost $(4 - \frac{3}{2Q})$ times the optimal cost, when at most Q white vertices appear between two consecutive black vertices. When exactly Q white vertices appear between two consecutive black vertices, the approximation bound can be slightly improved to $(4 - \frac{15}{8Q})$. This approximation bound is further improved to 2.5 when $Q = 2$.

1 Introduction

In this paper, we consider an extension of the classical traveling salesman problem (TSP). The problem is defined on an undirected graph, $G = (V, E)$, where a vertex set, $V = V_B \cup V_W$, is partitioned into a set of *black vertices*, V_B , and a set of *white vertices*, V_W , and an edge set, E , with edge costs $w(e)$ for all $e \in E$ satisfying the triangle inequality. The black and white traveling salesman problem (BWTSP) is to determine a minimum cost hamiltonian tour of G subject to the following restrictions:

1. *Cardinality constraint* in which the number of white vertices on “black to black” paths is bounded above by a positive integer constant Q , and
2. *Length constraint* in which the cost of any path between two consecutive black vertices is bounded above by a positive value L .

Clearly, BWTSP reduces to the classical TSP when $L = Q = \infty$, and is therefore NP-hard. An application of the directed BWTSP arises in short-haul airline operations ([4,13]). The flight leg between two stations p and q is determined by a white vertex v_{pq} and a maintenance station s corresponds to a

* Research was partially supported by MITACS and NSERC.

black vertex v_s . An arc represents a leg-leg, leg-maintenance, or maintenance-leg sequence. The problem is to determine a flying sequence such that the number of takeoffs and landings, as well as the total operating cost between any two maintenance sequences are bounded as above. The undirected case has applications in telecommunications ([6],[15]). Cosares et al. [6] and Wasem [15] describe an application of the undirected BWTSP arising in the design of telecommunication ring networks, in which black vertices are "ring offices" and white vertices are "hubs". In order to achieve a survivable synchronous optical network (SONET) architecture, any two consecutive ring offices on the network must be separated by at most Q hubs and a length not exceeding L . Another particular case of the BWTSP is the vehicle routing problem (VRP) where each client has unit demand, the vehicle has capacity Q , and maximal route length of the vehicle is at most L .

Attempts have been made to optimally solve BWTSP for small size problems ([2],[15]). Ghiani, Laporte and Semet recently developed an exact branch-and-cut algorithm for the undirected case ([10]). Mak and Boland [13] have proposed a simulated annealing algorithm for the directed BWTSP and have applied it to instances involving 36 vertices. Bourgeois, Laporte and Samet [2] proposed five heuristic algorithms for the BWTSP, along with extensive computational comparisons.

In this paper we are interested in designing efficient approximation algorithms with guaranteed performances for the BWTSP. We show that BWTSP cannot be approximated if the length constraint is specified. However, approximation algorithms with guaranteed performances can be designed when only the cardinality constraint is specified. BWTSP with the cardinality constraint $Q = 1$ occurs in routing papers with different names: bipartite traveling salesperson problem or k -delivery problem where $k = 1$. Anily and Hassin [1] have shown a 2.5-approximation algorithm for another generalization of this problem, known as the swapping problem. Their algorithm finds a perfect matching M , consisting of edges that connect black and white vertices, and it uses the Christofides-Serdyukov heuristic [4] to find a tour, T , of the black vertices. The final route consists of visiting the black vertices in the sequence specified by the tour T , using the matching edges in M . Later, Chalasani and Motwani [3] developed a 2-approximation algorithm for 1-delivery problem using some combinatorial properties of bipartite spanning trees and matroid intersection. We expand the idea proposed in [1] and present a $(4 - \frac{3}{2Q})$ -approximation algorithm, when the number of white vertices between two consecutive black vertices is bounded above by Q . The bound can be slightly improved to $(4 - \frac{15}{8Q})$, if the number of white vertices is exactly $Q \cdot |V_B|$. When $|V_W| = 2 \cdot |V_B|$, the bound can be improved to 2.5.

