LNCS 3879

Thomas Erlebach
Giuseppe Persiano (Eds.)

Approximation and Online Algorithms

Third International Workshop, WAOA 2005
Palma de Mallorca, Spain, October 2005
Revised Papers

Springer

Lecture Notes in Computer Science 3879

Thomas Erlebach Giuseppe Persiano (Eds.)

Approximation and Online Algorithms

Third International Workshop, WAOA 2005
Palma de Mallorca, Spain, October 6-7, 2005
Revised Papers

Springer

Volume Editors

Thomas Erlebach
University of Leicester
Department of Computer Science
University Road, Leicester, LE1 7RH, UK
E-mail: t.erlebach@mcs.le.ac.uk

Giuseppe Persiano
Università degli Studi di Salerno
Dipartimento di Informatica ed Applicazioni
Via S. Allende 2, 84081 Baronissi (SA), Italy
E-mail: giuper@dia.unisa.it

Library of Congress Control Number: 200692553

CR Subject Classification (1998): F.2.2, G.2.1-2, G.1.2, G.1.6, I.3.5, E.1

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

ISSN 0302-9743
ISBN-10 3-540-32207-8 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-32207-8 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springer.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11671411 06/3142 5 4 3 2 1 0

Preface

The third Workshop on Approximation and Online Algorithms (WAOA 2005) focused on the design and analysis of algorithms for online and computationally hard problems. Both kinds of problems have a large number of applications from a variety of fields. WAOA 2005 took place in Palma de Mallorca, Spain, on 6–7 October 2005. The workshop was part of the ALGO 2005 event that also hosted ESA, WABI, and ATMOS. The two previous WAOA workshops were held in Budapest (2003) and Rome (2004).

Topics of interest for WAOA 2005 were: algorithmic game theory, approximation classes, coloring and partitioning, competitive analysis, computational finance, cuts and connectivity, geometric problems, inapproximability results, mechanism design, network design, packing and covering, paradigms, randomization techniques, real-world applications, and scheduling problems. In response to the call for papers we received 68 submissions. Each submission was reviewed by at least three referees, and the vast majority by at least four referees. The submissions were mainly judged on originality, technical quality, and relevance to the topics of the conference. Based on the reviews, the Program Committee selected 26 papers.

We are grateful to Andrei Voronkov for providing the EasyChair conference system, which was used to manage the electronic submissions, the review process, and the electronic PC meeting. It made our task much easier.

We would also like to thank all the authors who submitted papers to WAOA 2005 as well as the local organizers of ALGO 2005.

November 2005

T. Erlebach
G. Persiano

Organization

Program Co-chairs

Thomas Erlebach	University of Leicester
Giuseppe Persiano	Università di Salerno

Program Committee

Evripidis Bampis	University of Evry
Markus Bläser	ETH Zürich
Thomas Erlebach	University of Leicester
Klaus Jansen	Universität Kiel
Christos Kaklamanis	University of Patras
Marc van Kreveld	Utrecht University
Pino Persiano	Università di Salerno
Guido Proietti	Università di L'Aquila
Kirk Pruhs	University of Pittsburgh
Yuval Rabani	Technion, Haifa
Adi Rosén	Technion, Haifa
Martin Skutella	Universität Dortmund
Roberto Solis-Oba	University of Western Ontario
Frits Spieksma	Katholieke Universiteit Leuven
Berthold Vöcking	RWTH Aachen

Referees

Helmut Alt
Ernst Althaus
Pasquale Ambrosio
Eric Angel
Claudio Arbib
Estie Arkin
Takao Asano
Vincenzo Auletta
Yossi Azar
Fabien Baille
Reuven Bar-Yehuda
Nadine Baumann
Davide Bilò
Vittorio Bilò
Hans Bodlaender
Joan Boyar
Ioannis Caragiannis
Reuven Cohen
José Correa
Yves Crama
Roberto De Prisco
Florian Diedrich
Benjamin Doerr
Leah Epstein
Aleksei Fishkin
Luca Forlizzi
Stefan Funke
Olga Gerber
Laurent Gourvès
Luciano Gualà
Jurriaan Hage
Han Hoogeveen
Sandy Irani
Panagiotis Kanellopoulos
Lasse Kliemann
Christian Knauer
Reinhard Koch
Ronald Koch
Ekkehard Köhler
Jochen Könemann
Alexander Kononov
Elias Koutsoupias
Annamaria Kovacs
Sofia Kovaleva
Darek Kowalski

Christian Laforest
Van Bang Le
Stefano Leonardi
Gitta Marchand
Maren Martens
Giovanna Melideo
Joe Mitchell
Luca Moscardelli
Alfredo Navarra
René van Oostrum
Evi Papaioannou
Paolo Penna
L. Shankar Ram
Fabrizio Rossi
Guido Schäfer
Stefano Smriglio
Rob van Stee
Gerard Tel
Nicolas Thibault
Ralf Thöle
Csaba Tóth
Marc Uetz
Carmine Ventre
Tjark Vredeveld
Egon Wanke
Gerhard Woeginger
Guochuan Zhang
Michele Zito

Table of Contents

"Almost Stable" Matchings in the Roommates Problem

David J. Abraham[1,⋆], Péter Biró[2,⋆⋆], and David F. Manlove[3,⋆⋆⋆]

[1] Computer Science Department, Carnegie-Mellon University, USA
dabraham@cs.cmu.edu
[2] Department of Algebra, and Department of Computer Science and Information Theory, Budapest University of Technology and Economics, Hungary
pbiro@cs.bme.hu
[3] Department of Computing Science, University of Glasgow, UK
davidm@dcs.gla.ac.uk

Abstract. An instance of the classical Stable Roommates problem (SR) need not admit a stable matching. This motivates the problem of finding a matching that is "as stable as possible", i.e. admits the fewest number of blocking pairs. In this paper we prove that, given an SR instance with n agents, in which all preference lists are complete, the problem of finding a matching with the fewest number of blocking pairs is NP-hard and not approximable within $n^{\frac{1}{2}-\varepsilon}$, for any $\varepsilon > 0$, unless P=NP. If the preference lists contain ties, we improve this result to $n^{1-\varepsilon}$. Also, we show that, given an integer K and an SR instance I in which all preference lists are complete, the problem of deciding whether I admits a matching with exactly K blocking pairs is NP-complete. By contrast, if K is constant, we give a polynomial-time algorithm that finds a matching with at most (or exactly) K blocking pairs, or reports that no such matching exists. Finally, we give upper and lower bounds for the minimum number of blocking pairs over all matchings in terms of some properties of a stable partition, given an SR instance I.

1 Introduction

The Stable Roommates problem (SR) is a classical combinatorial problem that has been studied extensively in the literature [3,9,7,4,15,8]. An instance I of SR contains an undirected graph $G = (A, E)$ where $A = \{a_1, \ldots, a_n\}$ and $m = |E|$. We assume that G contains no isolated vertices. We interchangeably refer to the vertices of G as the *agents*, and we refer to G as the *underlying graph* of I.

⋆ Part of this work was done whilst at Department of Computing Science, University of Glasgow, and Max-Planck-Institut für Informatik.

⋆⋆ Partially supported by the Center for Applied Mathematics and Computational Physics, and by the Hungarian National Science Fund (grant OTKA F 037301).

⋆⋆⋆ Supported by the Engineering and Physical Sciences Research Council (grant GR/R84597/01), and by Royal Society of Edinburgh/Scottish Executive Personal Research Fellowship.

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 1–14, 2006.

The vertices adjacent to a given agent $a_i \in A$ are the *acceptable* agents for a_i, denoted by A_i. If $a_j \in A_i$, we say that a_i *finds* a_j *acceptable.* (Note that the acceptability relation is symmetric, i.e. $a_j \in A_i$ if and only if $a_i \in A_j$.) Moreover we assume that in I, a_i has a linear order $\prec_{a_i}$ over A_i, which we refer to as a_i's *preference list.* If $a_j \prec_{a_i} a_k$, we say that a_i *prefers* a_j to a_k. Given $a_j \in A_i$, define $rank_{a_i}(a_j) = 1 + |\{a_k \in A_i : a_k \prec_{a_i} a_j\}|$.

Let M be a matching in I. If $\{a_i, a_j\} \in M$, we say that a_i is *matched* in M and $M(a_i)$ denotes a_j, otherwise a_i is *unmatched* in M. A *blocking pair* with respect to M is an edge $\{a_i, a_j\} \in E \backslash M$ such that (i) either a_i is unmatched in M, or a_i is matched in M and prefers a_j to $M(a_i)$, and (ii) either a_j is unmatched in M, or a_j is matched in M and prefers a_i to $M(a_j)$. Let $bp_I(M)$ denote the set of blocking pairs with respect to M in I (we omit the subscript if the instance is clear from the context). Matching M is *stable* in I if $bp_I(M) = \emptyset$.

Gale and Shapley [3] showed that an instance of SR need not admit a stable matching (see for example the SR instance I_r in Figure 1 where $r = 1$). Irving [7] gave an $O(m)$ algorithm that finds a stable matching or reports that none exists, given an instance I of SR. The algorithm in [7] assumes that in I, all preference lists are *complete* (i.e. $A_i = A \backslash \{a_i\}$ for each $a_i \in A$) and n is even, though it is straightforward to generalise the algorithm to the problem model defined here (i.e. the case of *incomplete lists*) [4]. Henceforth we denote by SRC the special case of SR in which all preference lists are complete.

As the problem name suggests, an application of SR arises in the context of campus accommodation allocation, where we seek to assign students to share two-person rooms, based on their preferences over one another. Another application occurs in the context of forming pairings of players for chess tournaments [10]. Very recently, a more serious application of SR has been studied, involving pairwise kidney exchange between incompatible patient-donor pairs [14]. Here, preference lists can be constructed on the basis of compatibility profiles between patients and potential donors.

Empirical results [12] suggest that, as n increases, the probability that a random SR instance with n agents admits a stable matching decreases steeply. Equivalently, as n grows large, these results suggest that an arbitrary matching in a random SR instance with n agents is likely to admit at least one blocking pair. In practical situations, a blocking pair $\{a_i, a_j\}$ of a given matching M need not always lead to M being undermined by a_i and a_j, since these agents might not realise that together they block M. For example, in situations where preference lists are not public knowledge, there may be limited channels of communication that would lead to the awareness of blocking pairs in practice. Nevertheless, it is reasonable to assert that the greater the number of blocking pairs of a given matching M, the greater the likelihood that M would be undermined by a pair of agents in practice. Hence, given an SR instance that does not admit a stable matching, one may regard a matching that admits 1 blocking pair as being "more stable" than a matching that admits 10 blocking pairs, for example. This motivates the problem of finding, given an SR instance I with no stable matching, a matching in I that admits the fewest number of blocking pairs [11,2]. Such a matching is, in the sense described here, "as stable as possible".

Given an SR instance I, define $bp(I) = \min\{|bp_I(M)| : M \text{ is a matching in } I\}$. Define MIN-BP-SR to be the problem of finding, given an SR instance I, a matching M in I such that $bp(M) = bp(I)$. (Note that, if I is an SRC instance where n is even, clearly M must be a perfect matching in I.) In Section 2, we show that MIN-BP-SR is NP-hard and very difficult to approximate. In particular we show that MIN-BP-SR is not approximable within $n^{\frac{1}{2}-\varepsilon}$, for any $\varepsilon > 0$, unless P=NP. The result holds even for complete preference lists.

We also consider the variant SRT of SR in which preference lists may include ties. Ties arise naturally in practical applications: for example in the kidney exchange context, two donors may be equally compatible for a given patient. We also denote by SRTC the special case of SRT in which all preference lists are complete. The definition of a blocking pair in the SRT and SRTC cases is identical to that given for SR (however the term "prefers" in the SR definition is interpreted as "strictly prefers" in the presence of ties). (Note that in [8], stable matchings in SRT and SRTC are referred to as *weakly stable* matchings, where three stability definitions are given; however weak stability is the more commonly-studied notion in the literature.) Clearly an instance of SRTC need not admit a stable matching. Moreover it is known [13,8] that the problem of deciding whether a stable matching exists, given an instance of SRTC, is NP-complete. Let MIN-BP-SRT denote the variant of MIN-BP-SR in which preference lists may include ties. In Section 2, we show that MIN-BP-SRT is not approximable within $n^{1-\varepsilon}$, for any $\varepsilon > 0$, unless P=NP. The result holds even if all preference lists are complete, there is at most one tie per list, and each tie has length 2.

We now remark on the format of the inapproximability results that we present for MIN-BP-SR and MIN-BP-SRT. We implicitly assume that a given instance I of the former problem is unsolvable, so that $bp(I) \geq 1$. Recall that the solvability or otherwise of I can be determined in $O(m)$ time [7,4]. Hence $bp(I)$ can be regarded as the objective function for measuring performance guarantee. On the other hand, given an instance I of MIN-BP-SRT, we do not assume that I is unsolvable, since the problem of deciding whether this is the case is NP-complete [13,8]. Hence possibly $bp(I) = 0$, and therefore we use $opt(I)$ to measure performance guarantee, where $opt(I) = 1 + bp(I)$. In fact our inapproximability result for MIN-BP-SRT shows that, given any $\varepsilon > 0$, it is NP-hard to distinguish between the cases that I admits a stable matching, and $bp(I) \geq n^{1-\varepsilon}$.

We also consider the case that we require a matching to admit *exactly* K blocking pairs. Define EXACT-BP-SR to be the problem of deciding, given an SR instance I and an integer K, whether I admits a matching M such that $bp(M) = K$. In Section 2 we show that EXACT-BP-SR is NP-complete (even for complete preference lists). However by contrast, in Section 3, we prove that EXACT-BP-SR is solvable in polynomial time if K is a constant. In particular we give an $O(m^{K+1})$ algorithm that takes as input an SR instance I and a constant integer K, and finds a matching M in I such that $bp(M) = K$, or reports that no

$$\begin{array}{llll} a_{4i+1}: a_{4i+2} & a_{4i+3} & a_{4i+4} & \quad (0 \leq i \leq r-1) \\ a_{4i+2}: a_{4i+3} & a_{4i+1} & a_{4i+4} & \\ a_{4i+3}: a_{4i+1} & a_{4i+2} & a_{4i+4} & \quad M_r^1 = \{\{a_{4i+1}, a_{4i+2}\} : 0 \leq i \leq r-1\} \\ a_{4i+4}: a_{4i+1} & a_{4i+2} & a_{4i+3} & \quad M_r^2 = M_r^1 \cup \{\{a_{4i+3}, a_{4i+4}\} : 0 \leq i \leq r-1\} \end{array}$$

Fig. 1. Instance I_r of SR and two matchings M_r^1, M_r^2 in I_r

such matching exists. We show how to adapt this algorithm to find a matching M in I such that $bp(M) \leq K$, or report that no such matching exists.

We next give a remark regarding related work. An alternative method has been considered in the literature for coping with instances of SR that do not admit a stable matching. Tan [16] defined a *stable partition* in a given instance I of SR, which is a generalisation of the concept of a stable matching in I. Following [12], a stable partition is a permutation Π of A satisfying the following two properties (which implicitly assume that if a_i is a fixed point of Π then a_i is appended to his own preference list):

(i) for each $a_i \in A$, a_i does not prefer $\Pi^{-1}(a_i)$ to $\Pi(a_i)$;
(ii) if a_i prefers a_j to $\Pi^{-1}(a_i)$ then a_j does not prefer a_i to $\Pi^{-1}(a_j)$.

Tan [16] showed that every instance I of SR admits a stable partition, and he also gave an $O(n^2)$ algorithm for finding such a structure in I. Moreover, starting from a stable partition, Tan [17] showed how to construct, also in $O(n^2)$ time, a largest matching M in I with the property that the matched pairs in M are stable *within themselves*. However such a matching may only be half the size of a maximum (cardinality) matching in I. Yet in many applications we seek to match as many agents as possible, and as discussed above, in order to satisfy this property, in many cases a certain number of blocking pairs may be tolerated. For example, suppose that $r \geq 1$ and consider the SR instance I_r and example matchings M_r^1, M_r^2 as shown in Figure 1. Since I_r is built up from r copies of insoluble SRC instances with 4 agents, Tan's algorithm is bound to construct a matching M in I_r of size r (such as M_r^1). Any such matching M satisfies $|bp_{I_r}(M)| \geq 2r$. However M_r^2 is a solution to MIN-BP-SR in I_r, where $|M_r^2| = 2r$ and $|bp_{I_r}(M_r^2)| = r$. In particular M_r^1 is half the size of M_r^2 and admits twice as many blocking pairs.

In Section 4, for a given SR instance I, we give upper and lower bounds for $bp(I)$ in terms of some properties of a stable partition in I.

2 Inapproximability of MIN-BP-SR and MIN-BP-SRT

In this section we present reductions showing the NP-hardness and inapproximability of each of MIN-BP-SR and MIN-BP-SRT. Define MIN-MM (respectively EXACT-MM) to be the problem of deciding, given a graph G and integer K, whether G admits a maximal matching of size at most (respectively exactly) K. Our reductions utilise the NP-completeness of EXACT-MM in cubic graphs, which we now establish.

Lemma 1. EXACT-MM *is NP-complete, even for cubic graphs.*

Proof. Clearly EXACT-MM belongs to NP. To show NP-hardness, we reduce from MIN-MM, which is NP-complete even for cubic graphs [6]. Let G (a cubic graph) and K (a positive integer) be an instance of the latter problem. Without loss of generality we may assume that $K \leq \beta(G)$, where $\beta(G)$ denotes the size of a maximum matching of G. Suppose that G admits a maximal matching M, where $|M| = k \leq K$. If $k = K$, we are done. Otherwise suppose that $k < K$. We note that maximal matchings satisfy the interpolation property [5] (i.e. G has a maximal matching of size j, for $k \leq j \leq \beta(G)$) and hence G has a maximal matching of size K. The converse is clear. □

We now define some notation. Let I be an instance of SR and let A be the set of agents in I. Given $a_i \in A$, we define a set of agents $P(a_i)$ to be a *prefix* of a_i's preference list in I if $P(a_i) \subseteq A_i$ and whenever $a_j \in P(a_i)$ and a_i prefers a_k to a_j, it follows that $a_k \in P(a_i)$. The following lemma will also be required by our reduction that establishes the inapproximability of MIN-BP-SR.

Lemma 2. *Let I be an instance of* SR *with underlying graph $G = (A, E)$. Let $a_i \in A$ and let $P(a_i)$ be a prefix of a_i's preference list in I. Then, for every $k \geq 1$, there exists an instance I' of* SR *with underlying graph $G' = (A', E')$, where $A \subseteq A'$, $|A'| = |A| + 2k$ and $E \subseteq E'$, satisfying the following two properties:*

1. *if M is any matching in I in which a_i is matched and $M(a_i) \in P(a_i)$ then there is a matching M' in I' such that $M \subseteq M'$ and $bp_{I'}(M') \cap (E' \backslash E) = \emptyset$;*
2. *if M' is any matching in I' in which a_i is matched and $M'(a_i) \notin P(a_i)$, or a_i is unmatched, then $|bp_{I'}(M') \cap (E' \backslash E)| \geq k$.*

(If I is an instance of SRC *then I' is also an instance of* SRC*.)*

Proof. Let $k \geq 1$ be given. We create a set B_k of new agents, where $B_k = \{b_2, \ldots, b_{2k+1}\}$. Let $A' = A \cup B_k$. Then $|A'| = |A| + 2k$ as required. The preference list of a_i in I' is as follows:

$$a_i : [[P(a_i)]] \quad b_2 \quad b_3 \quad \ldots \quad b_{2k+1} \quad | \quad [[A_i \backslash P(a_i)]]$$

where, for $S \subseteq A_i$, $[[S]]$ denotes those members of S listed in the order induced from a_i's preference list in I. For exposition purposes, we also denote a_i by b_1.

For $2 \leq r \leq 2k+1$, the preference list of b_r in I' is as follows:

$$b_r : b_{r+1} \quad b_{r+2} \quad \ldots \quad b_{2k+1} \quad b_1 \quad b_2 \quad \ldots \quad b_{r-1} \quad | \quad \ldots$$

where ... at the end of b_r's list denotes all agents in A in arbitrary strict order.

Let $B'_k = \{b_1\} \cup B_k$. For any agent $b_r \in B'_k$, the agents to the left of the symbol | in b_r's preference list in I' are called the *proper agents* for b_r.

Finally, every agent in $A \backslash \{a_i\}$ forms a preference list in I' by appending the members of B_k to their preference list in I (in arbitrary strict order). The definition of E' follows by construction of the preference lists in I'; hence $E \subseteq E'$.

Given a matching M' in I' and an agent $b_r \in B_k$ who is matched in M', define $pr(b_r, M')$ to be the set of agents whom b_r prefers to $M'(b_r)$.

To show (1) above, let M be a matching in I such that a_i is matched in M and $M(a_i) \in P(a_i)$. Let $M' = M \cup \{\{b_r, b_{k+r}\} : 2 \leq r \leq k+1\}$. Suppose that $\{b_r, b_s\} \in bp_{I'}(M') \cap (E' \backslash E)$, where $b_r, b_s \in B_k$ and $r < s$. We firstly suppose that $2 \leq r \leq k+1$. Then $M'(b_r) = b_{r+k}$. As $b_s \in pr(b_r, M') = \{b_{r+1}, \ldots, b_{r+k-1}\}$ and $|pr(b_r, M')| = k-1$, it follows that $M'(b_s) \in \{b_{r+k+1}, \ldots, b_{2k+1}, b_2, \ldots, b_{r-1}\}$, so that $b_r \notin pr(b_s, M')$, a contradiction. Now suppose that $k+2 \leq r \leq 2k+1$. Then $M'(b_r) = b_{r-k}$. As $b_s \in pr(b_r, M') \backslash \{b_1\} = \{b_{r+1}, \ldots, b_{2k+1}, b_2, \ldots, b_{r-k-1}\}$ and $|pr(b_r, M') \backslash \{b_1\}| = k-1$, it follows that $M'(b_s) \in \{b_{r-k+1}, \ldots, b_{r-1}\}$, so that $b_r \notin pr(b_s, M')$, a contradiction. Finally it is easy to see that $\{a_j, b_l\} \notin bp_{I'}(M') \cap (E' \backslash E)$ for any $a_j \in A$ and $b_l \in B_k$. Hence $bp_{I'}(M') \cap (E' \backslash E) = \emptyset$ as required.

To show (2) above, let M' be a matching in I', and suppose that a_i is matched in M' and $M'(a_i) \notin P(a_i)$, or a_i is unmatched in M'. Then there is an agent $b_j \in B'_k$ who is not matched to a proper agent in M'. Define E'' to be the edges in the subgraph of G' induced by B'_k. Suppose $|M' \cap E''| = t$. Then $t \leq k$. Also $2(k-t)$ agents in $B'_k \backslash \{b_j\}$ are not matched to a proper agent in M'. Now suppose that $\{b_r, b_s\} \in M' \cap E''$. Then $B'_k \backslash \{b_r, b_s\} \subseteq pr(b_r, M') \cup pr(b_s, M')$. Hence either $\{b_j, b_r\}$ or $\{b_j, b_s\}$ belongs to $bp_{I'}(M') \cap (E' \backslash E)$. Now suppose that $b_r \in B'_k \backslash \{b_j\}$ is not matched to a proper agent in M'. Then $\{b_j, b_r\} \in bp_{I'}(M') \cap (E' \backslash E)$. Hence $|bp_{I'}(M') \cap (E' \backslash E)| \geq t + 2(k-t) = 2k - t \geq k$ as required. □

Henceforth we adopt the following notation, given an instance I of SR. Given an agent a_i, a prefix $P(a_i)$ of a_i's preference list and an integer $k \geq 1$, the symbol $G_k(a_i)$ in a_i's preference list following the members of $P(a_i)$ denotes the introduction of the new agents in B_k together with their preference lists, and the insertion of the members of B_k in subscript order at the relevant point in a_i's preference list, as described by the proof of Lemma 2. Given two agents a_i, a_j and integers $k, l \geq 1$, usage of the symbols $G_k(a_i)$ and $G_l(a_j)$ in the preference lists of a_i and a_j respectively implies that the agents in B_k as introduced for a_i are disjoint from the agents in B_l as introduced for a_j.

We now present a gap-introducing reduction, starting from EXACT-MM, that establishes the hardness of approximating MIN-BP-SR.

Theorem 1. MIN-BP-SR *is not approximable within* $n^{\frac{1}{2}-\varepsilon}$*, for any* $\varepsilon > 0$*, unless P=NP. The result holds even for complete preference lists.*

Proof. Let $\varepsilon > 0$ be given. Let $G = (V, E)$ (a cubic graph) and K (a positive integer) be an instance of EXACT-MM. Assume that $V = \{v_1, \ldots, v_p\}$ and $q = |E|$. We assume that $2K \leq p$, for otherwise EXACT-MM trivially has a "no" answer. Let $t = \lceil \frac{1}{\varepsilon} \rceil$ and let $C = D = p^t$. For each i $(1 \leq i \leq p)$, let $v_{j_i}, v_{k_i}, v_{l_i}$ denote the three vertices adjacent to v_i in G. For each s $(1 \leq s \leq 4)$, let $U^s = \{u_i^s : 1 \leq i \leq p\}$. Let $U = \cup_{s=1}^4 U^s$, $H = \{h_1, h_2, \ldots, h_{p-2K}\}$, $X = \{x_1, x_2, \ldots, x_C\}$, $Y = \{y_1, y_2, \ldots, y_C\}$ and $Z = \{z_i^s : 1 \leq i \leq p \wedge 1 \leq s \leq 3\}$.

For each $\{v_i, v_j\} \in E$, define $\sigma_{i,j} = 1, 2, 3$ according as v_j is v_{j_i}, v_{k_i} or v_{l_i} respectively. Also define $W_{i,j}^s = \{w_{i,j}^{r,s} : 1 \leq r \leq C\}$ $(1 \leq s \leq 2)$ and

$$
\begin{array}{lll}
u_i^1 : z_i^1 \;\; u_{j_i}^{\sigma_{j_i,i}} \;\; [W_{i,j_i}^1] \;\; [W_{i,k_i}^1] \;\; [W_{i,l_i}^1] \;\; [H] \;\; [X] \;\; \ldots & & (1 \leq i \leq p) \\
u_i^2 : z_i^2 \;\; u_{k_i}^{\sigma_{k_i,i}} \;\; [X] \;\; \ldots & & (1 \leq i \leq p) \\
u_i^3 : z_i^3 \;\; u_{l_i}^{\sigma_{l_i,i}} \;\; [X] \;\; \ldots & & (1 \leq i \leq p) \\
u_i^4 : z_i^1 \;\; z_i^2 \;\; z_i^3 \;\; [X] \;\; \ldots & & (1 \leq i \leq p) \\
z_i^s : u_i^s \;\; u_i^4 \;\; [X] \;\; \ldots & & (1 \leq i \leq p \wedge 1 \leq s \leq 3) \\
h_k : [U^1] \;\; [X] \;\; \ldots & & (1 \leq k \leq p-2K) \\
x_r : [U] \;\; [Z] \;\; [H] \;\; [W] \;\; y_r \;\; \ldots & & (1 \leq r \leq C) \\
y_r : x_r \;\; G_D(y_r) \;\; \ldots & & (1 \leq r \leq C) \\
w_{i,j}^{r,1} : w_{j,i}^{r,1} \;\; u_i^1 \;\; w_{j,i}^{r,2} \;\; [X] \;\; \ldots & (1 \leq i < j \leq p \wedge \{v_i, v_j\} \in E \wedge 1 \leq r \leq C) & \\
w_{i,j}^{r,2} : w_{j,i}^{r,2} \;\; w_{j,i}^{r,1} \;\; [X] \;\; \ldots & (1 \leq i < j \leq p \wedge \{v_i, v_j\} \in E \wedge 1 \leq r \leq C) & \\
w_{j,i}^{r,1} : w_{i,j}^{r,2} \;\; u_j^1 \;\; w_{i,j}^{r,1} \;\; [X] \;\; \ldots & (1 \leq i < j \leq p \wedge \{v_i, v_j\} \in E \wedge 1 \leq r \leq C) & \\
w_{j,i}^{r,2} : w_{i,j}^{r,1} \;\; w_{i,j}^{r,2} \;\; [X] \;\; \ldots & (1 \leq i < j \leq p \wedge \{v_i, v_j\} \in E \wedge 1 \leq r \leq C) &
\end{array}
$$

Fig. 2. Preference lists in the constructed SR instance I

$W_{i,j} = W_{i,j}^1 \cup W_{i,j}^2$. (We remark that $\{v_i, v_j\}$ gives rise to both $\sigma_{i,j}$ and $\sigma_{j,i}$, and both $W_{i,j}$ and $W_{j,i}$.) Let $W = \cup_{\{v_i,v_j\} \in E} W_{i,j}$.

We create an instance I of SRC in which the set A of agents includes $U \cup Z \cup H \cup X \cup Y \cup W$ and also additional agents that arise from instances of gadgets that are constructed implicitly by the proof of Lemma 2. The preference lists of the agents in $U \cup Z \cup H \cup X \cup Y \cup W$ are shown in Figure 2. In a given agent a's preference list, the symbol $[S]$, for $S \subseteq U \cup Z \cup H \cup X$, denotes all members of S listed in increasing subscript order. Similarly, for $S \subseteq W$, the symbol $[S]$ denotes all members of S listed in arbitrary strict order. Also, the symbol $\ldots$ denotes all remaining agents (other than a) listed in arbitrary strict order. For certain agents in I, we now define a prefix $P(a)$ of a's preference list as follows. For each agent $a \in U \cup Z \cup H \cup W$, define $P(a)$ to be the set of agents whom a prefers to every member of X. For each agent $y_r \in Y$, define $P(y_r) = \{x_r\}$.

It may be verified that the number of agents in I is $n = 7p + p - 2K + 2C + 2CD + 4qC = 2p^{2t} + 6p^{t+1} + 2p^t + 8p - 2K$ (since G is cubic), which is polynomial in the size of the given instance of EXACT-MM.

Suppose that M is a maximal matching in G, where $|M| = K$. We create a matching M' in I as follows. Let $\{v_i, v_j\} \in E$ where $i < j$. Suppose firstly that $\{v_i, v_j\} \in M$. Let $s_1 = \sigma_{i,j}$ and let $s_2 = \sigma_{j,i}$. Add the pairs $\{u_i^{s_1}, u_j^{s_2}\}$, $\{u_i^s, z_i^s\}$ $(1 \leq s \neq s_1 \leq 3)$, $\{u_i^4, z_i^{s_1}\}$, $\{u_j^s, z_j^s\}$ $(1 \leq s \neq s_2 \leq 3)$, $\{u_j^4, z_j^{s_2}\}$, $\{w_{i,j}^{r,1}, w_{j,i}^{r,1}\}$, $\{w_{i,j}^{r,2}, w_{j,i}^{r,2}\}$ to M' $(1 \leq r \leq C)$. Now suppose that $\{v_i, v_j\} \notin M$. If v_j is unmatched in M, add the pairs $\{w_{i,j}^{r,1}, w_{j,i}^{r,2}\}$, $\{w_{i,j}^{r,2}, w_{j,i}^{r,1}\}$ $(1 \leq r \leq C)$ to M', otherwise add the pairs $\{w_{i,j}^{r,1}, w_{j,i}^{r,1}\}$, $\{w_{i,j}^{r,2}, w_{j,i}^{r,2}\}$ $(1 \leq r \leq C)$ to M'.

There remain $p - 2K$ agents in U^1 who are unmatched in M' – let $u_{t_1}^1, u_{t_2}^1, \ldots, u_{t_{p-2K}}^1$ denote these agents, where $t_1 < t_2 < \ldots < t_{p-2K}$. Add $\{u_{t_k}^1, h_k\}$ and

$\{u^s_{t_k}, z^{s-1}_{t_k}\}$ to M' ($2 \leq s \leq 4$, $1 \leq k \leq p - 2K$). Next add $\{x_r, y_r\}$ to M' ($1 \leq r \leq C$). Finally, since $M'(y_r) \in P(y_r)$ for each agent $y_r \in Y$, we may extend M' by adding the edges that follow from Property 1 of Lemma 2 as applied to $G_D(y_r)$.

For each i ($1 \leq i \leq p$), there exists a unique s ($1 \leq s \leq 3$) such that $\{u^s_i, z^s_i\} \in bp(M')$. It may be verified that, by the maximality of M in G, these are all the blocking pairs of M' in I, and hence $|bp(M')| = p$.

Conversely suppose that G does not admit a maximal matching of size K. Suppose for a contradiction that $bp(I) < C$. Let M' be a matching in I such that $|bp(M')| = bp(I) < C$. Clearly every agent must be matched in M', as I is an instance of SRC and n is even. Also by Property 2 of Lemma 2, it follows that $\{y_r, x_r\} \in M'$ for all $y_r \in Y$, for otherwise $|bp(M')| \geq C$, a contradiction. Hence for each $a \in U \cup Z \cup H \cup W$, it follows that $M'(a) \in P(a)$, for otherwise $\{x_r, a\} \in bp(M')$ for all $x_r \in X$, so that $|bp(M')| \geq C$, a contradiction.

Also for each i ($1 \leq i \leq p$), $\{u^4_i, z^{s'}_i\} \in M'$ for some s' ($1 \leq s' \leq 3$). It follows that $\{z^s_i, u^s_i\} \in M'$ ($1 \leq s \neq s' \leq 3$). Now suppose that $\{u^1_i, w^{r,1}_{i,j}\} \in M'$ for some i, j ($1 \leq i, j \leq p$) and r ($1 \leq r \leq C$). Then $\{w^{r,2}_{i,j}, w^{r,2}_{j,i}\} \in M'$, for otherwise $M'(w^{r,2}_{j,i}) \notin P(w^{r,2}_{j,i})$. Hence $\{w^{r,1}_{j,i}, u^1_j\} \in M'$, for otherwise $M'(w^{r,1}_{j,i}) \notin P(w^{r,1}_{j,i})$. Define

$$M = \left\{ \{v_i, v_j\} \in E : i < j \wedge \begin{pmatrix} \{u^{s_1}_i, u^{s_2}_j\} \in M' \text{ where } 1 \leq s_1, s_2 \leq 3 \vee \\ \{u^1_i, w^{r,1}_{i,j}\} \in M' \text{ where } 1 \leq r \leq C \end{pmatrix} \right\}.$$

It follows that M is a matching in G. Also each agent in H is matched in M' to an agent in U^1, so that $|M| \leq K$. But each agent $u^s_i \in U$ satisfies $M'(u^s_i) \in P(u^s_i)$, so that $|M| = K$. Now suppose that M is not maximal in G. Then there exists some edge $\{v_i, v_j\} \in E$ such that each of v_i and v_j is unmatched in M. Hence $\{u^1_i, h_k\} \in M'$ and $\{u^1_j, h_l\} \in M'$ for some $h_k, h_l \in H$. Let r ($1 \leq r \leq C$) be given. If $\{\{w^{r,1}_{i,j}, w^{r,1}_{j,i}\}, \{w^{r,2}_{i,j}, w^{r,2}_{j,i}\}\} \subseteq M'$ then $\{w^{r,1}_{j,i}, u^1_j\} \in bp(M')$. If $\{\{w^{r,1}_{i,j}, w^{r,2}_{j,i}\}, \{w^{r,2}_{i,j}, w^{r,1}_{j,i}\}\} \subseteq M'$ then $\{u^1_i, w^{r,1}_{i,j}\} \in bp(M')$. Hence $|bp(M')| \geq C$, a contradiction. Thus M is a maximal matching of size K in G, a contradiction. Hence $bp(I) \geq C = p^t$ after all.

Next we show that $p^{t-1} > n^{\frac{1}{2}-\varepsilon}$. Firstly recall that

$$n = 2p^{2t} + 6p^{t+1} + 2p^t + 8p - 2K. \tag{1}$$

As G is cubic, we may assume that $p \geq 4$. Hence Equation 1 implies that $n < 16p^{2t}$, and thus $p^{t-1} > 16^{\frac{1-t}{2t}} n^{\frac{1}{2}-\frac{1}{2t}}$. As $t \geq \frac{1}{\varepsilon}$, it follows that

$$p^{t-1} > 4^{\frac{1-t}{t}} n^{\frac{1}{2}-\frac{\varepsilon}{2}}. \tag{2}$$

But Equation 1 also implies that $n \geq p^{2t}$, since $2K \leq p$. As $p \geq 4$, it follows that $n \geq 4^{2t} \geq 4^{\frac{2(t-1)}{\varepsilon t}}$, and hence $4^{\frac{1-t}{t}} \geq n^{-\frac{\varepsilon}{2}}$. Thus by Inequality 2, it follows that $p^{t-1} > n^{\frac{1}{2}-\varepsilon}$ as required.

Hence the existence of an $(n^{\frac{1}{2}-\varepsilon})$-approximation algorithm for MIN-BP-SR implies a polynomial-time algorithm for EXACT-MM in cubic graphs. This is a contradiction to Lemma 1 unless P=NP. □

Corollary 1. EXACT-BP-SR *is NP-complete, even for complete preference lists.*

Proof. We use the same reduction as in the proof of Theorem 1 (for any $\varepsilon < 1$) and set $K' = p$. Clearly G admits a maximal matching of size K if and only if I admits a matching with exactly K' blocking pairs. □

We now consider the case where preference lists may include ties. For a given instance I of SRT, we define $opt(I) = 1 + bp(I)$ as discussed in Section 1. The following result establishes the hardness of approximating MIN-BP-SRT.

Theorem 2. MIN-BP-SRT *is not approximable within $n^{1-\varepsilon}$, for any $\varepsilon > 0$, unless P=NP. The result holds even if all preference lists are complete, there is at most one tie per list, and each tie is of length 2.*

Proof. This result follows by adapting the proof of Theorem 1; we outline only the modifications here. For the revised reduction, choose $t = \lceil \frac{2}{\varepsilon} \rceil$, $C = p$ and $D = p^t$. Let $F = p^{t-1}$. Also, for each $z_i^s \in Z$, the agents u_i^s and u_i^4 are tied in joint first place in the preference list of z_i^s. All other preference list entries are as before. We now create F copies of each agent in $a \in U \cup Z \cup H \cup W$ – each copy of a is denoted by $a(s)$ $(1 \leq s \leq F)$. In the preference list of $a(s)$ in I, we replace b by $b(s)$ for each agent $b \in U \cup Z \cup H \cup W$ who is a proper agent for a. In the preference list of each agent in X, we replace b by $b(1), \ldots, b(F)$ for each agent $b \in U \cup Z \cup H \cup W$. For each s $(1 \leq s \leq F)$, the *class of agents* $C(s)$ comprises those agents $a(s)$ such that $a \in U \cup Z \cup H \cup W$.

As in the proof of Theorem 1, if G admits a maximal matching of size K, we may construct a matching M' in I. However M' is modified as follows: if $\{a, b\} \in M'$ for $a, b \in U \cup Z \cup H \cup W$, we replace $\{a, b\}$ by $\{a(s), b(s)\}$ $(1 \leq s \leq F)$. The presence of the ties now implies that M' is stable in I, so that $opt(I) = 1$.

Conversely if G does not admit a maximal matching of size K, then as in the proof of Theorem 1, we let M' be any matching in I such that $|bp(M')| = bp(I)$. If $\{x_r, y_r\} \notin M'$ for some r $(1 \leq r \leq C)$, it follows that $|bp(M')| \geq D$. Otherwise, it may be verified that each class of agents $C(s)$ $(1 \leq s \leq F)$ contributes at least C blocking pairs of M', for if not then G admits a maximal matching of size K. Further, these F sets of blocking pairs are pairwise disjoint, so that $|bp(M')| \geq FC = D$. Hence $opt(I) \geq D + 1 = p^t + 1$.

Next we show that $p^t \geq n^{1-\varepsilon}$. For, we firstly note that $n = (8p - 2K + 4qC)F + 2C + 2CD$, so that

$$n = 8p^{t+1} + 8p^t - 2Kp^{t-1} + 2p. \tag{3}$$

Without loss of generality we may assume that $p \geq 9$. Hence Equation 3 implies that $n \leq 9p^{t+1}$, and thus $p^t \geq 9^{-\frac{t}{t+1}} n^{1-\frac{1}{t+1}}$. As $t \geq \frac{2}{\varepsilon}$, it follows that

$$p^t \geq 9^{-\frac{t}{t+1}} n^{1-\frac{\varepsilon}{2}}. \tag{4}$$

Equation 3 also implies that $n \geq 9^t$, since $2K \leq p$. It follows that $n \geq 9^{\frac{2t}{\varepsilon(t+1)}}$, and hence $9^{-\frac{t}{t+1}} \geq n^{-\frac{\varepsilon}{2}}$. Thus by Inequality 4, it follows that $p^t \geq n^{1-\varepsilon}$ as required.

Hence the existence of an $(n^{1-\varepsilon})$-approximation algorithm for MIN-BP-SRT implies a polynomial-time algorithm for EXACT-MM in cubic graphs. This is a contradiction to Lemma 1 unless P=NP. □

We denote by EXACT-BP-SRT the extension of EXACT-BP-SR to the SRT case. Corollary 1 may be strengthened for EXACT-BP-SRT as follows. It is known that the problem of deciding whether an SRTC instance I admits a stable matching is NP-complete [13,8]. Form an SRTC instance J by adding to I a new agent a_i such that $A_i = A\backslash\{a_i\}$ and $P(a_i) = \emptyset$, together with the new agents that are created by Lemma 2 as applied to a_i, with $k = K$. Clearly I admits a stable matching if and only if J admits a matching with exactly K blocking pairs. We have therefore proved:

Theorem 3. EXACT-BP-SRT *is NP-complete for each fixed* $K \geq 0$.

3 Polynomial-Time Algorithm for Fixed *K*

In this section we consider the case that I is an SR instance with underlying graph $G = (A, E)$ and $K \geq 1$ is a fixed constant. We give an $O(m^{K+1})$ algorithm that finds a matching M in I such that $|bp_I(M)| = K$, or reports that no such matching exists. Later, we show how to modify this algorithm if we require that $|bp_I(M)| \leq K$.

Our algorithm is based on generating subsets B of edges of G, where $|B| = K$ – these edges will form the blocking pairs with respect to a matching to be constructed in a subgraph of G. Given such a set B, we form a subgraph $G_B = (A, E_B)$ of G as follows. For each agent a_i incident to an edge $e = \{a_i, a_j\} \in B$, if e is a blocking pair of a matching M, it follows that $\{a_i, a_j\} \notin M$ and a_i cannot be matched in M to an agent whom he prefers to a_j in I. Hence we delete $\{a_i, a_j\}$ from E_B, and also we delete $\{a_i, a_k\}$ from E_B for any a_k such that a_i prefers a_k to a_j in I. If any such edge $\{a_i, a_k\}$ is not in B, then we require that $\{a_i, a_k\}$ is not a blocking pair of a constructed matching M. This can only be achieved if a_k is matched in M to an agent whom he prefers to a_i in I. Hence we invoke $truncate_{a_k}(a_i)$, which represents the operation of deleting $\{a_k, a_l\}$ from E_B, for any a_l such that a_k prefers a_i to a_l in I. Additionally we add a_k to a set P to subsequently check that a_k is matched in M.

Having completed the construction of G_B, we denote by I_B the SR instance with underlying graph G_B and preference lists obtained by restricting the preferences in I to E_B. By construction of G_B, it is immediate that any matching M in G_B satisfies $B \subseteq bp_I(M)$. To avoid any additional blocking pairs in I, we seek a stable matching in I_B in which all agents in P are matched. We apply Irving's algorithm for SR [4] to I_B – suppose it finds a stable matching M in I_B. If all agents in P are matched then, as we will show, $bp_I(M) = B$, and hence $|bp_I(M)| = K$ – thus we may output M and halt. If some agents in P are unmatched in M then we need not consider any other stable matching in I_B, since Theorem 4.5.2 of [4] asserts that the same agents are matched in all stable matchings in I_B. Hence (and also in the case that no stable matching in I_B is

for each $B \subseteq E$ such that $|B| = K$
  $E_B := E$; // $G_B = (A, E_B)$ is a subgraph of G
  $P := \emptyset$;
  for each agent a_i incident to some $\{a_i, a_j\} \in B$
    delete $\{a_i, a_j\}$ from E_B;
    for each agent a_k such that a_i prefers a_k to a_j in I
      delete $\{a_i, a_k\}$ from E_B;
      if $\{a_i, a_k\} \notin B$
        $truncate_{a_k}(a_i)$;
        $P := P \cup \{a_k\}$;
  if there is a stable matching M in I_B
    if every agent in P is matched in M
      output M and **halt**;
report that no matching with K blocking pairs exists;

Fig. 3. Algorithm K-BP

found), we may consider the next subset B. If we complete the generation of all subsets B without having output a matching M, we report that no matching with the desired property exists. The algorithm is displayed as Algorithm K-BP in Figure 3. The following theorem establishes its correctness and complexity.