The organization of this paper is as follows: In section 2, we show that the BWTSP is NP-hard when the length constraint is specified. Section 3 deals with the BWTSP when only the cardinality constraint is specified. Various approximation algorithms are provided for different variants of the cardinality constraint. Section 4 discusses our conclusions.

2 BWTSP with Length Constraint

We first show that the following problem is NP-complete. Given a complete weighted graph G , with black and white vertices, satisfying the triangle inequality, determine whether G has a BWTSP route wherein the cost of the path between two consecutive black vertices in the cycle is no more than L . The above result then implies that the problem of designing approximation algorithms for the BWTSP is NP-hard, if the length constraint is specified.

Let us consider an instance of the hamiltonian path problem. Let $G = (V, E)$ be the input graph with $|V| = n$. Consider the following graph G' . G' has n black vertices V_B and n copies of V (say, V_1, V_2, \dots, V_n) which are all white. Suppose $L = n + 1$ and $Q = \infty$. The cost of the edge between u and v is fixed as follows:

- (i) if $u \in V_B$ and $v \in V_B$, $w(u, v) = 2$,
- (ii) if $u \in V_B$ and $v \in V_i$, for any i , $w(u, v) = 1$,
- (iii) if $u \in V_i$ and $v \in V_j$, for any $i \neq j$, $w(u, v) = 2$,
- (iv) if $u \in V_i$, $v \in V_i$ and $(u, v) \in E$, $w(u, v) = 1$, and
- (v) if $u \in V_i$, $v \in V_i$ and $(u, v) \notin E$, $w(u, v) = 2$.

G' is a complete weighted graph satisfying the triangle inequality. Clearly, G' has a BWTSP route satisfying the length constraint, if and only if, the graph G has a hamiltonian path.

3 BWTSP with Only the Cardinality Constraint Specified

The result from the previous section implies the impossibility of finding approximate solutions to BWTSP when the length constraint is specified. Therefore, we consider the case where only the cardinality constraint is satisfied. In other words, Given a graph $G = (V, E)$ where $V = V_B \cup V_W$, $V_B \cap V_W = \emptyset$, with the edges satisfying the triangle inequality, determine a minimum cost traveling salesman tour such that the number of white vertices between two consecutive black vertices in the tour is at most a given integer Q .

Let $|V_B| = n$ and $|V_W| = m$. Without any loss of generality, we assume that $m \leq Q \cdot n$, otherwise an instance does not have a feasible solution. Also note that if $m \leq Q$, any Hamiltonian cycle satisfies the cardinality constraint and we get the classical traveling salesman problem. So in our paper, we assume that $Q < m \leq Q \cdot n$.

3.1 Lower Bounds

Let L^* be the length of the optimal tour of the BWTSP in $G = (V, E)$, satisfying the cardinality constraint. The fact that the given graph satisfies the triangle inequality implies that the cost of the optimal traveling salesman tour that visits only a subset of vertices is a lower bound of optimal tour of the BWTSP. Let L_B^* and L_W^* denote the lengths of the optimal traveling salesman tour of the black and white vertices respectively. Hence $L^* \geq L_B^*$ and $L^* \geq L_W^*$.

We define a Q -factor of G as a set of edges $E_Q \subseteq E$, such that for each black vertex $v \in V_B$, $\sigma(v) \leq Q$, and for each white vertex, $v \in V_W$, $\sigma(v) = 1$, where $\sigma(v)$ is the number of edges of E_Q incident on v .

Given a tour T_{BW} of black and white vertices, a white vertex w is said to be closer to a black vertex u than to a black vertex v in T_{BW} if the number of vertices between w and u in T_{BW} is less than the number of vertices between w and v in T_{BW} . Suppose in the optimal tour we connect each white vertex to the closest black vertex. If black to black path in the optimal tour has an odd number of white vertices, the middle white vertex can be connected to either of the black vertices. Our strategy of connection is such that each black vertex is allowed to be connected to one such middle vertex. This way each black vertex is connected to at most Q white vertices. Thus, the obtained set of edges is a Q -factor.

Let $L_{E_Q}^*$ be the total cost of edges connecting the black and white vertices, using the above rule on the optimal tour of the BWTSP. We can estimate the cost of each edge between black and white vertices by using the triangle inequality. When Q is even, clearly $L_{E_Q}^* \leq \frac{Q}{2}L^*$. A similar inequality results when Q is odd.

3.2 Approximation Algorithm when $Q < m \leq Q \cdot n$

We describe our approximation algorithm below. In the following we assume that in any tour of G there are at most Q white vertices between two consecutive black vertices. Each step is followed by a brief discussion and implementation details if needed.

Algorithm $BWTSP(n, m)$

Step 1: Construct a complete bipartite graph K in the following way. One part contains n black vertices, V_B , and the other part contains m white vertices, V_W .

Step 2: Find a minimum cost Q -factor E_Q of K .

We solve this problem in the following way. We add $Q - 1$ copies of each black vertex to the first part of bipartite graph. We also copy the edges incident on the black vertex. We then find a minimum cost matching M in the augmented complete bipartite graph K [7]. Note that the total cost of the edges in E_Q , denoted by $\|E_Q\|$, is at most $L_{E_Q}^*$.

Step 3: Transform graph G to graph \hat{G} as follows. Let h_v be the degree of black vertex v in the induced graph (V, E_Q) . For each black vertex v add $Q - h_v$ “dummy” white vertices, and associate them with black vertex v in the following way. Each dummy white vertex is connected to v with edge cost zero. The dummy vertices are connected to other black and white vertices with edge costs being the same as the edge costs with v . This way we get $Q \cdot n$ white vertices in total. It is easy to show that the triangle inequality is still satisfied in \hat{G} .

Consider any tour where the black vertices v_1, v_2, \dots, v_n are ordered, and the number of white vertices between consecutive black vertices v_i and v_{i+1} is $t = h_{v_i}$. Let these white vertices be in order b_1, b_2, \dots, b_t . We now augment

the path $v_i, b_1, b_2, \dots, b_t, v_{i+1}$ to $v_i, a_1, a_2, \dots, a_s, b_1, b_2, \dots, b_t, v_{i+1}$, for $s+t = Q$. Here a_1, a_2, \dots, a_s are the dummy white vertices associated with v_i . This means that the cost of the edge between v_i and a_1 is 0, and the cost of the edge between a_s and b_1 is the same as the cost of the edge between v_i and b_1 . Therefore the cost of the augmented black and white tour is the same as that of the original tour. Let $\hat{G}_W = (\hat{V}_W, \hat{E}_W)$ be the graph induced by the white vertices.

Step 4: Find a near optimal hamiltonian tour \hat{T}_W in graph $\hat{G}_W = (\hat{V}_W, \hat{E}_W)$.

This tour fixes the order of the white vertices in the proposed tour of the BWTSP. We use Christofides-Serdyukov algorithm [4] to obtain \hat{T}_W . Let $L_{\hat{T}_W}$ denote the length of \hat{T}_W . From the discussions in step 3, the optimal TSP tour \hat{T}_W^* involving all the white vertices of \hat{G}_W has the same cost as the optimal TSP tour of G , and therefore the cost of \hat{T}_W^* is less than L^* . So we have $L_{\hat{T}_W} \leq 1.5L^*$.

Step 5: Partition the tour \hat{T}_W into paths P_i on Q vertices, $i = 1, 2, \dots, n$ of minimum cost.

Let $P_i = (u_{i1}, u_{i2}, \dots, u_{iQ})$, $i = 1, 2, \dots, n$ be the minimum cost paths. Since there exist Q different ways to partition tour \hat{T}_W , the total cost of the paths in $P_i, i = 1, 2, \dots, n$ is at most $\frac{Q-1}{Q} L(\hat{T}_W)$.