Theorem 4. *Given an* SR *instance I and a fixed constant K, Algorithm K-BP finds a matching with exactly K blocking pairs, or reports that no such matching exists, in $O(m^{K+1})$ time.*

Proof. Suppose firstly that the algorithm outputs a matching M when the outermost loop considered a set B. We show that M is a matching in I such that $bp_I(M) = B$. As previously mentioned, $B \subseteq bp_I(M)$. We now show that $bp_I(M) \subseteq B$. For, suppose that $\{a_k, a_l\} \in (E \backslash B) \cap bp_I(M)$. Then $\{a_k, a_l\} \notin E_B$, as M is stable in I_B. Hence $\{a_k, a_l\}$ has been deleted by the algorithm. Thus without loss of generality $a_k \in P$, so that a_k is matched in M and a_k prefers $M(a_k)$ to a_l in I. Hence $\{a_k, a_l\} \notin bp_I(M)$ after all, so that $bp_I(M) = B$.

Now suppose that M is a matching in I such that $bp_I(M) = B$, where $|B| = K$. By the above paragraph, if, before considering B, the outermost loop had already output a matching M' when considering a subset B', then $bp_I(M') = B'$, and $|B'| = K$. Otherwise, when the outermost loop considers the subset B, it must be the case that no edge of M is deleted when constructing G_B. Hence $M \subseteq E_B$. Moreover M is stable in I_B, for if not then $e \in bp_{I_B}(M)$ for some $e \in E_B$, and hence $e \in bp_I(M)$. As $B \cap E_B = \emptyset$, it follows that $e \in bp_I(M) \backslash B$, a contradiction. Finally every member of P is matched in M, for suppose $a_k \in P$ is unmatched in M. As $a_k \in P$, there is some agent a_i such that a_i prefers a_k to a_j in I, where $\{a_i, a_j\} \in B$ and $\{a_i, a_k\} \notin B$. Hence $\{a_i, a_k\} \in bp_I(M) \backslash B$, a contradiction. Hence by [4, Theorem 4.5.2], Irving's algorithm finds a stable matching M' in I_B (possibly $M' = M$) such that all members of P are matched in M'. Thus the algorithm outputs M' in this case. By the above paragraph, $bp_I(M') = B$.

On the other hand suppose that there is no matching M in I such that $|bp_I(M)| = K$. By the first paragraph, if the algorithm outputs a matching M'

when the outermost loop considered a subset B, then $bp_I(M') = B$, a contradiction. Hence the algorithm reports that no such matching M exists.

Clearly the outermost loop iterates $O(m^K)$ times. Within a loop iteration, construction of G_B takes $O(m)$ time, as does the invocation of Irving's algorithm. All other operations are $O(m)$. □

Note that it is straightforward to modify Algorithm K-BP so that it outputs the largest stable matching taken over all subsets B – we may then find a matching M such that (i) $|bp_I(M)| = K$, and (ii) M is of maximum cardinality with respect to (i). This extension uses the fact that all stable matchings in I_B have the same size [4, Theorem 4.5.2], so that the choice of stable matching constructed by the algorithm is not of significance for Condition (ii).

Finally we remark that Algorithm K-BP may easily be modified in order to find a matching M such that $bp_I(M) \leq K$: the outermost loop iterates over all subsets B of E such that $|B| \leq K$. Again, one can find a maximum such matching if required. The time complexity of the algorithm remains unchanged.

4 Upper and Lower Bounds for $bp(I)$

In this section we present upper and lower bounds for $bp(I)$, given an SR instance I, in terms of properties of a stable partition as defined in Section 1. The following results concerning stable partitions were established by Tan [16].

Theorem 5 ([16]). *Given an* SR *instance I,*

1. *I admits a stable partition Π, which may be found in $O(n^2)$ time;*
2. *if C_i is an odd-length cycle in Π of length ≥ 1 (henceforth an* odd cycle*) in Π then C_i is an odd cycle in any stable partition of Π;*
3. *I admits a stable matching if and only if Π has no odd cycle of length ≥ 3.*

Let $\mathcal{C}$ denote the set of odd cycles of length ≥ 3 in a stable partition Π. Given $C_i \in \mathcal{C}$, let $d_i = \min_{a_j \in C_i} d_G(a_j)$, where $d_G(a_j)$ denotes the degree of vertex a_j in the underlying graph G of I. We firstly give an upper bound for $bp(I)$.

Lemma 3. *Given an* SR *instance I, the bound $bp(I) \leq \sum_{C_i \in \mathcal{C}}(d_i - 1)$ holds.*

Proof. We firstly remark that the upper bound is invariant for I by Part 2 of Theorem 5. It follows by [17, Proposition 4.1] and [16, Proposition 3.2] that, by deleting a vertex of minimum degree from each odd cycle of $\mathcal{C}$, and then by decomposing each even length cycle into pairs, we obtain a matching M that is stable in the instance J of SR so obtained. It then follows by Properties (i) and (ii) of Π as given in Section 1 that every blocking pair of M in I involves a deleted vertex, and moreover for any deleted vertex a_i, if $\Pi(a_i) = a_j$ then $\{a_i, a_j\} \notin bp_I(M)$ since a_j prefers $M(a_j) = \Pi(a_j)$ to a_i. It follows that $|bp_I(M)| \leq \sum_{C_i \in \mathcal{C}}(d_i - 1)$. □

In order to derive our lower bound for $bp(I)$, it will be helpful to utilise a construction due to Cechlárová and Fleiner [1] which involves transforming a given

$a_k^1 : a_k^2 \;\; a_i \;\; a_k^4$ $\qquad$ $a_k^2 : a_k^3 \;\; a_k^1$

$a_k^3 : a_k^6 \;\; a_k^2$ $\qquad$ $a_k^4 : a_k^1 \;\; a_k^5$

$a_k^5 : a_k^4 \;\; a_k^6$ $\qquad$ $a_k^6 : a_k^5 \;\; a_j \;\; a_k^3$

Fig. 4. Preference lists of the newly-introduced agents in I_e

SR instance I into an SR instance I_e as follows. In I_e, the preference lists of the agents in A are initially the same as the corresponding preference lists in I. We then replace each edge $e_k = \{a_i, a_j\}$ (where $i < j$) in the underlying graph of I by a 6-cycle involving vertices a_k^1, a_k^2, a_k^3, a_k^4, a_k^5, a_k^6. In a_i's preference list in I_e, a_j is replaced by a_k^1, whilst in a_j's preference list in I_e, a_i is replaced by a_k^6. The preference lists of the newly-introduced agents are shown in Figure 4.

Cechlárová and Fleiner [1] showed that a stable matching M in I corresponds to a stable matching M_e in I_e, and vice versa, as follows:

- $\{a_i, a_j\} \in M \Leftrightarrow \{a_i, a_k^1\}, \{a_k^2, a_k^3\}, \{a_k^4, a_k^5\}, \{a_k^6, a_j\} \in M_e$
- $\{a_i, a_j\} \notin M$ and a_i prefers $M(a_i)$ to $a_j \Rightarrow \{a_k^1, a_k^4\}, \{a_k^2, a_k^3\}, \{a_k^5, a_k^6\} \in M_e$
- $\{a_i, a_j\} \notin M$ and a_i prefers a_j to $M(a_i) \Rightarrow \{a_k^1, a_k^2\}, \{a_k^3, a_k^6\}, \{a_k^4, a_k^5\} \in M_e$
- $\{a_i, a_j\} \notin M \Leftarrow \{a_k^1, a_k^4\}, \{a_k^2, a_k^3\}, \{a_k^5, a_k^6\} \in M_e$ or $\{a_k^1, a_k^2\}$, $\{a_k^3, a_k^6\}$, $\{a_k^4, a_k^5\} \in M_e$

where $\{a_i, a_j\} = e_k$. Similarly, given stable partitions Π and Π_e in I and I_e respectively, we can prove that $\Pi(a_i) = a_j$ in an odd cycle if and only if, in Π_e:

- if $i < j$ then $\langle a_i, a_k^1, a_k^2, a_k^3, a_k^6, a_j \rangle$ is in an odd cycle and $\langle a_k^4, a_k^5 \rangle$ is a cycle;
- if $j < i$ then $\langle a_i, a_k^6, a_k^5, a_k^4, a_k^1, a_j \rangle$ is in an odd cycle and $\langle a_k^2, a_k^3 \rangle$ is a cycle.

Lemma 4. *Given an* SR *instance I, the bound $bp(I) \geq \left\lceil \frac{|\mathcal{C}|}{2} \right\rceil$ holds.*

Proof. It follows from the proof of Theorem 4 that $bp(I) = k$ if and only if k is the minimum number for which there exists a set S of k edges such that the SR instance I' obtained by deleting the edges in S from I admits a stable matching. To delete an edge $e_k = \{a_i, a_j\}$ from I is equivalent to deleting the two vertices a_k^1 and a_k^6 from I_e. That is, after deleting the above set S of edges, instance I' has a stable matching if and only if, after deleting the corresponding k pairs of vertices from I_e, the obtained instance I_e' has a stable matching. But by [17, Theorem 4.2], the number of odd cycles can decrease by at most one after deleting one vertex, so after deleting k edges from I, the number of odd cycles can decrease by at most $2k$ in I_e. Hence if $|\mathcal{C}| > 2k$, then I_e' still has at least one odd cycle of length ≥ 3, so neither I_e' nor I' can admit a stable matching. □

5 Concluding Remarks

The strong inapproximability results presented in this paper are perhaps surprising, in view of Theorem 5 and the various structural properties of a stable partition [16,17]. We conclude with two open problems.

Firstly, given an SR instance I and a matching M in I, it follows that $bp(M) \leq m = O(n^2)$. Is there an approximation algorithm for MIN-BP-SR with performance guarantee $o(m)$?

Secondly, it remains open to determine whether the bounds for $bp(I)$ presented in Section 4 are tight, and in particular to establish values of k_n and to obtain a characterisation of I_n such that I_n is an SR instance with n agents, in which $bp(I_n) = k_n$ and $bp(I_n)$ is maximum over all SR instances with n agents.

Acknowledgements

We would like to thank Katarína Cechlárová and Rob Irving for helpful discussions. The problem of finding a matching with at most K blocking pairs, for a fixed integer K, was suggested by Rob Irving.

References

1. K. Cechlárová and T. Fleiner. On a generalization of the stable roommates problem. *ACM Transactions on Algorithms*, 1(1):143–156, 2005.
2. K. Eriksson and P. Strimling. How unstable are matchings from decentralized mate search? Preprint, 2005. Submitted for publication.
3. D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.
4. D. Gusfield and R.W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, 1989.
5. F. Harary. Maximum versus minimum invariants for graphs. *J. Graph Theory*, 7:275–284, 1983.
6. J.D. Horton and K. Kilakos. Minimum edge dominating sets. *SIAM J. Discrete Mathematics*, 6:375–387, 1993.
7. R.W. Irving. An efficient algorithm for the "stable roommates" problem. *J. Algorithms*, 6:577–595, 1985.
8. R.W. Irving and D.F. Manlove. The Stable Roommates Problem with Ties. *J. Algorithms*, 43:85–105, 2002.
9. D.E. Knuth. *Mariages Stables*, Les Presses de L'Université de Montréal, 1976.
10. E. Kujansuu, T. Lindberg, and E. Mäkinen. The stable roommates problem and chess tournament pairings. *Divulgaciones Matemáticas*, 7(1):19–28, 1999.
11. M. Niederle and A.E. Roth. Market culture: How norms governing explording offers affect market performance. NBER working paper 10256, January 2004.
12. B.G. Pittel and R.W. Irving. An upper bound for the solvability probability of a random stable roommates instance. *Rand. Struct. Algorithms*, 5(3):465–486, 1994.
13. E. Ronn. NP-complete stable matching problems. *J. Algorithms*, 11:285–304, 1990.
14. A.E. Roth, T. Sönmez, and M. Utku Ünver. Pairwise kidney exchange. To appear in *Journal of Economic Theory*.
15. A.E. Roth and M.A.O. Sotomayor. *Two-sided matching: a study in game-theoretic modeling and analysis* Cambridge University Press, 1990.
16. J.J.M. Tan. A necessary and sufficient condition for the existence of a complete stable matching. *J. Algorithms*, 12:154–178, 1991.
17. J.J.M. Tan. Stable matchings and stable partitions. *International J. Computer Mathematics*, 39:11–20, 1991.

On the Minimum Load Coloring Problem

Extended Abtract

Nitin Ahuja[1], Andreas Baltz[2], Benjamin Doerr[3],
Aleš Přívětivý[4], and Anand Srivastav[2]

[1] Department of Mathematical Optimization, Technical University Braunschweig,
Pockelsstrasse 14, D-38106 Braunschweig, Germany
n.ahuja@tu-bs.de

[2] Department of Computer Science, Christian-Albrechts-University Kiel,
Christian-Albrechts-Platz 4, D-24098 Kiel, Germany
{aba, asr}@numerik.uni-kiel.de

[3] Max-Planck-Institute for Computer Science,
Stuhlsatzenhausweg 85, D-66123 Saarbrücken, Germany
doerr@mpi-sb.mpg.de

[4] Department of Applied Mathematics, Charles University,
Malostranské nám. 25, 11800 Praha, Czech Republic
privetivy@kam.mff.cuni.cz

Abstract. Given a graph $G = (V, E)$ with n vertices, m edges and maximum vertex degree Δ, the *load distribution* of a coloring $\varphi : V \to \{\text{red, blue}\}$ is a pair $d_\varphi = (r_\varphi, b_\varphi)$, where r_φ is the number of edges with at least one end-vertex colored red and b_φ is the number of edges with at least one end-vertex colored blue. Our aim is to find a coloring φ such that the (maximum) *load*, $l_\varphi := \max\{r_\varphi, b_\varphi\}$, is minimized. The problem has applications in broadcast WDM communication networks (Ageev et al., 2004). After proving that the general problem is NP-hard we give a polynomial time algorithm for optimal colorings of trees and show that the optimal load is at most $m/2 + \Delta \log_2 n$. For graphs with genus $g > 0$, we show that a coloring with load $\mathrm{OPT}(1 + o(1))$ can be computed in $O(n + g)$-time, if the maximum degree satisfies $\Delta = o(\frac{m^2}{ng})$ and an embedding is given. In the general situation we show that a coloring with load at most $\frac{3}{4}m + O(\sqrt{\Delta m})$ can be found in deterministic polynomial time using a derandomized version of Azuma's martingale inequality. This bound describes the "typical" situation: in the random multi-graph model we prove that for almost all graphs, the optimal load is at least $\frac{3}{4}m - \sqrt{3mn}$. Finally, we generalize our results to k–colorings for $k > 2$.

1 Introduction

We consider the following problem. We are given a graph $G = (V, E)$ on n vertices and m edges. The load of a k-coloring $\varphi : V \to \{1, \ldots, k\}$ is

$$\max_{i \in \{1,\ldots,k\}} |\{e \in E \mid \varphi^{-1}(i) \cap e \neq \emptyset\}|,$$

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 15–26, 2006.

the maximum number of edges with at least one end-point in color i, where the maximum is taken over all $i \in \{1, \ldots, k\}$. The problem of minimizing this load arises naturally in wavelength division multiplexing (WDM) networks with broadcast traffic: here, the nodes represent senders/receivers each of which wants to send messages to every other node via one of k available wavelength channels. The objective is to assign to each node a channel, such that the maximum traffic taken over all channels is minimized. Ageev et al. [1] consider scheduling aspects of the capacitated weighted version of this problem. Closely related is the k–*balanced graph partitioning problem* [6], where the aim is to find a set of edges of minimum capacity such that removing these edges partitions the graph into at most k roughly equally weighted and connected subgraphs.

In this paper the focus is on coloring the vertices of a graph with 2 colors, red and blue. For a *coloring* $\varphi : V \to \{\text{red, blue}\}$ we define the *load distribution* of φ by $d_\varphi := (r_\varphi, b_\varphi)$, where r_φ counts the number of edges incident with at least one red vertex, and b_φ is the number of edges incident with at least one blue vertex. The aim is to find a coloring φ such that the maximum load, $l_\varphi := \max\{r_\varphi, b_\varphi\}$, is minimized. In the following we shall skip the term "maximum" and refer to l_φ simply as the *load* of the coloring φ. We call the problem of finding a coloring φ that minimizes l_φ *Minimum Load Coloring Problem (MLCP).*

1.1 Our Results

After some preliminaries including the establishment of NP-hardness of the problem in Section 2, we show how to solve MLCP on trees optimally in $O(n^3)$ time (Section 3). Such an optimal solution is proven to have a load of at most $\frac{1}{2}m + \Delta \log_2 n$. Section 4 is concerned with graphs of genus $g > 0$. With a separator theorem proved with techniques from Djidjev [5] we obtain an $O(n+g)$-time algorithm for constructing a coloring with load bounded by $m/2+48\sqrt{g\Delta n}$. This is a $(1 + o(1))$-approximation in case $\Delta = o(\frac{m^2}{ng})$. In Section 5 we analyze arbitrary instances of the problem. We show that a random coloring has load $\frac{3}{4}m + O(\sqrt{\Delta m})$ with high probability. This immediately yields a randomized algorithm. Furthermore, using an algorithmic version of the Azuma-inequality, we derive a deterministic $O(n^3)$–time algorithm for computing colorings with the same load-bound. This is quite strong: in the random multi-graph model (and similarly in other random models), almost all graphs have no coloring with load less than $\frac{3}{4}m - \sqrt{3mn}$. In the last section we extend our results to $k > 2$ colors.

2 Preliminaries

In this section we state some basic facts. Let

$$l(G) := \min\{l_\varphi \mid \varphi : V \to \{\text{red,blue}\}\}$$

denote the optimal load of a graph $G = (V, E)$. Given a red-blue coloring φ, we shall denote the number of "cut edges" that connect a red vertex with a blue vertex by c_φ. We will refer to the set of red vertices as V_r and to the set of blue vertices as V_b.

Since every edge of G is counted as red or blue (or both), $l(G) \geq \frac{m}{2}$. Obviously, every red-blue coloring of G has load at most m, so we see that each two-coloring of a graph G is a 2-approximation of $l(G)$.

Let G be a star with $d+1$ vertices, then $l(G) = d$. In fact, the maximum degree Δ of the input graph is another lower bound on $l(G)$. It is also easy to find an optimal two-coloring of cycles and chains (graphs consisting of a single open path). Here, each of the two classes in an optimal coloring forms a connected component. This is already false for trees (cf. Section 3).

Let us observe that for regular graphs, MLCP is equivalent to MinBisection.

Lemma 1. *Let $k \in \mathbb{N}$. Let $G = (V, E)$ be a k-regular graph with $n := |V|$ even, and let $\varphi : V \to \{red, blue\}$ be an optimal coloring, then either $|V_r| = |V_b|$ or an optimal coloring with $|V_r| = |V_b|$ can be obtained by recoloring an arbitrary vertex of the larger color class.*

Proof. Suppose that $|V_r| > |V_b|$. The number of red edges is $r_\varphi = \frac{|V_r|\cdot k}{2} + \frac{c_\varphi}{2}$, and the number of blue edges is $b_\varphi = \frac{|V_b|\cdot k}{2} + \frac{c_\varphi}{2}$, hence

$$r_\varphi - b_\varphi = \frac{k}{2}(|V_r| - |V_b|) \geq k$$

since n is even. If we change the color of an arbitrary red vertex v into blue, the number of red edges decreases by at most k, while the number of blue edges increases by at most k. Consequently, l_φ does not increase and the resulting coloring is still optimal. On the other hand, l_φ must not decrease either. This means that r_φ has to stay the same or b_φ has to increase by at least k. Either of these events can occur only if v has only red neighbors. Since v is an arbitrary red vertex, we conclude that G consists of monochromatic components. If $|V_r| > |V_b| + 2$ we can recolor another red vertex v' without increasing l_φ. But choosing v' as a neighbor of v results in an overall decrease of l_φ contradicting the choice of φ as an optimal coloring. Hence $|V_r| = |V_b| + 2$, and thus recoloring v yields an optimal coloring with $|V_r| = |V_b|$. $\square$

Given a k-regular graph with an even number of vertices, we see by Lemma 1 that every optimal coloring φ induces a bisection of V (either at once or after recoloring an arbitrary vertex of the larger class) with

$$l_\varphi = \frac{n}{2} \cdot \frac{k}{2} + \frac{c_\varphi}{2}.$$

Since φ is optimal, c_φ, the size of the edge cut separating the classes V_r and V_b, is minimum, so we have a minimum bisection. On the other hand, every minimum bisection V_1, V_2 of V gives rise to a coloring with load

$$\frac{n}{2} \cdot \frac{k}{2} + \frac{|E(V_1, V_2)|}{2},$$

which is obviously optimal. Hence MLCP and MinBisection are equivalent on regular graphs. For $k \geq 3$, MinBisction on k-regular graphs is as hard as general MinBisection (see [3]). Since the decision version of MinBisection is NP-complete [7], and the load of any proposed solution for MLCP can be evaluated in polynomial time, we have established NP-completeness also for MLCP.

Theorem 1. *The decision version of MLCP is NP-complete.*

3 Polynomial Time Algorithms for Trees

In this section, we show how to efficiently compute an optimal solution for the MLCP on trees. We also show that any tree G with n vertices and maximum vertex degree Δ has load at most $l(G) \leq \frac{n-1}{2} + \Delta \log_2 n$. The key to prove this result is the following more general lemma.

Lemma 2. *Let $G = (V, E)$ be a tree on n vertices and let $m_1, m_2 \in \mathbb{N}$ such that $m_1 + m_2 = n - 1$. Then there is a red-blue coloring of V such that at least $m_1 + 1 - \Delta \log_2 n$ edges are monochromatic red and at least $m_2 + 1 - \Delta \log_2 n$ are monochromatic blue.*

Proof. We use induction. Clearly, the lemma holds for $n \leq 3$. Let us assume that the lemma holds for all trees on less than n vertices. Let $v \in V$ be a vertex such that deleting v breaks G into $k \geq 2$ components C_i, $i \in \{1, \ldots, k\}$, where the number of vertices n_i in component C_i is at most $n/2$. (To show the existence of v, assume for the sake of a contradiction that each vertex is the origin of at least one branch with more than $\frac{n}{2}$ nodes. Let v be a vertex whose maximum branch C is minimum, and let v' be the neighbor of v in C. Denote the maximum branch of v' by C'. Then, $|C'| \leq \max\{n - |C|, |C| - 1\} \leq \max\{\frac{n}{2} - 1, |C| - 1\} < |C|$ contradicting the minimality of $|C|$.) It is easy to see that there exist $I_1, I_2 \subseteq \{1, \ldots, k\}$ such that:

(i) $I_1 \cap I_2 = \emptyset$,
(ii) $|\{1, \ldots, k\} \setminus (I_1 \cup I_2)| = 1$,
(iii) $\sum_{i \in I_1} n_i \leq m_1$, and
(iv) $\sum_{i \in I_2} n_i \leq m_2$.

Note that either I_1 or I_2 can also be empty, but not both. Color the vertices of components with indices in I_1 (resp. I_2) with red (resp. blue). The central vertex v is arbitrarily colored red or blue. Let C_j be the component that is left uncolored, that is, $\{1, \ldots, k\} \setminus (I_1 \cup I_2) = \{j\}$. Let $m'_1 = m_1 - (\sum_{i \in I_1} n_i) - 1$ and $m'_2 = m_2 - \sum_{i \in I_2} n_i$. Then, $m'_1 + m'_2 = n_j - 1$ is a partition of the number of edges of C_j. By induction, there is a red-blue coloring of C_j such that at least $m'_1 + 1 - \Delta \log_2 n_j$ of its edges are monochromatic red and at least $m'_2 + 1 - \Delta \log_2 n_j$ are monochromatic blue. Now, the total number of monochromatic red edges is at least $\sum_{i \in I_1}(n_i - 1) + m'_1 - \Delta \log n_j \geq m_1 - |I_1| - \Delta \log_2(n/2)$, which is at least $m_1 + 1 - \Delta \log_2 n$. Similarly, the total number of monochromatic blue edges is at least $m_2 + 1 - \Delta \log_2 n$. □

We did not try to optimize the error term $\Delta \log_2 n$. It is clear that it has to contain a linear dependence on Δ — this is shown by stars — and a logarithmic dependence on the number of vertices. The latter is shown by a complete ternary tree (proof omitted). This example also demonstrates that in an optimal coloring the color classes may induce disconnected subgraphs. From the lemma, we easily deduce the following.

Theorem 2. *Let $G = (V, E)$ be a tree on n vertices with maximum vertex degree Δ. Then $l(G) \leq \frac{n-1}{2} + \Delta \log_2 n$.*

Note that the proof of Lemma 2 is constructive. We thus have an efficient algorithm computing colorings with load at most $\frac{n-1}{2} + \Delta \log_2 n$. However, it is also possible to compute optimal colorings for trees efficiently.

Theorem 3. *On trees with n vertices, MLCP can be solved in time $O(n^3)$.*

Proof. Let $G = (V, E)$ be a tree on n vertices. Let us consider G as being rooted in some arbitrary vertex a. We assign each $v \in V$ a distance dist_v given by the length of the path from a to v and view each edge $e \in E$ as pointing from lower to higher level nodes. So, we think of G as a directed tree with the root a at level 0, the successors $N(a) := \{v \in V \mid (a, v) \in E\}$ of a at level 1, etc. For each $v \in V$ we denote by T_v the induced subtree of G rooted in v, i.e., T_v is the subgraph of G induced by v and *all* of its (iterated) successors. We define for each *arbitrary* subtree G' of G with root a',

$$D_{G'} := \{(r, b) \mid (r, b) = d_\varphi \text{ for some coloring } \varphi \text{ of } G' \text{ with } \varphi(a') = \text{red}\},$$

the set of possible load distributions for G' (we may assume $\varphi(a') = \text{red}$ without loss of generality). Suppose, we can efficiently compute D_G. Since $|D_G| \leq n^2$, we can also efficiently find the load $l(G)$ of an optimal coloring by searching D_G for the load distribution with smallest maximum component. We will show that D_G can be determined in polynomial time by iteratively computing D_{T_v} for all $v \in V$, in *reverse breadth first order*. The iteration is based on two operations:

(i) Consider a subtree G' of G with root $a' \neq a$, $v \in V$ with $(v, a') \in E$, and the tree $v + G' := (V(G') \cup \{v\}, E(G') \cup \{(v, a')\})$ obtained by appending the edge (v, a') to G'. We define

$$v + D_{G'} := \{(r + 1, b) \mid (r, b) \in D_{G'}\} \cup \{(b + 1, r + 1) \mid (r, b) \in D_{G'}\}. \quad (1)$$

(ii) Consider two subtrees G'_1, G'_2 of G that do not intersect but in their joint root a'. Let $G'_1 + G'_2 := (V(G'_1) \cup V(G'_2), E(G'_1) \cup E(G'_2))$ denote the composite tree and define

$$D_{G'_1} + D_{G'_2} := \{(r_1 + r_2, b_1 + b_2) \mid (r_1, b_1) \in D_{G'_1}, (r_2, b_2) \in D_{G'_2}\}. \quad (2)$$

Since for each tree G' we defined $D_{G'}$ to contain only load distributions of colorings where the root of G' is colored red, it will be necessary to eventually flip colors in the course of our desired iteration. For convenience, let us denote the *inverse coloring* of a given coloring φ by $\overline{\varphi}$.

Claim 1. *For all subtrees $G' = (V', E')$ of G with root a' and all $v \in V$ with $(v, a') \in E$, $D_{v+G'} = v + D_{G'}$.*

Proof. Let $(r, b) \in D_{v+G'}$ and let $\varphi : V' \cup \{v\} \to \{\text{red, blue}\}$ be a coloring with $d_\varphi = (r, b)$ and $\varphi(v) = \text{red}$. Then $\varphi' := \varphi|_{V'}$ is a coloring of G'. If $\varphi'(a') = \text{red}$, then $(r', b') := d_{\varphi'} = (r-1, b) \in D_{G'}$ and thus $(r, b) = (r'+1, b) \in v + D_{G'}$, whereas if $\varphi'(a')$= blue, then $d_{\varphi'} = (r-1, b-1)$ and $\overline{\varphi'}$ induces a load distribution $d_{\overline{\varphi'}} = (r', b') := (b-1, r-1) \in D'_G$, so $(r, b) = (b'+1, r'+1) \in v + D_{G'}$.

Let $(r, b) \in v + D_{G'}$. There is a coloring $\varphi : V' \to \{\text{red, blue}\}$ with $\varphi(a') = \text{red}$ and either $d_\varphi = (r-1, b)$ or $d_\varphi = (b-1, r-1)$. In the first case, extending φ to $V' \cup \{v\}$ by coloring v red gives a coloring φ' of $v + G'$ with $d_{\varphi'} = (r, b)$, in the second case we similarly extend $\overline{\varphi'}$. □

Claim 2. *For all subtrees $G'_1 = (V'_1, E'_1), G'_2 = (V'_2, E'_2)$ intersecting only in their joint root a', $D_{G'_1+G'_2} = D_{G'_1} + D_{G'_2}$.*

Proof. Let $(r, b) \in D_{G'_1+G'_2}$ and let $\varphi : V'_1 \cup V'_2 \to \{\text{red, blue}\}$ be a coloring with $d_\varphi = (r, b)$ and $\varphi(a') = \text{red}$. Obviously, $\varphi|_{V'_1}$ and $\varphi|_{V'_2}$ are colorings of G'_1 and G'_2, respectively, with $\varphi|_{V'_1}(a') = \varphi|_{V'_2}(a') = \text{red}$ and $d_{\varphi|_{V'_1}} + d_{\varphi|_{V'_2}} = (r, b)$. Hence $D_{G'_1+G'_2} \subseteq D_{G'_1} + D_{G'_2}$.

On the other hand, if $(r, b) \in D_{G'_1} + D_{G'_2}$, then there are colorings φ_1, φ_2 of G'_1 and G'_2, respectively, with $d_{\varphi_1} = (r_1, b_1)$, $d_{\varphi_2} = (r_2, b_2)$, $(r_1 + r_2, b_1 + b_2) = (r, b)$, and $\varphi_1(a') = \varphi_2(a') = \text{red}$. Clearly, $\varphi' := \varphi_1 \cup \varphi_2$ is a coloring of $G'_1 + G'_2$ with $\varphi'(a') = \text{red}$ and $d_{\varphi'} = (r, b)$, thus $D_{G'_1} + D_{G'_2} \subseteq D_{G'_1+G'_2}$. □

As an easy consequence we observe the following fact.

Corollary 1. *For all $v \in V$,*

$$D_{T_v} = \sum_{v' \in N(v)} D_{v+T_{v'}} = \sum_{v' \in N(v)} v + D_{T_{v'}}.$$

Now the algorithm for computing $l(G)$ is straightforward:

1. Let level $:= \max\{\text{dist}_v \mid v \in V\} - 1$, $D_{T_{v'}} := \{(1, 0)\}$ for all $v' \in V$ with $\text{dist}_{v'} = \text{level} + 1$.
2. For all $v \in V$ with $\text{dist}_v = \text{level}$: compute $D_{T_v} = \sum_{v' \in N(v)} v + D_{T_{v'}}$.
3. Set level := level − 1.
4. If level ≥ 0 then go to 2.
5. Output $\min\{\max\{r, b\} \mid (r, b) \in D_{T_a}\}$.

Note that the time required for operation (1) is bounded by $2|D_{G'}| = O(n^2)$, since we have to consider each $(r, b) \in D_{G'}$ twice, and (r, b) takes at most n^2 values. Operation (2) consists of $|D_{G'_1}| \cdot |D_{G'_2}| = O(n^4)$ steps. The running time of the algorithm is dominated by the iterated calls of line 2, i.e., by the computations of D_{T_v}. Computing D_{T_v} involves $\deg(v)$ operations of type (2), where each summand is computed via a type (1) operation. Hence, the overall running time is bounded by $\sum_{v \in V} \deg(v) \cdot O(n^4 + n^2) = O(n^5)$. However, we can reduce the running time to $O(n^3)$ by neglecting "irrelevant" colorings. Note that, if (r, b_1) and $(r, b_2) \in D_{T_v}$ are possible load distributions for a tree T_v imposed by colorings φ_1 and φ_2, then the load distribution with larger second component, say (r, b_2), will be irrelevant for computing $l(G)$ (suppose, φ is an optimal coloring of G with $\varphi_{|T_v} = \varphi_2$, then replacing φ on T_v by φ_1 will not increase the load). Thus, for each r we have to store only $b := \min\{b' \mid (r, b') \in D_{T_v}\}$. Defining the set of *relevant load distributions*

$$\hat{D}_{G'} := \{(r, b) \mid (r, b) \in D_{G'}, b = \min\{b' \mid (r, b') \in D_{G'}\}\}$$

for each subtree G' of G, we have that $|\hat{D}_{G'}| = O(n)$. Obviously, $\hat{D}_G$ can be computed iteratively via operations similar to (1) and (2) that are performed on $\hat{D}_{G'}$ instead of $D_{G'}$ and thus require only $O(n)$ and $O(n^2)$ steps, respectively. This yields the desired $O(n^3)$ bound. The iterative procedure for computing D_G (or $\hat{D}_G$) can be easily modified such that it gives not only the optimal load, but also an optimal coloring. All we have to do is store, for each $(r, b) \in \hat{D}_{T_v}$ and each $v' \in N(v)$ a pair $((r', b'), i) =: p_{v'}(r, b)$, where $(r', b') \in \hat{D}_{T_{v'}}$ was used in the computation of (r, b) and $i \in \{1, 0\}$ indicates whether of not in computing (r, b) from (r', b') we swapped the colors of $T_{v'}$. Starting from an optimal load distribution $d = (r_0, b_0)$ we trace back the load computations via p and determine for each node an optimal color with the following algorithm.

1. Define $\varphi(a) :=$ red, $v := a$, $d := (r_0, b_0)$, $M = \emptyset$.
2. Set $M := M \cup \{(v, v', p_{v'}(d)) \mid v' \in N(v)\}$.
3. If $M = \emptyset$ then output φ and stop.
4. Let $(v, v', ((r', b'), i)) \in M$, set $M := M \setminus (v, v', ((r', b'), i))$.
5. Define $\varphi(v') := \begin{cases} \varphi(v) & \text{if } i = 0 \\ \{\text{red,blue}\} \setminus \varphi(v) & \text{otherwise.} \end{cases}$
6. Set $v := v'$, $d := (r', b')$ and go to 2.

This algorithm can be implemented to run in $O(n)$ time. Thus the time required to solve MLCP on trees with n vertices is $O(n^3)$ in total. This ends the proof of Theorem 3. □

4 An Approximation Algorithm for Graphs with Genus *g*

In this section, we show how a $(1 + o(1))$-approximate solution for the MLCP for graphs of genus $g > 0$ can be computed if $\Delta = o(\frac{m^2}{ng})$. Recall that the genus of a graph is the smallest integer g such that the graph can be drawn without crossing itself on a sphere with g "handles". The problem of determining the genus of a graph is NP-hard [12]. A trivial upper bound on the genus g of a graph with m edges and n vertices is $m - 1$ since each crossing of two edges can be eliminated by introducing a handle. A lower bound of $g \geq \frac{m-3n}{6} + 1$ can be obtained by generalizing Euler's formula for planar graphs (see [13]). The main idea of our algorithm is to partition V into two sets A and B such that

– the number of edges having both endpoints in A is at most $m/2$,
– the same holds for B,
– there are only $O(\sqrt{g\Delta n})$ edges between the sets A and B.

By coloring A and B with different colors, we obtain a coloring φ with $l_\varphi(G) \leq m/2 + c\sqrt{g\Delta n}$. Since $l(G) \geq m/2$, for $\Delta = o(\frac{m^2}{gn})$ we have a $(1 + o(1))$-approximate solution. A polynomial time algorithm finding a partition with small vertex separator for planar graphs ($g = 0$) was described in [8,4] and then extended for graphs of genus $g > 0$ in [5]. Let $E(A)$, $E(B)$, and $E(A, B)$ denote the sets of monochromatic edges in A, B, and the set of bichromatic edges connecting A and B, respectively. For our purpose we use the following theorem, given in [11].

Theorem 4 [11]. *Let G be a graph of genus $g > 0$, having nonnegative vertex weights summing to one such that no weight exceeds $2/3$. There is a partition of V into sets A and B, such that weight$(A) \leq 2/3$, weight$(B) \leq 2/3$, and $|E(A, B)| \leq 5\sqrt{3g\Delta n}$. Provided that we are given an embedding of G into its genus surface, there is an $O(n + g)$-time algorithm which finds such a partition.*

We can use this theorem in the following way: for any graph of genus $g > 0$ we assign to each vertex $v \in V$ a weight $w(v) = \frac{\deg(v)}{2m}$. The theorem yields a partition of V into A and B, such that $|E(A)| \leq \frac{2}{3}m$, $|E(B)| \leq \frac{2}{3}m$ and there are at most $5\sqrt{3g\Delta n}$ edges between A and B. This $\frac{2}{3}$ factor can be reduced to $\frac{1}{2}$ by iterating the algorithm on the bigger of the sets resulting from the partitioning. Both, the size of the edge separator and the running time, increase only by a constant factor. We summarize this in the following theorem. The proof is similar to the proof of Corollary 3 in [8], and thus will be given only in the full version of the paper.

Theorem 5. *Let G be a graph of genus $g > 0$. There is a partition of V into sets A, B, such that $|E(A)| \leq \frac{1}{2}m$, $|E(B)| \leq \frac{1}{2}m$, and $|E(A, B)| \leq 48\sqrt{g\Delta n}$. Provided that we are given an embedding of G into its genus surface, there is an algorithm which finds such a partition in time $O(n + g)$.*

Corollary 2. *Let G be any graph of genus $g > 0$. Given an embedding of G into its genus surface, a coloring φ with $l_\varphi(G) \leq m/2 + 48\sqrt{g\Delta n}$ can be constructed in time $O(n + g)$.*

For a planar graph G, we can similarly use the separator theorem from [4] to show that a coloring φ with $l_\varphi(G) \leq \frac{m}{2} + (6\sqrt{2} + 4\sqrt{3})\sqrt{\Delta n}$ can be constructed in time $O(n)$, provided that an embedding is given.

5 Randomized Approximation

5.1 Approximation for General Graphs

In this section, we study the MLCP on arbitrary graphs. Since the problem is NP–hard, approximate solutions are the best one can expect to find efficiently.

We first analyze the load of random colorings. With high probability, their load is less than $\frac{3}{4}m + O(\sqrt{\Delta m})$. This shows existence of such colorings, and also yields a randomized algorithm. Using an algorithmic version of the Azuma-inequality, we derive a deterministic algorithm for computing such colorings. Since $\frac{1}{2}m$ is a trivial lower bound for l_φ, these results yield a $(1.5 + o(1))$–approximation algorithm if $\Delta = o(m)$.

To analyze random colorings, we use the following martingale inequality[1] that can be found in McDiarmid [9]. It is an application of the well known inequality of Azuma [2]:

Lemma 3. *Let $X_1, \ldots, X_n$ be independent random variables taking values in some sets $A_1, \ldots, A_n$. Let $f : \prod_{i=1}^n A_i \to \mathbb{R}$ such that $|f(x) - f(y)| \leq c_i$ whenever x and y differ only in the ith coordinate. Let $X = (X_1, \ldots, X_n)$ and $\mu = \mathbb{E}(f(X))$. Then for any $\lambda \geq 0$,*

$$\mathbb{P}(f(X) - \mu \geq \lambda) \leq \exp\Big(-2\lambda^2 / \sum_{i=1}^n c_i^2\Big). \tag{3}$$

Theorem 6. *There is a coloring φ such that $l_\varphi \leq \frac{3}{4}m + \sqrt{(\ln 2)\Delta m}$. For all $q \geq 0$, a random coloring satisfies $\mathbb{P}\Big(l_\varphi \geq \frac{3}{4}m + q\sqrt{(\ln 2)\Delta m}\Big) \leq 2^{-q^2+1}$.*

Proof. We analyze the behavior of a random coloring. Let $\varphi : V \to \{\text{red}, \text{blue}\}$ such that $\mathbb{P}(\varphi(v) = \text{red}) = \frac{1}{2} = \mathbb{P}(\varphi(v) = \text{blue})$ independently for all $v \in V$. Clearly, if two colorings φ_1, φ_2 differ only in the color of some vertex $v \in V$, then $|r_{\varphi_1} - r_{\varphi_2}| \leq \deg(v)$. We compute $\mathbb{E}(r_\varphi) = \sum_{e \in E} \mathbb{P}(\exists v \in e : \varphi(v) = \text{red}) = \frac{3}{4}m$. Since $\sum_{v \in V} \deg(v)^2 \leq \sum_{v \in V} \deg(v)\Delta = 2\Delta m$, for $\lambda = \sqrt{(\ln 2)\Delta m}$, we have $\mathbb{P}(r_\varphi > \frac{3}{4}m + \lambda) < \frac{1}{2}$. Thus with positive probability, both r_φ and b_φ are at most $\frac{3}{4}m + \lambda$. In particular, a coloring with $l_\varphi \leq \frac{3}{4}m + \lambda$ exists. The second statement follows in a similar way. $\square$

The algorithm behind Theorem 6 can be efficiently derandomized.

Theorem 7. *A coloring φ such that $l_\varphi \leq \frac{3}{4}m + \sqrt{(\ln 4)\Delta m}$ can be constructed in $O(n^3)$ time.*

For the proof we invoke an algorithmic version of Azuma's martingale inequality proved by Srivastav and Stangier [10]. Let $\Omega = \{0,1\}^n$ be a probability space with probability measure $\mathbb{P}$ and let $\varphi : \Omega \to \mathbb{R}$ be a quadratic form. Let $X = (X_1, \ldots, X_n)$ be a vector of independent random variables with $X_k \in \{0,1\}$, for all $k \in \{1, \ldots, n\}$. Further, let $\mathbb{P}(X_k = 1) = p$ and $\mathbb{P}(X_k = 0) = 1 - p$ for all k and $p \in (0,1)$. We wish to bound the large deviation probability $\mathbb{P}\left(|\varphi(X) - \mathbb{E}(\varphi(X))| \geq \lambda\right)$, for $\lambda > 0$. If f satisfies a Lipschitz condition: $|\varphi(X) - \varphi(X')| \leq c_k$ if $X, X' \in \Omega$ differ only in the k-th component, then we can use the bounded difference inequality (3).

[1] One advantage of this version is that it can be formulated without introducing the martingale machinery used in its proof.

Theorem 8 [10]. *Let $\delta \in (0,1)$ such that $1-\delta \geq 2\exp\left(-2\lambda^2/\sum_{i=1}^n c_i^2\right)$. Then a vector $X \in \Omega$ which satisfies $|\varphi(X) - \mathbb{E}(\varphi(X))| \leq \lambda$ can be constructed in $O\left(n^3 \log(\delta^{-1})\right)$ time.*

Proof of Theorem 7. First we write the objective function l_φ, the load, as the maximum of two quadratic forms describing r_φ and b_φ respectively. We model a two coloring of the vertex set V as a vector $X = (X_1, \ldots, X_n) \in \Omega = \{0,1\}^n$, where for $i \in \{1, \ldots, n\}$, $X_i = 1$ if the vertex i is colored red and $X_i = 0$ if it is colored blue. Let (a_{ij}) be the adjacency matrix of the graph $G = (V, E)$ under consideration. We may identify a two-coloring $\varphi : V \to \{\text{red,blue}\}$ by $X \in \{0,1\}^n$, so for $X \in \{0,1\}^n$ let

$$r(X) = \sum_{i=1}^n \sum_{j=1}^n \frac{a_{ij}\, X_i\, X_j}{2} + \sum_{i=1}^n \sum_{j=1}^n a_{ij}\, X_i\, (1 - X_j)\ ,$$

and

$$b(X) = \sum_{i=1}^n \sum_{j=1}^n \frac{a_{ij}\, (1-X_i)\, (1-X_j)}{2} + \sum_{i=1}^n \sum_{j=1}^n a_{ij}\, X_i\, (1 - X_j)\ .$$

Note that $\varphi(i) = X_i$ for all $i \in \{1, \ldots, n\}$. So, $r(X) = r_\varphi$, $b(X) = b_\varphi$ and $l_\varphi = l(X) := \max\{r(X), b(X)\}$. Theorem 8 can be extended to cover also the maximum of two quadratic forms, $r(X)$ and $b(X)$, with minor modifications in the proof (the important thing is to be able to compute conditional expectations of the form $\mathbb{E}(f \mid X_1 = a_1, \ldots, X_k = a_k)$). Thus, applying Theorem 8 to $l(X)$ with $c_k = \deg(v_k)$, $\lambda = \sqrt{(\ln 4)\Delta m}$ and $\delta = 0.5$, we can construct a two-coloring $X \in \{0,1\}^n$ in $O(n^3)$ time that satisfies $l(X) \leq \frac{3}{4}m + \sqrt{(\ln 4)\Delta m}$. □

Note that the dependence on Δ cannot be avoided. This is shown by star graphs. Moreover, if $\Delta = o(m)$, then the bound of $(\frac{3}{4} + o(1))m$ cannot be improved in general. The complete graph $K_n = (\{1, \ldots, n\}, \binom{\{1,\ldots,n\}}{2})$ satisfies $l_\varphi \geq \frac{3}{8}n^2 - \frac{1}{4}n = (\frac{3}{4} + o(1))m$ for all colorings φ.