Step 6: Construct a bipartite multigraph H in the following way. One part contains the vertices V_B and the other part contains n vertices y_1, y_2, \dots, y_n where the element y_i represents path P_i computed in step 5. Now $(u, y), u \in V_B$ and $y \in \{y_1, y_2, \dots, y_n\}$, is an edge in H , if and only if there exists an edge (u, v') in E_Q (computed in step 2) such that $u \in V_B$ and v' is a vertex of the path represented by y .

Thus H is a bipartite multigraph and each vertex of H is of degree Q .

Step 7: Find a proper edge coloring of H in Q colors.

Each vertex in H has degree Q . According to König [11], the chromatic index of a bipartite multigraph with maximum degree h is h . It is also shown in [11] that in a bipartite multigraph, there exists a matching that saturates all the vertices with the maximum degree. Therefore, the edges of H can be colored using Q colors, and the set of edges of the same color covers the vertices of H . Let C_1, C_2, \dots, C_Q be the partition of the set of edges of E_Q where C_i contains all the i -colored edges. Note here that each C_i determines an assignment between black vertices and paths P_1, P_2, \dots, P_n .

Step 8: Select the set C_q from C_1, C_2, \dots, C_Q with minimum length.

Clearly, $\|C_q\| \leq \frac{1}{Q} \|E_Q\|$. Therefore, $\|C_q\| \leq \frac{1}{2} L^*$.

Step 9: Let v_i be the black vertex assigned to P_i . Construct two hamiltonian tours $R_1 = (v_1, u_{11}, u_{12}, \dots, u_{1Q}, v_2, u_{21}, u_{22}, \dots, u_{2Q}, \dots, v_i, u_{i1}, u_{i2}, \dots, u_{iQ}, v_{i+1}, \dots, v_n, u_{n1}, u_{n2}, \dots, u_{nQ})$ and $R_2 = (u_{11}, u_{12}, \dots, u_{1Q}, v_1, u_{21}, u_{22}, \dots, u_{2Q}, v_2, \dots, u_{i1}, u_{i2}, \dots, u_{iQ}, v_i, \dots, u_{n1}, u_{n2}, \dots, u_{nQ}, v_n)$. Remove the dummies from R_1 and R_2 , and take the tour, say R , with the minimal cost as a BWTSP tour of G .

3.3 Performance Analysis

We can estimate the total cost of each tour by separately estimating the cost of the edges between the white vertices and between the black and white vertices. The total length of the edges between the white vertices is the total cost of n paths obtained in step 4 and is at most $\frac{Q-1}{Q} L_{\hat{T}_W}$.

We now estimate the total edge cost of the edges between black and white vertices in tour R . Let us first consider the total cost of edges connected to the black vertex v_i in routes R_1 and R_2 . For route R_1 , suppose $(v_i, u_{ik}) \in C_q$ for some k , $1 \leq k \leq Q$, then

$$w(u_{i-1,Q}, v_i) + w(v_i, u_{i1}) \leq w(u_{i-1Q}, u_{i1}) + L(u_{i1}, u_{i2}, \dots, u_{ik}) + w(u_{ik}, v_i) + L(u_{i1}, u_{i2}, \dots, u_{ik}) + w(v_i, u_{ik}).$$

Here $L(P_i)$ indicates the cost of path P_i .

For tour R_2 we have

$$w(u_{iQ}, v_i) + w(v_i, u_{i+1,1}) \leq w(u_{ik}, v_i) + L(u_{ik}, \dots, u_{iQ}) + w(u_{ik}, v_i) + L(u_{ik}, \dots, u_{iQ}) + w(u_{iQ}, u_{i+1,1}).$$

Since $\min\{L_{R_1}, L_{R_2}\} \leq \frac{L_{R_1} + L_{R_2}}{2}$ we can now write

$$\begin{aligned} \min\{L_{R_1}, L_{R_2}\} &\leq (4\|C_Q\| + 2\frac{Q-1}{Q}L_{\hat{T}_W} + 2L_{\hat{T}_W})/2 \\ &\leq 2\|C_Q\| + \frac{Q-1}{Q}L_{\hat{T}_W} + L_{\hat{T}_W} \\ &\leq (4 - \frac{3}{2Q})L^*. \end{aligned}$$

Theorem 1: The black and white traveling salesman problem with only the cardinality constraint can be approximated to within $(4 - \frac{3}{2Q})$, where Q is the maximum number of consecutive white vertices that can appear in the route.

Running Time. The following computations dominate the running time of the algorithm.

1. Computing near optimal hamiltonian tour \hat{T}_W of $\hat{G}_W = (V_W, \hat{E}_W)$ (Step 4). The running time of Christofides-Serdyukov’s algorithm [4] to compute \hat{T}_W is dominated by the perfect matching problem in a subgraph of \hat{G}_W which, in the worst case, takes $O(|V_W|^3)$ time [12]. Since $|V_W| = Q \cdot n$, it takes $O(Q^3 n^3)$ time to compute \hat{T}_W .
2. Computing a Q -factor E_Q (Step 2). This problem has been shown to be equivalent to a matching problem in a complete bipartite graph K involving $O(Q \cdot n)$ vertices. Therefore, we can find the minimum cost Q -factor of K in $O(Q^3 n^3)$ time.
3. Finding a proper edge coloring of bipartite multigraph H (Step 6). We can use the algorithm proposed by Cole and Hopcroft [5] to find an edge coloring of the bipartite multigraph in $O(Q^2 n^2 \log n)$ time.

Thus, the approximation algorithm proposed to solve cardinality constrained *BWTSP* in a graph with n black vertices and m ($Q < m \leq Q \cdot n$) white vertices takes $O(Q^3 n^3)$ time to compute.

3.4 Approximation Algorithms When $m = Q \cdot n$

In this section we discuss ways of improving the performance bounds in the case when $m = Q \cdot n$. Let E_{BW}^* be the $2n$ edges of the optimal *BWTSP* solution L^* connecting the black and white edges. The cost of the Q -factor, described in section 3, can be alternately bounded by $\|E_{BW}^*\| + \frac{Q-2}{2} * L^*$. The argument for this new bound is almost the same once the edges of E_{BW}^* are separated. Also note that G and \hat{G} are the same in Step 3 of *BWTSP*(n, m). Let $\alpha = \frac{\|E_{BW}^*\|}{L^*}$, then we can write

$$\min\{L_{R_1}, L_{R_2}\} \leq 2\|C_Q\| + \frac{Q-1}{Q}L_{\hat{T}_W} + L_{\hat{T}_W} \leq (4 - \frac{7}{2Q} + \frac{2\alpha}{Q})L^*.$$

We describe below another algorithm, called *BWTSP2*($n, Q \cdot n$), where the number of white vertices between two consecutive black nodes is exactly Q . Most of the steps of Algorithm *BWTSP*(n, m) are the same in the new algorithm. Steps 3, 4 and 5 are replaced by step 3-4-5 and steps 8 and 9 are replaced by steps 8-9(a) and 8-9(b). As before, each new step is followed by a brief discussion and comments, if needed.

Algorithm *BWTSP2*($n, Q \cdot n$)

Step 3-4-5: Partition the white vertices into a set of paths, each containing exactly Q vertices, using the algorithm of Goemans and Williamson [9].

As described in [9], the exact path partitioning problem (partitioning $G_W = (V_W, E_W)$ into disjoint paths, each path containing exactly Q vertices) can be approximated to within $4(1 - \frac{1}{Q})(1 - \frac{1}{|V|})$, i.e. within $4(1 - \frac{1}{Q})$. Thus the set of paths we found in this step has a total cost of less than $4(1 - \frac{1}{Q})(1 - \alpha)L^*$.