5.2 Random Multi-graphs

In fact, in some sense almost all graphs have a load of $(\frac{3}{4} - o(1))m$. Without proof, we state the following.

Theorem 9. *Let $m \geq 12n$. For a random multi-graph $G = (V, E)$, $|V| = n$ obtained by choosing m edges from $\binom{V}{2}$ independently with repetition, we have $l(G) \geq \frac{3}{4}m - \sqrt{3mn}$ with probability $1 - 2^{-n}$.*

In other words, all but a fraction of less than 2^{-n} of the multi-graphs having n vertices and m edges have a load of at least $\frac{3}{4}m - \sqrt{3mn}$. If $n = o(m)$, this shows that almost all multi-graphs have a load of $(\frac{3}{4} - o(1))m$. The use of multi-graphs has mainly technical reasons. Unless m is close to $\binom{n}{2}$, most multi-graphs as above have only few multiple edges. Hence the random multi-graph model is close to the standard random graph model $G(n, p(n))$.

6 MLCP with More Than Two Colors

Most of our results have a natural extension to MLCP with more than two colors. For reasons of brevity and readability we omit the proofs, which are mostly similar (though more technical) to the ones for two colors.

- For any fixed number of colors, the MLCP is NP–complete.
- For any fixed number of colors, there is a polynomial time algorithm computing a minimal load coloring for trees.
- A tree G with m edges can be colored in k colors with load bounded by $\frac{m}{k} + O(\Delta(G) \log m)$.
- For all graphs $G = (V, E)$ there is a k-coloring with load at most $\frac{2k-1}{k^2} m + \sqrt{(\ln k)\Delta(G)m}$.
- For graphs on n vertices with genus $g > 0$ we can find a k-coloring with load bounded by $m/k + O(\sqrt{g\Delta n})$.

There are graphs having small load in some numbers of colors and large one in others. We give three examples.

(i) Let G be a graph consisting of two disjoint cliques on n vertices. Then the load in two colors is $\frac{1}{2}|E(G)|$, shown by coloring both cliques monochromatic in a different color. This is smallest possible for any graph. Let $\gamma = \sqrt{3} - 1$. In three colors, an optimal coloring will contain $(\gamma + o(1))n$ red vertices in the first clique, $(\gamma + o(1))n$ blue vertices in the second and $(1 - \gamma + o(1))n$ green vertices in each clique. This yields a load of $(2\sqrt{3} - 3 + o(1))n^2 \approx 0.4641|E(G)|$. Compared to the smallest possible value of $\frac{1}{3}|E(G)|$, this is quite large.

(ii) If G consists of three disjoint cliques of n vertices each, then the 3–color load is smallest possible with $\frac{1}{3}|E(G)|$, but the 2–color load is approximately $\frac{7}{12}|E(G)|$.

(iii) The same behavior is also displayed by trees. A complete 3–ary tree T has a 3–color load of $\frac{1}{3}|E(T)| + 2$. However, it can be proven to have a 2–color load of $\frac{1}{2}|E(G)| + \Omega(\log n)$, which is (up to the implicit constant) maximum possible for trees as shown in Theorem 2.

References

1. A.A. Ageev, A.V. Fishkin, A.V. Kononov and S.V. Sevastianov, *Open Block Scheduling in Optical Communication Networks.* Springer LNCS 2909 (2004), 13 - 26.
2. K. Azuma, *Weighted sums of certain dependent variables.* Tohoku Math. Journal 3(1967), 357 - 367.
3. P. Berman and M. Karpinski, *Approximation Hardness of Bounded Degree MIN-CSP and MIN-BISECTION*, Electronic Colloquium on Computational Complexity, Report No. 26 (2001).
4. K. Diks, H.N. Djidjev, O. Sýkora and I. Vrťo, *Edge Separators of Planar and Outerplanar Graphs with Applications*, Journal of Algorithms 14(1993), 258 - 279.

5. H.N. Djidjev, *A separator theorem.* Comptes Rendus de l'Academie Bulgare des Sciences 34(1981), 643 - 645.
6. G. Even, J. Naor, S. Rao and B. Schieber, *Fast Approximate Graph Partitioning Algorithms.* SIAM J. Comput. 28(6)(1999), 2187 - 2214.
7. M.R. Garey, D.S. Johnson and L. Stockmeyer, *Some simplified NP-complete graph problems*, Theoret. Comput. Sci. 1(33)(1976), 237 - 267.
8. R. J. Lipton and R. E. Tarjan, *A separator theorem for planar graphs.* SIAM Journal on Applied Mathematics 36(1979), 177 - 189.
9. C. McDiarmid, *Concentration.* In Probabilistic Methods for Algorithmic Discrete Mathematics, Volume 16 of Algorithms Combin.(1998), Springer, Berlin, 195 - 248.
10. A. Srivastav, *Derandomizing Martingale Inequalities.* Preprint (2005). A preliminary version appeared as A. Srivastav and P. Stangier, *On quadratic lattice approximations.* In Proc. of the 4th Internat. Symp. on Algorithms and Computation(1993), LNCS 762, Springer, 176 - 184.
11. O. Sýkora and I. Vrťo, *Edge Separators for Graphs of Bounded Genus with Applications.* In Proc. of the 17th International Workshop on Graph Theoretic Concepts in Computer Science (1992), 159 - 168.
12. C. Thomassen, *The graph genus problem is NP-complete.* Journal of Algorithms 10(4)(1989), 568 - 576.
13. D.B. West, Introduction to Graph Theory. Prentice Hall (1996).

Improved Approximation Algorithms for MAX NAE-SAT and MAX SAT

Adi Avidor*, Ido Berkovitch, and Uri Zwick**

School of Computer Science,
Tel-Aviv University, Tel-Aviv 69978, Israel
{adi, edoberko, zwick}@tau.ac.il

Abstract. MAX SAT and MAX NAE-SAT are central problems in theoretical computer science. We present an approximation algorithm for MAX NAE-SAT with a conjectured performance guarantee of 0.8279. This improves a previously conjectured performance guarantee of 0.7977 of Zwick [Zwi99]. Using a variant of our MAX NAE-SAT approximation algorithm, combined with other techniques used in [Asa03], we obtain an approximation algorithm for MAX SAT with a conjectured performance guarantee of 0.8434. This improves on an approximation algorithm of Asano [Asa03] with a conjectured performance guarantee of 0.8353. We also obtain a 0.7968-approximation algorithm for MAX SAT which is not based on any conjecture, improving a 0.7877-approximation algorithm of Asano [Asa03].

1 Introduction

An instance of MAX NAE-SAT (Maximum Not-All-Equal SAT) in the Boolean variables $x_1, \ldots, x_n$ is composed of a collection of *clauses* $C_1, \ldots, C_m$ with non-negative weights $w_1, \ldots, w_m$ associated with them. Each clause C_j is of the form *NAE*$(b_1, \ldots, b_{k_j})$. Each of the b_i's is a literal, i.e., a variable x_l or its negation $\bar{x}_l$ and $k_j \geq 2$. The clauses may be arbitrarily large and may not all be of the same size. A clause *NAE*$(b_1, \ldots, b_{k_j})$ is satisfied if at least one of the literal gets the value 1 and at least one of the literal gets the value 0. The goal is to assign the Boolean variables $x_1, \ldots, x_n$ values of 0 and 1 so that the total weight of the satisfied clauses is maximized. We let MAX NAE-$\{k\}$-SAT be the restriction of MAX NAE-SAT to instances in which all clauses are of size exactly k, and MAX NAE-k-SAT the restriction of MAX NAE-SAT to instances in which all clauses are of size at most k.

An instance of MAX SAT in the Boolean variables $x_1, \ldots, x_n$ is composed of a collection of *clauses* $C_1, \ldots, C_m$ with non-negative weights $w_1, \ldots, w_m$ associated with them. Each clause C_j is of the form $b_1 \vee \ldots \vee b_{k_j}$ where the b_i's are literals and $k_j \geq 1$. A clause $b_1 \vee \ldots \vee b_{k_j}$ is satisfied if at least one of the literal gets the value 1. The goal is again to assign the Boolean variables $x_1, \ldots, x_n$ values of 0 and 1 so that the total weight of the satisfied clauses is maximized. We let MAX $\{k\}$-SAT be the restriction of MAX SAT to instances in which all clauses are of size exactly k, and

* Research was supported by the Deutsch Fund.
** Research was supproted by the ISRAEL SCIENCE FOUNDATION (grant no. 246/01).

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 27–40, 2006.

MAX k-SAT the restriction of MAX SAT to instances in which all clauses are of size at most k.

MAX NAE-SAT is a generalization of both MAX SAT and MAX CUT. MAX CUT is the restriction of MAX NAE-$\{2\}$-SAT to instances without negations. An instance of MAX SAT can be converted to an instance of MAX NAE-SAT by replacing each clause $b_1 \vee \ldots \vee b_{k_j}$ by the clause $\mathit{NAE}(O, b_1, \ldots, b_{k_j})$, where O is a new variable that appears in all clauses. A solution $\alpha_1, \ldots, \alpha_n, \beta$ to the resulting MAX NAE-SAT instance, where $\alpha_1, \ldots, \alpha_n$ are the values assigned to $x_1, \ldots, x_n$ and β is the value assigned to O, can be converted to a solution $\alpha_1 \oplus \beta, \ldots, \alpha_n \oplus \beta$ of the original MAX SAT instance with the same cost. MAX NAE-SAT (MAX $\{k\}$-NAE-SAT) is also a generalization of MAX SET-SPLITTING (MAX k-SET-SPLITTING), which are the problems of 2-coloring the vertices of a hypergraph (k-uniform hypergraph) so as to maximize the number of non-monochromatic edges. More specifically, MAX SET-SPLITTING and MAX k-SET-SPLITTING are the restrictions of MAX NAE-SAT and MAX $\{k\}$-NAE-SAT to instances without negations.

Håstad [Hås01] showed that for every $k \geq 3$ and every $\varepsilon > 0$, if there is a $(1 - 2^{-k} + \varepsilon)$-approximation algorithm for MAX $\{k\}$-SAT, then $P = NP$. Hence, as both MAX SAT and MAX NAE-SAT generalize MAX $\{3\}$-SAT, we get that for every $\varepsilon > 0$, there is no $(7/8 + \varepsilon)$-approximation algorithm for these problems, unless $P = NP$.

The first approximation algorithm for MAX SAT was presented by Johnson [Joh74], who showed that the greedy algorithm achieves a performance guarantee of 1/2. Twenty years later, Yannakakis [Yan94] and then Goemans and Williamson [GW94] proposed two different 3/4-approximation algorithms.

In a seminal paper, Goemans and Williamson [GW95] used semidefinite programming to obtain 0.878-approximation algorithms for MAX CUT and MAX 2-SAT. Feige and Goemans [FG95] obtained an improved approximation algorithm for MAX 2-SAT with performance guarantee 0.931. An improved approximation algorithm for the general MAX SAT can be obtained by combining one of these MAX 2-SAT approximation algorithms and the previous 3/4-approximation algorithms. Such improvements include a 0.7584-approximation algorithm by Goemans and Williamson [GW95], a 0.765-approximation algorithm by Asano, Ono and Hirata [AOH96] and a 0.770-approximation algorithm by Asano [Asa97].

The approximation ratio of MAX 2-SAT was further improved to 0.935 by Matuura and Matsui [MM01a, MM01b], and then by Lewin, Livnat and Zwick [LLZ02] to 0.9401. An optimal, semidefinite programming based, 7/8-approximation algorithm for MAX 3-SAT was given by Karloff and Zwick [KZ97] (a rigorous proof of the conjectured approximation ratio of [KZ97] is given in [Zwi02].) A close to optimal 0.8721-approximation algorithm for MAX 4-SAT was given by Halperin and Zwick [HZ01].

A new rounding technique, "outward rotations", for rounding semidefinite programming solutions was introduced independently by Nesterov [Nes98], Ye [Ye01] and Zwick [Zwi99]. Using outward rotations, Han, Ye and Zhang [HYZ04], strengthening an earlier MAX NAE-SAT 0.7240-approximation algorithm of Andersson and Engerbretsen [AE98], obtained a 0.7499-approximation algorithm for MAX NAE-SAT. Zwick [Zwi99] obtained a 0.9087-approximation algorithm for MAX NAE-$\{3\}$-SAT.

Zwick also obtained an approximation algorithm for the MAX NAE-SAT and the MAX SAT problems with a conjectured approximation ratio of 0.7977.

A 0.7846-approximation algorithm for MAX SAT was given by Asano and Williamson [AW02]. This algorithm is based on linear programming with special rounding functions combined with several other MAX k-SAT algorithms. Asano and Williamson also gave a 0.8331-approximation algorithm for MAX SAT based on the previously conjectured 0.7977-approximation algorithm for MAX NAE-SAT. Finally, Asano [Asa03], using the same techniques and different rounding functions, gave a 0.7877-approximation algorithm for MAX SAT and an additional approximation algorithm with a conjectured performance guarantee of 0.8353.

The outward rotations technique was generalized by Feige and Langberg [FL01] to a new rounding technique named RPR^2 - Random Projection followed by Randomized Rounding. Feige and Langberg used RPR^2 to obtain an improved approximation algorithm for the "Light MAX CUT" problem. ("Light MAX CUT" is the MAX CUT problem restricted to instances of a small maximal cut.) Charikar and Wirth [CW04] extended Feige and Langberg "Light MAX CUT" results and demonstrated the applicability of the RPR^2 technique for maximizing quadratic forms and maximum correlation clustering. Lately, the results of Charikar and Wirth were extended by Alon *et al.* [AMMN05].

In this paper we use the RPR^2 technique to obtain new approximation algorithms for MAX NAE-SAT and MAX SAT. We give an approximation algorithm for MAX NAE-SAT with a conjectured performance guarantee of 0.8279. We also adjust Asano's [Asa03] MAX SAT approximation algorithm and obtain an approximation algorithm for MAX SAT with conjectured performance guarantee of 0.8434. In addition, we give a slightly improved 0.7968-approximation algorithm for MAX SAT which does not rely on any conjecture.

2 MAX NAE-SAT Approximation Algorithm

Our MAX NAE-SAT approximation algorithm starts by solving a semidefinite programming relaxation of the problem, which produces a sequence $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ of unit vectors in $\mathbb{R}^n$. The algorithm then uses the RPR^2 rounding technique to round these vectors to Boolean values.

2.1 A Semidefinite Programming Relaxation for MAX NAE-SAT

We let $x_{n+i} = \bar{x}_i$, for $1 \leq i \leq n$. The j-th clause of a MAX NAE-SAT instance is therefore of the form *NAE*$(x_{i_1}, \ldots, x_{i_{k_j}})$, where $1 \leq i_1, \ldots, i_{k_j} \leq 2n$ and $1 \leq j \leq m$. We denote the unit sphere in $\mathbb{R}^n$ by S^{n-1}, and the set of all permutations on $\{1, \ldots, k\}$ by $\mathcal{S}_k$. The semidefinite programming relaxation of MAX NAE-SAT is given in Figure 1. In this relaxation, a unit vector $\boldsymbol{v}_i \in S^{n-1}$ is assigned to each literal x_i, where $1 \leq i \leq 2n$. In addition, a scalar z_j is assigned to each clause, where $1 \leq j \leq m$. To ensure that $x_{n+i} = \bar{x}_i$, we require $\boldsymbol{v}_i \cdot \boldsymbol{v}_{n+i} = -1$, for $1 \leq i \leq n$. To check that this is indeed a relaxation of the MAX NAE-SAT instance, note that for every Boolean assignment $\alpha_1, \ldots, \alpha_n \in \{0, 1\}$ to the variables $x_1, \ldots, x_n$, the vectors

Max	$\sum_{j=1}^{m} w_j z_j$	
s.t.	$z_j \le \frac{k_j - \sum_{l=1}^{k_j} \boldsymbol{v}_{i_{\pi(l)}} \cdot \boldsymbol{v}_{i_{\pi(l+1)}}}{4}$	$C_j = \mathit{NAE}(x_{i_1}, \ldots, x_{i_{k_j}})$ $\pi \in \mathcal{S}_{k_j}, \quad k_j \le k_{max}$ $1 \le j \le m$
	$z_j \le 1$	$1 \le j \le m$
	$\boldsymbol{v}_i \cdot \boldsymbol{v}_{n+i} = -1$	$1 \le i \le n$
	$\boldsymbol{v}_{i_1} \cdot \boldsymbol{v}_{i_2} + \boldsymbol{v}_{i_1} \cdot \boldsymbol{v}_{i_3} + \boldsymbol{v}_{i_2} \cdot \boldsymbol{v}_{i_3} \ge -1$	$1 \le i_1, i_2, i_3 \le 2n$
	$\boldsymbol{v}_i \in S^{n-1}$	$1 \le i \le 2n$

Fig. 1. A semidefinite programming relaxation of MAX NAE-SAT

$\boldsymbol{v}_i = (2\alpha_i - 1, 0, \ldots, 0) \in S^{n-1}$, for $1 \le i \le n$, and $\boldsymbol{v}_i = -\boldsymbol{v}_{i-n}$, for $n+1 \le i \le 2n$, satisfy all the required constraints. Also, it is easy to check that

$$\mathit{NAE}(x_{i_1}, \ldots, x_{i_{k_j}}) = \min \left\{ 1, \min_{\pi \in \mathcal{S}_{k_j}} \frac{k_j - \sum_{l=1}^{k_j} \boldsymbol{v}_{i_{\pi(l)}} \cdot \boldsymbol{v}_{i_{\pi(l+1)}}}{4} \right\},$$

where $\mathit{NAE}(x_{i_1}, \ldots, x_{i_{k_j}})$ is defined here to be 1 if the clause is satisfied and 0 otherwise. (In the above expression, we interpret $\pi(k_j+1)$ to be $\pi(1)$.) These integral assignments also satisfy the so called "triangle constraints" $\boldsymbol{v}_{i_1} \cdot \boldsymbol{v}_{i_2} + \boldsymbol{v}_{i_1} \cdot \boldsymbol{v}_{i_3} + \boldsymbol{v}_{i_2} \cdot \boldsymbol{v}_{i_3} \ge -1$, for $1 \le i_1, i_2, i_3 \le 2n$. We write the "*NAE*" constraints only for clauses of size smaller than the parameter k_{max}, which will be chosen later. To ensure a polynomial number of constraints k_{max} must be chosen to be $O(\frac{\log n}{\log \log n})$. However, we will only need k_{max} to be some constant. Note that, if C_j is clause of size bigger than k_{max}, then in an optimal solution of the relaxation $z_j = 1$. This semidefinite program can be solved, to any desired precision, in polynomial time.

2.2 RPR^2 - Random Projection Followed by Randomized Rounding

RPR^2 parameterized by a function $f : \mathbb{R} \to [0, 1]$ is defined as follows:

1. Let $\boldsymbol{r}$ be a vector distributed according to the n-dimensional standard normal distribution $N(\boldsymbol{0}, I_n)$.
2. For $1 \le i \le n$, set the variable x_i to 1 independently with probability $f(\boldsymbol{v}_i \cdot \boldsymbol{r})$.

We typically use RPR^2 functions that satisfy $f(-x) = 1 - f(x)$, for any $x \in \mathbb{R}$. RPR^2 is a generalization of the outward rotations rounding technique ([Nes98, Ye01, Zwi99]) that was used to obtain previous MAX NAE-SAT approximation algorithms. More precisely, let $\phi(x) = (2\pi)^{-1/2} e^{-x^2/2}$ and $\Phi(x) = \int_{-\infty}^{x} \phi(t) dt$ be the probability density function and the cumulative distribution function of a standard normal random variable, respectively. Feige and Langberg [FL01] show that outward rotations with a rotation angle γ is equivalent to RPR^2 parameterized by the function $f_\gamma(x) = \Phi(x \cot \gamma)$.

2.3 The Algorithm

Our algorithm is parameterized by an RPR^2 function f and a perturbation probability $p \in [0, \frac{1}{2}]$:

1. Solve the MAX NAE-SAT semidefinite programming relaxation of Figure 1.
2. Round $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ using RPR^2 parameterized by f.
3. For $1 \le i \le n$, set the variable x_i to $\bar{x}_i$, independently, with probability p.

The perturbation step is introduced in order to handle clauses of size larger than k_{max}. We choose $p = \frac{2}{k_{max}}$.

2.4 Analysis

In this section we shortly describe the way used to obtain a lower bound on the performance ratio of our MAX NAE-SAT approximation algorithm.

For any dimension d and any $\boldsymbol{r} \in \mathbb{R}^d$, let $\phi(\boldsymbol{r}) = (2\pi)^{-d/2} \mathrm{e}^{-\boldsymbol{r}^T \cdot \boldsymbol{r}/2}$ be the probability density function of a d-dimensional standard normal random variable, and let $\phi_\Sigma(\boldsymbol{r}) = ((2\pi)^d \det(\Sigma))^{-1/2} \mathrm{e}^{-\boldsymbol{r}^T \Sigma^{-1} \boldsymbol{r}/2}$ be the probability density function of a d-dimensional normal random variable with expectation $\mathbf{0}$ and covariance matrix Σ. Let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in S^{n-1}$ be vectors corresponding to a clause *NAE*$(x_1, \ldots, x_k)$. By the definition of the RPR^2 procedure, the probability (over the choices of $\boldsymbol{r}$) that the clause *NAE*$(x_1, \ldots, x_k)$ is satisfied is

$$\begin{aligned} prob_f(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k) \stackrel{def}{=} 1 &- \int_{\mathbb{R}^n} f(\boldsymbol{v}_1 \cdot \boldsymbol{r}) \cdot \ldots \cdot f(\boldsymbol{v}_k \cdot \boldsymbol{r}) \phi(\boldsymbol{r}) d\boldsymbol{r} \\ &- \int_{\mathbb{R}^n} (1 - f(\boldsymbol{v}_1 \cdot \boldsymbol{r})) \cdot \ldots \cdot (1 - f(\boldsymbol{v}_k \cdot \boldsymbol{r})) \phi(\boldsymbol{r}) d\boldsymbol{r}. \end{aligned}$$

In addition, if the RPR^2 function f satisfies $f(-x) = 1 - f(x)$, then

$$prob_f(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k) = 1 - 2 \int_{\mathbb{R}^n} f(\boldsymbol{v}_1 \cdot \boldsymbol{r}) \cdot \ldots \cdot f(\boldsymbol{v}_k \cdot \boldsymbol{r}) \phi(\boldsymbol{r}) d\boldsymbol{r}.$$

Let V be the k by n matrix whose rows are the vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$. W.l.o.g., we may assume that $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ are linearly independent (otherwise we can take a maximal linearly independent subset of them.) By substituting $\boldsymbol{y} = V\boldsymbol{r}$ we get,

$$prob_f(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k) = 1 - 2 \int_{\mathbb{R}^k} f(y_1) \cdot \ldots \cdot f(y_k) \phi_{V^T V}(\boldsymbol{y}) d\boldsymbol{y}.$$

In particular, the probability $prob_f(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)$ depends only on the inner products $\boldsymbol{v}_i \cdot \boldsymbol{v}_j$, for $1 \le i < j \le k$. As the vectors are unit vectors, the probability depends only on the angles $\theta_{ij} = \arccos(\boldsymbol{v}_i \cdot \boldsymbol{v}_j)$, for $1 \le i < j \le k$. There seems to be no closed form formula for the latter integeral for most choices of f, even for $k = 2$. We therefore use numerical methods to compute $prob_f(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)$.

We let

$$value(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k) = \min\left\{1, \min_{\pi \in \mathcal{S}_k} \frac{k - \sum_{i=1}^{k} \boldsymbol{v}_{\pi(i)} \cdot \boldsymbol{v}_{\pi(i+1)}}{4}\right\}$$

be the contribution of the clause to the value of the MAX NAE-SAT semidefinite programming relaxation. In addition, we let

$$\hat{\alpha}_k(f) = \inf \frac{prob_f(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)}{value(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)}$$

where the infimum is taken over all k-tuples of vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in S^{k-1}$ that satisfy the "triangle inequalities" and for which $value(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k) > 0$. If the latter infimum is attained at $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$, we call the $\binom{k}{2}$-tuple of angles $(\theta_{12}, \ldots, \theta_{k-1,k})$ a *worst k-configuration* with respect to the RPR^2 function f.

In these notations, the probability that the clause *NAE*$(x_1, \ldots, x_k)$ is satisfied when using RPR^2 parameterized by f, followed by perturbation with probability p, is at least

$$\begin{aligned} prob_f(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)&(1 - p(1-p)^{k-1} - (1-p)p^{k-1}) \\ &+ (1 - prob_f(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k))(1 - p^k - (1-p)^k). \end{aligned}$$

Let $\varepsilon > 0$ be a small constant. We choose $k_{max} = \lceil 8/\varepsilon \rceil$. Then, it is not hard to verify that the latter expression is bounded below by $prob_f(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)(1-\varepsilon)$ for $k \le k_{max}$ and by $(1 - \mathrm{e}^{-2} - \varepsilon)$ for $k > k_{max}$. In this scenario we may define

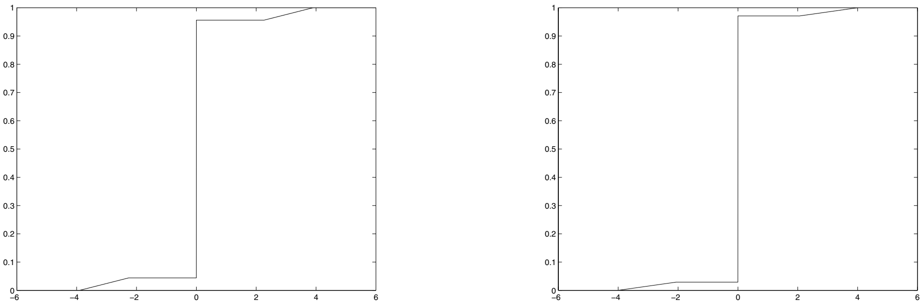
$$\alpha_k(f) = \begin{cases} \hat{\alpha}_k(f) - \varepsilon & \text{if } k \le k_{max} \\ 1 - \mathrm{e}^{-2} - \varepsilon & \text{if } k > k_{max} \end{cases}$$

As $value(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)$ is 1 for clauses of size bigger than k_{max}, the probability that the clause *NAE*$(x_1, \ldots, x_k)$ is satisfied is therefore at least $\alpha_k(f) value(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)$. Finally, we let $\alpha(f) = \min_{k \ge 2} \alpha_k(f)$.

Let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n \in S^{n-1}$ be an *optimal* solution of the semidefinite programming relaxation of a MAX NAE-SAT instance. Our algorithm produces an assignment with an expected cost of:

$$\begin{aligned} \sum_{j=1}^{m} w_j Pr[\text{clause } C_j \text{ is satisfied}] &\ge \sum_{j=1}^{m} w_j \alpha_{k_j}(f) \cdot value(\boldsymbol{v}_{i_1}, \ldots, \boldsymbol{v}_{i_{k_j}}) \\ &\ge \alpha(f) \sum_{j=1}^{m} w_j \cdot value(\boldsymbol{v}_{i_1}, \ldots, \boldsymbol{v}_{i_{k_j}}) \\ &\ge \alpha(f) OPT \end{aligned}$$

The last inequality holds as the value of the MAX NAE-SAT semidefinite programming relaxation is an upper bound on the value of the optimal assignment OPT. Therefore, $\alpha(f)$ is a lower bound on the performance ratio of our approximation algorithm.



(a) (b)

Fig. 2. (a) The RPR^2 function f_{NAE} used in the MAX NAE-SAT approximation algorithm. (b) The RPR^2 function f_{SAT} used in the MAX-SAT approximation algorithm.

2.5 The RPR^2 Function

In this subsection we describe the RPR^2 function f used to obtain an improved approximation ratio. We note that for various choices of f, the minimum $\min_{k\geq 2}\alpha_k(f)$ is attained at two values of k (which are less than the k_0 parameter of the previous subsection). We call a worst k-configuration of such a value of k a *worst configuration*.

Our search for a good RPR^2 function was inspired by the previously MAX NAE-SAT approximation algorithms of [Zwi99]. More specifically, the previous MAX NAE-SAT approximation algorithm used outward rotations with rotation angle $\gamma = 0.4555$. Equivalently, the algorithm rounded the semidefinite programming solution using RPR^2 with the function $f_\gamma(x) = \Phi(x\cot(0.4555))$. Extensive numerical experiments led to the conjecture that for any rotation angle γ and for any $k \geq 4$, the worst k-configurations with respect to $f_\gamma(x) = \Phi(x\cot\gamma)$ are the $\binom{k}{2}$-tuples $(\arccos(1-\frac{4}{k}),\ldots,\arccos(1-\frac{4}{k}))$.

In our algorithm we use the piecewise linear RPR^2 function $f_{\text{NAE}} : \mathbb{R} \to [0,1]$ connecting between the points $(-\infty,0)$, $(-3.9,0)$, $(-2.262,0.044)$, $(0,0.044)$, $(0,0.956)$, $(2.262,0.956)$, $(3.9,1)$ and $(\infty,1)$. The function f_{NAE} is shown in Figure 2(a). Numerical experiments with the function f_{NAE} lead us to believe that the worst k-configurations for this function, for any $k \geq 4$, are again configurations in which $\boldsymbol{v}_i\cdot\boldsymbol{v}_j = 1-\frac{4}{k}$, for every $1\leq i<j\leq k$. We thus conjecture:

Conjecture 1. For any $k \geq 4$ the infimum in $\hat{\alpha}_k(f_{\text{NAE}})$ is attained when for every $1 \leq i < j \leq k$, $\boldsymbol{v}_i\cdot\boldsymbol{v}_j = 1-\frac{4}{k}$.

The conjecture implies that $\hat{\alpha}_k(f_{\text{NAE}}) > 0.8279$ for all $k \geq 2$. We can choose the parameter ε of subsection 2.4, to be small enough to have $\alpha(f_{\text{NAE}}) > 0.8279$. Our algorithm achieves its worst case ratio on instances in which all clauses are of size 2 or 12. More specifically, for a worst instance the solution of the semidefinite programming $\boldsymbol{v}_1,\ldots,\boldsymbol{v}_n$ satisfies that $\boldsymbol{v}_{i_1}\cdot\boldsymbol{v}_{i_2} \simeq -0.7638$ for every clause *NAE*(x_{i_1},x_{i_2}) and $\boldsymbol{v}_{i_{l_1}}\cdot\boldsymbol{v}_{i_{l_2}} = 1-\frac{4}{12}$ $(1\leq l_1<l_2\leq 12)$ for every clause *NAE*$(x_{i_1},\ldots,x_{i_{12}})$.

In our search for an optimal RPR^2 function we considered various piecewise linear symmetric monotone RPR^2 functions with up to eight turnings. Note that the function f_{NAE} used has only six turnings. It should be mentioned that the symmetric RPR^2 functions with two turnings (and which are usually referred to as s-linear RPR^2 functions [FL01]), achieve approximation ratios worst than outward rotations. Minor improvements can be achieved by combining s-linear RPR^2 functions with outwards rotations. We believe our choice of RPR^2 function is not far from being optimal.

3 MAX SAT Approximation Algorithms

As MAX NAE-SAT generalizes MAX SAT, our results so far immediately imply a MAX SAT approximation algorithm with a conjectured approximation ratio of 0.8279. In this section we present our approximation algorithms for MAX SAT. We first describe the methods used to obtain previous MAX SAT approximations algorithms.

3.1 Asano's MAX SAT Approximations Algorithms

As before let $x_{n+i} = \bar{x}_i$, for $1 \le i \le n$. Goemans and Williamson [GW94] formulated MAX SAT as the following integer programming (IP) problem:

$$\begin{array}{lll} \max & \sum_{j=1}^{m} w_j z_j & \\ s.t. & z_j \le \sum_{l=1}^{k_j} y_{i_l} & \begin{array}{c} C_j = x_{i_1} \vee x_{i_2} \vee \ldots \vee x_{i_{k_j}}, \\ 1 \le j \le m \end{array} \\ & y_i + y_{n+i} = 1 & 1 \le i \le n \\ & y_i \in \{0,1\} & 1 \le i \le 2n \\ & z_j \in \{0,1\} & 1 \le j \le m \end{array}$$

If the last two integrality constraints are relaxed, and the variables y_i and z_j are allowed to take on any values between 0 and 1, then an LP relaxation of MAX SAT is obtained. Let (y^*, z^*) be an optimal solution of the LP relaxation of MAX SAT. Goemans and Williamson [GW94] used the following rounding: Let $g : [0,1] \to [0,1]$ be a rounding function. For $1 \le i \le n$, set the variable x_i to be 1 independently with probability $g(y^*)$.

Asano [Asa03], following [AW02], suggested two families of rounding functions:

$$f_3^a(y) = \begin{cases} 1 - \frac{a}{(4a^2)^y} & \text{if } y \in [0, \frac{1}{2}] \\ \frac{(4a^2)^y}{4a} & \text{if } y \in [\frac{1}{2}, 1] \end{cases} \quad \text{and} \quad f_4^a(y) = \begin{cases} ay + 1 - a & \text{if } \ y \in [0, 1 - y_a] \\ \frac{ay}{2} + \frac{1}{2} - \frac{a}{4} & \text{if } y \in [1 - y_a, y_a] \\ ay & \text{if } \ y \in [y_a, 1] \end{cases}$$

where $y_a = \frac{1}{a} - \frac{1}{2}$.

Asano showed that using the rounding function $f_3^a(y)$ for $1/2 \le a \le \sqrt{\mathrm{e}}/2 = 0.824360\ldots$, the approximation ratio obtained for clauses of size k is at least:

$$\zeta_k^a = \begin{cases} a & \text{if } k = 1 \\ 1 - \frac{1}{4} a^{k-2} & \text{if } k \ge 2 \end{cases}.$$

$$\begin{array}{lll}
\text{Max} & \displaystyle\sum_{j=1}^{m} w_j z_j & \\
\text{s.t.} & \left.\begin{array}{ll}
z_j \le \displaystyle\sum_{l=1}^{k_j} y_{i_l} & \\
z_j \le \frac{1}{k_j-1} \displaystyle\sum_{1\le p<q\le k_j} \frac{3-\boldsymbol{v}_0\cdot\boldsymbol{v}_{i_p}-\boldsymbol{v}_0\cdot\boldsymbol{v}_{i_q}-\boldsymbol{v}_{i_p}\cdot\boldsymbol{v}_{i_q}}{4} & k_j \ge 2 \\
z_j \le \frac{1}{\binom{k_j-1}{2}} \displaystyle\sum_{1\le l_1<l_2<l_3\le k_j} u_{i_{l_1} i_{l_2} i_{l_3}} & k_j \ge 3 \\
z_j \le \frac{k_j+1-\sum_{l=0}^{k_j} \boldsymbol{v}_{i_{\pi(l)}}\cdot\boldsymbol{v}_{i_{\pi(l+1)}}}{4} & \pi \in \hat{\mathcal{S}}_{k_j} \\
 & k_j \le k_{max}
\end{array}\right\} & \begin{array}{c} 1 \le j \le m \\ C_j = x_{i_1} \vee \ldots \vee x_{i_{k_j}} \end{array} \\
 & y_i = \frac{1-\boldsymbol{v}_0\cdot\boldsymbol{v}_i}{2} & 1 \le i \le 2n \\
 & u_{i_1 i_2 i_3} \le \frac{4-\sum_{l=0}^{3} \boldsymbol{v}_{i_{\pi(l)}}\cdot\boldsymbol{v}_{i_{\pi(l+1)}}}{4} & \pi \in \hat{\mathcal{S}}_3 \\
 & u_{i_1 i_2 i_3} \le 1 & 1 \le i_1 < i_2 < i_3 \le k_j \\
 & z_j \le 1 & 1 \le j \le m \\
 & \boldsymbol{v}_i \cdot \boldsymbol{v}_{n+i} = -1 & 1 \le i \le n \\
 & \boldsymbol{v}_{i_1}\cdot\boldsymbol{v}_{i_2} + \boldsymbol{v}_{i_1}\cdot\boldsymbol{v}_{i_3} + \boldsymbol{v}_{i_2}\cdot\boldsymbol{v}_{i_3} \ge -1 & 0 \le i_1, i_2, i_3 \le 2n \\
 & \boldsymbol{v}_i \in S^n & 0 \le i \le 2n
\end{array}$$

Fig. 3. A semidefinite programming relaxation of MAX SAT

In addition, he showed that if the rounding function $f_4^a(y)$ for $\sqrt{\mathrm{e}}/2 \le a \le 1$ is used, then the approximation ratio obtained for clauses of size k is at least:

$$\eta_k^a = 1 - \max\left\{ a^k\left(1-\tfrac{1}{k}\right)^k,\ \ \tfrac{a^{k-2}}{4},\ \ \tfrac{a^k}{2}\left(1-\tfrac{1-y_a}{k-1}\right)^{k-1},\ \ \tfrac{1}{2^k}\left(1+\tfrac{a}{2}-\tfrac{a}{k}\right)^k \right\},$$

for $k \ge 2$ and $\eta_k^a = a$ for $k = 1$.

To obtain an improved approximation algorithm for MAX SAT Asano used a hybrid approach that was also used by Asano and Williamson [AW02]. In this approach several algorithms are run in parallel to obtained a solution and the solution with the maximal value is returned.

More specifically, the algorithm first solves a semidefinite programming relaxation for MAX SAT which incorporates all relaxations (LP and SDPs) used in pervious algorithms (see Figure 3 discussed in the next subsection). If the solution is rounded using the rounding procedure of Goemans and Williamson [GW94] with the rounding function f_3^a, the MAX 2-SAT rounding procedure of Feige and Goemans [FG95] and the rounding procedure of Halperin and Zwick [HZ01] for MAX 3-SAT, a performance guarantee of 0.7877 is obtained. If the solution is rounded using the rounding procedure of Goemans and Williamson [GW94] with the rounding function f_4^a and the MAX NAE-SAT rounding procedure of [Zwi99], a conjectured performance guarantee of 0.8353 is obtained.

In the next subsections we use the hybrid approach with improved algorithms to obtain an improved approximation algorithms for MAX SAT.

APPROXMAX-SAT(g, S, p)

1. Solve the MAX SAT semidefinite programming relaxation of Figure 3. (W.l.o.g $\boldsymbol{v}_0 = (1, 0, \ldots, 0) \in \mathbb{R}^{n+1}$.)
2. Return the maximal solution between
   (a) For $1 \le i \le n$, set $x_i = 1$ independently with probability $g(y_i)$
   (b) i. Let $\boldsymbol{r} = (0, r_1, \ldots, r_n)$, where $r_1, \ldots, r_n$ are independent standard normal variables. For $1 \le i \le n$, set $x_i = 1$ if $\boldsymbol{v}_i \cdot \boldsymbol{r} \le S(\boldsymbol{v}_0 \cdot \boldsymbol{v}_i)$
   ii. For $1 \le i \le n$, set $x_i = \bar{x}_i$ independently with probability p

Fig. 4. Algorithm APPROXMAX-SAT

3.2 A Semidefinite Programming Relaxation for MAX SAT

The semidefinite programming relaxation of MAX SAT is shown in Figure 3. As in the semidefinite programming relaxation of MAX NAE-SAT, each Boolean variable x_i corresponds to a unit vector $\boldsymbol{v}_i$. Here, the additional vector $\boldsymbol{v}_0$ corresponds to the value FALSE and the vector $-\boldsymbol{v}_0$ corresponds to the value TRUE. We use $\hat{\mathcal{S}}_k$ to denote the set of all permutation of $\{0, 1, \ldots, k\}$, i_0 to denote the index 0 and $\pi(k+1)$ to denote $\pi(0)$. As before to ensure a program of polynomial size we should take $k_{max} = O(\frac{\log n}{\log \log n})$, but eventually we take k_{max} to be some constant.

In an integral solution all the vectors correspond to the value FALSE are set to $\boldsymbol{v}_0$ and all the vectors correspond to the value TRUE are set to $-\boldsymbol{v}_0$. Hence, in an integral solution, the expression $\frac{1}{2}(1 - \boldsymbol{v}_0 \cdot \boldsymbol{v}_i)$ is 1 if and only if $\boldsymbol{v}_i = -\boldsymbol{v}_0$ and 0 if and only if $\boldsymbol{v}_i = \boldsymbol{v}_0$. Similarly, the expression $\frac{1}{4}(3 - \boldsymbol{v}_0 \cdot \boldsymbol{v}_{i_p} - \boldsymbol{v}_0 \cdot \boldsymbol{v}_{i_q} - \boldsymbol{v}_{i_p} \cdot \boldsymbol{v}_{i_q})$ is 1 only if and only if at least one of the vectors $\boldsymbol{v}_{i_p}, \boldsymbol{v}_{i_q}$ is $-\boldsymbol{v}_0$. It can be easily verified that $u_{i_1 i_2 i_3}$ is 1 if and only if at least one of the vectors $\boldsymbol{v}_{i_1}, \boldsymbol{v}_{i_2}, \boldsymbol{v}_{i_3}$ is $-\boldsymbol{v}_0$.

3.3 A Hybrid 0.7968-Approximation Algorithm for MAX SAT

Our first hybrid algorithm combines the LP rounding of Asano and Williamson [AW02] with a perturbation of the threshold rounding suggested for MAX 2-SAT by Lewin, Livnat and Zwick [LLZ02]. Our algorithm is given in Figure 4. The algorithm is parameterized by an LP rounding function $g : [0, 1] \to [0, 1]$, a threshold function $S : [-1, 1] \to \mathbb{R}$ and a perturbation probability $p \in [0, \frac{1}{2}]$. We choose the LP rounding function of Asano and Williamson $g = f_3^a$ and the threshold function $S(x) = -\cot(0.5583 \arccos(x) + 0.6466)\sqrt{1 - x^2}$ used by Lewin, Livnat and Zwick.

The analysis of the algorithm is similar to the analysis of the MAX NAE-SAT algorithm. More specifically, for a clause $x_1, \ldots, x_k$ with corresponding vectors $\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ we denote by $prob_{LLZ}(\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)$ the probability that the clause is satisfied using the rounding of step (2(b)i). In addition, let

$$value(\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_k) = \begin{cases} \frac{1 - \boldsymbol{v}_0 \cdot \boldsymbol{v}_1}{2} & \text{if} \quad k = 1 \\ \frac{3 - \boldsymbol{v}_0 \cdot \boldsymbol{v}_1 - \boldsymbol{v}_0 \cdot \boldsymbol{v}_2 - \boldsymbol{v}_1 \cdot \boldsymbol{v}_2}{4} & \text{if} \quad k = 2 \\ \min\left\{1, \frac{1}{\binom{k-1}{2}} \sum_{1 \le i_1 < i_2 < i_3 \le k} u_{i_1 i_2 i_3}\right\} & \text{if} \quad k \ge 3 \end{cases}$$

where

$$u_{i_1 i_2 i_3} = \min\left\{1, \min_{\pi \in \hat{S}_3} \frac{4 - \sum_{l=0}^{3} \boldsymbol{v}_{i_{\pi(l)}} \cdot \boldsymbol{v}_{i_{\pi(l+1)}}}{4}\right\}.$$

A lower bound on the approximation ratio of step (2(b)i) for clauses of size k is therefore

$$\beta_k = \inf \frac{prob_{LLZ}(\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)}{value(\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)}.$$

where the infimum is taken over all $(k+1)$-tuples of vectors $\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in S^k$ that satisfy the "triangle inequalities" and for which $value(\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_k) > 0$. Note that, for $k \geq 3$, $\beta_k \geq \frac{\beta_3}{k}$

$$\begin{aligned} prob_{LLZ}[x_1 \vee \ldots \vee x_k &= \text{TRUE}] \\ &\geq \frac{1}{\binom{k}{3}} \sum_{1 \leq i_1 < i_2 < i_3 \leq k} prob_{LLZ}[x_{i_1} \vee x_{i_2} \vee x_{i_3} = \text{TRUE}] \\ &\geq \frac{1}{\binom{k}{3}} \sum_{1 \leq i_1 < i_2 < i_3 \leq k} \beta_3 u_{i_1 i_2 i_3} \geq \frac{\beta_3}{k} value(\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_k). \end{aligned}$$

Adding the perturbation step, the performance ratio of the rounding (2b) for clauses of size k is at least $\beta_k(1-p)^k + (1-\beta_k)(1-(1-p)^k)$.