This step can be performed in time $O(n^2 \log n)$ [9] which was later improved to $O(n^2)$ [8]. Let P_W be the set of paths.

Step 8-9(a): Find a near-optimal tour T_B in $G_B = (V_B, E_B)$.

Step 8(b): Construct a tour involving the edges of C_q, P_W and T_B .

This tour uses the edges of C_q and the edges of the paths in P_W , found in step 3-4-5, twice and the edges of T_B once. Thus we can get a feasible solution of *BWTSP* with a cost less than $(1.5 + 8(1 - \frac{1}{Q})(1 - \alpha) + \frac{2}{Q}(\alpha + \frac{Q-2}{2}))L^*$, i.e. less than $(10.5 - \frac{10}{Q} - (8 - \frac{10}{Q})\alpha)L^*$.

We now have two solutions with the costs of $(4 - \frac{7}{2Q} + \frac{2\alpha}{Q})L^*$ and $(10.5 - \frac{10}{Q} - (8 - \frac{10}{Q})\alpha)L^*$ respectively. We can choose the one with the smaller cost to be our final solution. It is interesting to verify that the two cost functions have the same

value for all Q when $\alpha = \frac{13}{16}$. Substituting $\frac{13}{16}$ for α , the approximation ratio is then $4 - \frac{15}{8Q}$. The gain of $\frac{3}{8Q}$ from our previous ratio of $4 - \frac{3}{2Q}$ is meaningful for small Q .

Further Improvement when $m = 2n$. In the following we show that for $m = 2n$ ($Q = 2$), the approximation bound can be improved to 2.5. This is due to the fact that both the white-white edges (denoted by E_{WW}^*) and the black-white edges (denoted by E_{BW}^*) of an optimal BWTSP solution L_{BW}^* can be efficiently approximated. Note that there are n edges in E_{WW}^* and $2n$ edges in E_{BW}^* . The algorithm is formally described below.

Algorithm $BWTSP3(n, 2 \cdot n)$

Step 1: Construct a bipartite graph K in the following way. One part contains n black vertices of V_B and the other part contains all $2n$ white vertices of V_W .

Only the edges of G connecting the black and white vertices are present in K .

Step 2: Find a minimum cost 2-factor E_2 of K .

Since E_{BW}^* is a 2-factor, then $\|E_2\| \leq \|E_{BW}^*\|$.

Step 3: Find a minimum cost perfect matching M of $G_W = (V_W, E_W)$.

Clearly $\|M\| \leq \|E_{WW}^*\|$. Consider the induced graph $(V, M \cup E_2)$. This graph is a collection of cycles. Each cycle involving black and white vertices is a tour (on a subset of vertices) satisfying the cardinality constraint. We represent each cycle by $CYCLE(v)$ where v is an arbitrary black vertex in the cycle. Let V_A be the set of arbitrary black vertices chosen to represent the cycles.

We note that $\|M \cup E_2\| = \|M\| + \|E_2\| \leq \|E_{BW}^*\| + \|E_{WW}^*\| = L^*$.

Step 4: Find a near-optimal TSP tour T_A of G_A , the subgraph of G induced by the vertices of V_A .

Step 5: Starting from an arbitrary black vertex $v \in V_A$ we make the round of all vertices of $CYCLE(v)$. After that we move to the next black vertex in tour T_A . The order in which the vertices appear in this walk define a tour T .

The cost of T is bounded by the total cost of the edges of $M \cup E_2$ and edges in tour T_A . So we have $L_T = L_{T_A} + \|M\| + \|E_2\| \leq \frac{5}{2}L^*$ and therefore, the algorithm $BWTSP3(n, 2n)$ is a 2.5-approximation algorithm.