The probability $prob_{LLZ}(\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)$ may be written as a $(k-1)$-dimensional integral. However, this integral does not seem to have an analytical representation, even for $k = 2$. We used numerical methods to compute lower bounds on β_k. In particular, $\beta_1 > 0.9834, \beta_2 > 0.9401$ and $\beta_3 > 0.8610$. It is possible to obtain a rigorous proof for the latter three bounds using a tool such as $\mathcal{R}$EAL$\mathcal{S}$EARCH [Zwi02]. However, this would require a tremendous amount of work.

In this scenario, for any $p_1, p_2 \geq 0$ that satisfy $p_1 + p_2 = 1$, the approximation ratio of our algorithm is bounded below by the approximation ratio of an algorithm that runs the rounding of (2a) with probability p_1 and the rounding of (2b) with probability p_2. We can therefore formulate an optimization problem over the variables a, p, p_1, p_2 for which a feasible solution gives values for a and p and a lower bound on the performance guarantee:

$$\begin{array}{llll} \max & \beta & & \\ s.t. & p_1(0.9834(1-p) + (1-0.9834)p) + p_2\zeta_k^a & \geq \beta & k=1 \\ & p_1(0.9401(1-p)^2 + (1-0.9401)(1-(1-p)^2) + p_2\zeta_k^a & \geq \beta & k=2 \\ & p_1(\frac{0.8610}{k}(1-p)^k + (1-\frac{0.8610}{k})(1-(1-p)^k) + p_2\zeta_k^a & \geq \beta & k \geq 3 \\ & p_1 + p_2 = 1 & & \\ & \frac{1}{2} \leq a \leq \frac{\sqrt{e}}{2} & & \\ & 0 \leq p \leq \frac{1}{2} & & \\ & 0 \leq p_1, p_2 \leq 1 & & \end{array}$$

A feasible solution is $p_1 = 0.732178, p_2 = 0.267822, a = 0.731649$ and $p = 0.008741$ giving us an approximation ratio of $\beta = 0.7968$. In this setting, the constraints of

APPROXMAX-SAT–$RPR^2(g, f, p)$

1. Solve the MAX SAT semidefinite programming relaxation of Figure 3.
2. Return the maximal solution between
   (a) For $1 \leq i \leq n$, set $x_i = 1$ independently with probability $g(y_i)$
   (b) i. Let $\boldsymbol{r} = (r_0, r_1, \ldots, r_n)$, where $r_0, r_1, \ldots, r_n$ are independent standard normal variables. For $0 \leq i \leq n$, set $x_i = 1$ independently with probability $f(\boldsymbol{v}_i \cdot \boldsymbol{r})$
   ii. If $x_0 = 1$, set $x_i = \bar{x}_i$, for $0 \leq i \leq n$
   iii. For $1 \leq i \leq n$, set $x_i = \bar{x}_i$ independently with probability p

Fig. 5. An RPR^2 based approximation algorithm for MAX SAT

$k = 2, 15$ are tight and the constraints of $k = 1, 7$ are almost tight. Hence, our algorithm achieves its worst case ratio on instances in which all clauses are of size 2 or 15.

We note that the addition of any combination of the Goemans and Williamson algorithm [GW94] with the rounding function f_4^a (for any $\sqrt{\mathrm{e}}/2 \leq a \leq 1$) and the algorithms of Halperin and Zwick [HZ01] with or without perturbation does not improve the approximation ratio.

3.4 An Improved Approximation Algorithm Using RPR^2

Our improved hybrid algorithm combines the LP rounding of Asano [Asa03] and RPR^2. Our algorithm is given in Figure 5. The algorithm is parameterized by a rounding function $g : [0, 1] \to [0, 1]$, an RPR^2 function f and a perturbation probability p. We choose $g = f_4^a$, $p = \frac{2}{k_{max}}$ and an RPR^2 function f_{SAT} that resembles the f_{NAE} used in the previous section. The function f_{SAT} is a piecewise linear connecting between the points $(-\infty, 0)$, $(-4, 0)$, $(-2.064, 0.029)$, $(0, 0.029)$, $(0, 0.971)$,$(2.064, 0.971)$, $(4, 1)$ and $(\infty, 1)$. The RPR^2 function f_{SAT} is given in Figure 2(b).

The analysis of our hybrid algorithm resembles the one of the MAX NAE-SAT algorithm. Similar arguments shows that for sufficiently large n, the parameter $\alpha_{k+1}(f_{SAT})$ (as defined in Subsection 2.4) is a lower bound on the approximation ratio of step (2b) for clauses of size k. Hence, a lower bound on the performance guarantee of our hybrid algorithm may be obtained by solving the following optimization problem:

$$\begin{array}{ll} \max & \beta \\ s.t. & p_1\alpha_{k+1}(f_{SAT}) + p_2\eta_k^a \geq \beta \quad k \geq 1 \\ & p_1 + p_2 = 1 \\ & \frac{\sqrt{\mathrm{e}}}{2} \leq a \leq 1 \\ & 0 \leq p_1, p_2 \leq 1 \end{array}$$

We conjecture that conjecture 1 holds for f_{SAT} as well. Based on the conjecture we calculated lower bounds on $\hat{\alpha}_k(f_{SAT})$. Using the arguments of subsection 2.5, for a proper choice of k_{max} these bounds are also bounds on $\alpha_k(f_{SAT})$. Using these bounds, a feasible solution is $p_1 = 0.648682, p_2 = 0.351318, a = 0.840105$ yielding a conjectured approximation ratio of $\beta = 0.8434$. In this setting, the constraints of $k = 1$, $k = 2$ and $k = 5$ are tight, i.e., our algorithm achieves its worst case ratio on instances in which all clauses are of size 1, 2 or 5.

We note that the addition of any combination of Goemans and Williamson algorithm [GW94] with the rounding function f_3^a (for any $1/2 \leq a \leq \sqrt{\mathrm{e}}/2$), Lewin, Livnat and Zwick algorithm [LLZ02], Halperin and Zwick algorithms [HZ01] with or without perturbation does not yield an improved approximation ratio. Again, in our search for an optimal RPR^2 function we explored various piecewise linear symmetric monotone functions with up to eight turning points.

4 Concluding Remarks

We used the RPR^2 technique to obtain approximation algorithms for the MAX NAE-SAT and MAX SAT problems with conjectured approximation ratios of 0.8279 and 0.8434, respectively. We also used the MAX 2-SAT algorithm of Lewin Livnat and Zwick to obtain an 0.7968-approximation algorithm for the MAX SAT problem.

References

[AE98] G. Andersson and L. Engebretsen. Better approximation algorithms for SET SPLITTING and NOT-ALL-EQUAL SAT. *Information Processing Letters*, 65:305–311, 1998.

[AMMN05] N. Alon, K. Makarychev, Y. Makarychev, and A. Naor. Quadratic forms on graphs. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, Maryland*, pages 486–493, 2005.

[AOH96] T. Asano, T. Ono, and T. Hirata. Approximation algorithms for the maximum satisfiability problem. *Nordic Journal of Computing*, 3:388–404, 1996.

[Asa97] T. Asano. Approximation algorithms for MAX SAT: Yannakakis vs. Goemans-Williamson. In *Proceedings of the 3nd Israel Symposium on Theory and Computing Systems, Ramat Gan, Israel*, pages 24–37, 1997.

[Asa03] T. Asano. An improved analysis of Goemans and Williamson's LP-relaxation for MAX SAT. *FCT 2003*, LNCS 2751:2–14, 2003.

[AW02] T. Asano and D. P. Williamson. Improved approximation algorithms for MAX SAT. *Journal of Algorithms*, 42:173–202, 2002.

[CW04] M. Charikar and A. Wirth. Maximizing Quadratic Programs: Extending Grothendieck's Inequality. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, Rome, Italy*, pages 54–60, 2004.

[FG95] U. Feige and M. X. Goemans. Approximating the value of two prover proof systems, with applications to MAX-2SAT and MAX-DICUT. In *Proceedings of the 3rd Israel Symposium on Theory and Computing Systems, Tel Aviv, Israel*, pages 182–189, 1995.

[FL01] U. Feige and M. Langberg. The RPR^2 rounding technique for semidefinite programs. In *Proceedings of the 28th Int. Coll. on Automata, Languages and Programming, Crete, Greece*, pages 213–224, 2001.

[GW94] M. X. Goemans and D. P. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7:656–666, 1994.

[GW95] M. X. Goemans and D. P. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *Journal of the ACM*, 42:1115–1145, 1995.

[Hås01] J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001.

[HYZ04] Q. Han, Y. Ye, and J. Zhang. Improved Approximation for Max Set Splitting and Max NAE SAT. *Discrete Applied Mathematics*, 142(1-3):133–149, 2004.

[HZ01] E. Halperin and U. Zwick. Approximation algorithms for MAX 4-SAT and rounding procedures for semidefinite programs. *Journal of Algorithms*, 40:184–211, 2001.

[Joh74] D. S. Johnson. Approximation algorithms for combinatorical problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[KZ97] H. Karloff and U. Zwick. A $7/8$-approximation algorithm for MAX 3SAT? In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, Miami Beach, Florida*, pages 406–415, 1997.

[LLZ02] M. Lewin, D. Livnat, and U. Zwick. Improved rounding techniques for the MAX 2-SAT and MAX DI-CUT problems. In *Proceedings of the 9th IPCO, Cambridge, Massachusetts*, pages 67–82, 2002.

[MM01a] S. Matuura and T. Matsui. 0.863-approximation algorithm for MAX DICUT. In *Approximation, Randomization and Combinatorial Optimization: Algorithms and Techniques, Proceedings of APPROX-RANDOM'01, Berkeley, California*, pages 138–146, 2001.

[MM01b] S. Matuura and T. Matsui. 0.935-approximation randomized algorithm for MAX 2SAT and its derandomization. Technical Report METR 2001-03, Department of Mathematical Engineering and Information Physics, the University of Tokyo, Japan, September 2001.

[Nes98] Y. E. Nesterov. Semidefinite relaxation and nonconvex quadratic optimization. *Optimization Methods and Software*, 9:141–160, 1998.

[Yan94] M. Yannakakis. On the approximation of maximum satisfiability. *Journal of Algorithms*, 17:475–502, 1994.

[Ye01] Y. Ye. A .699-approximation algorithm for Max-Bisection. *Mathematical Programming*, 90(1, Ser. A):101–111, 2001.

[Zwi99] U. Zwick. Outward rotations: a tool for rounding solutions of semidefinite programming relaxations, with applications to MAX CUT and other problems. In *Proceedings of the 31th Annual ACM Symposium on Theory of Computing, Atlanta, Georgia*, pages 679–687, 1999.

[Zwi02] U. Zwick. Computer assisted proof of optimal approximability results. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, California*, pages 496–505, 2002.

The Hardness of Network Design for Unsplittable Flow with Selfish Users*

Yossi Azar and Amir Epstein

School of Computer Science, Tel-Aviv University, Tel-Aviv, 69978, Israel
azar@tau.ac.il, amirep@tau.ac.il

Abstract. In this paper we consider the network design for selfish users problem, where we assume the more realistic unsplittable model in which the users can have general demands and each user must choose a single path between its source and its destination. This model is also called atomic (weighted) network congestion game. The problem can be presented as follows : given a network, which edges should be removed to minimize the cost of the worst Nash equilibrium?

We consider both computational issues and existential issues (i.e. the power of network design). We give inapproximability results and approximation algorithms for this network design problem. For networks with linear edge latency functions we prove that there is no approximation algorithm for this problem with approximation ratio less then $(3+\sqrt{5})/2 \approx 2.618$ unless $P = NP$. We also show that for networks with polynomials of degree d edge latency functions there is no approximation algorithm for this problem with approximation ratio less then $d^{\Theta(d)}$ unless $P = NP$. Moreover, we observe that the trivial algorithm that builds the entire network is optimal for linear edge latency functions and has an approximation ratio of $d^{\Theta(d)}$ for polynomials of degree d edge latency functions. Finally, we consider general continuous, non-decreasing edge latency functions and show that the approximation ratio of any approximation algorithm for this problem is unbounded, assuming $P \neq NP$. In terms of existential issues we show that network design cannot improve the maximum possible bound on the price of anarchy in the worst case.

Previous results of Roughgarden for networks with n vertices where each user controls only a negligible fraction of the overall traffic showed optimal inapproximability results of $4/3$ for linear edge latency functions, $\Theta(d/\ln d)$ for polynomial edge latency functions and $n/2$ for general continuous non-decreasing edge latency functions. He also showed that the trivial algorithm that builds the entire network is optimal for that case.

1 Introduction

1.1 Selfish Routing

A major component of any large-scale network system is the routing mechanism, namely choosing a communication path between a sender and a receiver of traffic.

* Research supported in part by the German-Israeli Foundation.

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 41–54, 2006.

In most cases, such as the Internet, wireless networks, or overlay networks built on top of the Internet, traffic from a sender to a receiver is sent over a *single* path; splitting the traffic causes the problem of packet reassembly at the receiver and thus is generally avoided. When choosing routing paths, the typical objective is to minimize the total latency. In most of these network systems it is infeasible to maintain one centralized authority that imposes efficient routing strategies on the network traffic. As a result users act independently and "selfishly": each user tries to minimize his traffic cost based on current network traffic.

This problem can be mathematically formalized using classical game theory as follows. The network users are viewed as independent agents participating in a non-cooperative game. Each agent wishes to use the minimum latency path from its source to its destination, given the link congestion caused by the rest of the agents. This system is said to be in Nash Equilibrium if no agent has an incentive to change his path from its source to its destination. It is well known that Nash Equilibria do not in general optimize the social welfare (see, e.g, "The Prisoner's Dilemma" [7, 15]) and can be far from the global optimum.

Equilibria can be defined for pure strategies, where a single path is chosen by each user and for mixed strategies, where a probability distribution over the paths is used instead of a single path. Our hardness results hold for pure strategies and hence also for mixed strategies. Nash equilibrium requires mixed strategies, but in some cases pure strategies suffice [9, 14, 17].

The degradation of network performance caused by the lack of a centralized authority can be measured using the worst-case coordination ratio (price of anarchy) suggested by Koutsoupias and Papadimitriou [10] and Papadimitriou [16] which is the ratio between the worst possible Nash Equilibrium and the social optimum, see, e.g., [1,4–6,10,11,16,19–21].

Braess's paradox is the counterintuitive phenomenon that removing edges from a network can improve its performance. This paradox was first discovered by Braess [3] and later reported by Murchland [12]. Braess's paradox motivates the following network design problem for improving the performance of a network with selfish users: How can we design selfish users networks to minimize the inefficiency inherent in Nash equilibrium?

Previous results of Roughgarden [18] for networks of n vertices with single source-sink pair where each user controls only a negligible fraction of the overall traffic showed optimal inapproximability results of $4/3$ for linear edge latency functions, $\Theta(d/\ln d)$ for polynomials of degree d edge latency functions and $n/2$ for general continuous non-decreasing edge latency functions. He also showed that the trivial algorithm that builds the entire network is optimal. For linear and polynomial edge latency functions these follow from price of anarchy results of Roughgarden and Tardos [21].

1.2 Our Results

We prove the following results for the network design problem for general networks with unsplittable flow:

- For linear latency functions we prove that for any $\epsilon > 0$ there is no $(\beta - \epsilon)$-approximation algorithm for network design where $\beta = (3 + \sqrt{5})/2 \approx 2.618$, assuming $P \neq NP$. Price of anarchy results appearing in [1] imply that this hardness result is optimal.
- For latency functions which are polynomials of degree d we prove that there is no approximation algorithm for network design, with approximation ratio less then $d^{\Theta(d)}$, assuming $P \neq NP$. Price of anarchy results appearing in [1] imply that the trivial algorithm has an approximation ratio of $d^{\Theta(d)}$. We note that our hardness result is $\Omega(d^{d/4})$ where the trivial algorithm's approximation ratio is $O(2^d d^{d+1})$.
- For general continuous, non-decreasing latency functions we show that the approximation ratio of any polynomial time approximation algorithm for NETWORK DESIGN is unbounded, assuming $P \neq NP$.

The above results deal with the computational issues related to the power of network design. We also consider the existential issues. Specifically we also consider the question whether network design can reduce the maximum bound on the price of anarchy in the worst case. We answer this negatively.

- For linear edge latency functions there is a network with coordination ratio at least $\beta - \epsilon$ where $\beta = (3 + \sqrt{5})/2 \approx 2.618$ for any $\epsilon > 0$, for polynomials of degree d edge latency functions there is a network with coordination ratio at least $\Omega(d^{d/4})$ and for general latency functions (continuous and non-decreasing) there is a network with unbounded coordination ratio such that in these networks network design cannot decrease the cost of the worst Nash equilibrium.

All our results hold for pure strategies and hence also for mixed strategies, since these are hardness and non existential results.

Techniques: To prove our hardness results we first prove hardness results to SELECTIVE NETWORK DESIGN which is an harder problem than NETWORK DESIGN. Then we show a general way to transform many types of hardness results of selective network design to hardness results of network design.

1.3 Paper Structure

The paper is organized as follows. Section 2 includes formal definitions and notations. In section 3 we prove inapproximability results for NETWORK DESIGN and observe the approximation ratio of the trivial algorithm for linear and polynomial latency functions. In section 4 we consider the existential issues of NETWORK DESIGN and show that it cannot reduce the maximum bound on the price of anarchy.

2 Definitions and Preliminaries

2.1 The Model

We consider the following model which is called weighted network congestion game: there is a directed graph $G = (V, E)$. Each edge $e \in E$ is given a load-dependent latency function $f_e : \mathcal{R}^+ \rightarrow \mathcal{R}^+$. There are n users, where user j ($j = 1, \ldots, n$) has a bandwidth request defined by a tuple (s_j, t_j, w_j), where $s_j, t_j \in V$ are the source/destination pair, and $w_j \in \mathcal{R}^+$ corresponds to the required bandwidth. We denote the set of (simple) $s_j - t_j$ paths by $\mathcal{Q}_j$. Request j can be assigned to any path Q from the set of paths $\mathcal{Q}_j$, such that the required bandwidth w_j has to be reserved along the path Q.

We assume that the users are non-cooperative and each one wishes to minimize its own cost with no regard to the global optimum. In Pure strategies user j selects a single path $Q \in \mathcal{Q}_j$ and assigns his request to it. Each user is aware of the choices made by all other users when making his decision.

2.2 Pure Strategies Definition

First, we give some simpler notations we use for a system $\mathcal{S} = (Q_1, \ldots, Q_n)$ of pure strategies. Let Q_j be the path associated with request j. We define $J(e) = \{j | e \in Q_j\}$ the set of requests assigned to a path containing the edge e. The load on edge e is defined by: $l_e = \sum_{j \in J(e)} w_j$.

For the optimal routes let Q_j^* be the path associated with request j. We define $J^*(e) = \{j | e \in Q_j^*\}$ the set of requests assigned to a path containing the edge e. We denote the load on edge e by l_e^*.

Definition 1. *The latency of user j for assigning his request in system $\mathcal{S}$ to path Q (instead of path Q_j) is defined as:*

$$c_{Q,j} = \sum_{(e \in Q) \wedge (e \in Q_j)} f_e(l_e) + \sum_{(e \in Q) \wedge (e \notin Q_j)} f_e(l_e + w_j). \tag{1}$$

2.3 Nash Equilibrium and Coordination Ratio

Nash equilibrium is characterized by the property that there is no incentive for any user to change its strategy and defined as follows

Definition 2 (Nash Equilibrium). *A system $\mathcal{S}$ is said to be in pure Nash Equilibrium if and only if for every $j \in \{1, \ldots, n\}$ and $Q \in \mathcal{Q}_j$, $c_{Q_j,j} \leq c_{Q,j}$.*

Definition 3. *The cost $C(\mathcal{S})$ for a given system $\mathcal{S}$ of pure strategies is defined as the total latency incurred by $\mathcal{S}$, that is $C(\mathcal{S}) = \sum_{e \in E} f_e(l_e) l_e$.*

We are interested in estimating the worst-case coordination ratio when pure Nash equilibrium exists. We denote the optimal system of pure strategies by $\mathcal{S}^*$.

Definition 4 (Coordination Ratio). *The coordination ratio is defined as $R = max_{\mathcal{S}} \frac{C(\mathcal{S})}{C(\mathcal{S}^*)}$, where the maximum is taken over all strategies $\mathcal{S}$ in Nash equilibrium.*

2.4 Formalizing the Network Design Problem

Let $C(H, \mathcal{S})$ be the total latency incurred by a given system $\mathcal{S}$ of pure strategies in Nash equilibrium for a subgraph H of G. If there is a user j such that $\mathcal{Q}_j = \emptyset$ in the subgraph H then $C(H, \mathcal{S}) = \infty$. We denote by $C(H)$ the maximum cost obtained for the graph H, where the maximum is taken over all strategies $\mathcal{S}$ in Nash equilibrium for the graph H. We note that for unsplittable flow we do not know how to compute the value $C(H)$ in polynomial time, while for the case of splittable flow (or alternatively where each user controls a negligible amount of the traffic) the value $C(H)$ can be recovered from the subgraph H in polynomial time via convex programming for positive convex functions (see [2]). Now we define the network design and selective network design problems for unsplittable flow.

The Network Design Problem: Given a weighted network congestion game with directed graph $G = (V, E)$, find a subgraph H of G that minimizes $C(H)$.

The Selective Network Design Problem: Given a weighted network congestion game with directed graph $G = (V, E)$ and $E_1 \subseteq E$, find a subgraph H of G containing the edges of E_1 that minimizes $C(H)$.

The above formulation of the SELECTIVE NETWORK DESIGN problem is itself interesting, but the main purpose of the presentation of this problem is for proving inapproximability results for the NETWORK DESIGN problem. In particular we first prove hardness results for the selective network design problem (which is a harder problem than the network design problem and hence it is easier to show hardness results for this problem) and then we modify the instance of the selective network design problem used in the proof of inapproximability of selective network design to an instance of the network design problem to show its inapproximability result.

3 Inapproximabilty of Network Design

In this section we consider the computational issues of NETWORK DESIGN. Specifically we prove inapproximabilty results for NETWORK DESIGN and observe the approximation ratio of the trivial algorithm for linear and polynomial latency functions.

3.1 Linear Latency Functions

In this section we consider the case where the latency of each edge is linear in the edge congestion. Specifically $f_e(x) = a_e x + b_e$ for each edge $e \in E$, where a_e and b_e are nonnegative reals. Let $\beta = (3 + \sqrt{5})/2 \approx 2.618$.

A trivial algorithm for the problem outputs the entire network G. We begin by observing that this trivial algorithm for NETWORK DESIGN is a β-approximation algorithm, where the latency functions are linear. This will follow easily from a result of Awerbuch et al. [1]. They proved that in every network with linear latency functions and unsplittable flow, the cost of unsplittable flow at Nash equilibrium is at most β times that of every other feasible unsplittable flow.

Proposition 1. *([1]) For linear latency functions and weighted demands let S^* be a system of strategies and let S be a system of strategies in Nash equilibrium. Then $C(S) \leq \beta \cdot C(S^*)$.*

Corollary 1. *The trivial algorithm is a β-approximation for linear latency functions and weighted demands.*

Proof. Consider an instance of the problem with subgraph H of G minimizing $C(H)$. Let S and S^* denote systems of strategies at Nash equilibrium for the graphs G and H, respectively. Since S^* can be viewed as a system of strategies for the graph G, it follows from proposition 1 that $C(G,S) \leq \beta \cdot C(G,S^*)$ and hence $C(G) \leq \beta \cdot C(H)$.

The main result of this section is a lower bound on the approximation ratio of any polynomial algorithm (unless P=NP).

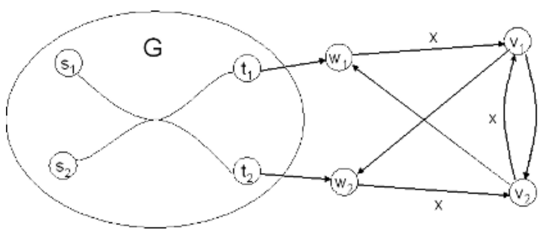


Fig. 1. Proof of Theorem 1

Theorem 1. *For linear latency functions and weighted demands assuming $P \neq NP$ there is no $(\beta - \epsilon)$-approximation algorithm for SELECTIVE NETWORK DESIGN (recall that $\beta = (3+\sqrt{5})/2 \approx 2.618$).*

Proof. We reduce from the problem 2 Directed Disjoint Paths (2DDP): Given a directed graph $G = (V, E)$ and distinct vertices $s_1, s_2, t_1, t_2 \in V$, are there s_i-t_i paths P_1 and P_2, such that P_1 and P_2 are vertex disjoint? Fortune et al. [8] proved that this problems is NP-complete. We will show that for linear latency functions and weighted demands $(\beta - \epsilon)$-approximation algorithm for the SELECTIVE NETWORK DESIGN problem can be used to distinguish "yes" and "no" instances of 2DDP in polynomial time. Consider an instance I of 2DDP, as above. We add the vertices w_1, w_2, v_1 and v_2 to the vertex set V and include directed edges (t_1, w_1), (t_2, w_2), (w_1, v_1), (w_2, v_2), (v_1, v_2), (v_2, v_1), (v_1, w_2) and (v_2, w_1) as shown in Figure 1. We denote the new network by $G' = (V', E')$. Let $E_1 := E' - E$ be the group of edges that the subgraph H of G' should contain. We define the following linear latency functions f for the edges of E': the edges

$(w_1, v_1), (w_2, v_2), (v_1, v_2), (v_2, v_1)$ are given the latency functions $f(x) = x$ and all other edges are given the latency functions $f(x) = 0$. We later choose $\phi = \frac{1+\sqrt{5}}{2}$ which is the golden ratio. We consider an atomic weighted network congestion game with six players that uses the network G'. Player 1 has a bandwidth request (s_1, v_1, ϕ) (player 1 has to move ϕ units of bandwidth from s_1 to v_1), player 2 has a bandwidth request (s_2, v_2, ϕ), player 3 has a bandwidth request $(v_1, v_2, 1)$, player 4 has a bandwidth request $(v_2, v_1, 1)$, player 5 has a bandwidth request $(s_1, t_1, 1)$ and player 6 has a bandwidth request $(s_2, t_2, 1)$. The new instance I' can be constructed from I in polynomial time. To complete the proof, it suffices to show the following two statements.

1. If I is a "yes" instance of 2DDP, then G' contains a subgraph H of G' with $C(H) = 2\phi^2 + 2$.
2. If I is a "no" instance of 2DDP, then $C(H) \geq 2(\phi+1)^2+2\phi^2$ for all subgraphs H of G'.

Recall that the subgraph H of G' should contain the edges in E_1. To prove (1), let P_1 and P_2 be vertex-disjoint paths in G, respectively, and obtain H by deleting all edges of G not contained in some P_i. Then, H is a subgraph of G' that contains the paths $s_1 - t_1 - w_1 - v_1$, $s_2 - t_2 - w_2 - v_2$, $v_1 - v_2$, $v_2 - v_1$, $s_1 - t_1$ and $s_2 - t_2$. These paths are the direct paths of players $1-6$ respectively. The optimal solution S_1 is obtained when each player chooses its direct path and this solution is the only Nash equilibrium for I' in which the costs of players $1 - 6$ are $\phi^2, \phi^2, 1$, 1,0 and 0 respectively. The total cost $C(H, S_1) = 2\phi^2 + 2$. This solution is the unique Nash Equilibrium, since the dominant strategy of each of the players $1, 2, 5, 6$ is to choose its direct path which is its unique simple path and given these strategies of players $1, 2, 5, 6$ the best response of each of the players 3 and 4 is its direct path. For (2), we may assume that H contains $s_1 - t_1$ and $s_2 - t_2$ paths. In this case the paths $s_1 - t_1$ and $s_2 - t_2$ are not disjoint and hence H must contain $s_1 - t_2$ and $s_2 - t_1$ paths. Let S_2 be the system of strategies where player 1 uses its indirect path $s_1 - t_2 - w_2 - v_2 - v_1$, player 2 uses its indirect path $s_2 - t_1 - w_1 - v_1 - v_2$, player 3 uses its indirect path $v_1 - w_2 - v_2$, player 4 uses its indirect path $v_2 - w_1 - v_1$, player 5 uses its direct path $s_1 - t_1$ and player 6 uses its direct path $s_2 - t_2$. Then this is a Nash equilibrium and the costs of players $1 - 6$ are $2\phi + 1$, $2\phi + 1$, $\phi + 1$, $\phi + 1$, 0 and 0 respectively. The total cost $C(H, S_2) = 2(\phi + 1)^2 + 2\phi^2$. The ratio of the total costs $C(H, S_2)$ and $C(H, S_1)$ is :

$$\frac{2(\phi + 1)^2 + 2\phi^2}{2\phi^2 + 2}.$$

We choose $\phi = \frac{1+\sqrt{5}}{2}$ which is the golden ratio and get a ratio $\beta = \phi + 1 \approx 2.618$. This completes the proof.

We call a family X of latency functions **nice** if all of its functions are non-negative, continuous and non-decreasing and the family is closed under non-negative linear combinations. Note that ,obviously, linear and polynomial latency functions satisfy this definition.

The following Lemma provides a way to transform inapproximability result of SELECTIVE NETWORK DESIGN to inapproximability result of NETWORK DESIGN.

Lemma 1. *Given a direct reduction from a hard problem Q to SELECTIVE NETWORK DESIGN for a nice family of latency functions that shows that it is hard to c-approximate selective network design, then one can create a similar reduction from Q to NETWORK DESIGN for the same family of latency functions that shows that it is hard to c-approximate network design, if the following condition applies : for every instance of selective network design created by the reduction with weighted network congestion game consisting of graph $G' = (V', E')$, $E_1 \subseteq E'$ and every subgraph $H \subseteq G'$ that has been considered in the proof (i.e. that contains E_1) it holds that in the worst Nash equilibrium each player has a unique best response (best strategy).*

Proof. For every instance of SELECTIVE NETWORK DESIGN created by the reduction with weighted network congestion game consisting of graph $G' = (V', E')$, $E_1 \subseteq E'$ and every subgraph $H \subseteq G'$ that has been considered in the proof (i.e. that contains E_1) we do the following. Let $\delta > 0$. For each edge $e \in E_1$ we make the following local modification. First we split the edge by adding a new vertex w_e and replacing the edge $e = (u, v)$ by the two edges $e_1 = (u, w_e)$ and $e_2 = (w_e, v)$. The new edges e_1 and e_2 will posses the latency function $\frac{1}{2} f_e$. Then we add two players with requests (u, w_e, δ) and (w_e, v, δ). We denote the modified network created from H by $H^* = (V^*, E^*)$. Since the costs of the players change continuously as a function of δ, for sufficiently small constant δ it holds that in the new weighted network congestion game the worst Nash equilibrium remains a Nash equilibrium where each player uses its original strategy and this strategy is its unique best response (the new players choose their unique strategy). Moreover, the total cost changes continuously as a function of δ and hence the new total cost is arbitrarily close to the original total cost as a function of δ. Additionally, each of the edges in E_1 cannot be deleted since it is a unique strategy of a new player. Hence the inapproximablity proof for SELECTIVE NETWORK DESIGN is also a proof for NETWORK DESIGN.

Unfortunately we cannot use Lemma 1 to prove Theorem 2 according to the result of Theorem 1, hence we have to modify the weighed network congestion game used in the proof of Theorem 1 to satisfy the condition required by Lemma 1.

Theorem 2. *For linear latency functions and weighted demands assuming $P \neq NP$ there is no $(\beta - \epsilon)$-approximation algorithm for NETWORK DESIGN (recall that $\beta = (3 + \sqrt{5})/2 \approx 2.618$).*

Proof. We modify the weighted network congestion game defined in the proof of Theorem 1 as follows : Let $\epsilon > 0$. First we modify the network $G' = (V', E')$ shown in Figure 1 and obtain the network $G'' = (V'', E'')$ shown in Figure 2. Next we modify the requests of players 3 and 4. Player 3 has a bandwidth request $(z_1, z_2, 1)$ (its previous request was $(v_1, v_2, 1)$) and player 4 has a bandwidth

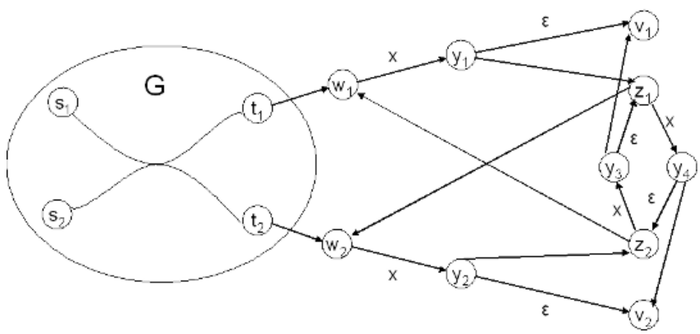


Fig. 2. Proof of Theorem 2

request $(z_2, z_1, 1)$ (its previous request was $(z_2, v_1, 1)$). The direct paths of players $1-6$ are $s_1 - t_1 - w_1 - y_1 - v_1$, $s_2 - t_2 - w_2 - y_2 - v_2$, $z_1 - y_4 - z_2$, $z_2 - y_3 - z_1$, $s_1 - t_1$ and $s_2 - t_2$ respectively. The indirect paths of players $1-4$ are $s_1 - t_2 - w_2 - y_2 - z_2 - y_3 - v_1$, $s_2 - t_1 - w_1 - y_1 - z_1 - y_4 - v_2$, $z_1 - w_2 - y_2 - z_2$, $z_2 - w_1 - y_1 - z_1$ respectively. Now it is easy to verify according to the proof of Theorem 1 that the following properties hold:

1. The optimum which is the best Nash equilibrium is obtained when each player chooses its direct path.
2. The worst Nash equilibrium is obtained when each of the players $1-4$ chooses its indirect path and players $5, 6$ choose their direct path.
3. In the best and worst Nash equilibria the total cost was increased by at most 8ϵ.
4. In the best and worst Nash equilibria each player has a unique best response (best setrategy).

Let $E_1 = E'' - E$ be the group of edges that the subgraph H of G'' should contain. It follows from the above properties and the proof of Theorem 1 that the above modified weighted network congestion game can be used to prove Theorem 1. It also follows that for every subgraph considered in the new proof of Theorem 1 which uses the modified weighted network congestion game, in the worst Nash equilibrium each player has a unique best response (best startegy). Applying Lemma 1 completes the proof.

3.2 Polynomial Latency Functions

In this section we consider the case where the latency of each edge is a polynomial of degree d in the edge congestion. Specifically $f_e(x) = \sum_i a_{e,i} x^i$ for each edge $e \in E$, where $a_{e,i}$ are nonnegative reals.

Proposition 2. *([1]) For polynomial of degree d latency functions and weighted demands let S^* be a system of strategies and let S be a system of strategies in Nash equilibrium. Then $C(S) \le O(2^d d^{d+1}) \cdot C(S^*)$.*

Corollary 2. *The trivial algorithm is a $O(2^d d^{d+1})$-approximation for linear latency functions and weighted demands.*

The main results of this section are lower bounds on the approximation ratio of any polynomial algorithm for weighted demands (unless P=NP).

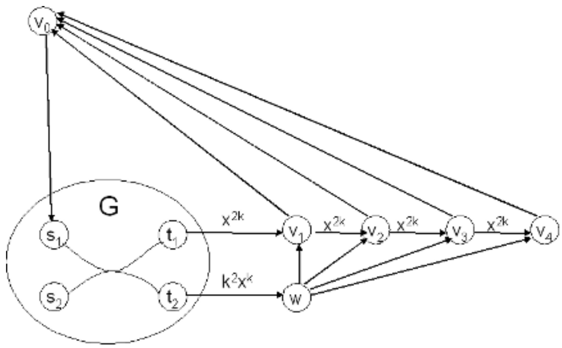


Fig. 3. Proof of Theorem 3. In this example $n = 4$.

Theorem 3. *For polynomials of degree d latency functions and weighted demands assuming $P \neq NP$ there is a lower bound of $\Omega(d^{d/4})$ on the approximation ratio of any polynomial time approximation algorithm for SELECTIVE NETWORK DESIGN.*

Proof. Let $c = 2$,let $d = 2k$ (we can assume that d is even), let $n = k\sqrt{k}/c$. We reduce from the problem 2 Directed Disjoint Paths (2DDP): Given a directed graph $G = (V, E)$ and distinct vertices $s_1, s_2, t_1, t_2 \in V$, are there s_i-t_i paths P_1 and P_2, such that P_1 and P_2 are vertex disjoint? Fortune et al. [8] proved that this problems is NP-complete. We will show that for polynomials of degree d latency functions and weighted demands $O(d^{d/4})$-approximation algorithm for the SELECTIVE NETWORK DESIGN problem can be used to distinguish "yes" and "no" instances of 2DDP in polynomial time. Consider an instance I of 2DDP, as above. We now build the graph $G' = (V', E')$ shown in Figure 3. Let $E_1 = E' - E$ be the group of edges that the subgraph H of G' should contain. We begin by adding the vertices w and $v_0, \ldots, v_n$ to the vertex set V and include directed edges (v_0, s_1), (t_1, v_1), (t_2, w), (v_i, v_{i+1}) for $i = 1, \ldots, n-1$, (v_i, v_0) for $i = 1, \ldots, n$ and (w, v_i) for $i = 1, \ldots, n$. Next we add the edge latency functions. Edges (t_1, v_1) and (v_i, v_{i+1}) for $i = 1, \ldots, n-1$ will possess the latency function $f(x) = x^{2k}$, edge (t_2, w) will possess the latency function $f(x) = k^2x^k$, all other edges will possess the latency function $f(x) = 0$. Let $\delta > 0$ be sufficiently small. We consider an atomic weighted network congestion game with $n+3$ players that use the network G'. Player 1 has a bandwidth request (s_2, v_n, k). For $i = 2 \ldots n+1$ player i has a bandwidth request $(v_{i-2}, v_{i-1}, c\sqrt{k})$. Player $n+2$ has a bandwidth request (s_1, t_1, δ) and player $n+3$ has a bandwidth request

(s_2, t_2, δ). The new instance I' can be constructed from I in polynomial time. To complete the proof, it suffices to show the following two statements.

1. If I is a "yes" instance of 2DDP, then G' contains a subgraph H of G' with $C(H) = k^{k+2}2^{2k} + k^{k+3}$.
2. If I is a "no" instance of 2DDP, then $C(H) \geq k^{2k+4}$ for all subgraphs H of G'.

To prove (1), let P_1 and P_2 be vertex-disjoint paths in G, respectively, and obtain H by deleting all edges of G not contained in some P_i. Then, H is a subgraph of G'. There is one simple path for each player. The optimal solution is obtained when each player chooses its direct path as follows. Player 1 chooses the path $s_2 - t_2 - w - v_n$,player 2 chooses the path $v_0 - s_1 - t_1 - v_1$, for $i = 3, \dots n+1$ player i chooses the path $v_{i-2} - v_{i-1}$, player $n+2$ chooses the path $s_1 - t_1$ and player $n+3$ chooses the path $s_2 - t_2$. This solution is the only Nash equilibrium for I', in which $C(H, S) = \sum_{e \in E} f_e(l_e)l_e = n(2\sqrt{k})^{2k+1} + k^2 \cdot k^{k+1} = k\sqrt{k}/2(2\sqrt{k})^{2k+1} + k^2 \cdot k^{k+1} = k^{k+2}2^{2k} + k^{k+3}$. For (2), we may assume that H contains $s_1 - t_1$ and $s_2 - t_2$ paths. In this case H must contain $s_1 - t_2$ and $s_2 - t_1$ paths to satisfy the requests for paths $s_1 - t_1$ and $s_2 - t_2$. If player 1 uses its indirect path $s_2 - t_1 - v_1 - v_2 - \dots - v_n$, for $i = 2 \dots n+1$ player i uses its indirect path $v_{i-2} - v_0 - s_1 - t_2 - w - v_{i-1}$, player $n+2$ uses its direct path $s_1 - t_1$ which must exist and player $n+3$ uses its direct path $s_2 - t_2$ if it exists, otherwise it uses its indirect path $s_2 - t_1 - v_1 - v_0 - s_1 - t_2$, then this is a Nash equilibrium with $C(H, S) \geq k^2 \cdot k^{2k+2} + k \cdot k^{2k+3/2}/2 = k^{2k+4} + k^{2k+5/2}/2$. To show that this is a Nash equilibrium we have to show that no player benefits from changing its path. We assume that player $n+3$ uses its indirect path $s_2 - t_1 - v_1 - v_0 - s_1 - t_2$. The analysis of the case when player $n+3$ uses its direct path $s_2 - t_2$ follows from this case. The cost of player 1 on path $s_2 - t_1 - v_1 - v_2 - \dots - v_n$ is $k\sqrt{k}/2 \cdot k^{2k} = k^{2k+3/2}/2$. The cost of player 1 on path $s_2 - t_2 - w - v_n$ is $k^2 \cdot (k^2 + k + \delta)^k > k^{2k+2}$, which is greater. For $i = 2 \dots n+1$ the cost of player i on path $v_{i-2} - v_0 - s_1 - t_2 - w - v_{i-1}$ is $k^2 \cdot (k^2 + \delta)^k \geq k^{2k+2}$. The cost of player i on path $v_{i-2} - v_{i-1}$ is $(k + 2\sqrt{k})^{2k} > k^2 \cdot (k^2 + \delta)^k$ for sufficiently small δ (but at least one divided by a polynomial in k). Players $n+2$ and $n+3$ cannot decrease their cost by changing path (if one exists). This completes the proof.

Theorem 4. *For polynomials of degree d latency functions and weighted demands assuming $P \neq NP$ there is a lower bound of $\Omega(d^{d/4})$ on the approximation ratio of any polynomial time approximation algorithm for NETWORK DESIGN.*

Proof. In any Nash equilibrium considered in the proof of Theorem 3 every player has a unique best response, hence the result follows from Lemma 1.

3.3 General Latency Functions

In this section we consider the case where the latency of each edge is continuous and non-decreasing in the edge congestion. We show that the approximation ratio of any approximation algorithm is unbounded even as a function of n.

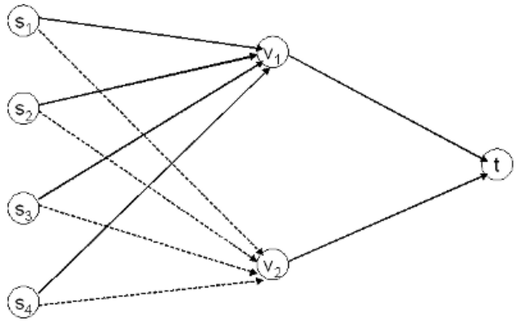


Fig. 4. Proof of Theorem 5. In this example $n = 4$.

Theorem 5. *For general continuous, non-decreasing latency functions assuming $P \neq NP$ the approximation ratio of any polynomial time approximation algorithm for NETWORK DESIGN is unbounded.*

Proof. We show that it is NP-hard to differentiate between zero cost and positive cost. We reduce from the NP-complete problem PARTITION: we are given q positive integers $\{a_1, a_2, \ldots, a_q\}$ and seek for a subset $T \subseteq \{1, 2, \ldots, q\}$ such that $\sum_{j \in T} a_j = \frac{1}{2} \sum_{j=1}^{q} a_j$ [13]. Consider an instance I of PARTITION, as above. We now build the directed graph $G = (V, E)$ shown in Figure 4. Let $n = q$, let $A = \sum_{j=1}^{q} a_j$, $V = \{s, t, v_1, v_2, \ldots, v_n\}$ and E includes the edges (s_i, v_1) for $i = 1, \ldots, n$, (s_i, v_2) for $i = 1, \ldots, n$, (v_1, t) and (v_2, t). The edges (v_1, t) and (v_2, t) will posses the latency function f satisfying $f(x) = 0$ for $x \leq A/2$ and $f(x) = x - A/2$ for $x \geq A/2$, all other edges will posses the latency function $f(x) = 0$. We consider an atomic weighted network congestion game with n players that uses the network G. For $i = 1 \ldots n$ player i has a bandwidth request (s_i, t, a_i).

The new instance I' can be constructed from I in polynomial time. To complete the proof, it suffices to show the following two statements.