4 Conclusions

We have designed approximation algorithms with guaranteed performances for the black and white traveling salesman problem. We have shown that BWTSP cannot be approximated if the length constraint is specified. However, approximation algorithms with guaranteed performances can be designed when the cardinality constraint Q is only specified. For arbitrary Q , the designed algorithm has an approximation ratio of less than 4. This ratio is slightly improved for smaller values of Q if the number of white vertices is exactly $Q \cdot n$ where n is the number of black vertices. The approximation bound of 2.5 is possible for the BWTSP when the number of white vertices is exactly $2n$.

References

1. Anily, S., Hassin, R.: The swapping problem. *Networks* 22, 419–433 (1992)
2. Bourgeois, M., Laporte, G., Samet, F.: Heuristics for the black and white traveling salesman problem. *Computers and Operations Research* 30, 75–85 (2003)
3. Chalasani, P., Motwani, R.: Approximating capacitated routing and delivery problems. *SIAM Journal on Computing* 28, 2133–2149 (1999)
4. Christofides, N.: The traveling salesman problem. In: Christofides, N., Mingozzi, A., Toth, P., Sandi, C. (eds.) *Combinatorial Optimization*, pp. 315–318 (1979)
5. Cole, R., Hopcroft, J.: On edge coloring bipartite graphs. *SIAM Journal on Computing* 11, 540–546 (1982)
6. Cosares, S., Deutsch, D.N., Saniee, I., Wasem, O.J.: SONET Toolkit: A decision support system for designing robust and cost effective fibre-optic networks. *Interfaces* 25, 20–40 (1995)
7. Dinitz, D.: The solution of two assignment problems. In: Fridman, A.A. (ed.) *Russian; Studies in Discrete Optimization*, Nauka, Moscow, pp. 333–348 (1976)
8. Gabow, H.N., Pettie, S.: The dynamic vertex minimum problem and its application to clustering-type approximation algorithms. In: Penttonen, M., Schmidt, E.M. (eds.) *SWAT 2002. LNCS*, vol. 2368, pp. 190–199. Springer, Heidelberg (2002)
9. Goemans, M.X., Williamson, D.P.: A general approximation technique for constrained forest problems. *SIAM Journal on Computing* 24(2), 296–317 (1995)
10. Ghiani, G., Laporte, G., Semet, F.: The black and white traveling salesman problem. *Operations Research* 54, 366–378 (2006)
11. König, D.: Über Graphen und ihre Anwendungen. *Math. Annalen* 77, 453–465 (1916)
12. Lawler, E.L.: *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York (1976)
13. Mak, V., Boland, N.: Heuristic approaches to the asymmetric traveling salesman problem with replenishment arcs. *International Transactions in Operations Research* 7, 431–437 (2000)
14. Talluri, K.T.: The four-day aircraft maintenance routing problem. *Transportation Science* 32, 43–53 (1998)
15. Wasem, O.J.: An algorithm for designing rings in survivable fibre networks. *IEEE Transactions on Reliability* 40, 428–432 (1991)