1. If I is a "yes" instance of PARTITION, then G contains a subgraph H of G with $C(H) = 0$.
2. If I is a "no" instance of PARTITION, then $C(H) > 0$ for all subgraphs H of G.

To prove (1), let the subset Y be the solution to the instance I, we obtain H by deleting all edges (s_i, v_2) for $i \in Y$ and deleting all edges (s_i, v_1) for i not in Y. Each player has a unique path (strategy) in the graph H. The load on each of the edges (v_1, t) and (v_2, t) is $A/2$ and hence $C(H, S) = 0$. For (2), we may assume that H contains $s_i - t$ path for each $i = 1, \ldots, n$. Let Y' be the subset of players using paths containing the edge (v_1, t) (all other players use paths containing the edge (v_2, t)), then it holds that the load of one of the edges (v_1, t) and (v_2, t) is greater then $A/2$ and hence $C(H, S) > 0$.

4 The Limitation on the Power of Network Design

In this section we consider the existential issues of NETWORK DESIGN. Specifically we consider the question whether network design can reduce the maximum bound on the price of anarchy. We answer this negatively.

Theorem 6. *For any $\epsilon > 0$ and for linear latency functions there is a network with coordination ratio at least $\beta - \epsilon$ in which NETWORK DESIGN cannot decrease the cost of the worst Nash equilibrium (recall that $\beta = (3 + \sqrt{5})/2 \approx 2.618$).*

Proof. The proof follows from the weighted network congestion game with the graph G'' constructed in the proof of Theorem 2 where the graph G is contracted to a single vertex. For each edge in the graph G'' we apply the local modification described in the proof of Lemma 1 and obtain a new weighted network congestion game with coordination ratio at least $\beta - \epsilon$ where edges cannot be removed.

Theorem 7. *For polynomial of degree d latency functions there is a network with coordination ratio at least $\Omega(d^{d/4})$ in which NETWORK DESIGN cannot decrease the cost of the worst Nash equilibrium.*

Proof. The proof follows from the weighted network congestion game with the graph G' constructed in the proof of Theorem 3 where the graph G is contracted to a single vertex. For each edge in the graph G' we apply the local modification described in the proof of Lemma 1 and obtain a new weighted network congestion game with coordination ratio at least $\Omega(d^{d/4})$ where edges cannot be removed.

Theorem 8. *For general latency functions (continuous and non-decreasing) there is a network with unbounded coordination ratio such that in this network NETWORK DESIGN cannot decrease the cost of the worst Nash equilibrium.*

Proof. We prove the result by showing a weighted network congestion game for network with edges that cannot be removed (since each edge is a unique path of a player). In this game there is Nash equilibrium with zero cost and Nash equilibrium with positive cost as follows. We consider a weighted network congestion game that uses the network defined in the proof of Theorem 5 and shown in Figure 4. We denote the new network by $G = (V, E)$. Let the number of source vertices $n = 4$ and let $A = 12$. We define the following players: players $1-4$ have bandwidth requests $(s_1, t, 2)$, $(s_2, t, 3)$, $(s_3, t, 2)$, $(s_4, t, 3)$ respectively. For each $i = 1 - 4$ we add two players with requests $(s_i, v_1, 1)$ and $(s_i, v_2, 1)$. Additionally we add two players with requests $(v_1, t, 1)$ and $(v_2, t, 1)$. When players $1, 2$ choose their simple paths containing the edge (v_1, t), players $3, 4$ choose their simple paths containing the edge (v_2, t) and all other players use their unique path, then this is the optimal solution and it is also the best Nash equilibrium with cost $C(H, S_1) = 0$. Additional Nash equilibrium is obtained when players $1, 3$ choose their simple paths containing the edge (v_1, t), players $2, 4$ choose their simple path containing the edge (v_2, t) and all other players use their unique path. The cost of this Nash equilibrium $C(H, S_2) > 0$.

References

1. B. Awerbuch, Y. Azar, and A. Epstein. The price of routing unsplittable flow. In *Proc. 37th ACM Symp. on Theory of Computing*, 2005. To appear.
2. M. Beckmann, C.B. McGuire, and C.B. Winsten. *Studies in Economics of Transportation.* Yale University Press, 1956.
3. D. Braess. Uber ein paradoxon der verkehrsplanung. *Unternehmensforschung*, 12:258–268, 1968.
4. G. Christodoulou and E. Koutsoupias. The price of anarchy of finite congestion games. In *Proc. 37th ACM Symp. on Theory of Computing*, 2005. To appear.
5. A. Czumaj, P. Krysta, and B. Vöcking. Selfish traffic allocation for server farms. In *Proc. 34th ACM Symp. on Theory of Computing*, pages 287–296, 2002.
6. A. Czumaj and B. Vöcking. Tight bounds for worst-case equilibria. In *Proc. 13rd ACM-SIAM Symp. on Discrete Algorithms*, pages 413–420, 2002.
7. P. Dubey. Inefficiency of nash equilibria. *Mathematics of Operations Research*, 11(1):1–8, 1986.
8. S. Fortune, J.E. Hopcroft, and J.C. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.
9. D. Fotakis, S. Kontogiannis, and P. Spirakis. Selfish unsplittable flows. In *International Colloquium on Automata, Languages and Programming - ICALP '04*, 2004.
10. E. Koutsoupias and C.H. Papadimitriou. Worst-case equilibria. In *Proc. 16th Symp. on Theoretical Aspects of Comp. Science*, pages 404–413, 1999.
11. M. Mavronicolas and P. Spirakis. The price of selfish routing. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 510–519, 2001.
12. J.D. Murchland. Braess's paradox of traffic flow. *Transportation Research*, 4:391–394, 1970.
13. M.R. Garey nad D.S. Johnson. *Computers and Intractabilty: A Guide to the Theory of NP-Completeness.* Freeman, 1979.
14. J. F. Nash. Equilibrium points in n-person games. In *Proceedings of National Academy of Sciences*, volume 36, pages 48–49, 1950.
15. G. Owen. *Game Theory.* Academic Press, third edition, 1995.
16. C.H. Papadimitriou. Algorithms, games and the internet. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 749–753, 2001.
17. R. W. Rosenthal. A class of games possesing pure-strategy nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.
18. T. Roughgarden. Designing networks for selfish users is hard. In *Proc. 42nd IEEE Symp. on Found. of Comp. Science*, pages 472–481, 2001.
19. T. Roughgarden. The price of anarchy is independent of the network topology. In *Proc. 34th ACM Symp. on Theory of Computing*, pages 428–437, 2002.
20. T. Roughgarden. The maximum latency of selfish routing. In *Proc. 15rd ACM-SIAM Symp. on Discrete Algorithms*, pages 973–974, 2004.
21. T. Roughgarden and É. Tardos. How bad is selfish routing. In *Proc. 41st IEEE Symp. on Found. of Comp. Science*, pages 93–102, 2000.

Improved Approximation Algorithm for Convex Recoloring of Trees

Reuven Bar-Yehuda[1], Ido Feldman[1], and Dror Rawitz[2]

[1] Department of Computer Science, Technion, Haifa 32000, Israel
`{reuven, idofeld}@cs.technion.ac.il`
[2] Caesarea Rothschild Institute, University of Haifa, Haifa 31905, Israel
`rawitz@cri.haifa.ac.il`

Abstract. A pair (T, C) of a tree T and a coloring C is called a *colored tree*. Given a colored tree (T, C) any coloring C' of T is called a *recoloring* of T. Given a weight function on the vertices of the tree the *recoloring distance* of a recoloring is the total weight of recolored vertices. A coloring of a tree is *convex* if for any two vertices u and v that are colored by the same color c, every vertex on the path from u to v is also colored by c. In the *minimum convex recoloring problem* we are given a colored tree and a weight function and our goal is to find a convex recoloring of minimum recoloring distance.

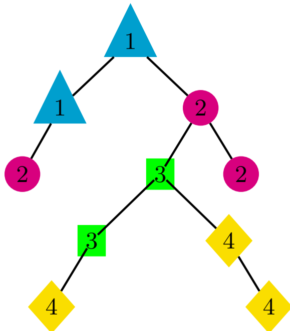
The minimum convex recoloring problem naturally arises in the context of *phylogenetic trees*. Given a set of related species the goal of phylogenetic reconstruction is to construct a tree that would best describe the evolution of this set of species. In this context a convex coloring correspond to *perfect phylogeny*. Since perfect phylogeny is not always possible the next best thing is to find a tree which is as close to convex as possible, or, in other words, a tree with minimum recoloring distance.

We present a $(2+\varepsilon)$-approximation algorithm for the minimum convex recoloring problem, whose running time is $O(n^2 + n(1/\varepsilon)^2 4^{1/\varepsilon})$. This result improves the previously known 3-approximation algorithm for this NP-hard problem.

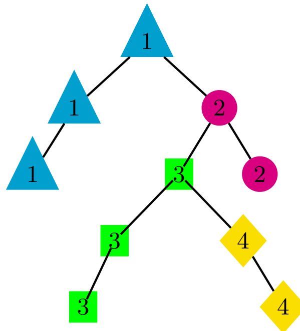
1 Introduction

Problem statement and motivation. Given a tree $T = (V, E)$ with n vertices, a *coloring* of the tree is a function $C : V \to \mathcal{C}$, where $\mathcal{C}$ is a set of colors. A pair (T, C) of a tree and a coloring is called a *colored tree*. A coloring C of a tree is *convex* if for every two vertices u and v such that $C(u) = C(v) = c$ the color of every vertex on the path from u to v is also c. That is, a coloring is convex if the set of vertices colored by c induces a (possibly empty) subtree for every color $c \in \mathcal{C}$. Examples of a non-convex coloring and a convex coloring are given in Fig. 1. Given a colored tree (T, C) any coloring C' of T is called a *recoloring* of T. A vertex u is *recolored* if $C(v) \neq C'(v)$. Given a non-negative weight function w on the vertices of T the *recoloring distance* of C' is the total weight of recolored vertices. For example, given the coloring in Fig. 1(a) and

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 55–68, 2006.



(a) Non-convex coloring



(b) Convex coloring

Fig. 1. Transforming a non convex coloring into a convex coloring

assuming unit weights, the recoloring cost of the coloring in Fig. 1(b) is 2. In the *minimum convex recoloring problem* we are given a colored tree (T, C) and a non-negative weight function w and our goal is to find a convex recoloring C' of minimum recoloring distance.

The minimum convex recoloring problem was first introduced by Moran and Snir [8], who showed that this problem arises in the context of *phylogenetic trees*. Given a set of related species the goal of phylogenetic reconstruction is to construct a tree that would best describe the evolution of this set of species. In such a phylogenetic tree the leaves represent the species, while internal vertices represent extinct species. A *character* is an attribute shared by the entire set of species represented by the tree. Such a character has different *states*, and each species is associated with one of these states. For example, the character may be as simple as the existence of wings, and the states are *wings*, and *no wings*. It is not hard to see that the states of a given character correspond to a set of colors, and that the states associated with the species correspond to a coloring of the tree. A natural biological constraint is that the tree does not contain *reverse* or *convergent* transitions with respect to every character. A reverse transition occurs when some species has a common character state with an old ancestor while its direct ancestor is associated with a different character state. In a convergent transition two species share a character state which is different from the character state of their least common ancestor. The absence of reverse and convergent transitions implies that the path in the tree connecting two species with some character state must contain only species with an identical character state. In other words, a character with respect to which there are no convergent and reverse transitions is a convex coloring of the tree. Hence, our goal is to construct a tree in which every character is a convex coloring. This problem is known as the *perfect phylogeny problem* (see, e.g., [1–4]).

Since perfect phylogeny is not always possible the next best thing is to find a tree which is as close to convex as possible with respect to each character. However, the meaning of *close to convex* must be defined first. One possible measure of closeness is the *parsimony score* which is the number of mutated edges summed over all characters [5, 6]. Another measure is the *phylogenetic number* [7] which is defined as the maximum number of connected components induced by a single state. In [8] Moran and Snir defined a natural distance from a phylogenetic tree to a convex one—the *recoloring distance*. We note that the parsimony score and the phylogenetic number do not specify a distance to an actual convex coloring of the given tree. Moreover, there are trees with large phylogenetic numbers and parsimony scores that can be made convex only by changing the color of one vertex, while other trees with small phylogenetic numbers that can become convex only by changing the color of a large number of vertices.

For more details about phylogenetic trees and other applications of the minimum convex recoloring problem we refer the reader to [8, 9].

Previous Results. The minimum convex recoloring problem was first defined by Moran and Snir [8]. They showed that the problem is NP-Hard even on strings (trees with two leaves) with unit weights. In addition, they presented dynamic programming based algorithm for computing an optimal convex recoloring of trees. The running time of this algorithm is $O(n \cdot n^* \cdot \Delta^{n^*+2})$, where n^* is the number of colors that violate convexity in the input tree, and Δ is the maximum degree of vertices in the tree. In a followup paper [9] Moran and Snir presented a 3-approximation algorithm based on the *local ratio technique* [10–14], and a 2-approximation algorithm for strings.

Our Results. We obtain a polynomial time $(2+\varepsilon)$-approximation algorithm for the minimum convex recoloring problem. Our algorithm depends on an accuracy parameter $k \geq 2$, and consists of two phases. The first phase is a local ratio algorithm in which we manipulate the weights such that the original weighted colored tree is transformed into a weighted colored tree we call k-simple. The approximation ratio of this phase is $2 + \frac{2}{k-1}$ and the running time is $O(n^2)$. In the second phase of the algorithm we use dynamic programming to compute an optimal solution. The running time of this phase is $O(n^2 + nk^2 2^k)$. For example, if we set $k = \log n/2 + 1$ we get a $(2 + 4/\log n)$-approximation algorithm whose time complexity is $O(n^2)$. In addition, our dynamic programming algorithm for computing an optimal convex recoloring (for general colored trees) is faster than the best previously known algorithm presented in [8], since the running time of our algorithm (in terms of n^* and Δ) is $O(n^2 + n \cdot n^* \cdot \Delta^{n^*})$.

Overview. The remainder of the paper is organized as follows. Section 2 contains most of our definitions and notation. Our dynamic programming algorithm is given in Sect. 3. In Sect. 4 we define k-simple trees and analyze the algorithm from the previous section on the special case of k-simple trees. In Sect. 5 we present our $(2 + \frac{2}{k-1})$-approximation algorithm.

2 Preliminaries

In this section we focus on two main issues. First, we define the notion of convex partial colorings, and show that it is sufficient look for a convex partial recoloring of a given colored tree. Next, we examine the form of an optimal solution.

2.1 Partial Colorings

A *partial coloring* of a tree T is function $C : V \to \mathcal{C} \cup \{\varnothing\}$, where $\varnothing$ stands for *no color*. That is, if $C(v) = \varnothing$ then v is assumed to be *uncolored*. A pair (T, C) of a tree and a partial coloring is called a *partially colored tree*. A partial coloring C is called convex if it can be extended to a (total) convex coloring.

Observation 1. *A convex partial coloring can be completed in $O(n^2)$ time.*

We consider an extended version of the minimum convex recoloring problem in which both the input and output colorings may be partial. That is, we are given a tree $T = (V, E)$, and a partial coloring C of the tree and our goal is to compute a convex partial recoloring C'. We say that a recoloring C' of C *discolors* a vertex u if $C(u) \neq \varnothing$ and $C'(u) \neq C(u)$. That is, C' discolors u if it changes or removes its original color. Given a non-negative weight function w on the vertices of T the *recoloring distance* of C' is the total weight of discolored vertices. (Informally, this means that we pay for removing a color, but not for applying a color.) Hence, we may assume without loss of generality that if $w(u) = 0$ then $C(u) = \varnothing$, and vice versa. Observe that since coloring uncolored vertices cost nothing, turning a convex partial coloring into a convex coloring incurs no cost.

Observation 2. *The weight of an optimal convex partial recoloring is equal to the weight of an optimal convex recoloring.*

Given a partially colored tree (T, C), a vertex set $X \subseteq V$ is called a *cover* if there is a convex partial recoloring C' such that X is the set of vertices that are discolored by C'. For a set of vertices $U \subseteq V$ and a weight function w we define $w(U) = \sum_{u \in U} w(u)$. Hence, the cost of a cover X is defined as $w(X)$, and the *recoloring distance* of a corresponding convex partial coloring C' is $w(C') = w(X)$.

2.2 Form of an Optimal Solution

Given a subset of vertices $U \subseteq V$ we denote the set of colors that are used to color U by $C(U)$, i.e., $C(U) = \{c \in \mathcal{C} : C(u) = c \text{ and } u \in U\}$. Notice that $C(u)$ does not include $\varnothing$. Given a subtree T' of T we denote the set of colors used in T' by $C(T')$, i.e., $C(T') = C(V(T'))$.

Given colored tree (T, C), a *color block* in T is a maximal set of vertices which induces a monochromatic subtree. A c-block is a color block colored by c. (For example, in Fig. 1(a) the tree contains two 2-blocks, and one 3-block.) If C is a convex coloring then for every color c there exists only one c-block. Moran and Snir [9] referred to a coloring C' as an *expanding* recoloring of C if in each block of C' at least one vertex v is not recolored, i.e., $C'(v) = C(v)$.

Observation 3 ([9]). *Let (T, C, w) be a weighted colored tree. Then there exists an expanding optimal convex recoloring of the tree.*

It follows that there exists an optimal convex (partial) recoloring C' that uses only colors that were originally used by C. Next, we show that there exists an optimal partial recoloring in which each vertex has a limited choice of colors, and a vertex colored by $\varnothing$ is not located on the path between two c-blocks for some color c.

Given a tree T we denote by $T \setminus v$ the set of subtree obtained when v is removed from T. Given a colored tree (T, C), we say that v *separates* c (with respect to C) if there are at least two subtrees in $T \setminus v$ that contain a vertex u such that $C(u) = c$. The *separation number* $\text{SEP}_v(c)$ of a vertex v with respect to a color $c \neq \varnothing$ is defined as the number of subtrees in $T \setminus v$ that contain a vertex u such that $C(u) = c$. Let $S(v)$ be the set of colors that are separated by v, i.e., let $S(v)$ be the set of colors for which the separation number is greater than 1. That is, $S(v) = \{c : \text{SEP}_v(c) > 1\}$. We define $\Sigma_v = |S(v)|$ and $\Pi_v = \prod_{c \in S(v)} \text{SEP}_v(c)$. (If $\Sigma_v = 0$ then $\Pi_v = 1$.) An example is given in Fig. 2.

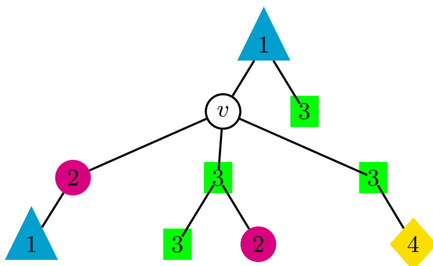


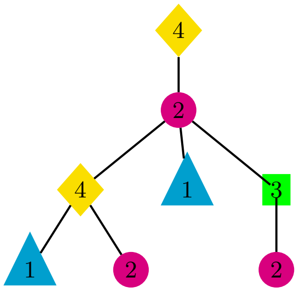
Fig. 2. $S(v) = \{1, 2, 3\}$, $\Sigma_v = 3$, and $\Pi_v = \text{SEP}_v(1) \cdot \text{SEP}_v(2) \cdot \text{SEP}_v(3) = 2 \cdot 2 \cdot 3 = 12$

Definition 1. *Given a colored tree* (T, C) *we define the* color set *of* v *by* $G(v) \triangleq S(v) \cup \{C(v), \varnothing\}$. *(Recall that* $C(v)$ *may be* $\varnothing$.*) A partial recoloring* C' *is called* good *if (1)* $C'(v) \in G(v)$ *for every* $v \in V$, *and (2) if* $C'(v) = \varnothing$, *then* v *does not separate any color* c *with respect to* C'.

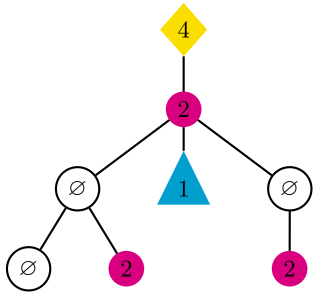
In the next lemma we show that there exists a good optimal convex partial recoloring. Hence, we can concentrate on finding good partial recolorings, and, in particular, it is enough to design an algorithm that computes an optimal good partial recoloring in order to solve the problem.

Lemma 1. *Let* (T, C, w) *be a weighted colored tree. Then there exists a good optimal convex partial recoloring* C'.

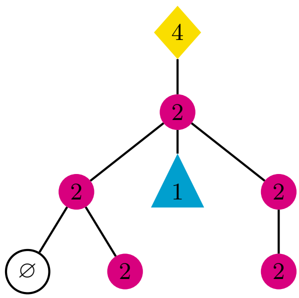
Proof. Let C' be an optimal convex recoloring, and let X be the corresponding cover. We construct a good optimal partial recoloring C'' that correspond to the same cover X. First, we set $C''(v) = C(v)$ for every $v \notin X$. Next, we discolor the vertices in X. Observe that, with respect to C', every $v \in X$ separates at most 1 color (since X is a cover). Hence, for every $v \in X$ that separates a color c, we define $C''(v) = C'(v) = c$, and otherwise, we define $C''(v) = \varnothing$. (See Fig. 3 for an illustration). Clearly, C'' is good. Moreover, since we only changed vertices in X, we get that $w(C'') = w(C')$. Also, C'' is convex, since it can be extended to C'. $\square$



(a) Non-convex coloring



(b) Convex partial recoloring



(c) Good convex partial recoloring

Fig. 3. Example of a good convex partial recoloring

Lemma 2. *Let $v \in V$ be a vertex and $T' \in T \setminus v$ be a subtree. If C' is a good partial recoloring, then $C'(T') \subseteq C(T') \cup \{\varnothing\}$.*

Proof. Consider a vertex u in T'. If $C'(u) = C(u)$ or $C'(u) = \varnothing$ we are done. Otherwise, since $C'(u) \neq C(u)$ we know that $u \in X$. It follows that u separates $C'(u)$, and thus $C'(u) \in C(T')$. □

3 Dynamic Programming Algorithm

In this section we describe a dynamic programming algorithm for computing an optimal convex partial recoloring whose running time is $O(n^2 + \sum_{v \in V} \Sigma_v \cdot (\deg(v) + \Pi_v))$, where $\deg(v)$ is the degree of the vertex v. This expression becomes polynomial in n and exponential in k for the special case of k-simple trees (defined in the next section).

Throughout this section we treat the input tree T as a rooted tree. This is done by choosing an arbitrary root s. Let $v \in V$ be a vertex with r children (i.e., $\deg(v) = r$). We denote the ith child of v by v_i, and the parent of v by v_0. As before the set of subtrees obtained by the removal of V is denoted by $T \setminus v$. We denote the subtree of the ith child by $T_i(v)$, and by $T(v)$ the tree rooted at v.

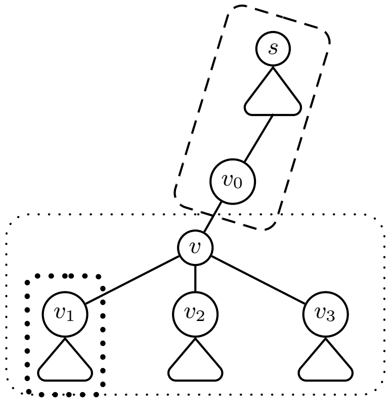


Fig. 4. $T_0(v)$ is marked by the dashed line, $T_1(v)$ is marked by the thick dotted line, and $T(v)$ is marked by the thin dotted line

We also denote by $T_0(v)$ the subtree obtained by removing $T(v)$ from T. (If the tree was unrooted then $T_0(v)$ would be the subtree $T_0(v)$ of the parent v_0.) See Fig. 4.

Let C' be a convex good partial recoloring of T, and consider a vertex v that is colored by c. If $c \neq \varnothing$ then C' induces a partition of $\mathcal{C} \setminus \{c\}$. A color $d \neq c$ that is used in $T_i(v)$ cannot be used in $T_j(v)$ where $i \neq j$. If $c = \varnothing$ then C' induces a partition on $\mathcal{C}$. In both cases, v partitions the color set into $r + 1$ mutually disjoint color sets. If $c \neq \varnothing$ then c may be used to color vertices in more than one subtree from $T \setminus v$. Obviously, the use of this color is possible only if the vertices colored by it form a subtree.

Definition 2. *Let (T, C) be a colored tree, let c be a color, and let $v \in V$ be a vertex with r children. We say that $(D_0, \ldots, D_r)$ is a* good partition with respect to v and c *if $D_i \subseteq C(T_i(v))$ for $i \in \{0, \ldots, r\}$, and $(D_0, \ldots, D_r)$ is a partition of $\mathcal{C} \setminus \{c\}$ ($\mathcal{C}$ when $c = \varnothing$).*
$\textsc{good}(v, c)$ *denotes the set of all good partitions with respect to v and c.*

Observation 4. *Let v be a vertex, and let c be a color. Then, $|\textsc{good}(v, c)| \leq \Pi_v$.*

Let v be a vertex, and let c be a color. Also, let C be a coloring that is consitent with some good partition $(D_0, \ldots, D_r)$ with respect to v and c. Then, if v is colored by c, the colors in D_0 cannot be used in $T(v)$. We refer to these colors as *forbidden* with respect to v.

Definition 3. *Let $v \in V$ be a vertex. Define,*

$$\textsc{forb}(v) = \{D : \exists c \in G(v)\ \exists (D_1, \ldots, D_r),\ (D, D_1, \ldots, D_r) \in \textsc{good}(v, c)\}$$

Thus, $D \in \textsc{forb}(v)$ if there is a good partition, where the color set D is used only in $T_0(v)$.

Lemma 3. $|\text{FORB}(v)| \leq 2^{\Sigma_v}$ *for every* $v \in V$.

Proof. Let v be a vertex with r children, let c be a color, and let i and j be indices such that $0 \leq i < j \leq r$. Observe that if $c \in C(T_i(v))$ and $c \in C(T_j(v))$, then v separates c. Thus, if v does not separate a color c' then either $c' \in D_0$ for every $(D_0, \ldots, D_r) \in \text{GOOD}(v, c)$, or $c' \notin D_0$ for every $(D_0, \ldots, D_r) \in \text{GOOD}(v, c)$. Therefore, $|\text{FORB}(v)| \leq 2^{|\{c' : v \text{ separates } c'\}|} = 2^{\Sigma_v}$. □

We now turn to design a dynamic programming algorithm for computing the optimal good convex partial recoloring of a given colored tree. We construct an optimal solution bottom-up, by storing intermediate values on the vertices of the tree.

Definition 4. *Let* v *be a vertex in* T*, let* $c \in \mathcal{C} \cup \{\varnothing\}$*, and let* $D \in \text{FORB}(v)$.

- *A good convex recoloring* C' *of* $T(v)$ *is a* (v, D)-coloring *if it is a recoloring in which the colors in* D *are not used to color* $T(v)$*, i.e., it is a recoloring of* $T(v)$ *such that* $C'(T(v)) \cap D = \emptyset$. $\text{OPT}(v, D)$ *denotes the weight of an optimal* (v, D)*-coloring.*
- *A good convex recoloring* C' *of* $T(v)$ *is a* (v, c, D)-coloring *if it is a recoloring in which the colors in* D *are not used to color* $T(v)$ *and* v *is colored by* c*, i.e., it is a* (v, D)*-coloring of* $T(v)$ *such that* $C'(v) = c$. $\text{OPT}(v, c, D)$ *denotes the weight of an optimal* (v, c, D)*-coloring.*

Observe that, for a tree T with root s, the weight of an optimal recoloring of the whole tree is $\text{OPT}(s, \emptyset)$.

We compute OPT recursively using the following rules:

1. $R(v, D) = \min_{c \in G(v) \setminus D} R(v, c, D)$.
2. If $C(v) = c$ then

$$R(v, c, D) = \min_{(D, D_1, \ldots, D_r) \in \text{GOOD}(v,c)} \left\{ \sum_{i=1}^{r} R'(v_i, c, \mathcal{C} \setminus D_i) \right\}$$

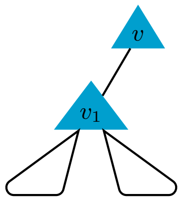
otherwise

$$R(v, c, D) = w(v) + \min_{(D, D_1, \ldots, D_r) \in \text{GOOD}(v,c)} \left\{ \sum_{i=1}^{r} R'(v_i, c, \mathcal{C} \setminus D_i) \right\}$$

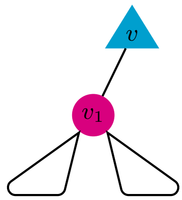
where $R'(v, c, D) \triangleq \min \{R(v, D \cup \{c\}), R(v, c, D)\}$.

where v is a vertex with r children, $D \in \text{FORB}(v)$, and and $c \in G(v) \setminus D$. (Recall that v_i is the ith child of v for every $i \in \{1, \ldots, r\}$.)

In Rule 1 we go through all (v, c, D)-colorings and find the best one that colors the subtree $T(v)$. In Rule 2 we try to glue colorings of the subtrees $T_1(v), \ldots, T_r(v)$ to a coloring of v. Notice that, for every $i \in \{1, \ldots, r\}$, if v is colored by c then either c is not used in $T_i(v)$, or it is used to color v_i. (See Fig. 5.)



(a) Case 1: $R(v_1, c, D)$



(b) Case 2: $R(v_1, d, D \cup \{d\})$

Fig. 5. Combining a recoloring of $T_1(v)$ with a recoloring of v

Theorem 1. *Let v be a vertex with r children, let $D \in \textsc{forb}(v)$ be a set of colors, and let $c \in G(v) \setminus D$. Then, $R(v, c, D) = \textsc{opt}(v, c, D)$, and $R(v, D) = \textsc{opt}(v, D)$.*

Proof. We proof the theorem by induction on the tree. In the induction base v is a leaf, and in this case

$$R(v, c, D) = \begin{cases} w(v) & C(v) \neq c, \\ 0 & \text{otherwise}, \end{cases}$$

and

$$R(v, D) = \begin{cases} w(v) & C(v) \in D \\ 0 & \text{otherwise}, \end{cases}$$

as required.

Next, for the inductive step we assume that $R(v_i, D) = \textsc{opt}(v_i, D)$ and $R(v_i, c, D) = \textsc{opt}(v_i, c, D)$ for every $i \in \{1, \ldots, r\}$. We first prove that $R(v, c, D) \leq \textsc{opt}(v, c, D)$. Let C' be an optimal (v, c, D)-coloring. Let D_i be the set of colors which C' uses in coloring $T_i(v)$. That is, $D'_i = C'(T_i(v)) \setminus \{c\}$ for every $i \in \{0, \ldots, r\}$. Since C' is convex and $C'(v) = c$, we must have that for $i \neq j$, $D_i \cap D_j = \emptyset$. By Lemma 2, it follows that $D_i \subseteq C(T_i(v)) \setminus \{c\}$ for every i. Thus, $\bigcup_{i=0}^r D_i \subseteq \bigcup_{i=0}^r C'(T_i(v)) \setminus \{c\}$. If $\bigcup_{i=0}^r D_i \subsetneq \bigcup_{i=0}^r C'(T_i(v)) \setminus \{c\}$ we include each missing color d in an arbitrary set D_i satisfying $d \in C'(T_i(v))$. Hence, $(D, D_1, \ldots, D_r) \in \textsc{good}(v, c)$. Furthermore, the recoloring of $T_i(v)$ that is induced by C' is a good recoloring of $T_i(v)$ for every $i \in \{1, \ldots, r\}$. In addition this recoloring of $T_i(v)$ does not use colors from $\mathcal{C} \setminus (D_i \cup \{c\})$, and if it uses c then $C'(v_i) = c$. Therefore, $R'(v_i, c, \mathcal{C} \setminus D_i) \leq w(T_i(v))$ for every $i \in \{1, \ldots, r\}$, and it follows that $R(v, c, D) \leq \textsc{opt}(v, c, D)$.

Next we show that $\textsc{opt}(v, c, D) \leq R(v, c, D)$. Let $(D, D_1, \ldots, D_r)$ be a good partition which minimizes the RHS of Rule 2. Let C'_i be the recoloring of $T_i(v)$ whose weight is $R'(v_i, c, \mathcal{C} \setminus D_i)$ for $i \in \{1, \ldots, r\}$. We obtain a recoloring of $T(v)$ as follows: $C'(v) = c$ and $C'(u) = C'_i(u)$ for every u that belong to $T_i(v)$. By its construction C' is a good convex recoloring of $T(v)$ that does not use colors from D and such that $C'(v) = c$. Hence, there exists a good convex recoloring C' of $T(v)$ whose weight is $R(v, c, D)$. Therefore, $\textsc{opt}(C, c, v) \leq R(v, c, D)$.

It remains to show that:

$$\textsc{opt}(v, D) = \min_{c \in G(v) \setminus D} \textsc{opt}(v, c, D) = \min_{c \in G(v) \setminus D} R(v, c, D) = R(v, D)$$

and we are done. □

The number of entries computed by the dynamic programming algorithm is: $O(\sum_{v \in V} |G(v)| \cdot |\textsc{forb}(v)|)$. An additional $O(nc)$ space is needed for storing $G(v)$ for every vertex v. Hence, the space complexity of the algorithm is $O(nc + \sum_{v \in V} \Sigma_v \cdot 2^{\Sigma_v})$.

For each entry of the form $R(v, D)$ the running time is $O(|G(v)|)$. For each entry of the form $R(v, c, D)$ we need to go through all possibilities of good partitions of the form $(D, D_1, \ldots, D_r)$, and for each such good partition we perform $O(\deg(v))$ operations. Observe that for every v and c we actually go through every good partition in $\textsc{good}(v, c)$ for computing the values of $R(v, c, D)$ for all $D \in \textsc{forb}(v)$. Since every good partition is visited exactly once, we invest $O(|\textsc{good}(v, c)| \cdot \deg(v))$ operations for every pair of vertex v and color c. Hence, by Observation 4, this brings us to $O(\sum_{v \in V} \Sigma_v \cdot \Pi_v \cdot \deg(v))$. An additional $O(n^2)$ is needed to computing $G(v)$ for every v. Hence, the total running time is $O(n^2 + \sum_{v \in V} \Sigma_v \cdot \Pi_v \cdot \deg(v))$.

Note that the computation of $R(v, c, D)$ can be modified also to output a corresponding solution. This can be done by keeping track of which option was taken in both rules. Afterwards we can reconstruct the optimal recoloring in a top down manner.

Next, we explain how to improve the running time of the algorithm. Let v be a vertex v with r children, and c a color. Also, let $\mathcal{D} = (D_0, \ldots, D_r), \mathcal{D}' = (D'_0, \ldots, D'_r) \in \textsc{good}(v, c)$. We define $\Delta(\mathcal{D}, \mathcal{D}') \triangleq \{i : D_i \neq D'_i\}$. We also define $R(v, c, \mathcal{D}) = \sum_i R'(v_i, c, \mathcal{C} \setminus D_i)$, and $w(v, c, \mathcal{D}, \mathcal{D}') \triangleq R(v, c, \mathcal{D}) - R(v, c, \mathcal{D}')$. If we know the value of $R(v, c, \mathcal{D})$ and $|\Delta(\mathcal{D}, \mathcal{D}')| = 2$ we can compute $R(v, c, \mathcal{D}')$ by using the equation $R(v, c, \mathcal{D}) = R(v, c, \mathcal{D}') + w(v, c, \mathcal{D}, \mathcal{D}')$ in $O(1)$ time.

For a vertex $v \in V$, and a color c we say that an ordering $\mathcal{D}_1, \ldots, \mathcal{D}_{|\textsc{good}(v,c)|}$ of the set $\textsc{good}(v, c)$ is a *close order* if $\Delta(\mathcal{D}_i, \mathcal{D}_{i+1}) = 2$ for every i. We now describe how to construct a close order of $\textsc{good}(v, c)$ for any $v \in V$ with r children and a color c. A good partition $(D_0, ..., D_r) \in \textsc{good}(v, c)$ can be described by a word $\sigma_1 \cdots \sigma_m$ such that $\sigma_i = j$ if $c_i \in D_j$. (Notice that $\sigma_i = j$ is possible only if $c_i \in C(T_j(v))$.) An order of this set of words in which the hamming distance between every two consecutive words is 1 defines a close order on $\textsc{good}(v, c)$. Such a close order can be obtained using the description in [15].

We examine the time complexity for computing the value of $R(v, c, D)$ for every vertex v, $D \in \textsc{forb}(v)$, and $c \in G(v) \setminus D$. Computing the value corresponding to the first member in the close order of $\textsc{good}(v, c)$ takes $O(\deg(v))$ time. For any other member the computation takes $O(1)$. Therefore, the time complexity for a vertex v is

$$O(\sum_{c \in G(v)} (\deg(v) + |\textsc{good}(v, c)|)) = O(\sum_{c \in G(v)} (\deg(v) + \Pi_v)) \ .$$

It follows that the total running time is: $O(n^2 + \sum_{v \in V} \Sigma_v \cdot (\deg(v) + \Pi_v))$.

4 Simple Trees

In this section we define the notion of a *k-simple* tree. Then, we show that the running time of the dynamic programming algorithm from the previous section amounts to $O(n^2 + n \cdot k^2 2^k)$ in the special case of k-simple trees.

Let (T, C) be a colored tree and $u \in V$ be a vertex. We say that u is a (t, d)-*separator* if there are t different colors $c_1, \ldots, c_t$ such that for $1 \leq i \leq t$, $\text{SEP}_u(c_i) \geq d$. Observe that in this case $\Pi_v \geq d^t$. Also, notice that if v is a (t, d)-separator with $r \geq t$ children, then for every $c_i \in \{c_1, \ldots, c_t\}$ there are d vertices $u_1^i, \ldots, u_d^i$ on d different components of $T \setminus v$ such that $C(u_j^i) = c_i$ and $w(u_j^i) > 0$ for every $1 \leq j \leq d$. We refer to such set of $t \cdot d$ vertices as a (t, d)-*separating witness of* v.

Definition 5. *Let* (T, C) *be a colored tree, and define*

$$SEP \triangleq \{(2, k), (3, 4), (4, 3), (k, 2)\}$$

where $k \geq 2$. *We say that the colored tree is* k-simple *if* v *is not a* (t, d)*-separator for every* $v \in V$ *and* $(t, d) \in SEP$.

Observation 5. *Let* (T, C) *be* k*-simple for* $k \geq 2$. *Then,* $\Sigma_v < k$ *for every* $v \in V$.

Consider a vertex v in a k-simple tree. Since $\Sigma_v < k$, v separates at most $k - 1$ colors, and therefore $|G(v)| \leq k + 1$.

Lemma 4. *Let* (T, C) *be* k*-simple for* $k \geq 2$. *Then,* $\Pi_v = O(\deg(v) \cdot k \cdot 2^k)$ *for every* $v \in V$.

Proof. Let $\mathcal{C} = \{c_1, \ldots, c_m\}$, and consider a vertex $v \in V$. Without loss of generality we assume that $\text{SEP}_v(c_i) \geq \text{SEP}_v(c_{i+1})$ for every i. We show that the following conditions hold: (1) $\text{SEP}_v(c_1) \leq \deg(v)$, (2) $\text{SEP}_v(c_2) \leq k - 1$, (3) $\text{SEP}_v(c_3) \leq 3$, (4) $\text{SEP}_v(c_4) \leq 2$, and (5) $\text{SEP}_v(c_i) \leq 1$ for every $i \geq k$. Hence, $\Pi_v \leq \deg(v) \cdot (k-1) \cdot 3 \cdot 2^{\Sigma_v - 2} = O(\deg(v) \cdot k \cdot 2^k)$.

First, $\text{SEP}_u(c_1) \leq |T \setminus u| = \deg(v)$. Also, if $\text{SEP}_v(c_2) \geq k$ then u is a $(2, k)$-separator; if $\text{SEP}_v(c_3) \geq 4$ then v is a $(3, 4)$-separator; if $\text{SEP}_v(c_4) \geq 3$ then v is a $(4, 3)$-separator; and if $\text{SEP}_v(c_k) \geq 2$ then v is a $(k, 2)$-separator. All in contradiction to the fact that (T, C) is k-simple. $\square$

Now we analyze the running time of the dynamic programming algorithm for the special case of k-simple trees. Since $\Sigma_v < k$ and $\Pi_v = O(\deg(v) \cdot k 2^k)$ for every v, the total running time is

$$O(n^2 + \sum_{v \in V} k \cdot \deg(v) \cdot k 2^k) = O(n^2 + n \cdot k^2 2^k) .$$

The number of triplets of the form (v, c, D) computed by the dynamic programming algorithm is $O(nc + \sum_{v \in V} k \cdot 2^k) = O(nc + n \cdot k 2^k)$.

5 Local Ratio Algorithm

In this section we develop an algorithm that given a colored tree and an accuracy parameter k, computes a $(2+\frac{2}{k-1})$-approximate convex partial recoloring. The running time of the algorithm is $O(n^2 + n \cdot k^2 \cdot 2^k)$.

The algorithm consists of two phases. In the first phase we use the local ratio technique. We manipulate the weights such that the original weighted colored tree is transformed into a k-simple tree. The approximation ratio of this phase is $2+\frac{2}{k-1}$ and the running time is $O(n^2)$. In the second phase of the algorithm we use our dynamic programming algorithm to compute an optimal solution. We note that if we set $k = \frac{\log n}{2} + 1$ the approximation guarantee is $(2+\frac{4}{\log n})$ and the time complexity is $O(n^2)$.

The local ratio technique [10–13] is based on the Local Ratio Theorem, which applies to optimization problems of the following type. The input is a non-negative weight vector $w \in \mathbb{R}^n$ and a set of feasibility constraints $\mathcal{F}$. The problem is to find a solution vector $x \in \mathbb{R}^n$ that minimizes (or maximizes) the inner product $w \cdot x$ subject to the constraints $\mathcal{F}$.

Theorem 2 (Local Ratio [12]). *Let $\mathcal{F}$ be a set of constraints and let w, w_1, and w_2 be weight vectors such that $w = w_1 + w_2$. Then, if x is r-approximate both with respect to $(\mathcal{F}, w_1)$ and with respect to $(\mathcal{F}, w_2)$, for some r, then x is also an r-approximate solution with respect to $(\mathcal{F}, w)$.*

Algorithm **CR-LR** is our local ratio approximation algorithm. It uses our dynamic programming algorithm which is referred to as Algorithm **CR-DP**. Apart from a weighted colored tree, the input to our algorithm includes an accuracy parameter k. As we shall see this algorithm computes $(2+\frac{2}{k-1})$-approximate solutions, and its running time is polynomial is n and exponential in k.

We first analyze the time complexity of the algorithm. Observe that given a vertex v, checking whether v is a (t, d)-separator, where $(t, d) \in \text{SEP}$, can be done in linear time. Since in each weight subtraction the weight of at least one vertex becomes zero, after no more than n subtraction there are no (t, d)-separators left in the given tree. Hence, the local ratio phase of the algorithm

---

Algorithm 1 : **CR-LR**(T, C, w, k)

---

if (T, C) is k-simple **then**
  Return **CR-DP**(T, w)
else
  Find $v \in V$ and $(t, d) \in \text{SEP}$ such that v is a (t, d)-separator
  Find a (t, d)-separating witness U of v
  Let $\varepsilon = \min_{u \in U} w(u)$
  Define $w_1(u) = \begin{cases} \varepsilon & u \in U, \\ 0 & \text{otherwise} \end{cases}$
  Return **CR-LR**$(T, w - w_1)$
end if

---

can be implemented to run in $O(n^2)$ time. Moreover, since the input to the dynamic programming algorithm is a k-simple tree, the total running time is $O(n^2 + n \cdot k^2 \cdot 2^k)$.

The computed solution is feasible since we use our dynamic programming algorithm the solution returned is feasible. It remains to show that the solution returned is $(2 + \frac{2}{k-1})$-approximate. We prove this by induction on the recursion. At the recursive base the solution returned is optimal, since it is computed by the dynamic programming algorithm. For the inductive step, we assume that the solution returned by the recursive call is $(2+\frac{2}{k-1})$-approximate with respect to $w - w_1$. We show that every solution is $(2 + \frac{2}{k-1})$-approximate with respect to w_1. Thus, by the Local Ratio Theorem the solution is $(2 + \frac{2}{k-1})$-approximate with respect to w as well.

Lemma 5. *Every convex partial recoloring is* $(2+\frac{2}{k-1})$*-approximate with respect to* w_1.

Proof. Obviously, there are four possible types of w_1 that correspond to the four members of SEP. Let v be a (t, d)-separator, where $(t, d) \in$ SEP, and let U be the (t, d)-separating witness. Consider a cover X that corresponds to a partial convex recoloring C'. It is not hard to see that $w_1(X) \leq \varepsilon \cdot td$.