Author Index

- Afshani, Peyman 459
Alon, Noga 394, 428
Aluru, Srinivas 1
Apostolico, Alberto 360
Arpe, Jan 296
- Bhattacharya, Binay 559
Bodlaender, Hans L. 86
Borodin, Allan 504
Boros, Endre 222
Borys, Konrad 222
Brandes, U. 254
Brown, Dan 51
Buchin, Kevin 97
Buragohain, Chiranjeeb 210
- Chan, Timothy M. 383
Chang, Ching-Lueh 285
Chen, Danny Z. 4, 232
Chen, Jianer 349, 537
Chen, Shihyen 482
Chen, Zhizhong 493
Chin, Francis Y.L. 2, 526
Chiniforooshan, Ehsan 459
Chor, Benny 75
Christodoulou, George 187
Czygrinow, A. 515
- Deng, Xiaotie 264
Dessimoz, Christophe 151
Dorrigiv, Reza 459
Duchesne, Jean-Eudes 27
- El-Mabrouk, Nadia 27
Elbassioni, Khaled 222
Erten, C. 254
- Fang, Qizhi 439
Farzan, Arash 459
Fellows, Michael R. 75, 86
Ferrante, Alessandro 417
Fleischer, Rudolf 439
Fomin, Fedor V. 65, 165
Fowler, J. 254
Frati, F. 254
- Frid, Yelena 51
Fung, Stanley P.Y. 176
- Gaspers, Serge 65
Geyer, M. 254
Giraud, Mathieu 27
Glaßer, Christian 307
Gourvès, Laurent 187
Gurvich, Vladimir 222
Gusfield, Dan 16, 51
Gutner, Shai 394
Gutwenger, C. 254
- Hańćkowiak, M. 515
Hansen, Kristoffer Arnsfelt 274, 448
Harkins, Ryan C. 129
Harutyunyan, Hovhannes 372
Healy, Mark A. 4
Heggernes, Pinar 406
Hitchcock, John M. 129
Hong, S. 254
Hu, Yuzhuang 559
Hüffner, Falk 140
- Irving, Robert W. 548
Iwama, Kazuo 108, 264
- Jansen, Maurice J. 470
- Katoh, Naoki 243
Kaufmann, M. 254
Knauer, Christian 97
Kobourov, S.G. 254
Komusiewicz, Christian 140
Kononov, Alexander 559
Kratochvíl, Jan 118
Kriegel, Klaus 97
- Langston, Michael A. 86
Ledergerber, Christian 151
Li, Jian 439
Li, Ming 3
Liotta, G. 254
Liu, Sheng 198
Liu, Yunlong 349

- Lu, Songjian 537
 Lyuu, Yuh-Dauh 285

 Ma, Bin 40
 Makino, Kazuhisa 222
 Manlove, David F. 548
 Maraachlian, Edward 372
 Miltersen, Peter Bro 274
 Mirzazadeh, Mehdi 459
 Misiolek, Ewa 232
 Moser, Hannes 140
 Mutzel, P. 254

 Nakashima, Takuya 108
 Niedermeier, Rolf 140

 Pandurangan, Gopal 417
 Papadopoulos, Charis 406
 Park, Kihong 417
 Pascual, Fanny 187
 Pergel, Martin 118
 Poon, Chung Keung 176

 Qi, Qi 264

 Ragan, Mark A. 75, 86
 Razgon, Igor 75
 Regan, Kenneth W. 470
 Reischuk, Rüdiger 296
 Rosamond, Frances A. 75, 86
 Rudolf, Gabor 222

 Saurabh, Saket 65
 Schulz, André 97
 Seidel, Raimund 97
 Selman, Alan L. 307
 Shapira, Asaf 428
 Simjour, Narges 459

 Snir, Sagi 75
 Sørensen, Troels Bjerre 274
 Stav, Uri 428
 Stepanov, Alexey A. 165
 Sun, Aries Wei 264
 Sun, Xiaoxun 439
 Suri, Subhash 210
 Symvonis, A. 254

 Tagliacollo, Claudia 360
 Tanigawa, Shin-ichi 243
 Tasaka, Toyotaka 264
 Tóth, Csaba D. 210

 Wang, Chao 4
 Wang, Jianxin 349
 Wang, Lusheng 493
 Wang, Zhanyong 493
 Weyer, Mark 86
 Wu, Xiaodong 4
 Wu, Yufeng 16

 Xin, Lei 40

 Ye, Yuli 504
 Yu, Fuxiang 318

 Zarrabi-Zadeh, Hamid 383, 459
 Zhang, Jian 198
 Zhang, Kaizhong 40, 482
 Zhang, Liyu 307
 Zhang, Shengyu 338
 Zhang, Yong 526
 Zheng, Feifeng 176
 Zheng, Xizhong 327
 Zhou, Yunhong 210
 Zhu, Binhai 198
 Zhu, Hong 526