On the other hand, in a convex partial recoloring, for every $v \in V$ there is at most one color $c \in \mathcal{C}$ such that v separates c. Therefore, at least $(t - 1)(d - 1)$ vertices in U must be recolored. Thus, $w(X) \geq \varepsilon \cdot (t-1)(d-1)$. It follows that the weight of every cover X is within a factor of $\frac{td}{(t-1)(d-1)}$ from the optimum with respect to w_1. The lemma follows, since for $(2, k)$ and $(k, 2)$ we get $\frac{td}{(t-1)(d-1)} = \frac{2k}{k-1} = 2 + \frac{2}{k-1}$ and for $(3, 4)$ and $(4, 3)$ we get $\frac{td}{(t-1)(d-1)} = 2$. □

Acknowledgment. We thank the anonymous referees for their helpful comments and suggestions.

References

1. Bodlaender, H.L., Fellows, M.R., Warnow, T.J.: Two strikes against perfect phylogeny. In: 19th International Colloquium on Automata, Languages, and Programming. Volume 623 of LNCS., Springer (1992) 273–283
2. Gusfield, D.: Efficient algorithms for inferring evolutionary trees. Networks **21** (1991) 19–28
3. Kannan, S., Warnow, T.: Inferring evolutionary history from DNA sequences. SIAM Journal on Computing **23** (1994) 713–737
4. Kannan, S., Warnow, T.: A fast algorithm for the computation and enumeration of perfect phylogenies. SIAM Journal on Computing **26** (1997) 1749–1763
5. Semple, C., Steel, M.: Phylogenetics. Volume 22 of Mathematics and its Applications series. Oxford University Press (2003)
6. Sankoff, D.: Minimal mutation trees of sequences. SIAM Journal on Applied Mathematics **28** (1975) 35–42

7. Goldberg, L.A., Goldberg, P.W., Phillips, C.A., Sweedyk, E., Warnow, T.: Minimizing phylogenetic number to find good evolutionary trees. Discrete Applied Mathematics **71** (1996) 111–136
8. Moran, S., Snir, S.: Convex recoloring of trees and strings: definitions, hardness results, and algorithms. In: 9th Workshop on Algorithms and Data Structures. (2005) To appear.
9. Moran, S., Snir, S.: Efficient approximation of convex recoloring. In: 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems. (2005) To appear.
10. Bar-Yehuda, R., Even, S.: A local-ratio theorem for approximating the weighted vertex cover problem. Annals of Discrete Mathematics **25** (1985) 27–46
11. Bafna, V., Berman, P., Fujito, T.: A 2-approximation algorithm for the undirected feedback vertex set problem. SIAM Journal on Discrete Mathematics **12** (1999) 289–297
12. Bar-Yehuda, R.: One for the price of two: A unified approach for approximating covering problems. Algorithmica **27** (2000) 131–144
13. Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., Schieber, B.: A unified approach to approximating resource allocation and scheduling. Journal of the ACM **48** (2001) 1069–1090
14. Bar-Yehuda, R., Bendel, K., Freund, A., Rawitz, D.: Local ratio: a unified framework for approximation algorithms. ACM Computing Surveys **36** (2004) 422–463
15. Er, M.: On generating the n-ary reflected gray codes. IEEE Transactions on Computers **33** (1984) 739–741

Exploiting Locality: Approximating Sorting Buffers

Reuven Bar-Yehuda and Jonathan Laserson

Computer Science Department, Technion, Haifa 32000, Israel
{reuven, joni}@cs.technion.ac.il

Abstract. The Sorting Buffers problem is motivated by many applications in manufacturing processes and computer science, among them car-painting and file servers architecture. The input is a sequence of items of various types. All the items must be processed, one by one, by a service station. We are given a random-access sorting buffer with a limited capacity. Whenever a new item arrives it may be moved directly to the service station or stored in the buffer. Also, at any time items can be removed from the buffer and assigned to the service station. Our goal is to give the service station a sequence of items with minimum type transitions. We generalize the problem to allow items with different sizes and type transitions with different costs. We give a polynomial-time 9-approximation algorithm for the maximization variant of this problem, which improves the best previously known 20-approximation algorithm.

1 Introduction

In the sorting buffers problem, the input is a sequence of items of various types. All the items must be processed, one at a time, by a service station. When the service station processes two consecutive items of different types we say that there is a type transition. Type transitions are expensive, and the goal is to give the service station a sequence of items with as few type transitions as possible. To achieve this task we are given a random-access sorting buffer with a limited capacity. Whenever a new item arrives it may be moved directly to the service station or stored in the sorting buffer. Also, at any time items can be removed from the sorting buffer and then assigned to the service station. Thus, the service station processes a sequence of items which is a permutation of the input sequence. Using the sorting buffer, we need to rearrange the input sequence so that the number of type transitions is minimized, or equivalently (for the maximization variant), so that the number of items which are followed by an item of the same type is maximized.

The sorting buffers problem is motivated by many applications in manufacturing processes. For example, during the manufacturing process in a car plant (e.g. the Daimler-Benz car plant in Germany), the cars arrive one after the other, from an assembly-line, to the painting center where each car is painted with its own top coat. If two consecutive cars are to be painted in different colors, a color

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 69–81, 2006.

change is required. Since each such color change causes a waste of paint and requires cleaning chemicals, it makes sense to rearrange the sequence of cars in a way that cars of the same color preferably appear in consecutive positions. For this purpose, a small garage with a limited capacity is built before the painting center, such that cars can be transferred from the assembly line to the garage, and later from the garage to the painting center. The garage acts as a sorting buffer and is used to deliver larger subsequences of cars of the same color.

This problem has also many application in computer science. For example, a file server receives a sequence of read/write requests to files stored on its disk. In addition to the time it takes to read or write the data to a file, more time is wasted by locating the file, opening it and closing it after the request is handled. One can minimize this overhead time by using a sorting buffer to group requests for the same file together and have them handled in sequence. In a similar way, this technique can be implemented in communication networks to group requests which deal with the same server and save the startup cost.

Another application is in computer graphics. During the process of polygon rendering, a set of polygons is processed one by one. A change of attributes in two consecutive polygons is denoted as state-change. As the number of state-changes decreases, the performance improves. By rearranging the sequence of polygons such that polygons with similar attributes are processed consecutively, one can effectively boost performance. In this case also, a sorting buffer can come in handy.

1.1 Our Contribution

We present a polynomial time 9-approximation algorithm for the maximization variant of the sorting buffers problem. This result improves the best previously known 20-approximation algorithm, obtained in [1]. The algorithm we introduce is also applicable to a generalized variant of the problem, in which each item is assigned a size and a nonnegative profit. We gain the profit assigned to an item if at the service station it is followed by another item of the same type (see formal definition in Problem 3). The goal is to gain maximum profit. The generalized problem becomes the original maximization problem if all the profits are equal.

We prove some combinatorial lemmas about the optimal solutions for this problem, and use the Local-Ratio Technique [3] [4] to obtain a polynomial-time 9-approximation algorithm for the generalized problem. This result can be easily converted to a simple solution in the primal-dual schema [5].

1.2 Previous Work

The first constant-approximation algorithm for the sorting buffers problem was given by Kohrt and Pruhs [1]. They gave a 20-approximation algorithm for the maximization variant of the problem. Their algorithm also uses the local-ratio technique. Kohrt et al. also noted that the problem can be solved exactly in polynomial time if either the number of types or the buffer size is constant.

The best approximation result known for the minimization problem is actually an on-line algorithm with a competitive ratio of $O(\log^2 k)$, where k is the size of the buffer. Räcke et al. [2] gave a deterministic bounded-waste strategy which achieved this result.

A related problem is studied by Epping and Hochstättler in [7]. In this problem, r queues are used to rearrange the items instead of a random-access sorting-buffer. Epping et al. show equivalence between their problem and the multiple sequence alignment problem known from molecular biology. They provide a dynamic programming algorithm which solves their problem exactly.

Another related problem is the bandwidth-allocation problem, which is studied in [6]. The input is a set of intervals, each with a width and a profit. The goal is to choose a subset of these intervals with maximum total profit such that at any point t, the total width of the intervals intersecting t is not larger than 1. Bar-Noy et al. were able to achieve a 5-approximation algorithm for this NP-hard problem. We will show later that the generalized maximization problem for sorting buffers is also a generalization of the bandwidth-allocation problem, and hence the generalized maximization problem is also NP-hard.

2 Preliminaries

The rest of this paper is organized as follows. In Section 2 we give a formal description of the problem, and make some observations on optimal solutions. These observations allow us to represent the problem differently, as a maximization problem. We also make some observations on a subclass of feasible solutions denoted as "good" and show how to turn any feasible solution to a good one. In Section 3 we generalize the problem by adding a profit function, and introduce the local-ratio schema which will be used on the generalized problem. In Section 4 we provide the rest of the details necessary for applying the schema, and obtain our approximation algorithm.

2.1 The Model

The input is a sequence of items $\sigma = \sigma_1, \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_n$ which are only characterized by a specific attribute. To simplify things, we will assume that the items are packages, and that they are characterized by color. The input sequence is processed from left to right by a *sorting buffer* which is a random access buffer with storage capacity for k packages. During this process, packages may be stored in the buffer and later they are placed back into the sequence. The resulting sequence is the *output sequence* (this is the sequence given to the service station).

We can formalize the rearrangement process as follows. The process consists of n steps, where at step i ($i = 1, 2, \ldots, n$) at most one of these actions occur:

1. Any subset of the packages currently in the sorting buffer may be removed from the buffer and placed back in the sequence (right after σ_i), in any order.
2. If space permits, σ_i may be removed from the sequence and stored in the sorting buffer.

We assume that the sorting buffer is initially empty, and at the end of the process the buffer has to be empty again. Intuitively, we can picture the buffer as a truck which makes one pass along a line of packages, when the packages are occasionally loaded on and off the truck along the way.

The goal is to rearrange the input sequence in a way that packages with the same color preferably appear at consecutive positions in the output sequence. Let each maximal subsequence of packages of the same color be denoted as *color block*. Between two different color blocks there is a *color change*. Then, the goal is to minimize the number of color changes in the output sequence.

Problem 1 (Minimum Color Changes). Given a sequence of packages σ, rearrange it using a sorting buffer of capacity k to minimize the number of color changes in the output sequence.

A *solution* S to the above problem is a rearrangement of σ. Let the integer $drop_S(\sigma_i)$ denote the rearrangement step of S on which σ_i was removed from the buffer, where $drop_S(\sigma_i) = i$ if σ_i was not stored in the buffer at all. We denote by $B_S(j)$ the set of packages which are in the buffer at the beginning of step j of S.

2.2 Observations About the Optimal Solution

As noted in [2] and in [1], the following two lemmas hold for any input sequence:

Lemma 1. *If two packages of the same color are adjacent in the input sequence, then there is an optimal solution where these two packages are adjacent in the output sequence.*

Lemma 2. *For any optimal solution we may assume that for any color, the order of the packages of this color in the input sequence is preserved in the output sequence.*

Lemma 1 allows us to consider any color block in the input sequence as one big package. In other words, we can now replace every color block of t packages with one package of the same color, and assign that package a *size* of t. Having said that, we can now assume that the input sequence has no adjacent packages of the same color. Furthermore, we can scale the sizes with respect to the sorting buffer capacity, i.e. the buffer will have capacity 1 instead of k, and each package will have a size of $\frac{t}{k}$ instead of t. We will denote by $Size(\sigma_i)$ the size of package σ_i, and for any set of packages A, we will denote by $Size(A)$ the total size of the packages in A.

Now we turn to look at the maximization variant of the problem. If we have to pay one dollar for every color change in the output sequence, then we save a dollar whenever there are two adjacent packages in the output sequence which share the same color. According to Lemma 2, it suffices to consider only dollars saved by these adjacent packages which preserve their order from the input sequence. Each such pair of packages is called a *color-saving*. The number of color changes is minimized when the number of dollars we save is maximized, i.e. when we make the maximum number of color-savings.

Problem 2 (Maximum Color-Savings). Given a sequence of packages in different colors and sizes with no two adjacent packages of the same color, rearrange it using a sorting buffer of capacity 1 to maximize the number of color-savings in the output.

Problems 1 and 2 are equivalent because we can restrict ourselves to schedules which comply with the assumptions of Lemma 1 and Lemma 2. However, a constant approximation algorithm to the maximization problem is probably not a constant approximation algorithm to the minimization problem, and while we give a constant approximation algorithm for Problem 2, such algorithm for Problem 1 is not known.

We now extend our notation and given $\sigma = \sigma_1, \sigma_2, \dots \sigma_n$ we use r_i to denote the ith package with color r in σ and $\overline{r_i}$ to denote the index of that package in σ (i.e. $r_i = \sigma_{\overline{r_i}}$). For each color r and index i we call $r_i - r_{i+1}$ a *pair* and we say that r_i is the *first package* of the pair and r_{i+1} the *last package* of the pair. If in the output sequence of a solution S, r_{i+1} appears adjacent to the right of r_i we say that the pair $r_i - r_{i+1}$ is a *color-saving* in S.

As an example of the problem and the notation we adopt, consider the following. The input sequence is $a_1b_1c_1a_2c_2b_2c_3a_3$ (the letters denote colors and the indexes distinguish between packages of the same color). There are 8 packages in the sequence. Assume all the packages have the same size, and that the buffer has room for 2 packages (i.e. $Size(\sigma_i) = 0.5$ for all $i = 1, 2, \dots, 8$). One of the optimal solutions S, has the output sequence $a_1a_2b_1b_2c_1c_2c_3a_3$. S stores b_1 and c_1 in the buffer, drops b_1 after a_2 (at step $\overline{a_2}$), stores c_2, and drops c_1 and c_2 at step $\overline{b_2}$. The output sequence has 3 color-changes and 4 color-savings out of possible 5, with $a_2 - a_3$ the only pair which is not a color-saving.

If $r_i - r_{i+1}$ is a color-saving in S, denote $j = drop_S(r_i)$. If $j < \overline{r_{i+1}} - 1$, we say that it is a *passive* color-saving. In this case, in order to make a color-saving, r_{i+1} is not stored in the buffer, while all the packages $\{\sigma_{j+1}, \sigma_{j+2}, \dots \sigma_{\overline{r_{i+1}}-1}\}$ are. We call these packages the *clearance zone* of $r_i - r_{i+1}$. Notice that a package cannot be in more than one clearance-zone. In the above example, the color savings $a_1 - a_2$ and $b_1 - b_2$ are passive, with $drop_S(a_1) = \overline{a_1} = 1$ and $drop_S(b_1) = \overline{a_2} = 4 < 6 = \overline{b_2} - 1$. The clearance zone of $a_1 - a_2$ is $\{b_1, c_1\}$ and the clearance zone of $b_1 - b_2$ is $\{c_2\}$.

With this terminology, we can make further assumptions on the optimal solution. We now assume that every package that gets on the buffer does it for a reason - either to make a color-saving, or to help another package make a color-saving (a passive one). We further assume that in the latter case, the package leaves the buffer as soon as it is no longer needed. And lastly, if a package gets on the buffer in order to make a color-saving, but that color-saving is passive (e.g. the package is dropped before reaching its destination), we assume that it is because one of the packages in the clearance zone starts a color-saving (otherwise - why not go all the way and make an active color-saving?).

Lemma 3. *For any optimal solution we may assume:*

1. *If r_i is stored in the buffer then either r_i is the first package of a color-saving or r_i is in the clearance zone of another color-saving.*
2. *Let $c_s - c_{s+1} - c_{s+2} - \cdots - c_{s+t}$ be a maximal sequence of passive color-savings from the same color c. Let r_j be a package in a clearance zone of one of these color-savings, and assume r_j is not the first package of a color-saving. Then, r_j is removed from the buffer at step $\overline{c_{s+t}}$.*
3. *If r_i is stored in the buffer and it is the first-package of a passive color-saving, then one of the packages in the clearance zone of that saving is the first-package of a color-saving.*

Proof. Given any solution S, we can easily transform it into one that follows the Lemma's conditions without loss of performance. We simply prevent S from storing any package that does not satisfy the conditions of part 1, and remove from the cache any package which satisfy the conditions of part 2 as soon as the buffer reaches c_{s+t} (together with all other packages in the buffer of the same color). It is easily seen that these changes in S did not interfere with any of the color-savings it had made. For part 3, if S stores r_i in the buffer and no package in the clearance-zone of $r_i - r_{i+1}$ starts a color-saving, then we can change S to carry r_i all the way to r_{i+1} (without storing any of the packages that were in the clearance-zone). Clearly, this change also does not reduce S's performance. □

Corollary 1. *Let r_i and b_j be packages, such that $b_j \in B_S(\overline{r_i})$ in a solution S. If r_i is not stored in the buffer and b_j is not starting a color-saving then $r_{i-1} - r_i$ is a color-saving in S.*

Proof. According to part 1 of Lemma 3, b_j was in the clearance zone of another color-saving $c_s - c_{s+1}$. Let c_{s+t} be the last package in the maximal sequence of passive color-savings to which $c_s - c_{s+1}$ belongs. Notice that since all the color-savings in the above sequence are passive, any package between b_j and c_{s+t} which is not stored in the buffer is the last-package of a color-saving of color c. Now, because b_j is still in the buffer even though it is not starting a color-saving we know (according to part 2 of the lemma) that $\overline{b_j} < \overline{r_i} \leq \overline{c_{s+t}}$. Since r_i is not stored in the buffer, it implies that r_i is the last-package of a color-saving of color c, and specifically, that $r_{i-1} - r_i$ is a color-saving in S. □

2.3 Deleting Pairs from the Input Sequence

We recall that the input sequence is a line of packages of different colors, and a pair consists of two consecutive packages of the same color. Given an input sequence $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_n$ and a pair $r_i - r_{i+1}$ in σ, we can *delete* the pair $r_i - r_{i+1}$ by switching the color of all the packages $\{r_j\}_{j \geq i+1}$ to a new color s (i.e. for each $j \geq i+1$ the package r_j becomes s_{j-i}). Let $\sigma' = \sigma'_1, \sigma'_2, \ldots, \sigma'_n$ be the input sequence after the deletion. It is easily seen that except in the case of

$r_i - r_{i+1}$, a pair $\sigma_a - \sigma_b$ is in σ if and only if the pair $\sigma'_a - \sigma'_b$ is in σ'. As an example, consider the sequence $a_1b_1a_2b_2a_3b_3a_4b_4a_5$. If we delete the pair $a_2 - a_3$, the sequence changes to $a_1b_1a_2b_2c_1b_3c_2b_4c_3$.

If we know that we cannot gain a profit by making a color-saving $r_i - r_{i+1}$, then deleting that pair from the input sequence does not affect the optimum solution. We will use this fact extensively in the following sections, and we will also use it now to make another assumption on the input sequence.

Let $r_i - r_{i+1}$ be a pair in the input sequence. Notice that if $Size(r_i) > 1$ and the total size of the packages between r_i and r_{i+1} is also greater than 1, a feasible solution cannot make the color-saving $r_i - r_{i+1}$. Therefore, we can delete that pair from the input sequence. By repeating this process until no such pairs exist, we get the following:

Corollary 2. *If $r_i - r_{i+1}$ is a pair in the input sequence and $Size(r_i) > 1$ then the total size of the packages between r_i and r_{i+1} is at most 1.*

2.4 Classification of Intersecting Color-Savings

For every package r_i and pair $b_j - b_{j+1}$, if $\overline{r_i} \in [\overline{b_j}, \overline{b_{j+1}}]$ we say that r_i and $b_j - b_{j+1}$ *intersect*. Define $\mathcal{I}(r_i)$ to be the set of pairs intersecting r_i.

Let S be a solution and r_i a package. We classify every color-saving $I \in \mathcal{I}(r_i)$ of S into three types:

- Type A: If $I \in \{r_{i-1} - r_i, \quad r_i - r_{i+1}\}$.
- Type B: If r_i is in the clearance-zone of I.
- Type C: Otherwise.

The following two observations are immediate from the definition:

Lemma 4. *Among the color-savings, there is at most one of type B.*

Proof. Immediate, since r_i cannot be in more than one clearance-zone. □

Lemma 5. *If $b_j - b_{j+1}$ is of type C then $b_j \in B_S(\overline{r_i})$*

Proof. Since $b_j - b_{j+1}$ is not of type A or B it implies $\overline{b_j} < \overline{r_i} \leq drop_S(b_j)$ and the lemma follows. □

2.5 A Good Solution

Given σ, a sequence of packages, let $r_i - r_{i+1}$ be the pair whose first-package is the last to appear in σ ("the pair which starts last"). We say that a solution S is *good* if S either makes the $r_i - r_{i+1}$ color-saving, or, otherwise, it has a reason not to (for example - the buffer is full when r_i is reached). In a sense, a good solution is a solution which is "maximal" with respect to the last pair.

Definition 1 (good). *Let $r_i - r_{i+1}$ be the pair which starts last. Then, S is* good *if one of the following is true:*

1. *$r_i - r_{i+1}$ is a color-saving in S.*
2. *$i > 1$ and $r_{i-1} - r_i$ is a color-saving in S.*
3. *If $r_i - r_{i+1}$ is not a color-saving in S, S cannot be trivially changed to include it. Specifically:*
   - *Changing S to store r_i until step $\overline{r_{i+1}} - 1$ will render it infeasible.*
   - *If $B_S(\overline{r_i}) = \emptyset$, then changing S to store all the packages between r_i and r_{i+1} will render it infeasible.*

Notice that if condition 3 is false regarding a solution S, then S can be easily changed, without damaging existing color-savings, to include the $r_i - r_{i+1}$ color-saving and thus become good. We denote by *make_good*(S) the function that applies the above procedure to a solution S and returns the (good) result.

The following lemma states some facts about the state of the buffer after it reaches r_i in a good solution:

Lemma 6. *Let $r_i - r_{i+1}$ be the pair which starts last in σ and let S be a good solution which does not make the $r_i - r_{i+1}$ and $r_{i-1} - r_i$ color-savings. Then, at step $\overline{r_i}$:*

1. *There is no room to store r_i in the buffer (i.e. $Size(B_S(\overline{r_i})) + Size(r_i) > 1$).*
2. *All the packages in $B_S(\overline{r_i})$ are first-packages of color-savings.*

Proof. For part 1, assume on the contrary that it is possible to store r_i in the buffer at step $\overline{r_i}$. Then, since S is good, there is not enough room to store r_i all the way to r_{i+1}. Therefore, there must be another package b_j which S stores in the buffer after step $drop_S(r_i)$. Why is b_j in the buffer? It cannot start a color-saving, since r_i is the last package which starts a color-saving. So according to part 1 of Lemma 3, b_j is in the clearance zone of another color-saving $c_k - c_{k+1}$ (where $\overline{c_k} < \overline{r_i}$), and that clearance zone must lie entirely after $drop_S(r_i)$. To summarize, we have $\overline{c_k} < \overline{r_i} \leq drop_S(r_i) \leq drop_S(c_k)$, which means c_k was stored in the buffer. By part 3 of Lemma 3, it follows that there is a color-saving which starts in the clearance zone of $c_k - c_{k+1}$ and hence after r_i, a Contradiction.

For part 2, let $b_j \in B_S(\overline{r_i})$, and assume on the contrary that b_j is not the first-package of a color-saving. Then, according to Corollary 1, $r_{i-1} - r_i$ is a color-saving in S. contradiction. □

3 Local Ratio Schema

In order to use the local-ratio technique, we must have a profit function we can work with. Thus, we need to further generalize the problem by assigning a *profit* to every pair. When a pair becomes a color-saving, we gain the profit which was assigned to the pair. The goal is to make the maximum profit. This problem is equivalent to the Maximum Color-Savings Problem if we assign each pair a profit of 1.

Problem 3 (Maximum Color Savings with Profits).

Input:
- A sequence of packages in different colors and sizes with no two adjacent packages of the same color.
- A nonnegative profit assigned to every pair in the sequence.

Goal:
Rearrange the sequence using a sorting buffer of capacity 1 to make color-savings with maximum profit.

Notice that as long as the profit is nonnegative, all the lemmas and corollaries which were proved earlier in this paper also apply to optimal solutions of this generalized problem (with the same proofs).

This problem contains the bandwidth-allocation problem [6]. Indeed, we can represent each interval as a pair of packages $r_1 - r_2$ and set its profit to the profit of the interval. We set the size of r_1 as the width of the interval. We organize the packages such that pairs intersect iff their corresponding intervals intersect. Next, we insert a heavy ($Size > 1$) package before the last package of each pair, so no passive color-savings could be made (The heavy packages we add are from distinct colors so no new pairs are created). Now, every color-saving made by a feasible solution in our problem corresponds to a scheduled instance in the bandwidth-allocation problem. Since the bandwidth-allocation problem is NP-hard, it follows Problem 3 is NP-Hard too.

We are now going to examine a general instance of the above problem. Let $\mathcal{P}$ be the set of all pairs in the input sequence σ. Given a solution S, let x be a vector of the boolean variables $\{x_I | I \in \mathcal{P}\}$ such that $x_I = 1$ iff I is a color-saving in S ($x_I = 0$ otherwise). We call x the *color-savings vector* of S. The profit made by a solution S can be represented by the inner product $p \cdot x$ where x is the color-savings vector of S and p is the profit vector, with p_I the profit gained if I is a color-saving in S.

A solution S is an *r-approximation* to an instance of Problem 3, if $p \cdot x \geq \frac{1}{r} \cdot p \cdot x^*$, where x is the color-savings vector of S and x^* is the color-savings vector of an optimal solution. An algorithm is an *r-approximation algorithm* if for every instance of the problem it computes an r-approximation.

Theorem 1 (Local Ratio Theorem). *Let σ be the input sequence of an instance of Problem 3, and let p, p_1, and p_2 be profit vectors such that $p = p_1 + p_2$. Let S be a solution to the above instance, and let x be its color-savings vector. Then, if S is an r-approximation with respect to p_1 and with respect to p_2, then S is also an r-approximation with respect to p.*

Proof. Let S^*, S_1^*, S_2^* be optimal solutions of the instance with respect to the profit vectors p, p_1, and p_2 respectively, and let x^*, x_1^*, x_2^* be their corresponding color-savings vectors. Then:

$$p \cdot x = p_1 \cdot x + p_2 \cdot x \geq \frac{1}{r} \cdot p_1 \cdot x_1^* + \frac{1}{r} \cdot p_2 \cdot x_2^* = \frac{1}{r} \cdot (p_1 \cdot x_1^* + p_2 \cdot x_2^*) \geq \frac{1}{r} p \cdot x^*$$

□

3.1 Schema

We present a generic schema based on the local-ratio technique to approximate the maximum color-savings problem.

1. Delete all pairs with zero profit from the input sequence. Let $\mathcal{P}$ be the set of all the remaining pairs.
2. If $\mathcal{P} = \emptyset$, return the *empty solution* (no package is stored in the buffer).
3. Decompose p by $p = p_1 + p_2$ (The decomposition will be discussed later).
4. Solve the problem recursively using p_2 as the profit function. Let S' be the solution returned.
5. return $S = make\_good(S')$.

We now analyze the quality of the solution produced by the above schema.

Lemma 7. *Let r be a constant. Suppose that the method for decomposing the profit function is such that:*

1. *p_2 is nonnegative.*
2. *There is a pair $I \in \mathcal{P}$ such that $p_2(I) = 0$.*
3. *Every good solution is an r-approximation with respect to p_1.*

Then, the solution S returned by the schema is an r-approximation.

Proof. First of all, since in each recursive call one of the pairs has a zero profit ($p_2(I) = 0$), at least one pair is deleted in every call. Thus the number of recursive calls is bounded by the finite number of pairs, and hence the algorithm terminates in polynomial time.

Second, the first step in which pairs with zero profit are deleted clearly does not change the optimal value. Thus, it is sufficient to show that S is an r-approximation with respect to the new input sequence. The proof is by induction on the number of recursive calls. At the basis of the recursion, the returned solution is optimal (and hence an r-approximation), since no pairs remain in the input. For the inductive step, assume that S' is an r-approximation with respect to p_2. Then, since $S = make\_good(S')$ has (at least) all the color-savings in S' and p_2 is nonnegative, it follows that S is an r-approximation with respect to p_2. Since S is good, it is also an r-approximation with respect to p_1. By the Local-Ratio Theorem, it is an r-approximation with respect to p. □

4 Applying the Schema

We call a pair a *heavy pair* if its first-package has a size greater than $\frac{1}{2}$, and a *light pair* otherwise. We are now going to apply the above schema to two types of instances of the Maximum Color-Savings Problem with Profits - a light type and a heavy type. In the light type all the pairs are light and by applying the schema we will obtain a 6-approximation. In the heavy type, all the pairs are heavy and we will obtain a 3-approximation.

Using these results, the following algorithm returns a 9-approximation solution. Let σ be the input sequence and p the profit function. Then:

1. Let σ' be the resulting sequence after deleting all the heavy pairs in σ.
2. Apply the schema to σ' (light instance) and let S' be the returned solution.
3. Let σ'' be the resulting sequence after deleting all the light pairs in σ.
4. Apply the schema to σ'' (heavy instance) and let S'' be the returned solution.
5. Return the solution, between S' and S'', which gains maximum profit with respect to p.

Theorem 2. *The solution returned by the above algorithm is a* 9*-approximation.*

Proof. Let S^* be the optimal solution, with profit P^*. Let P' and P'' be the profits S^* gained from light pairs and heavy pairs, respectively, such that $P^* = P' + P''$. Then, if $P' \geq \frac{2}{3}P^*$, S' is a 9-approximation. Otherwise, $P'' \geq \frac{1}{3}P^*$ and S'' is a 9-approximation. Hence, the better solution of the two is always a 9-approximation. □

4.1 Applying the Schema on a Heavy Instance

Consider an instance of the Maximum Color-Savings problem with profits, in which all the pairs are heavy. In order to apply the schema it remains to show how to decompose the nonnegative profit function p to $p = p_1 + p_2$ such that all the conditions of Lemma 7 are satisfied. Let $r_i - r_{i+1} \in \mathcal{P}$ be the pair which starts last (recall that $\mathcal{P}$ refers to the pairs in the input sequence after pairs with zero profit have been deleted). Now, we can define the profit function p'_1 as follows:

$$p'_1(I) = \begin{cases} 1 & I \in \mathcal{I}(r_i) \\ 0 & Otherwise \end{cases} .$$

Claim. Every good solution is a 3-approximation with respect to p'_1

Proof. First, we will show that the profit of a good solution is at least 1. Let S be a good solution. If either one of the color-savings $r_{i-1} - r_i$ and $r_i - r_{i+1}$ are made by S then we are done. Otherwise, by Lemma 6, every package in the buffer at step $\overline{r_i}$ is the first package of a color-saving. Since all pairs are heavy, the buffer is either empty or has exactly one package. In the latter case, it follows that the package in the buffer is the first package of a color-saving which intersects r_i, and hence here also S makes a profit of 1.

We are left with the case the buffer is empty when it reaches r_i. This case is not possible: By Lemma 6, there is no place in the buffer to store r_i, which implies $Size(r_i) > 1$. But if that is true, S can be trivially changed to store all the packages between r_i and r_{i+1} in the empty buffer (because by Corollary 2 their total size is no more than 1). This contradicts the fact that S is a good solution which does not make the $r_i - r_{i+1}$ color-saving.

Second, we will prove that the maximum profit is at most 3. Let S be any feasible solution. Classify the color-savings of S in $\mathcal{I}(r_i)$ to 3 types, as in Section 2.4. S can make a profit of at most 2 from type A color-savings. If r_i is not stored in the buffer, S does not profit from type B color-savings and gains at most 1 (because all pairs are heavy) from type C. If r_i is stored in the buffer, S does not profit from type C color-savings and gains at most 1 from type B. In both cases, S profits no more than 3. □

We note that for every $\epsilon \geq 0$, every good solution is a 3-approximation with respect to $\epsilon p'_1$. It is easily seen that by choosing $\epsilon_0 = max\{\epsilon | p - \epsilon p'_1 \geq 0\}$ to define $p_1 = \epsilon_0 p'_1$ and $p_2 = p - \epsilon_0 p'_1$ we ensure that one of the pairs has a p_2-profit of 0 and still keep all the prices nonnegative. This decomposition satisfies all the conditions of Lemma 7, and it allows us to apply the schema on any heavy instance of the problem to receive a solution which is a 3-approximation.

4.2 Applying the Schema on a Light Instance

Consider an instance of the Maximum Color-Savings problem with profits, in which all the pairs are light. In order to obtain a 6-approximation we are going to decompose the problem once more. For each color r, a pair $r_i - r_{i+1}$ is *even* (*odd*, respectively) if i is even (odd). We call an instance of the maximum color-savings with profits problem *reduced* if every package belongs to at most one pair, or in other words, if there are at most 2 packages of each color. We observe that if we delete all the even (odd) pairs, we are left with a reduced instance. We will later show that by applying the schema to a reduced-light instance, we can obtain a 3-approximation. The following algorithm will thus yield a 6-approximation:

1. Let σ' be the resulting sequence after deleting all the even pairs in σ.
2. Apply the schema to σ' (reduced-light) and let S' be the returned solution.
3. Let σ'' be the resulting sequence after deleting all the odd pairs in σ
4. Apply the schema to σ'' (reduced-light) and let S'' be the returned solution.
5. Return the solution, between S' and S'', which gains maximum profit with respect to p.

Lemma 8. *The solution returned by the above algorithm is a* 6*-approximation.*

Proof. Let S^* be the the optimum solution, with profit P^*. Let P' and P'' be the profits S^* gained from even and odd pairs, respectively ($P^* = P' + P''$). Then, either $P' \geq \frac{1}{2}P^*$ or $P'' \geq \frac{1}{2}P^*$. Since S' and S'' are 3-approximations with respect to σ' and σ'', the better solution of the two is a 6-approximation. □

Applying the Schema on a Reduced-Light Instance. It remains to show how to apply the schema on a reduced-light instance to obtain a 3-approximation. As in the previous subsection, we need to show how to decompose the nonnegative profit function p by $p = p_1 + p_2$ such that all the conditions of Lemma 7 are satisfied. Since the instance is reduced, all the pairs in $\mathcal{P}$ are of the form $b_1 - b_2$ where b is a color. Let $r_1 - r_2 \in \mathcal{P}$ be the pair which starts last, and define $\delta \triangleq 1 - Size(r_1)$ (notice that $\delta \geq \frac{1}{2}$). We define p'_1 as follows:

$$p'_1(b_1 - b_2) = \begin{cases} \delta & b_1 - b_2 = r_1 - r_2 \\ Size(b_1) & b_1 - b_2 \in \mathcal{I}(r_1) \setminus \{r_1 - r_2\} \\ 0 & Otherwise \end{cases} .$$

Claim. Every good solution is a 3-approximation with respect to p'_1

Proof. First, we will show that the profit of a good solution is at least δ. Let S be a good solution. If $r_1 - r_2$ is a color-saving in S then we are done. Otherwise, by Lemma 6 we know that $Size(B_S(\overline{r_1})) > 1 - Size(r_1) = \delta$. Let b_i be a package in $B_S(\overline{r_1})$. Then, by part 2 of Lemma 6, b_i is the first-package of a color-saving in S. Since the instance is reduced it follows that $i = 1$, $b_2 \notin B_S(\overline{r_1})$, and hence $b_1 - b_2$ is a color-saving in S which intersects r_1. Therefore, S gains $p'_1(b_1 - b_2) = Size(b_1) = Size(b_i)$ for every $b_i \in B_S(\overline{r_1})$. It follows that S makes a profit of at least $Size(B_S(\overline{r_1})) > \delta$.

Second, we will prove that the maximum profit is at most 3δ. Let S be any feasible solution. Classify the color-saving of S in $\mathcal{I}(r_1)$ into 3 types, as in Section 2.4. S can make a profit of at most δ from type A color-savings (namely $r_1 - r_2$). If r_1 is not stored in the buffer, S does not profit from type B color-savings and gains at most $Size(B_S(\overline{r_1})) \leq 1$ from type C, for a total of no more than $\delta + 1$. If r_1 is stored in the buffer, S can profit at most $Size(B_S(\overline{r_1})) \leq 1 - Size(r_1) = \delta$ from type C color-savings and at most $\frac{1}{2}$ from type B (because there is no more than one color-savings of type B, and it is light), for a maximum total of $2\delta + \frac{1}{2}$. In both cases, S profits no more than 3δ. $\square$

As before, by choosing $\epsilon_0 = \max\{\epsilon | p - \epsilon p'_1 \geq 0\}$ to define $p_1 = \epsilon_0 p'_1$ and $p_2 = p - \epsilon_0 p'_1$ we get the required decomposition, and obtain a 6-approximation algorithm for heavy instances.

References

1. J. S. Kohrt and K. Pruhs. A constant approximation algorithm for sorting buffers. *Proceedings of the Sixth Latin American Symposium (LATIN 2004)*, volume 2976 of Lecture Notes in Computer Science, pages 193-202. Springer-Verlag, 2004.
2. H. Räcke, C. Sohler, and M. Westermann. Online Scheduling for Sorting Buffers. *Proceedings of the 10th ESA (Rome)*, pp. 820–832, 2002.
3. R. Bar-Yehuda. One for the price of two: a unified approach for approximating covering problems. *Algorithmica* 27, 131–144, 2000.
4. R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics 25*, 27-46, 1985.
5. R. Bar-Yehuda and D. Rawitz. On the Equivalence between the Primal-Dual Schema and the Local-Ratio Technique. *Proceedings of RANDOM-APPROX 2001*, p.24–35, 2001.
6. A. Bar-Noy, R. Bar-Yehuda, A. Freund , J. Naor , B. Schieber. A unified approach to approximating resource allocation and scheduling, *Journal of the ACM (JACM)*, v.48 n.5, p.1069-1090, September 2001.
7. Th. Epping, W. Hochstättler. Storage and Retrieval of Car Bodies by the Use of Line Storage Systems. *Technical report* btu-lsgdi-001.02, BTU Cottbus, Germany, 2002.

Approximate Fair Cost Allocation in Metric Traveling Salesman Games

M. Bläser* and L. Shankar Ram

Institut für Theoretische Informatik, ETH Zürich, CH-8092, Zürich, Switzerland
{mblaeser, lshankar}@inf.ethz.ch

Abstract. A traveling salesman game is a cooperative game $\mathcal{G} = (N, c_D)$. Here N, the set of players is the set of cities (or the vertices of the complete graph) and c_D is the characteristic function where D is the underlying cost matrix. For all $S \subseteq N$, define $c_D(S)$ to be the cost of a minimum cost Hamiltonian tour through the vertices of $S \cup \{0\}$ where $0 \notin N$ is called as the *home* city. Define $\text{Core}(\mathcal{G}) = \left\{x \in \Re^N : x(N) = c_D(N) \text{ and } \forall S \subseteq N, x(S) \leq c_D(S)\right\}$ as the core of a traveling salesman game $\mathcal{G}$. Okamoto [15] conjectured that for the traveling salesman game $\mathcal{G} = (N, c_D)$ with D satisfying triangle inequality, the problem of testing whether $\text{Core}(\mathcal{G})$ is empty or not is NP–*hard*. We prove that this conjecture is true. This result directly implies the NP–*hardness* for the general case when D is asymmetric. We also study approximate fair cost allocations for these games. For this, we introduce the cycle cover games and show that the core of a cycle cover game is non–empty by finding a fair cost allocation vector in polynomial time. For a traveling salesman game, let $\epsilon\text{–Core}(\mathcal{G}) = \left\{x \in \Re^N : x(N) \geq c_D(N) \text{ and } \forall S \subseteq N, x(S) \leq \epsilon \cdot c_D(S)\right\}$ be an ϵ–approximate core, for a given $\epsilon > 1$. By viewing an approximate fair cost allocation vector for this game as a sum of exact fair cost allocation vectors of several related cycle cover games, we provide a polynomial time algorithm demonstrating the non–emptiness of the $\log_2(|N|-1)$– approximate core by exhibiting a vector in this approximate core for the asymmetric traveling salesman game. We also show that there exists an $\epsilon_0 > 1$ such that it is NP–*hard* to decide whether $\epsilon_0\text{–Core}(\mathcal{G})$ is empty or not.

1 Introduction

The cooperative game related with the traveling salesman problem is very well–studied. Any cooperative game is characterized by the set of players (or agents) and a cost function that is defined for any coalition of these players. In a traveling salesman game, the players are the cities which the salesman has to visit. The cost function is intuitively the cost incurred by visiting a given subset of the cities, and returning to the home city.

Several problems can be posed with respect to a given combinatorial optimization game. One prominent question is to test the non-emptiness of the core of a game. Probably [18] is the first paper which studied a cooperative game, namely, the assignment game. The underlying combinatorial optimization problem is the assignment problem (or equivalently, the maximum weighted matching problem on bipartite graphs). Testing

* Author's new address: FR Informatik, Universität des Saarlandes, Postfac 151150, 66041 Saarbrücken, Germany. email: mblaeser@cs.uni-sb.de

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 82–95, 2006.

the core non-emptiness of this game is essentially the same as the polynomial solvability of the optimization problem by the Hungarian method [14]. Another example is the minimum spanning tree game wherein the core was shown to be non-empty by an explicit construction of a vector in the core [2,11]. In these examples and some more, a clear relationship exists between the polynomial solvability of the underlying optimization problem and testing the non-emptiness of the core of the game.

Another characterization of the core non-emptiness of a game is from linear programming. A result of Deng et. al. [3] states that a necessary and sufficient condition for the core of a maximum packing game and a minimum covering game to be non-empty is that the linear programming relaxations of these problems have integral optimal solutions. Note that the underlying optimization problems in this case are NP–*hard*. Other characterizations in this direction are for the facility location games [13], partition games [7], and delivery games [10] to mention a few.

On the other hand, several papers deal with the intractability of the core non–emptiness of certain games. For example, Deng et. al. [3], showed that testing the non-emptiness of the core of the minimum coloring game is NP–*complete*. The underlying combinatorial optimization problem in the case is also NP–*hard*. Thus, this reinstates again the relationship between these two problems. Goemans and Skutella [9] showed the NP–*completeness* of the core non-emptiness of a facility location game.

In this paper, we study traveling salesman games, introduced by Potters et. al. [17]. More formally, a cooperative game is given by the tuple (N, f) where $N = \{1, 2, \dots, n\}$ and $f : 2^N \rightarrow \Re$ is a characteristic function. In the case of a traveling salesman game, $N = \{1, 2, \dots, n\}$ (the cities) with a given symmetric distance matrix D (referred to as a cost function defined on all pairs of cities) on the set of cities and for a subset $S \subseteq N$, we have the characteristic function $c_D(S)$ defined to be the cost of a minimum cost Hamiltonian tour which visits all the cities in $S \cup \{0\}$ where $0 \notin N$ is called the home city or home node. Note that the cost matrix D is defined over all pairs (i, j) where $i, j \in N \cup \{0\}$. The core of a game (N, f) is defined to be the following:

$$\text{Core}(N, f) = \{x \in \Re^n : x(N) = f(N) \text{ and } \forall S \subseteq N, x(S) \leq f(S)\}$$

where $x(S) = \sum_{i \in S} x(i)$ with $x = (x(i))_{i=1\dots n}$. The interpretation of this definition for the traveling salesman game can be motivated as follows. Consider home node 0 as the home city of a professor who has to give talks at the universities located in vertices $1, \dots, n$. The total travel cost is $c_D(N)$. So, the problem is to find a fair cost allocation (a vector in the core) such that no coalition S will split off because they pay more than the actual cost of an optimal subtour through $S \cup \{0\}$ and invite the professor to visit only the universities $i \in S$.

Various aspects of the traveling salesman games have already been covered in the literature. Tamir [19] showed that a metric (i.e., satisfying triangle inequality) traveling salesman game with at most four players always has a non-empty core and also the existence of a game with six players whose core is empty. Further, Faigle et. al. [6] designed an instance of a 2-dimensional Euclidean game with six players such that the core is empty. More recently, Okamoto [15] showed that the problem of deciding whether a general traveling salesman game has an empty core or not, is NP–*hard*. But for the special case of metric traveling salesman games, the same question was left open and conjectured to be NP–*hard*. In this paper, we show that this is indeed the case. In fact, we prove that testing the core–emptiness of a $\{1, 2\}$ traveling salesman game where the costs on any pair of cities is either one or two and the costs are symmetric (i.e., cost on a pair (i, j) is the same as that on (j, i)) is NP–*hard*. This also proves that it is NP–*hard* to decide if the core of an asymmetric traveling salesman game with triangle inequality, is empty or not. Note that an asymmetric traveling salesman game is a generalization of the symmetric game. We then consider approximate fair cost allocations, i.e., find a cost allocation vector $x \in \Re^N$ such that $\forall S \subseteq N$, $x(S) \leq \epsilon \cdot c_D(S)$ and $x(N) \geq c_D(N)$, for some ϵ. Our reduction also yields that it is NP–*hard* to find an ϵ_0–approximate cost allocation vector for some $\epsilon_0 > 1$ for the asymmetric traveling salesman game, using a result of Berman et. al. [1].

We introduce cycle cover games on the same underlying complete directed graph, where the characteristic function is the cost of a minimum cost cycle cover. We show that the core is always non–empty for such a game and provide a $O(|N|^3)$ time algorithm for finding a fair cost allocation vector. We also show that an approximate fair cost allocation vector for an asymmetric traveling salesman game is the sum of exact fair cost allocation vectors of several related cycle cover games.

The question of finding an approximate fair cost allocation vector has already been considered for several cooperative games where testing the core non-emptiness problem is NP–*hard*. Faigle et. al. [6] find a 1.5-approximate fair cost allocation vector for symmetric traveling salesman game. For this, they make use of the well known Christofides' approximation algorithm for symmetric traveling salesman optimization problem. In this paper, we provide a polynomial time algorithm that finds a $\log_2(|N| - 1)$-approximate cost allocation vector for the asymmetric traveling salesman game. We make use of an approximation algorithm for the minimum asymmetric traveling salesman problem of Frieze et. al [8].

2 Preliminaries

Let $N = \{1, 2, ..., n\}$. Define $D : (N \cup \{0\}) \times (N \cup \{0\}) \rightarrow \{1, 2\}$ to be an $(n + 1) \times (n + 1)$ symmetric matrix. Let $c_D : 2^N \rightarrow \mathbb{Z}$ be such that $\forall S \subseteq N$,

$$c_D(S) = \min_{\rho : S \rightarrow S} \left\{ d(0, \rho(i_1)) + \sum_{j=1}^{|S|-1} d(\rho(i_j), \rho(i_{j+1})) + d(\rho(i_{|S|}), 0) \right\}$$

over all permutations ρ on $S = \{i_1, i_2, \ldots, i_{|S|}\}$. In other words, $c_D(S)$ is the cost of a minimum cost Hamiltonian tour through $S \cup \{0\}$, with $0 \notin N$ called the home node, when we consider the complete graph on $N \cup \{0\}$. The tuple (N, c_D) is the symmetric traveling salesman game. The core of the game is defined as

$$\text{Core}(N, c_D) = \{x \in \Re^n : x(N) = c_D(N) \text{ and } \forall S \subseteq N, x(S) \leq c_D(S)\}$$

where $x(S) = \sum_{i \in S} x(i)$ with $x = (x(i))_{i=1 \ldots n}$. Any vector $x \in \text{Core}(N, c_D)$ is called a *fair cost allocation vector*. Whenever x is a vector, $x(i)$ will refer to the corresponding value at the ith coordinate.

Consider the following decision problem : given a matrix D, is Core$(N, c_D) = \emptyset$ or not?

We denote the problem as Core–ΔTS or the problem of testing the core non-emptiness of a metric traveling salesman game. Note that the input to the decision problem is the matrix D and not the function $c_D(.)$. We remark that one does not need to compute $c_D(N)$ and hence testing whether (N, c_D) has an empty core may be easier than testing membership in the core, i.e., whether a given x satisfies the two properties of fair cost allocation. We show that Core–ΔTS is NP–*hard* by a polynomial time reduction from the following SAT problem (3SAT4), also called the Bounded Occurence Satisfiability problem:

Given a boolean formula ϕ as a conjunction of disjunctive clauses with exactly three literals per clause and the number of occurences of a literal is four, does there exist a truth assignment to the variables of the formula such that all the clauses are satisfied?

3SAT4 was shown to be NP–*complete* in [20] . Recently, it was shown in [1], that it is NP–*hard* to approximate the corresponding maximization problem to within a constant $c > 1$.

3 The Reduction

In this section, we elaborate the polynomial time reduction from 3SAT4 to Core–ΔTS.

3.1 The Basic Gadgets

The usual reductions to the traveling salesman problem make use of special components called gadgets or devices. A gadget forces an optimal Hamiltonian tour to have a special structure. We use gadgets similar to those given in [5,16] – the former reduces from a NP–*hard* problem related to Linear Equations, while the latter reduces from 3SAT4 – for the reductions to the minimum symmetric traveling salesman problem. A basic gadget used in the construction is the ex-OR device, shown in Fig. 1(a). The structure of the device is so that there can be only two possible traversals of this gadget by any optimum Hamiltonian tour since the gadget is connected to the rest of the graph only at the boundary vertices. We shall think of an ex-OR subgraph as two edges connected by an extrinsic device (Fig. 1(b)). This will be useful in visualizing the Hamiltonian cycle in the whole graph.

For each variable of the boolean formula, we have a device as shown in Fig. 2. It has two paths, one for each truth value of the variable. We refer to these paths as "true path" and "false path" respectively. Each path is an arrangement of 29 ex-ORs - four

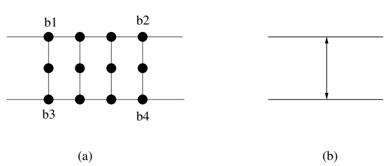


Fig. 1. (a)The ex-OR gadget. $b1, b2, b3, b4$ are called boundary vertices. (b) Representation of an ex-OR device.

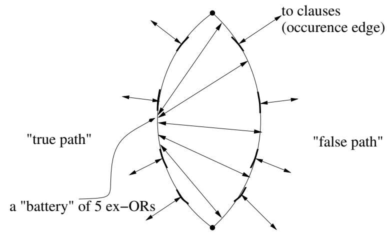


Fig. 2. The variable gadget. There are four "occurence" edges corresponding to the four occurences of literal x or $\bar{x}$, in the respective paths.

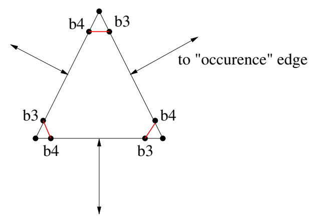


Fig. 3. The clause gadget. There are three ex-OR devices corresponding to the three literals of the clause. $b3, b4$ vertices are the boundary vertices of the corresponding ex-ORs.

of them are connected to the clause devices (one for each occurence of a literal called *occurence edge*), and the others (five "batteries" or "series" of five ex-ORs each) are connected within to the other path of the same variable device. The intuition behind such a construction is consistency, i.e., to ensure that an optimal tour does not traverse both paths. So, any optimal Hamiltonian tour traverses exactly one of the two paths and also all the vertices of this path appear successively on the tour.

For each clause, we have a triangle device with each edge connected to the occurence edge of the literal in the clause via an ex-OR device. Please refer to Fig. 3. Note that there are three edges between the boundary points of adjacent ex-OR devices of the gadget. These edges will be referred to as *boundary-boundary* edges. This is an important difference from the clause gadget of [16], which will be essential for the NP–*hardness* proof.

3.2 The Construction

We now describe the actual graph that will be constructed from a given boolean formula. Let $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be the given boolean formula where each $C_i = (a_i \vee b_i \vee c_i)$. Also any variable v appears at most four times as the literal v and at most four times as the literal $\bar{v}$ in ϕ. We construct graph G as follows. Fix an order of the variables and connect the variable gadgets as a series, as shown in Fig. 4. The set of all m clause gadgets are connected so that the $3m$ corners are pairwise connected amongst themselves and also to the first and last vertices of the variable series.

The distance matrix D for this graph G is simply : $d(i,j) = 1$ if $(i,j) \in E$, and otherwise $d(i,j) = 2$. This means that all the edges which are mentioned in the

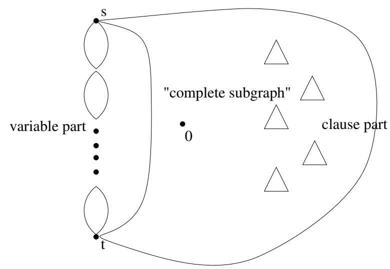


Fig. 4. The graph G. Home node "0" is not considered to be part of G. Corners of clause gadgets, the nodes s and t, and the home node "0" form a complete subgraph of $G \cup \{0\}$.

construction are of cost one and the remaining edges (note that an instance of a TSP game is a complete graph) are of cost two. We consider G to be the graph consisting of only these cost one edges.

3.3 Structure of an Optimal Tour

We show the following lemmas on the structure of optimal Hamiltonian tours in G.

Definition 1. *Nodes of G which are traversed by an optimal Hamiltonian tour with one edge of cost one and another of cost two are called endpoints. Those nodes which are traversed with both edges of cost two are called double endpoints and will be seen as two endpoints each.*

Lemma 1. *Let $\mathcal{A}$ be a truth assignment to the variables of ϕ such that maximum number of clauses are satisfied. If, under $\mathcal{A}$, ϕ has k unsatisfied clauses, then there exists an optimal Hamiltonian tour through the vertices of G with k or $k+1$ endpoints, depending on k being even or odd respectively. Moreover, these endpoints are present in the clause part of G.*

Proof : Consider the following tour. The variable part of G is traversed according to the assignment $\mathcal{A}$, i.e., if $a_i = 1$ in $\mathcal{A}$, then we take the "true path" of the variable a_i, otherwise the "false path" of the variable. In the clause part, the tour traverses the satisfied clause gadgets first (this means that at most two edges of such a triangle are not covered by the variable part traversal of the tour). Then, in the unsatisfied clause gadgets, one endpoint per each gadget is introduced. This is because, any Hamiltonian tour , for optimality, needs to leave an unsatisfied clause from a non-corner vertex at least once and this vertex becomes an endpoint. For parity reasons, one may have to introduce another endpoint. Thus, in this tour there are either k or $k+1$ endpoints and all of them are introduced in the clause devices. It remains to show that such a tour is optimal. The proof of this fact is essentially the same as given in [16] which exhaustively lists the various possibilities of traversal and in each case how one can modify the tour to have the required structure without increasing costs. However, there is one issue that needs to be taken care of. The clause gadget of [16] and ours differ in the introduction of additional cost one edges between the adjacent boundary points in our construction (the *boundary-boundary* edges). So, we need to make sure that

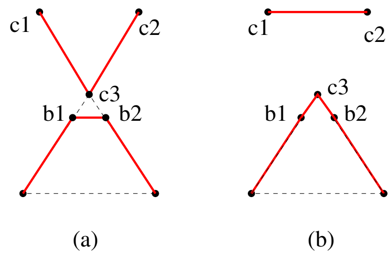


Fig. 5. (a) shows an optimal Hamiltonian tour using the *boundary-boundary* edge (b_1, b_2). Such a tour can be modified as shown in (b).

the traversal of any optimal Hamiltonian tour through our clause gadget can also be assumed to follow the same traversal pattern as that of the gadget given in [16]. This modification is illustrated in Fig. 5. In Fig. 5(a), the optimal tour uses the boundary edge (b_1, b_2). This can be overcome by the modification suggested in (b). Since for optimality, any clause gadget can be entered only from corner vertices, the vertices c_1, c_2 are indeed corner vertices and hence are connected by an edge of cost one. Putting it all together, the Lemma follows. □

Lemma 2. *Let n be the number of vertices in G. If ϕ is satisfiable, the cost of an optimal Hamiltonian tour in G is n. If ϕ is unsatisfiable and there exists an optimal (in the sense of satisfying maximum number of clauses) assignment with k unsatisfied clauses, then the cost of an optimal Hamiltonian tour in G is $n + \lceil \frac{k}{2} \rceil$.*

Proof : Consider the optimal Hamiltonian tour constructed in Lemma 1. If ϕ is satisfiable or in other words $k = 0$, then this tour has no endpoints. So, its cost is n. Otherwise, the tour has k or $k+1$ endpoints. Each endpoint has one edge of cost two in the tour, and hence the number of cost two edges in the tour is $\frac{k}{2}$ or $\frac{k+1}{2}$ when k is even or odd respectively, i.e., $\lceil \frac{k}{2} \rceil$ cost two edges. Thus, the cost of the tour is $(n - \lceil \frac{k}{2} \rceil) \cdot 1 + \lceil \frac{k}{2} \rceil \cdot 2 = n + \lceil \frac{k}{2} \rceil$. □

4 Hardness Results

Let n be the number of vertices in G. Define $N = \{1, 2, \ldots, n\}$, the vertices of G. Also, let $c_D : 2^N \to \Re$, be defined as follows. For any $S \subseteq N$, $c_D(S)$ is the cost of a minimum cost tour through the vertices of $S \cup \{0\}$. Recall that in our construction, the home node 0 is connected to the corner vertices of all clause gadgets by cost one edges.

Theorem 1. *If $N = \{1, 2, \ldots, n\}$ and D is a $(n+1) \times (n+1)$ symmetric matrix satisfying triangle inequality, then the problem of deciding if Core(N, c_D) is empty or not, is* NP–hard.

Proof: We show the NP–*hardness* of Core–ΔTS by showing the following equivalent claim.

Claim : ϕ is satisfiable if and only if Core(N, c_D) is non-empty.

Suppose ϕ is satisfiable. We show a fair cost allocation in the TSP game (N, c_D) thereby proving the core to be non-empty. Since ϕ is satisfiable, by Lemma 2, the cost of an optimal Hamiltonian tour is $n+1$. Note that this tour passes through the home node '0'. Let us define the vector $x \in \Re^n$ to be $(\frac{n+1}{n}, \frac{n+1}{n}, \ldots, \frac{n+1}{n})$. We claim that $x \in$Core(N, c_D). Clearly, $x(N) = n+1 = c_D(N)$. Consider any $S \subseteq N$. We have, $c_D(S) \geq |S|+1$. But, $x(S) = \sum_{i \in S} \frac{n+1}{n} = (1+\frac{1}{n})|S| = |S|+\frac{|S|}{n} \leq |S|+1 \leq c_D(S)$. Hence, $x \in$Core(N, c_D).

Now, suppose ϕ is unsatisfiable. Consider an optimal truth assignment $\mathcal{A}$ (optimal in terms of maximum number of satisfiable clauses) which satisfies all but k clauses ($k > 2$). We deal with $k = 1, 2$ cases later. Depending on the truth value of a variable in $\mathcal{A}$, let T denote all the vertices of G occuring in the "true paths" of all variables (i.e., if variable $v_i = 1$ in $\mathcal{A}$, then we take the path in the variable device of v_i corresponding to the literal v_i, otherwise that of $\bar{a}_i$) , the vertices of all the ex-OR batteries on the "false paths" and finally all the remaining vertices of satisfied clause devices (with respect to $\mathcal{A}$). This means that we consider all the vertices in the variable part of G except those present in the occurence edges on the "false paths" with respect to $\mathcal{A}$. It also implies that vertices of ex-OR devices of occurence edges in "true paths" present on the satisfied clauses (with respect to $\mathcal{A}$) are also in T. Let C denote the corner vertices of the unsatisfied clause gadgets. Let $R := N \backslash \{T \cup C\}$. Suppose, for contradiction, that Core$(N, c_D) \neq \emptyset$. Let $x \in$ Core(N, c_D). Since $x \in \Re^n$ is a fair cost allocation vector, the following structural properties must hold for x.

Lemma 3. *If x is a fair cost allocation vector, and T is defined as above, then $x(T) \leq |T| + 1$.*

Proof : There is a Hamiltonian tour through the vertices of T which uses only cost one edges, as follows. First, traverse the vertices of T on the variable part of G according to the assignment $\mathcal{A}$. Now, consider the vertices of satisfied clause devices that have not yet been covered by the tour. There are three possibilities as to the arrangement of these vertices :

1. only the corner vertices of a satisfied clause device are not traversed.
2. an edge of the satisfied clause device and the corner vertices are not traversed.
3. two edges of the satisfied clause device alongwith the corner vertices are not traversed.

Clearly, all these vertices can be traversed with cost one edges by recalling the fact that all corner vertices and the home node 0 are interconnected by cost one edges. Hence, $c_D(T) = |T| + 1$. Since $x(T) \leq c_D(T)$, the claim follows. □

Lemma 4. *Let C be the set of corner vertices of unsatisfied clause gadgets, and R be the set of remaining vertices in unsatisfied clause gadgets. If x is a fair cost allocation vector, then $x(C) + x(R) \geq |C| + |R| + \lceil \frac{k}{2} \rceil$.*

Proof : Since x is a fair cost allocation vector, $x(N) = c_D(N)$. Now, there is a Hamiltonian tour through the vertices of G of cost $n + \lceil \frac{k}{2} \rceil$, by Lemma 2. So, $c_D(N) = n + \lceil \frac{k}{2} \rceil + 1$, by recalling the fact that $c_D(N)$ is an optimal tour including the home

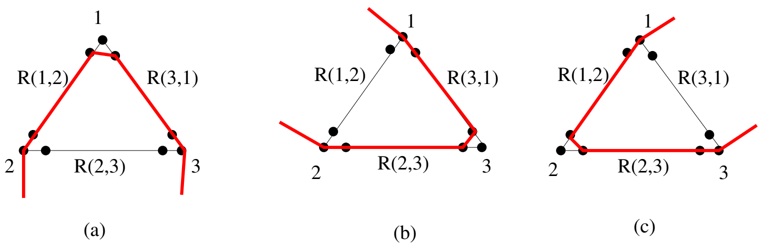


Fig. 6. (a), (b), (c) show the traversals of three tours H_1, H_2, H_3 respectively on an unsatisfied clause gadget. In H_1, the corner vertex 1 and all the vertices of the ex-OR on the edge $R(2,3)$ are not traversed. The complete tours can be visualized easily since all the corner vertices are pairwise connected.

node 0. Also, $x(N) = x(C) + x(R) + x(T)$, since the sets C, R, T are disjoint. Now, by Lemma 3, $x(T) \leq |T| + 1$, and hence the Lemma follows. □

We consider three tours H_1, H_2, H_3 through the vertices of $C \cup R$ as shown in Fig. 6, Let an unsatisfied clause gadget C_i, for $i = 1, 2, \ldots, k$, be given by $(3i-2, 3i-1, 3i, R(3i-2, 3i-1), R(3i-1, 3i), R(3i, 3i-2))$, where the first three are the corner vertices and $R(p,q)$ denotes the vertices in the ex-OR gadget between (p,q) of C_i and the corresponding occurence edge in the variable part of G. Thus, the tour H_1 is then $\{2, R(1,2), R(3,1), 3, 5, R(5,4), R(6,4), 6, \ldots, 3k-1, R(3k-2, 3k-1), R(3k, 3k-2), 3k, 2\}$. The tours H_2 and H_3 are similarly defined. Let H be one of these tours with the maximum $x(.)$ value, i.e., $x(H) := \max\{x(H_1), x(H_2), x(H_3)\}$. This implies that $x(H) \geq \frac{1}{3}\{x(H_1) + x(H_2) + x(H_3)\}$.

For all $u \in C \cup R$, $x(u)$ contributes twice in the sum $x(H_1) + x(H_2) + x(H_3)$. Therefore,

$$x(H) \geq \frac{1}{3}\{2x(C) + 2x(R)\} = \frac{2}{3}\left\{|C| + |R| + \left\lceil \frac{k}{2} \right\rceil\right\}$$

But, $|H| = |H_1| = |H_2| = |H_3| = \frac{2}{3}\{|C| + |R|\}$ and $x(H) \leq |H| + 1$, by the definition of the tours H_j, i.e., $\frac{2}{3}\left\{|C| + |R| + \left\lceil \frac{k}{2} \right\rceil\right\} \leq x(H) \leq \frac{2}{3}|C| + \frac{2}{3}|R| + 1$, a contradiction when $k \geq 3$.

We employ the following technique in order to overcome the difficulty in getting a contradiction for $k \leq 2$. Instead of considering the formula ϕ, we look at the formula $\phi' = \phi_1 \wedge \phi_2 \wedge \phi_3$. The formula ϕ' is a conjunction of the formulas ϕ_i, for $i = 1, 2, 3$, where each ϕ_i is a copy of the old formula ϕ but with new, distinct variables. This means that ϕ' has $3n$ variables and $3m$ clauses. It is easy to see that both ϕ and ϕ' are equivalent because the variables of each ϕ_i are distinct. Now, if there is an optimal truth assignment that satisfies all but k clauses of ϕ, then there is an optimal truth assignment that satisfies all but $3k$ clauses of ϕ'. Thus, when $k = 1$ or $k = 2$, the number of unsatisfied clauses in ϕ' is respectively 3 and 6. The above proof holds good for ϕ'. Hence, Core$(N, c_D) = \emptyset$. This proves the claim that ϕ is satisfiable if and only if Core (N, c_D) is non–empty. Clearly, the construction of the graph G from ϕ can be done in polynomial time (in the size of ϕ and the number of variables). Therefore, Core–ΔTS is NP–*hard*. □

Theorem 2. *If* $N = \{1, 2, \dots, n\}$ *and* D *is a* $(n+1) \times (n+1)$ *matrix, not necessarily symmetric but satisfying triangle inequality, then the problem of deciding if Core*(N, c_D) *is empty or not, is* NP–hard.

Proof : Let the problem mentioned in the statement of the theorem be referred to as Core–ΔATS. But since Core–ΔTS is shown to be NP–*hard* by Theorem 1, and it is a special case of Core–ΔATS, the claim of the theorem follows. □

5 Approximate Fair Cost Allocation

Since the core emptiness problem is NP-*hard* for traveling salesman games, we turn our attention towards finding approximate fair cost allocation vectors. Define, for a (N, f) game and a given $\epsilon > 1$, an ϵ–*approximate* core as :

$$\epsilon\text{–Core}(N, f) = \left\{x \in \Re^N : x(N) \geq f(N) \text{ and } \forall S \subseteq N, x(S) \leq \epsilon \cdot f(S)\right\}$$

Towards finding such approximate fair cost allocations, we introduce new games called as cycle cover games.

5.1 Cycle Cover Games

Let $G = (V, E)$ be a complete directed graph with a cost function $D : E \to \Re$. A cycle cover $\mathcal{C}$ in G is a collection of vertex–disjoint cycles that span V. A minimum cost cycle cover is a cycle cover of minimum cost with respect to D. We define a cycle cover game to be the tuple (N, f_D) where $N = V$, and $f_D : 2^V \to \Re$ is defined for a subset $S \subseteq N$ as the cost of a minimum cost cycle cover on the vertices of S. For this game, we show that the core is non-empty by finding a fair cost allocation vector in polynomial time.

Theorem 3. *For a cycle cover game* (N, f_D)*, the core is not empty. A fair cost allocation vector in the core can be found in* $O(|N|^3)$ *time.*

Proof: Consider the following integer program formulation for the minimum cost cycle cover problem:

$$\min \sum_{i,j \in N} d(i,j) y_{ij} \quad \text{subject to}$$

$$\sum_{j \in N \setminus \{i\}} y_{ij} = 1 \quad \forall i \in N \quad \text{and} \quad \sum_{i \in N \setminus \{j\}} y_{ij} = 1 \quad \forall j \in N$$

$$\text{where} \quad y_{ij} \in \{0, 1\}$$

We relax the final set of constraints to $y_{ij} \geq 0$, to obtain a linear program $\mathcal{L}(N)$. It is known that in fact $\mathcal{L}(N)$ has an integer optimum solution. Next, we consider the dual program of $\mathcal{L}(N)$.

$$\max \sum_{v \in N} x^+(v) + \sum_{v \in N} x^-(v) \quad \text{where} \quad x^+(i) + x^-(j) \leq d(i,j) \quad \forall i,j \in N$$

Let us denote the dual program by $\mathcal{D}(N)$. Let $x(v) = x^+(v) + x^-(v)$ for all $v \in N$. We claim that an optimal solution $\mathbf{x} = (\mathbf{x}(v))_{v \in N}$ of $\mathcal{D}(N)$ is a fair cost allocation vector to the cycle cover game (N, f_D). By the duality theorem, we know that the optimal value of the objective function of $\mathcal{L}(N)$ which is $f_D(N)$ by definition, is the same as $\mathbf{x}(N)$. Consider any subset $S \subset N$. Let C_S denote a minimum cost cycle cover on S, i.e., the cost of this cycle cover is $f_D(S)$. Now, $f_D(S) = \sum_{C \in C_S} \sum_{(u,v) \in C} d(u,v) \geq \sum_{C \in C_S} \sum_{(u,v) \in C} (\mathbf{x}^+(u) + \mathbf{x}^-(v)) = \sum_{u \in S} (\mathbf{x}^+(u) + \mathbf{x}^-(u)) = \sum_{u \in S} \mathbf{x}(u) = \mathbf{x}(S)$. Here, $C \in C_S$ denotes a cycle in the cycle cover and the inequality in the middle follows because of the feasibility of $\mathbf{x}$ in $\mathcal{D}(N)$. Thus, we have shown that $\mathbf{x} \in \Re^N$ is a fair cost allocation vector of the cycle cover game (N, f_D), thereby showing the non-emptiness of the core. Also $\mathbf{x}$ is computed in $O(|N|^3)$ time using the algorithm of [4] which is a primal-dual type algorithm. □

5.2 A Traveling Salesman Game as a Combination of Several Cycle Cover Games

We show how one can view a traveling salesman game to be a *combination* of several cycle cover games. Formally, what we prove is that an approximate fair cost allocation vector for a traveling salesman game can be seen as the sum of (exact) fair cost allocation vectors of several related cycle cover games. We provide an algorithm to find such an approximate fair cost allocation vector, followed by the proof of the claimed degree of approximation.

Here, the traveling salesman game refers to the asymmetric traveling salesman game where the cost matrix fulfills the triangle inequality. For the purpose of proving the main theorem of this section, we adapt the approximation algorithm of Frieze et. al. [8], for the asymmetric traveling salesman optimization problem. This approximation algorithm achieves a performance guarantee of $\log_2(|V|)$, where V is the set of vertices of the underlying complete directed graph.

The algorithm to find an approximate fair cost allocation vector for an asymmetric traveling salesman game is given in Fig. 1. Note that the home node “0” is included only in the first cycle cover game.

Theorem 4. *Let (N, c_D) be an asymmetric traveling salesman game, with D satisfying triangle inequality. If $\mathbf{x}^*$ is the vector returned by Algorithm 1 for this game, then it is a $\log_2(|N| - 1)$–approximate fair cost allocation vector. The running time of the algorithm is $O(|N|^3)$.*

Proof : First, let us consider the following linear program for asymmetric traveling salesman problem, $\mathcal{T}(N)$:

$$\min \sum_{i,j \in N \cup \{0\}} d(i,j) y_{ij} \quad \text{subject to}$$

Algorithm 1

Input: An asymmetric game (N, c_D) with a complete directed graph on $N \cup \{0\}$, and D satisfying triangle inequality.

Output: A vector $\mathbf{x}^* \in \Re^{|N|}$.

1: Set $j := 0$, $V_j := N$, and let $\mathbf{x}^* \in \Re^{|N|}$ be the all zero vector.
2: Compute a fair cost allocation vector $\mathbf{x}_j \in \Re^{|V_j|+1}$ for the cycle cover game on the complete graph induced on $V_j \cup \{0\}$. Let $\mathcal{C}$ be a minimum cost cycle cover in this graph.
3: Set, for all $1 \le i \le |N|$, $\mathbf{x}^*(i) := \mathbf{x}_j(i) + \frac{\mathbf{x}_j(0)}{|N|}$ and then set $j := 1$. Let $\mathbf{z}_0 \in \Re^{|N|}$ denote the current $\mathbf{x}^*$.
4: Pick one vertex from each cycle of $\mathcal{C}$ such that the vertex set picked, V_j, does not contain "0".
5: **while** $|V_j| \ge 2$ **do**
6: Compute a minimum cost cycle cover $\mathcal{C}$ in the induced complete graph on V_j.
7: Compute a fair cost allocation vector $\mathbf{x}_j \in \Re^{|V_j|}$ by using Theorem 3 for the cycle cover game (V_j, c_D).
8: Update $\mathbf{x}^*(i) := \mathbf{x}^*(i) + \sum_{j: i \in V_j} \mathbf{x}_j(i)$, for all $i \in N$.
9: $j := j + 1$.
10: Pick exactly one vertex from each of the cycles of the cycle cover $\mathcal{C}$. Set V_j to be the set of such vertices.
11: **end while**
12: return the vector $\mathbf{x}^*$.

$$\sum_{j \in N \cup \{0\} \setminus \{i\}} y_{ij} = 1 \quad \forall i \in N \text{ and } \sum_{i \in N \cup \{0\} \setminus \{j\}} y_{ij} = 1 \quad \forall j \in N$$

$$\sum_{i \in S, j \in N \cup \{0\} \setminus S} y_{ij} \ge 1 \ \ \forall S \subseteq N \ \text{ and } \sum_{j \in S, i \in N \cup \{0\} \setminus S} y_{ij} \ge 1 \ \ \forall S \subseteq N$$

$$\text{where } \ y_{ij} \ge 0$$

The third and the fourth set of constraints together are usually referred to as *subtour elimination* constraints. Note the inclusion of the home node "0" in the program. This program is the asymmetric version of the program for symmetric game given in [6]. It can be easily verified that the actual integer linear program corresponding to $\mathcal{T}(N)$ has an optimum value $c_D(N)$. When the subtour elimination constraints are dropped from $\mathcal{T}(N)$, the linear program obtained is the same as the linear program $\mathcal{L}(N \cup \{0\})$ formulated in the proof of Theorem 3. This can be seen as follows: the only issue is to verify that not having the in-degree and out-degree constraints at home node "0" is equivalent to having the constraints. Suppose "0" appears in more than one cycle of an optimal solution y (we can assume that y is integral). Let $u, v \in N$ be such that $y_{u0} = 1 = y_{0v}$ where u, v are in the same cycle in this cycle cover y. Then by changing y_{uv} from 0 to 1 and resetting y_{u0}, y_{0v} both to 0, we obtain a solution of cost at most that of y as $d_{u0} + d_{0v} \ge d_{uv}$ by triangle inequality.

From the algorithm, $\mathbf{x}^*(N) = \sum_{j=0}^{k} \sum_{i \in N} \mathbf{x}_j(i)$ where k is such that $|V_k| = 1$, i.e. the number of times the while–loop gets executed. By duality theorem, this means that $\mathbf{x}^*(N)$ is the sum of the costs of all k cycle covers computed in the algorithm. Now, the union of all these cycle covers is an Eulerian graph (the in-degree of any vertex is equal to its out-degree). But, any Hamiltonian tour obtained by short-cutting through such an Eulerian graph is of cost at most that of the whole Eulerian graph because of triangle inequality. Hence, $\mathbf{x}^*(N) \geq c_D(N)$ since $c_D(N)$ is the cost of an optimal Hamiltonian tour.

Consider any subset $R \subset N$. We claim that $\mathbf{x}^*(R) \leq \log_2(|R|)c_D(R)$. By definition, $\mathbf{x}^*(R) = \mathbf{z}_0(R) + \sum_{j=1}^{k} \mathbf{x}_j(R \cap V_j)$. First, we show that $\mathbf{z}_0(R) \leq c_D(R)$. Now, since $\mathbf{x}_0 \in \Re^{|N|+1}$ (refer to step 2 of the algorithm) is an exact fair cost allocation vector for the cycle cover game on $N \cup \{0\}$, we have $\mathbf{x}_0(R \cup \{0\})$ is at most the cost of a minimum cost cycle cover on $R \cup \{0\}$. But, by definition, $\mathbf{x}_0(R \cup \{0\}) = \mathbf{x}_0(R) + \mathbf{x}_0(0) = \{\mathbf{x}_0(R) + |R|\frac{\mathbf{x}_0(0)}{|N|}\} + (|N| - |R|)\frac{\mathbf{x}_0(0)}{|N|} \geq \mathbf{z}_0(R)$ since $|N| - |R| > 0$. Thus, $\mathbf{z}_0(R)$ is at most the cost of a minimum cost cycle cover on $R \cup \{0\}$ which is at most $c_D(R)$, the cost of an optimal Hamiltonian tour through $R \cup \{0\}$. Next, we show that for all $0 < j \leq k$, $\mathbf{x}_j(R \cap V_j) \leq c_D(R)$. Denote by TSP_j, the optimal value of the linear program $\mathcal{T}(V_j \cap R)$. Then, since any feasible solution to $\mathcal{T}(V_j \cap R)$ is a cycle cover on $V_j \cap R$, we have that $\sum_{j=1}^{k} \mathbf{x}_j(R \cap V_j)$ is bounded by $\sum_{j=1}^{k} TSP_j$. The only non-zero TSP_j values are those for which $|V_j \cap R| \neq 0$. By triangle inequality, we have that for all j, $TSP_j \leq TSP_0$ where TSP_0 is the cost of an optimal solution to the LP, $\mathcal{T}(N \cap R)$. As shown before, $\mathbf{z}_0(R) \leq c_D(R)$. Hence, $\mathbf{x}^*(R) \leq \mathbf{z}_0(R) + \sum_{j \geq 1 : |V_j \cap R| \neq 0} TSP_0 \leq c_D(R) + (\log_2(|R|) - 1)TSP_0 \leq \log_2(|R|)c_D(R)$. The last inequality is true because the linear program optimal value is a lower bound on the integer optimal value. Since, $R \subset N$, $|R| \leq n - 1$. So, for any $R \subset N$, we have $\mathbf{x}^*(R) \leq \log_2(|N| - 1)c_D(R)$.

From the above two paragraphs, we deduce that $\mathbf{x}^* \in \Re^N$ is a $\log_2(|N| - 1)$- approximate fair cost allocation vector for the asymmetric traveling salesman game (N, c_D).

As for the running time of the algorithm, to find a minimum cost cycle cover there is $O(|N|^3)$ algorithm due to [4]. Also, as mentioned in Theorem 3, finding a fair cost allocation vector for a cycle cover game takes $O(|N|^3)$ time. The while–loop is executed at most $\log_2(|N|) - 1$ times, where $|V_{j+1}| \leq |V_j|/2$ with $j = 0, 1, \ldots, k$ where $|V_k| = 1$ and $V_0 = N$. Thus the total running time of the algorithm is $O(\sum_{i=0}^{k} (|N|/2^i)^3) = O(|N|^3)$. □

Since it is NP–*hard* to approximate 3SAT4 to within a certain constant $c > 1$ [1] and by Theorem 2, we have:

Theorem 5. *Let (N, c_D) be an asymmetric traveling salesman game, with D satisfying triangle inequality. There exists an $\epsilon_0 > 1$ such that it is* NP–hard *to decide whether ϵ_0–Core(N, c_D) is empty or not. In other words, it is* NP–hard *to find an ϵ_0–approximate fair cost allocation vector for the game.* □

Acknowledgements

We thank the anonymous referees for useful comments.

References

1. P. Berman, M. Karpinski, and A. D. Scott. Approximation hardness and satisfiability of bounded occurrence instances of SAT. Technical Report TR03-022, ECCC, 2003.
2. C. G. Bird. On cost allocation for a spanning tree: a game theoretic approach. *Networks*, 6:335–350, 1976.
3. X. Deng, T. Ibaraki, and H. Nagamochi. Algorithmic aspects of the core of combinatorial optimization games. *Math. Oper. Res.*, 24:751–766, 1999.
4. J. Edmonds and L. Johnson. Matching: A well-solved class of integer linear programs. In *Proceedings of Calgary International conference on combinatorial structures and their applications, Gordon and Breach*, pages 89–92, 1970.
5. L. Engebretsen. An explicit lower bound for TSP with distances one and two. *Algorithmica*, 35(4):301–319, 2003.
6. U. Faigle, S. P. Fekete, W. Hochstättler, and W. Kern. On approximately fair cost allocation in Euclidean TSP games. *OR Spektrum*, 20:29–37, 1998.
7. U. Faigle and W. Kern. On the core of ordered submodular cost games. *Math. Program.*, 87:483–499, 2000.
8. A. Frieze, G. Galbiati, and F. Maffioli. On the worst–case performance of some algorithms for the asymmetric travelling salesman problem. *Networks*, 12:23–39, 1982.
9. M. X. Goemans and M. Skutella. Cooperative facility location games. In *Proc. 11th SODA*, pages 76–85, 2000.
10. D. Granot, H. Hamers, and S. Tijs. On some balanced, totally balanced and submodular delivery games. *Math. Program.*, 86:355–366, 1999.
11. D. Granot and G. Huberman. Minimum cost spanning tree games. *Math. Program.*, 21:1–18, 1981.
12. H. Kaplan, M. Lewenstein, N. Shafrir, and M. Sviridenko. A 2/3 approximation for maximum asymmetric TSP by decomposing directed regular multi graphs. In *Proc. of the 44th Annual IEEE Symposium on Foundations of Computer Science*, 2003.
13. A. Kolen. Solving covering problems and the uncapacitated plant location algorithms. *Eur. J. Oper. Res.*, 12:266–278, 1983.
14. H. W. Kuhn. The Hungarian method for the assignment problem. *Nav. Res. Logist. Q.*, 2:83–97, 1955.
15. Y. Okamoto. Traveling salesman games with the Monge property. *Disc. Appl. Math.*, 138:349–369, 2004.
16. C. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Math. Oper. Res.*, 18:1–11, 1993.
17. J. A. M. Potters, I. J. Curiel, and S. H. Tijs. Traveling salesman games. *Math. Program.*, 53:199–211, 1992.
18. L. Shapley and M. Shubik. The assignment game I: the core. *Int. J. Game Theory*, 1:111–130, 1972.
19. A. Tamir. On the core of a traveling salesman cost allocation game. *Oper. Res. Lett.*, 8:31–34, 1988.
20. C. A. Tovey. A simplified NP-complete satisfiability problem. *Disc. Appl. Math.*, 8:85–89, 1984.

Rounding of Sequences and Matrices, with Applications

Benjamin Doerr, Tobias Friedrich, Christian Klein, and Ralf Osbild

Max-Planck-Institut für Informatik, Saarbrücken, Germany

Abstract. We show that any real matrix can be rounded to an integer matrix in such a way that the rounding errors of all row sums are less than one, and the rounding errors of all column sums as well as all sums of consecutive row entries are less than two. Such roundings can be computed in linear time. This extends and improves previous results on rounding sequences and matrices in several directions. It has particular applications in just-in-time scheduling, where balanced schedules on machines with negligible switch over costs are sought after. Here we extend existing results to multiple machines and non-constant production rates.

1 Introduction

In this paper, we analyze a rounding problem with connections to different areas in discrete mathematics, computer science, and operations research. Roughly speaking, we show that any real matrix can be rounded to an integer one in such a way that the rounding errors of all row and column sums are less than one, and the rounding errors of all sums of consecutive row entries are less than two.

Let m, n be positive integers. For some set S, we write $S^{m \times n}$ to denote the set of $m \times n$ matrices with entries in S. For real numbers a, b let $[a..b] := \{z \in \mathbb{Z} | a \le z \le b\}$.

Theorem 1. *Let $X \in \mathbb{R}^{m \times n}$ having integral column sums. Then there is a $Y \in \mathbb{Z}^{m \times n}$ such that*

$$\forall j \in [1..n] : \sum_{i=1}^{m} (x_{ij} - y_{ij}) = 0,$$

$$\forall b \in [1..n], i \in [1..m] : \Big| \sum_{j=1}^{b} (x_{ij} - y_{ij}) \Big| < 1.$$

Such a matrix Y can be computed in time $O(mn)$.

It is easy to see that the second condition implies that for all $a, b \in [1..n]$ and $i \in [1..m]$ we have $|\sum_{j=a}^{b} (x_{ij} - y_{ij})| < 2$. Also, the theorem can easily be extended to matrices having arbitrary column sums. See Section 3 for the details.

Theorem 1 extends and improves a number of results from different applications.

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 96–109, 2006.

1.1 Rounding of Sequences

One of the most basic rounding results states that any sequence $x_1, \ldots, x_n$ of numbers can be rounded to an integer one $y_1, \ldots, y_n$ in such a way that the rounding errors $|\sum_{j=a}^{b}(x_j - y_j)|$ are less than one for all $a, b \in [1..n]$. Such roundings can be computed efficiently in linear time by a one-pass algorithm resembling Kadane's scanning algorithm (described in Bentley's Programming Pearls [4]). Extensions in different directions have been obtained in [9,10,13,16,18]. This rounding problem has found a number of applications, among others in image processing [1,17].

Theorem 1 yields a multi-sequence analogue of this result. Assume that we have m sequences $x_1^{(i)}, \ldots, x_n^{(i)}$, $i \in [1..m]$, such that for all $k \in [1..n]$, the k-th terms sum up to at most one (that is, $\sum_{i=1}^{m} x_k^{(i)} \leq 1$). Then we may simultaneously round the sequences such that (i) all errors $|\sum_{j=a}^{b}(x_j^{(i)} - y_j^{(i)})|$ are less than two and (ii) no two sequences have a 1 in the same position, that is, $y_j^{(i_1)} = y_j^{(i_2)} = 1$ implies $i_1 = i_2$.

While we solve this problem in linear time, one has to be more careful than in the one-dimensional case. A simple greedy algorithm may produce a rounding error of $\Omega(\log m)$ as shown in Section 5.1.

1.2 Linear Discrepancy in More Than Two Colors

Let $k \in \mathbb{N}_{\geq 2}$. Denote by E_k the set of the k unit vectors in $\mathbb{R}^k$ and by $\overline{E}_k$ the convex hull of E_k. In other words, $\overline{E}_k = \{v \in [0,1]^k \mid \|v\|_1 = 1\}$. Let $\mathcal{H} = (X, \mathcal{E})$ be a hypergraph. The linear discrepancy problem of $\mathcal{H}$ in k colors is to find for given mixed coloring $p : X \to \overline{E}_k$ a pure coloring $q : X \to E_k$ such that

$$\operatorname{lindisc}(\mathcal{H}, p, q) := \max_{E \in \mathcal{E}} \left\| \sum_{x \in E} (p(x) - q(x)) \right\|_\infty$$

is small. The linear discrepancy of $\mathcal{H}$ in k colors is $\operatorname{lindisc}(\mathcal{H}, k) := \max_p \min_q \operatorname{lindisc}(\mathcal{H}, p, q)$. This notion introduced in [11] extends the classical linear discrepancy notion (see e.g. Beck and Sós [3]), which refers to two colors only.

Let $\mathcal{H}_n$ be the hypergraph of intervals in $[n]$, that is, $\mathcal{H}_n = ([n], \{[a..b] \mid a, b \in [n]\}$. Then Theorem 2, a slight variant of Theorem 1, shows $\operatorname{lindisc}(\mathcal{H}_n, k) < 2$ for all n and k. Theorem 4 shows that for all $k \geq 3$ and all n, $\operatorname{lindisc}(\mathcal{H}_n, k) \geq 1.5 - 6n^{-1/2}$. The lower bound shows that the bound $\operatorname{lindisc}(\mathcal{H}_n, k) < 1$ only holds for $k = 2$. Note that $\mathcal{H}_n$ is a unimodular hypergraph, and that we have $\operatorname{lindisc}(\mathcal{H}, 2) < 1$ for all unimodular hypergraphs.

1.3 Baranyai's Rounding Lemma and Applications in Statistics

Baranyai [2] used a similar rounding result to obtain his famous results on coloring and partitioning complete uniform hypergraphs. He showed that any matrix can be rounded in a way that the errors in all rows, all columns and the whole matrix are less than one. He used a formulation as flow problem to prove this statement.

Independently, this result was obtained by Causey, Cox and Ernst [6]. In statistics, there are two applications for such rounding results [8]. Note first that instead of rounding to integers, our results also applies to rounding to multiples of any other base (e.g., whole multiples of one percent). This can be used in statistic to improve the readability of data tables. A second reason to apply such rounding procedures is confidentiality. Frequency counts that directly or indirectly disclose small counts may permit the identification of individual respondents. In this case, rounding to multiples of e.g. 10 can prevent such risks. However, in both applications one would like to have that rounding errors in columns and rows are small. This allows to use the rounded matrix to obtain information on the row and column totals.

Our result allows to retrieve further reliable information from the rounded matrix, namely also on the sums of consecutive elements in rows. Such queries make sense if there is a linear ordering on statistical attributes. Here is an example. Let x_{ij} be the number of people in country i that are j years old. Say Y is such that $\frac{1}{1000}Y$ is a rounding of $\frac{1}{1000}X$ as in Theorem 1. Now $\sum_{j=20}^{40} y_{ij}$ is the number of people in country i that are between 20 to 40 years old, apart from an error of less than 2000. Note that such guarantees are not provided by the results of Baranyai and Causey, Cox and Ernst.

Also, our result is algorithmically highly efficient. Both Baranyai, who was not interested in algorithmic issues, and Causey, Cox and Ernst used a reduction of the rounding problem to a flow or transportation problem. Though such problems can be solved relatively efficiently, our linear time solution clearly beats their runtimes.

1.4 Flexible Transfer Line Scheduling

Surprisingly, our matrix rounding problem remains non-trivial if all columns are equal. This problem occurs as a scheduling problem. In the *flexible transfer line scheduling problem* we try to produce m different goods on a single machine in a balanced manner. We know the demands $d_i \in \mathbb{N}$, $i \in [1..m]$, for each good in advance. We assume that our machine (typically a mixed-model assembly line) can produce any good in one unit of time. Furthermore, there are no switch-over costs, that is, we may change from one product to another at no cost.

Our goal is to design a production schedule for $n = \sum_{i=1}^{m} d_i$ time steps such that exactly d_i units of product i are produced. Moreover, at any time and for any product we want our production rate to be close to the average rate $r_i = d_i/n$: After j time steps, we hope to have produced jr_i units of product i. Such production lines are a central part of many just-in-time systems, see e.g. Monden's work [14,15] on Toyota's production system.

Denote by p_{ij} the number of units of product i produced up to time step j. In the *maximum deviation just-in-time scheduling problem* (MDJIT), our aim is to keep the maximum deviation of these production numbers from the aimed at values jr_i small. In other words, we are looking for a schedule minimizing $\max\{|p_{ij} - jr_i| \mid i \in [1..m], j \in [1..n]\}$.

For this problem, Steiner and Yeomans [19] as well as Brauner and Crama [5] give a number of interesting results. In particular, they show that the MDJIT can be solved with maximum error less than one. Via Theorem 1, we extend this result to significantly more general settings. (i) We allow non-constant production rates. Instead of only prescribing that the total production of d_i units of product i ideally should be obtained by producing r_i units in each time step, we allow arbitrary aimed at production rates r_{ij} for each product i and time step j. Of course, $\sum_{i=1}^{m} r_{ij}$ should be one for each time step since we assumed that we may produce a single item each time. This generalized setting makes sense if we know or expect changing demands over a period of time.

(ii) We also allow the use of more than one machine. If we have k machines, we may simply use larger rates satisfying $\sum_{i=1}^{m} r_{ij} = k$. In fact, we are quite flexible in this respect. We may use a different number of machines each time step, that is, have $\sum_{i=1}^{m} r_{ij} = k_j$ with different k_j. We may also have non-integral k_j and in this case use between $\lfloor k_j \rfloor - 1$ and $\lceil k_j \rceil$ machines.

1.5 Lower Bounds

We also present a non-trivial lower bound for the error in arbitrary intervals. Earlier works only regarded errors in initial intervals $[1..t]$. From the view-point of balanced schedules approximating average expected demands, it also makes sense to investigate errors in arbitrary intervals. For upper bounds, the simple triangle inequality argument of Lemma 5 extends any upper bound for initial intervals to twice this bound for arbitrary intervals. For lower bounds, things are more complicated. In particular, the example of Brauner and Crama [5] showing a lower bound of $1 - 1/m$ for initial intervals yields no better bound for arbitrary intervals. We present a three product instance (in the simple model with constant rates and one machine) such that any schedule produces an error of at least $1.5 - \varepsilon$. Note that this also yields an error of $0.75 - \varepsilon$ for initial intervals, that cannot be derived from existing works.

2 The Algorithm

In this section, we present an algorithm solving the matrix rounding problem of Theorem 1. For a region $R \subseteq [1..m] \times [1..n]$, the rounding error in R is $|\sum_{(i,j)\in R}(x_{ij} - y_{ij})|$. Our aim is to achieve low rounding errors in all columns and in all intervals of rows. Note that by subtracting integer part, we may always assume that $X \in [0,1)^{m\times n}$.

We denote by X_i and X^j the i-th row and j-th column of X, respectively. We define the partial sums $s_{ij} := \sum_{\ell=1}^{j} x_{i\ell}$ for all $i \in [1..m]$ and $j \in [1..n]$.

2.1 Basic Algorithm

Here we consider the *restricted* problem with uniform column sums $\|X^j\|_1 = 1$ for all $j \in [1..n]$. Note that in this case each column of the rounded matrix Y

contains just a single 1. The solution to this special problem is later on called *basic algorithm*.

First we give a motivation for the solution. By Lemma 5, it suffices to keep the errors

$$\left|\sum_{j=1}^{b}(x_{ij} - y_{ij})\right|, \quad \forall i \in [1..m],\ \forall b \in [1..n], \tag{1}$$

small in all initial intervals. For the moment, consider a single row $i \in [1..m]$. The idea is to place 1s into Y_i between the row indices where the partial sums of row X_i exceed the next integral values at that time. Formally, we require to place the k-th 1 in row i onto position y_{ij}, where j is some column index in the range $I_i^k := [a_i^k..b_i^k]$ with limits $a_i^k := \min\{j \in [1..n] \mid k-1 < s_{ij}\}$ and $b_i^k := \max\{j \in [1..n] \mid s_{ij} < k \vee (s_{ij} = k \wedge x_{ij} \neq 0)\}$. We call I_i^k the *k-th index interval of row i*. One particularity of this definition is, that no 1 is placed onto a 0 (say $x_{ij} = 0$), if the row sum s_{ij} is integral. This way, all errors in Equation (1) are less than 1.

The algorithm works as follows. The columns of Y are computed successively, Y^j at time $j \in [1..n]$, that is, we have to place a single 1 into Y^j. To select an appropriate position in column Y^j, we regard the set C^j of all index intervals that contain j and whose corresponding entries in Y are still zeros, i.e., $I_i^k \in C^j$, if and only if $j \in I_i^k$ and $y_{ih} = 0$ for all $h \in I_i^k$, $h < j$. Now, C^j contains implicitly all the positions where the 1 could be placed. From those we choose the position ℓ that belongs to the earliest ending interval $[a_\ell..b_\ell]$ of C^j. (In case of a tie we choose the uppermost row.) Then we set column Y^j to the ℓ-th canonical unit column vector, i.e., $y_{\ell j} = 1$. Then we proceed with Y^{j+1} in the same way.

The index intervals I_i^k can be computed as follows. The *initial step* of the algorithm is to determine the limits a_i^1 and b_i^1 of the intervals I_i^1 for all rows X_i, $i \in [1..m]$. For that purpose, each partial row sum is computed up to the first entry where the sum is no longer smaller than 1 or until we reach the end of the row. (The latter case is indicated by any index larger than n.) The values $a_i := \min(I_i^1)$, $b_i := \max(I_i^1)$ and $s_i := s_{i,b_i}$ are stored in three arrays of length m each. With this information we compute the first column Y^1. Each time after we have placed a 1 in Y, an *update step* is necessary, because then the demand of a current index interval for a 1 is just satisfied. Hence we replace this interval by its succeeding interval $I_i^{\lceil s_i \rceil + 1}$. This can be done similar to the initial step. We continue computing the partial row sum of X_i up to the first entry where the sum is no longer smaller than the next integral value (which is $\lceil s_i \rceil + 1$) or until we reach the end of the row. As before the current values of the interval limits and the sum so far are stored in the arrays.

COMPUTEROUNDING($X \in [0,1]^{m \times n}$)

$\triangleright$ Initialization
for $i \leftarrow 1$ **to** m
  do $s[i] \leftarrow 0$
    $b[i] \leftarrow 0$

```
            (a[i], b[i], s[i]) ← GetNextInterval(i)
    ▷ Main Loop
    for j ← 1 to n
        do C ← {i ∈ [1..m] | j ∈ [a[i]..b[i]]}
            ℓ ← argmin_{i∈C} b[i]
            Y^j ← ℓ-th unit column vector
            (a[ℓ], b[ℓ], s[ℓ]) ← GetNextInterval(ℓ)
    return Y ∈ {0,1}^{m×n}

GetNextInterval(i)
    j ← b[i] + 1
    while j ≤ n and x_ij = 0
        do j ← j + 1
    if j > n
      then return (n + 2, n + 2, s[i])
    a[i] ← j
    k ← ⌈s[i]⌉ + 1
    while s[i] + x_ij ≤ k
        do s[i] ← s[i] + x_ij
            if s[i] = k
              then return (a[i], j, s[i])
            j ← j + 1
            if j > n
              then return (a[i], j, s[i])
    return (a[i], j − 1, s[i])
```

2.2 Time and Space Complexity

For the time being we ignore the calls of GetNextInterval in the analysis of the runtime. Then the initialization loop has runtime $\Theta(m)$ and the main loop, which is executed exactly n times, needs $\Theta(m)$ time for each of the three non-trivial assignments. Together that takes $\Theta(mn)$ time.

It remains to add the time spend in GetNextInterval. Be aware that the row index i never changes within this procedure. Hence its total runtime can be estimated by multiplying the maximal time spend in a single row X_i by m. Each of the commands in GetNextInterval can be executed in constant time except the while loop. Since this loop successively increases j – which is swapped to $b[i]$ when the procedure returns – $\Theta(n)$ time is needed for each row X_i. It follows that the runtime of the entire algorithm is $\Theta(mn)$.

The algorithm only needs to keep track of the m current intervals and the m accumulated row sums. So $\Theta(m)$ space suffices in addition to the space needed for input and output.

2.3 Correctness

To show that our algorithm returns a valid solution, we have to show that (i) each column vector Y^j contains *exactly* one 1 and (ii) each index interval gets assigned *exactly* one column with 1. For this it will be convenient to assume integrality of the row sums, i.e., $\sum_{j=1}^{n} x_{ij} \in \mathbb{N}$ for all i. This can by achieved by adding additional columns at the end. If the algorithm returns a valid solution even for these columns, it is also correct for the original matrix. Note that it is not necessary to actually compute these additional columns, i.e., they are only needed for the analysis. The following lemma gives the main property of the algorithm. It shows that at each step there are enough unsatisfied intervals to choose from.

Lemma 1. *Let k_{ij} be the number of intervals which have started until column j in the i-th row. Then $\sum_{i=1}^{m} k_{ij} \geq j$ for all $j \in [1..n]$.*

Proof (by induction on j). For $j = 1$ at least one interval has to start due to the norm condition $\sum_{i=1}^{m} x_{i1} = 1$ for the first column. Now assume the lemma has been established until column j. If there are already *more* than j intervals, there is nothing to prove for $j + 1$. So let us assume that there are exactly j intervals so far, that is to say, $\sum_{i=1}^{m} k_{ij} = j$. Since $\sum_{i=1}^{m} s_{ij} = \sum_{i=1}^{m} \sum_{\ell=1}^{j} x_{i\ell} = \sum_{\ell=1}^{j} \sum_{i=1}^{m} x_{i\ell} = \sum_{\ell=1}^{j} 1 = j$, we get $\sum_{i=1}^{m} k_{ij} = \sum_{i=1}^{m} s_{ij}$. With $0 \leq s_{ij} \leq k_{ij}$ and $k_{ij} \in \mathbb{N}$ for all $i \in [1..m]$, it follows that $S^j = K^j$ and hence $S^j \in \mathbb{N}^m$. This means that all intervals have ended until column j. So at least one interval has to start at position $j + 1$, analogously to the start of the induction base. So $\sum_{i=1}^{m} k_{(i+1),j} \geq \sum_{i=1}^{m} k_{ij} + 1 \geq j + 1$. □

That there is (i$_{\leq 1}$) *no column with more than one* 1 is guaranteed by the algorithm as it chooses the uppermost 1 in the case that there are two closest ending intervals at one time. Due to Lemma 1 the algorithm has passed at least j intervals till the j-th column and has by construction satisfied only $j - 1$ of them. Therefore the algorithm can always satisfy at least one interval and will (i$_{\geq 1}$) *not return any empty column.*

Also (ii$_{\leq 1}$) *no interval will get more than one* 1, because a 1 is only assigned to unsatisfied intervals. We furthermore know $\|X^j\|_1 = 1$ for all columns j and hence $\sum_{j=1}^{n} \sum_{i=1}^{m} x_{ij} = n$. The integrality assumption of the row sums gives that we have exactly n intervals overall. Since each column contains exactly one 1, we have assigned n 1s to intervals. Due to the pigeonhole principle there is (ii$_{\geq 1}$) *no interval with no assigned* 1 because there is no interval with more than one 1.

2.4 Error Bounds

Lemma 2. $\left| \sum_{\ell=1}^{j} (x_{i\ell} - y_{i\ell}) \right| < 1$ *for all $i \in [1..m]$ and $j \in [1..n]$.*

Proof. x_{ij} belongs to the k_{ij}-th interval in the i-th row, that is, to $I_i^{k_{ij}}$. The algorithm assigns to each interval exactly one 1 (cf. Section 2.3). So depending on whether the 1 that corresponds to $I_i^{k_{ij}}$ is in some column at most j or later, $\sum_{\ell=1}^{j} y_{i\ell}$ is either $k_{ij} - 1$ or k_{ij}, respectively. Hence we have $k_{ij} - 1 < \sum_{\ell=1}^{j} x_{i\ell} \leq k_{ij}$ as well as $k_{ij} - 1 \leq \sum_{\ell=1}^{j} y_{i\ell} \leq k_{ij}$, where the second sum equals k_{ij} if the first sum does. This shows $\left| \sum_{\ell=1}^{j} x_{i\ell} - \sum_{\ell=1}^{j} y_{i\ell} \right| < 1$. □

Lemma 3. $\left|\sum_{j=a}^{b}(x_{ij}-y_{ij})\right| < 2$ *for all* $1 \le a \le b \le n$ *and* $i \in [1..m]$.

Proof. This follows immediately from Lemma 2 using Lemma 5. □

The results of the basic algorithm can be subsumed as follows.

Theorem 2. *Let* $X \in [0,1]^{m\times n}$ *with* $\|X^j\|_1 = 1$ *for all* $j \in [1..n]$. *Then there is a* $Y \in \{0,1\}^{m\times n}$ *such that* $\|Y^j\|_1 = 1$ *and*

$$\forall b \in [1..n], i \in [1..m] : \left|\sum_{j=1}^{b}(x_{ij}-y_{ij})\right| < 1.$$

Such a matrix Y *can be computed in time* $O(mn)$.

The following example shows that the above error bound is tight for our algorithm, i.e. that it may indeed generate errors arbitrarily close to two. To see this let $\varepsilon \in (0,1/2)$ and

$$X_\varepsilon := \begin{pmatrix} \varepsilon & 1-\varepsilon/2 & 1-2\varepsilon & \varepsilon/2 & \varepsilon \\ (1-\varepsilon)/2 & \varepsilon/4 & \varepsilon & 1/2-\varepsilon/4 & (1-\varepsilon)/2 \\ (1-\varepsilon)/2 & \varepsilon/4 & \varepsilon & 1/2-\varepsilon/4 & (1-\varepsilon)/2 \end{pmatrix}.$$

This yields the index intervals [1..1] and [2..5] for the first row, and [1..3] and [4..5] for the second and third row. Hence the algorithm puts the first 1 into row one, followed by 1s into row two and three. This yields an error of $(1-\varepsilon/2)+(1-2\varepsilon) = 2-5\varepsilon/2$ in the interval [2..3] in the first row.

2.5 Naïve Generalization

We now show that the basic algorithm of Section 2.1 can be utilized for input matrices with arbitrary column sums $\|X^j\|_1 = c_j \in \mathbb{N}$ for $j \in [1..n]$. In this case, the output matrix $Y \in \mathbb{N}^{m\times n}$ has to satisfy $\|Y^j\|_1 = c_j$. The error on arbitrary intervals is still at most two. First we show how to reduce this generalization to the unitary problem and solve it with the basic algorithm in $\Theta(m^2 n)$ time. We then modify the algorithm in such a way that it can handle the general problem directly in time $\Theta(mn)$. Note that we can still assume $x_{ij} \in [0,1)$ (and hence $c_j < m$) for all $j \in [1..n], i \in [1..m]$ as discussed in Section 2.

A simple way to solve the general problem is to preprocess the input by expanding each vector X^j into c_j identical vectors $\tilde{X}^{\ell_j}, \ldots, \tilde{X}^{\ell_j+c_j-1}$ each of the form $(x_{1j}/c_j, \ldots, x_{mj}/c_j)^T$. With this preprocessing we obtain a new matrix $\tilde{X}$ having $\sum_{j=1}^{n} c_j$ columns, each having sum one. The basic algorithm applied to $\tilde{X}$ yields a matrix $\tilde{Y}$ with errors at most two on arbitrary intervals.

In a postprocessing step we then condense for each $j \in [1..n]$ the c_j output vectors $\tilde{Y}^{\ell_j}, \ldots, \tilde{Y}^{\ell_j+c_j-1}$ to one vector Y^j (having column sum c_j) by summing them up. This yields a solution Y to the original problem. Since all intervals

$[a..b] \subseteq [1..n]$ of the general problem correspond to an interval $[\ell_a..(\ell_b + c_b - 1)]$ of the expanded problem, Y satisfies the properties of Theorem 1.

Observe that this approach may produce entries of value two in the solution. This can happen if an unsatisfied interval ends in the expansion of an input vector and the following index interval ends "close enough" after this expansion. The behavior of the expanding algorithm and the solution it computes can be characterized as follows.

Lemma 4. *Let $\hat{c}_j, j \in [1..n]$, be the number of index intervals that end in (or directly after) the expansion of X^j and are not satisfied before the expansion.*

(a) No index interval is fully contained in the expansion.
(b) $\hat{c}_j \leq c_j$.
(c) The basic algorithm applied to the expanded matrix will first satisfy the $\hat{c}_j$ unsatisfied intervals ending in the expansion. If $\hat{c}_j < c_j$ it will then satisfy the $c_j - \hat{c}_j$ first ending unsatisfied intervals (all of them ending after the expansion).

Proof. The first claim follows since all entries are smaller than one, the second claim follows directly from the correctness of the basic algorithm.

For the third claim observe that there are two types of unsatisfied intervals in the expansion: those ending in (or directly after) it and those continuing afterward. As argued for the second claim, the unsatisfied intervals ending in the expansion are satisfied by the algorithm. Furthermore, all other crossing intervals end after the expansion and hence later than these $\hat{c}_j$ intervals. Thus the algorithm will distribute the remaining 1s to these intervals. □

2.6 Linear Time Generalization

Since expanding X and running the basic algorithm worsens the runtime, we now give an algorithm that simulates this approach and needs nothing more than the basic algorithm of Section 2.1. To achieve this the algorithm has to satisfy c_j intervals instead of just a single one in each step $j \in [1..n]$. According to Lemma 4(c), this can be done in two distribution steps: First identify the $\hat{c}_j$ unsatisfied index intervals ending in the expansion of X^j and assign them a 1. Then satisfy the remaining $c_j - \hat{c}_j$ earliest ending index intervals in the data structure. According to Lemma 4(a) it is not necessary to update and search the data structure after each assigned 1. Instead this can be postponed until the end of each distribution step.

The first distribution step can be done in time $\Theta(m)$ by scanning the data structure once and extracting the $\hat{c}_j$ just ending intervals. Then 1 is added to the entries in Y^j corresponding to those index intervals and their consecutive index intervals are added to the data structure.

For the second distribution step we first extract the $(c_j - \hat{c}_j)$-th earliest ending interval. This too is possible using $\Theta(m)$ time (see e.g. Chapter 10, Medians and Order Statistics, in Cormen et al. [7]). Knowing this interval, the algorithm can locate the other $(c_j - \hat{c}_j) - 1$ earliest ending intervals by just doing a pass over the data structure, again taking $\Theta(m)$ time. Finally, as after the first step, we add 1 to each entry in Y^j corresponding to those index intervals and update the data structure by adding their consecutive index intervals.

Since each update of the data structure takes constant time, the generalized algorithm still needs time $\Theta(mn)$.

The only detail still missing is how to detect if an interval would end inside the expansion of a column X^j and how to compare the endpoints of index intervals ending in the same expansion. For this, first consider the unexpanded input. Let $x_{i,j-1}$ be the last entry belonging to the k-th interval. Then $s_{i,j-1} \leq k < s_{i,j}$ holds. But in the expanded input, the interval would still have a value of $0 \leq k - s_{i,j-1} < x_{i,j} < 1$ left to cover vectors in $\tilde{X}^{\ell_j}, \ldots, \tilde{X}^{\ell_j + c_j - 1}$ of X^j. Since the expansion of X^j has entries x_{ij}/c_j in the i-th row, the interval would continue for

$$\ell := \left\lfloor \frac{k - s_{i,j-1}}{x_{ij}/c_j} \right\rfloor$$

entries into the expansion of X^j.

Hence the end of each index interval is represented by a tuple (j, ℓ) instead of just by the number j as in the basic algorithm. Interval endpoints can then be compared lexicographically.

All in all we can conclude that Theorem 1 holds.

3 Extensions

In this section, we provide two easy extensions of Theorem 1 that are useful in some of the applications described in the introduction. First, it is easy to see that we immediately obtain rounding errors of less than two in arbitrary intervals in rows. This is supplied by the following lemma.

Lemma 5. *Let Y be a rounding of X such that the errors $|\sum_{j=1}^{b}(x_{ij} - y_{ij})|$ in all initial intervals of rows are at most d. Then the errors in arbitrary intervals of rows are at most $2d$, that is, for all $i \in [1..m]$ and all $1 \leq a \leq b \leq n$,*

$$\left| \sum_{j=a}^{b} (x_{ij} - y_{ij}) \right| \leq 2d.$$

Proof. Let $i \in [1..m]$ and $1 \leq a \leq b \leq n$. Then

$$\begin{aligned} \left| \sum_{j=a}^{b} (x_{ij} - y_{ij}) \right| &= \left| \sum_{j=1}^{b} (x_{ij} - y_{ij}) - \sum_{j=1}^{a-1} (x_{ij} - y_{ij}) \right| \\ &\leq \left| \sum_{j=1}^{b} (x_{ij} - y_{ij}) \right| + \left| \sum_{j=1}^{a-1} (x_{ij} - y_{ij}) \right| \leq 2d. \end{aligned}$$

□

Second, we may extend Theorem 1 to include matrices having non-integral column sums.

Theorem 3. *Let $X \in \mathbb{R}^{m\times n}$. Then there is a $Y \in \mathbb{Z}^{m\times n}$ such that*

$$\forall j \in [1..n] : \Big|\sum_{i=1}^{m}(x_{ij} - y_{ij})\Big| < 2,$$

$$\forall b \in [1..n], i \in [1..m] : \Big|\sum_{j=1}^{b}(x_{ij} - y_{ij})\Big| < 1.$$

Such a matrix Y can be computed in time $O(mn)$.

Proof. For an arbitrary matrix X, we add an extra row taking what is missing towards integral column sums: Let $\tilde{X} \in [0,1)^{(m+1)\times n}$ be such that $\tilde{x}_{ij} = x_{ij}$ for all $i \in [1..m]$, $j \in [1..n]$, and $\tilde{x}_{m+1,j} = \lceil\sum_{i=1}^{m} x_{ij}\rceil - \sum_{i=1}^{m} x_{ij}$ for all j.

Clearly $\tilde{X}$ has integral column sums. Using Theorem 1, we can compute a rounding $\tilde{Y} \in \{0,1\}^{(m+1)\times n}$ of $\tilde{X}$ as described in Theorem 1. Note that there are no rounding errors in the columns, i.e., we have $\sum_{i=1}^{m+1} \tilde{y}_{ij} = \sum_{i=1}^{m+1} \tilde{x}_{ij}$ for all $j \in [1..n]$.

Define $Y \in \{0,1\}^{m\times n}$ by $y_{ij} = \tilde{y}_{ij}$ for all $i \in [1..m]$, $j \in [1..n]$. Now the errors in the columns are $|\sum_{i=1}^{m}(x_{ij} - y_{ij})| = |\tilde{x}_{m+1,j} - \tilde{y}_{m+1,j}|$. By Lemma 5, all single entry rounding errors $|x_{ij} - y_{ij}|$ are less than two, proving the first set of inequalities.

The errors in initial intervals in row 1 to m naturally remain unchanged, proving the second set of inequalities. □

4 Lower Bounds

We present a new lower bound for the matrix rounding problem. Theorem 4 shows that there are $3 \times n$ matrices such that any rounding has an error of $1.5-\varepsilon$ in *arbitrary intervals*. Via a triangle inequality argument similar to Lemma 5, this matrix also yields an error of $0.75 - \varepsilon$ in *initial intervals*. The latter is particularly interesting for the MDJIT problem (see Section 1.4), where Steiner and Yeomans [19] showed a lower bound of $1-1/m$ by means of an $m\times m$ matrix. So for the three-part type MDJIT problem we could raise the lower bound from 2/3 to 3/4.

Theorem 4 (Lower Bound). *For all $\varepsilon \in (0,1)$ there are problem instances $X \in [0,1]^{3\times n}$ such that for all solutions $Y \in \{0,1\}^{3\times n}$ there are $i \in [1..3]$ and $1 \le a \le b \le n$ with $\left|\sum_{j=a}^{b}(x_{ij} - y_{ij})\right| \ge 1.5 - \varepsilon$.*

Proof. Let $n > 1.5/\varepsilon^2$ and $X \in [0,1]^{3\times n}$ with

$$X := \begin{pmatrix} 1-\varepsilon & 1-\varepsilon & & 1-\varepsilon \\ \varepsilon-\varepsilon^2 & \varepsilon-\varepsilon^2 & \cdots & \varepsilon-\varepsilon^2 \\ \varepsilon^2 & \varepsilon^2 & & \varepsilon^2 \end{pmatrix}.$$

Assume that there is a valid solution Y with $\left|\sum_{j=a}^{b}(x_{ij}-y_{ij})\right| < 1.5-4\varepsilon$ for all $i \in [1..3]$ and $1 \le a \le b \le n$. By choice of n, there is at least one column j having a 1 in the third row. Let $p \ge 0$ and $q \ge 0$ be the number of consecutive columns equal to $(1,0,0)^T$ to the left and right of column j, respectively. Thus

$$Y = \begin{pmatrix} \cdots & 0 & 1\ldots1 & 0 & 1\ldots1 & 0 & \cdots \\ & ? & 0\ldots0 & 0 & 0\ldots0 & ? & \\ & ? & \underbrace{0\ldots0}_{p\text{ times}} & \underset{\substack{\uparrow \\ \text{column } j}}{1} & \underbrace{0\ldots0}_{q\text{ times}} & ? & \end{pmatrix}.$$

The rounding error of the interval $[(j-p-1)\,..\,(j+q+1)]$ in the first row is $\left|\sum_{\ell=j-p-1}^{j+q+1}(x_{1,\ell}-y_{1,\ell})\right| = (p+q+3)\cdot(1-\varepsilon)-(p+q) = 3\cdot(1-\varepsilon)-\varepsilon\cdot(p+q)$. Since this is less than $1.5-4\varepsilon$, we have $p+q > (3\cdot(1-\varepsilon)-1.5+4\varepsilon)/\varepsilon = 1.5/\varepsilon+1$. The error of the interval $[j-p..j+q]$ in the second row now is $\left|\sum_{\ell=j-p}^{j+q}(x_{2,\ell}-y_{2,\ell})\right| = (p+q+1)\cdot(\varepsilon-\varepsilon^2) > (1.5/\varepsilon+2)\cdot(\varepsilon-\varepsilon^2) = 1.5+0.5\varepsilon-2\varepsilon^2 > 1.5-4\varepsilon$. This contradicts our assumption. □

5 Alternative Approaches

5.1 Greedy Algorithm

A greedy algorithm traverses the matrix X column by column and sets the 1s in Y only based on the columns previously read. The 1 is assigned to a row i with the highest difference between the accumulated sum s_{ij} and the number of 1s in this row so far. That this may produce a rounding error of $\Omega(\log n)$ can be shown by the following example:

$$X := \begin{pmatrix} \frac{1}{n} & 0 & 0 & \cdots & 0 & 0 \\ \frac{1}{n} & \frac{1}{n-1} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{1}{n} & \frac{1}{n-1} & \frac{1}{n-2} & \cdots & 0 & 0 \\ \frac{1}{n} & \frac{1}{n-1} & \frac{1}{n-2} & \cdots & \frac{1}{2} & 0 \\ \frac{1}{n} & \frac{1}{n-1} & \frac{1}{n-2} & \cdots & \frac{1}{2} & 1 \end{pmatrix} \in [0,1]^{n\times n}$$

The greedy algorithm returns the identity matrix whereby the discrepancy of the interval [1,n-1] in the last row becomes $\left|\sum_{j=1}^{n-1}(x_{n,j}-y_{n,j})\right| = \sum_{j=2}^{n} 1/j = H_n - 1 > \log n - 1$ with H_n being the harmonic number of n.

5.2 Row Intervals

If one accepts a quadratic runtime we can extend Theorem 2 in such a way that not only the initial row intervals, but also the initial column intervals are small:

Theorem 5. *Let $X \in [0,1)^{m\times n}$. Then there is a $Y \in \{0,1\}^{m\times n}$ such that*

$$\forall b \in [1..n], i \in [1..m] : \Big|\sum_{j=1}^{b}(x_{ij} - y_{ij})\Big| < 1,$$

$$\forall b \in [1..m], j \in [1..n] : \Big|\sum_{i=1}^{b}(x_{ij} - y_{ij})\Big| < 1.$$

Such a matrix Y can be computed in time $O(m^2n^2)$.

Proof. Knuth [13] showed how to round a sequence of n real numbers x_i to $y_i \in \{\lfloor x_i \rfloor, \lceil x_i \rceil\}$ such that for two given permutations σ_1 and σ_2, we have $\sum_{i=1}^{k}(x_{\sigma_1(i)} - y_{\sigma_1(i)}) < 1$ as well as $\sum_{i=1}^{k}(x_{\sigma_2(i)} - y_{\sigma_2(i)}) < 1$ for all k. To apply this to our problem of rounding a matrix $X \in \mathbb{R}^{m\times n}$, we first assume integrality of the row and column sums without loss of generality as detailed in Section 3. Consider all elements x_{ij} of the matrix X as the sequence to be rounded. With a permutation σ_1, which enumerates the x_{ij} row by row, Knuth's two-way rounding gives $\sum_{i=1}^{k}\sum_{j=1}^{n}(x_{ij} - y_{ij}) < 1$ for all $k \in [1..m]$. Note that the integrality of the row sums yields by induction $\sum_{i=1}^{k}\sum_{j=1}^{n}(x_{ij} - y_{ij}) = 0$ for all k, which in turn shows for the initial row intervals $\sum_{j=1}^{b}(x_{ij} - y_{ij}) < 1$ for all $b \in [1..n]$ and $i \in [1..m]$. For initial column intervals one can achieve $\sum_{i=1}^{b}(x_{ij} - y_{ij}) < 1$ for all $b \in [1..m]$ and $j \in [1..n]$ in an analogous manner by choosing a permutation σ_2, which enumerates the x_{ij} column by column. His proof employs integer flows in a certain network [12]. On account of this he only achieves a runtime of $O(m^2n^2)$. □

Note that both inequalities in Theorem 5 are actually $\left|\sum(x_{ij} - y_{ij})\right| \le mn/(mn+1)$.

Acknowledgments

The authors wish to thank Pavol Hell for pointing out the relation to controlled rounding.

References

1. T. Asano. Digital halftoning: Algorithm engineering challenges. *IEICE Trans. on Inf. and Syst.*, E86-D:159–178, 2003.
2. Zs. Baranyai. On the factorization of the complete uniform hypergraph. In *Infinite and finite sets (Colloq., Keszthely, 1973; dedicated to P. Erdős on his 60th birthday), Vol. I*, pages 91–108. Colloq. Math. Soc. Jánōs Bolyai, Vol. 10. North-Holland, Amsterdam, 1975.
3. J. Beck and V. T. Sós. Discrepancy theory. In R. Graham, M. Grötschel, and L. Lovász, editors, *Handbook of Combinatorics*, pages 1405–1446. Elsevier, 1995.
4. J. L. Bentley. Algorithm design techniques. *Commun. ACM*, 27:865–871, 1984.

5. N. Brauner and Y. Crama. The maximum deviation just-in-time scheduling problem. *Discrete Appl. Math.*, 134:25–50, 2004.
6. B. D. Causey, L. H. Cox, and L. R. Ernst. Applications of transportation theory to statistical problems. *Journal of the American Statistical Association.*
7. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms.* MIT Press, Cambridge, MA, 1990.
8. L. H. Cox and L. R. Ernst. Controlled rounding. *Informes*, 20(4):423–432, 1982.
9. B. Doerr. Lattice approximation and linear discrepancy of totally unimodular matrices. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 119–125, 2001.
10. B. Doerr. Global roundings of sequences. *Information Processing Letters*, 92:113–116, 2004.
11. B. Doerr and A. Srivastav. Multicolour discrepancies. *Combinatorics, Probability and Computing*, 12:365–399, 2003.
12. L. R. Ford, Jr., and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.
13. D. E. Knuth. Two-way rounding. *SIAM J. Discrete Math.*, 8:281–290, 1995.
14. Y. Monden. What makes the Toyota production system really tick? *Industrial Eng.*, 13:36–46, 1981.
15. Y. Monden. *Toyota Production System.* Industrial Engineering and Management Press, Norcross, GA, 1983.
16. K. Sadakane, N. Takki-Chebihi, and T. Tokuyama. Combinatorics and algorithms on low-discrepancy roundings of a real sequence. In *ICALP 2001*, volume 2076 of *Lecture Notes in Computer Science*, pages 166–177, Berlin Heidelberg, 2001. Springer-Verlag.
17. K. Sadakane, N. Takki-Chebihi, and T. Tokuyama. Discrepancy-based digital halftoning: Automatic evaluation and optimization. In *Geometry, Morphology, and Computational Imaging*, volume 2616 of *Lecture Notes in Computer Science*, pages 301–319, Berlin Heidelberg, 2003. Springer-Verlag.
18. J. Spencer. *Ten lectures on the probabilistic method*, volume 64 of *CBMS-NSF Regional Conference Series in Applied Mathematics.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1994.
19. G. Steiner and S. Yeomans. Level schedules for mixed-model, just-in-time processes. *Management Science*, 39:728–735, 1993.

A Note on Semi-online Machine Covering

Tomáš Ebenlendr[1], John Noga[2], Jiří Sgall[1], and Gerhard Woeginger[3]

[1] Mathematical Institute, AS CR, Žitná 25, CZ-11567 Praha 1, The Czech Republic
{ebik, sgall}@math.cas.cz
[2] Department of Computer Science, California State University, Northridge, CA 91330, USA
jnoga@ecs.csun.edu
[3] Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
gwoegi@win.tue.nl

Abstract. In the machine cover problem we are given m machines and n jobs to be assigned (scheduled) so that the smallest load of a machine is as large as possible. A semi-online algorithm is given in advance the optimal value of the smallest load for the given instance, and then the jobs are scheduled one by one as they arrive, without any knowledge of the following jobs. We present a deterministic algorithm with competitive ratio $11/6 \leq 1.834$ for machine covering with any number of machines and a lower bound showing that no deterministic algorithm can have a competitive ratio below $43/24 \geq 1.791$.

1 Introduction

In the machine cover problem we are given m identical machines and n jobs to be assigned (scheduled) so that the smallest load of a machine is as large as possible.

The motivation for this objective function comes from applications where the jobs correspond to supplies (like fuel tanks) needed to keep the machines alive, and the overall goal is to keep the whole system alive as long as possible. The same objective was studied before for example in [5], where some additional motivation can be found.

Similarly to the classical makespan problem, the ideal schedule is perfectly balanced. Thus the exact solution is NP-hard, and using similar techniques as for makespan scheduling, approximation schemes can be constructed even for uniformly related machines [6,2,1,4].

It is easy to see that in the online setting with jobs arriving one by one, no non-trivial deterministic algorithm is possible [3]. If m jobs with processing times equal to 1 arrive, the algorithm has to assign them to distinct machines, as this may be the whole sequence. Then $m-1$ jobs with processing time m arrive, and the online algorithm achieves objective 1 while the optimum is m.

With this in mind, Azar and Epstein [3] considered semi-online algorithms which are given in advance the value of the optimum. Among other results, they showed that a simple greedy algorithm is $2-1/m$ competitive, this is optimal for $m = 2, 3, 4$ for deterministic algorithms, and no semi-online deterministic algorithm for $m \geq 4$ is better than 1.75-competitive.

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 110–118, 2006.

1.1 Our Results

We focus on semi-online algorithms for large m. We present a deterministic algorithm with competitive ratio $11/6 \leq 1.834$ for machine covering with any number of machines. This is the first semi-online algorithm whose competitive ratio is strictly smaller than 2.

We also present a lower bound showing that no deterministic algorithm can have a competitive ratio below $43/24 \geq 1.791$. This improves the previous lower bound of 1.75 and is reasonably close to the upper bound.

2 Preliminaries

We are given m machines and n jobs with processing times (or size) $p_j \geq 0$. A *schedule* is an assignment of jobs to machines $S : \{1, \dots, n\} \to \{1, \dots, m\}$.

The *load* of machine i is the sum of the processing times of the jobs assigned to that machine, denoted by $L_i = \sum_{j \in S^{-1}(i)} p_j$. A machine is L-*covered* (for a number L) if its load is at least L in the given schedule ($L_i \geq L$).

The objective is to maximize the minimal load of a machine, $\min_i L_i$. An optimal schedule for the given instance I is denoted $OPT(I)$ and its objective value is denoted $L^{\mathrm{OPT}}(I)$.

A semi-online algorithm A is given in advance the value $L^{\mathrm{OPT}} = L^{\mathrm{OPT}}(I)$ (and the value m). Then the jobs of the instance I are scheduled one by one as they arrive, without any knowledge of the following jobs. Its objective on the given instance I is denoted $L^{\mathrm{A}}(I)$. The algorithm is called R-competitive if $L^{\mathrm{OPT}}(I) \leq R \cdot L^{\mathrm{A}}(I)$ for any instance I.

Note that, given the desired competitive ratio R, a semi-online algorithm knows the covering level L^{OPT}/R which it needs to achieve. After some partial sequence, if there exists an assignment with m' of L^{OPT}-covered machines, then the algorithm actually needs to guarantee that it has at least m' of L^{OPT}/R-covered machines. The reason is that the instance can continue with $m - m'$ jobs with $p_j = L^{\mathrm{OPT}}$. Intuitively, this means that if the number of machines is sufficiently large, the exact value of m does not really matter.

Since the value L^{OPT} is known to the algorithm, we may always assume that the instances are rescaled so that L^{OPT} and L^{OPT}/R are convenient numbers (as specified later in the paper).

We call a job *huge* if $p_j \geq L^{\mathrm{OPT}}/R$. Every reasonable algorithm schedules huge jobs on separate machines, because scheduling such a job in any other way wastes the jobs that are assigned to the same machine.

3 The Upper Bound

We analyze our algorithm using an appropriate weight function—a classical technique used for bin packing and related problems.

A weight function $w : \mathcal{R}^+ \to \mathcal{R}^+$ assigns a weight to each job, based on its processing time. The weight of job j is denoted $w_j = w(p_j)$, the total weight of jobs is denoted $W = \sum_j w_j$. Finally, the weight of machine i is defined as

$$W_i = \sum_{j \in S^{-1}(i)} w_j .$$

We illustrate the use of weight functions on a greedy algorithm Init which is known to be $(2-1/m)$-competitive [3]. Assume that $L^{\mathrm{OPT}} = 2-1/m$ (otherwise scale the instance). Fill schedules all jobs greedily on one machine, called an active machine, until it is 1-covered; then it uses a new active machine. As an exception, huge jobs (with $p_j \geq 1$) are always scheduled on a new machine. If no new machine is available, all the remaining jobs are scheduled on the last machine. (This description slightly deviates from [3], however, the behavior is different only when all machines are already 1-covered, so it does not matter for the analysis.)

We define the weight function as $w_j = 2$ for huge jobs (i.e., for jobs with $p_j \geq 1$) and $w_j = p_j$ otherwise. Now every $(2-1/m)$-covered machine has weight $W_i \geq 2-1/m$. Since OPT covers all the machines, it follows that $W \geq 2m-1$. On the other hand, every 1-covered machine generated by Fill has weight $W_i \leq 2$, possibly with the exception of the last machine. Assume that only $m' < m$ machines are 1-covered at the end of the algorithm. Then the 1-covered machines have weight $W_i \leq 2$ each, the last active machine has weight $W_i < 1$, and the remaining machines are empty. Thus the total weight is strictly less than $2m' + 1 \leq 2m - 1$, a contradiction.

To improve upon Fill, we use two active machines in place of a single one. This allows us to avoid the situation when the active machine is almost 1-covered by small jobs and a job of size $1-\varepsilon$ arrives, causing the final load to be close to 2 in Fill.

Theorem 3.1. *There exists a semi-online algorithm for machine cover which is* $11/6$*-competitive.*

Proof. Without loss of generality, we assume that $L^{\mathrm{OPT}} = 11$ (otherwise scale the instance). We design an algorithm A so that each machine is 6-covered.

The weight function and the total weight. We define the weight function as follows:

$$w_j = \begin{cases} 10 & \text{if } p_j \geq 6 \text{ (huge jobs)} \\ \min(5, p_j) & \text{otherwise} \end{cases}$$

Every 11-covered machine has weight $W_i \geq 10$: It contains either a single huge job, or two jobs of weight 5 (with $p_j \in [5, 6)$), or one job of weight 5 and some jobs with $p_j < 5$ and total weight at least 5, or only jobs with $p_j < 5$ of total weight at least 11.

Since OPT has all the m machines 11-covered, the total weight is at least $W \geq 10m$.

The invariants of the algorithm. Our algorithm is designed so that at any time, the total weight of 6-covered machines is at most 10 times their number. In addition, the total weight of jobs on machines that are not 6-covered is strictly less than 10.

Strictly speaking, the invariants may be violated when all the machines but the last one are 6-covered. This final phase of the algorithm needs to be handled separately.

The algorithm. Intuitively, we would like to design the algorithm so that the weight of each machine is at most 10. However, it is not possible to maintain this for each machine. In some cases the algorithm creates pairs of machines with weights at most 9 and 11. The key is to try to create a machine with load (and thus weight) between 2 and 4; upon arrival of a job with $p_j \geq 4$ it is 6-covered with weight at most 9.

The main part of the algorithm is described in Table 1. The algorithm maintains two active machines i and h. All the other machines are at all times either 6-covered or empty.

The leftmost three columns describe four different types of configurations of the algorithm by the conditions on the active machines. The remaining columns describe where a new job is scheduled, depending on its size, and which actions are taken to get back to one of the permitted type of configuration. If a new active machine is requested by the algorithm and none is available, the algorithm enters its final phase described later.

As a rule not included in Table 1, whenever a huge job ($p_j \geq 6$) arrives, it is scheduled on an empty machine, which is 6-covered afterwards. If no empty machine is available, the algorithm enters its final phase described later.

Table 1. The main loop of the 11/6-competitive algorithm

Old configuration			New	Action		New
Label	Active machines		job j	Put j on		config.
INIT	$L_h = 0$	$L_i < 2$	$p_j < 2$	i	if $L_i + p_j < 2$	INIT
					otherwise swap $i \leftrightarrow h$	GOOD
			$p_j \in [2, 4)$	h		GOOD
			$p_j \geq 4$	h		BIG
BIG	$L_h \geq 4$ $W_h \leq 5$	$L_i < 2$	$p_j < 2$	i	if $L_i + p_j < 2$	BIG
					otherwise swap $i \leftrightarrow h$	GOOD
			$p_j \geq 2$	h	close h, get a new active machine h	INIT
GOOD	$L_h \in [2, 4)$	$L_i < 6$	$p_j < 4$	i	if $L_i + p_j < 6$	GOOD
					otherwise close i, get a new active machine i	GOOD
			$p_j \geq 4$	h		SPEC
SPEC	$L_h \geq 6$ $W_h \leq 9$	$L_i < 6$	any	i	if $L_i + p_j < 6$	SPEC
					otherwise close i and h, get new machines i and h	INIT

Initially, the active machines are chosen arbitrarily; they are empty and the configuration is INIT. BIG denotes a configuration in which the active machine h actually always contains a single job with $p_j \in [4, 6)$ (as is easily verified by the inspection of Table 1); this guarantees the condition $W_h \leq 5$. GOOD denotes the safe configuration from the intuitive description above with $L_h \in [2, 4)$. Finally, SPEC is a possible successor configuration of GOOD where h is 6-covered with weight at most 9; we still consider this machine active even though no more jobs are scheduled on it. The condition $W_h \leq 9$ in SPEC follows since h contains a single job with $p_j \in [4, 6)$ and possibly some other jobs with total load and weight less than 4.

It is easily verified that when an active machine is closed in BIG or GOOD configurations, its weight is at most 10. When both machines are closed in SPEC, we have $W_h \leq 9$ and $W_i \leq 11$. Also the active 6-covered machine h in SPEC has $W_h \leq 9$. Summarizing, the invariant concerning the weight of 6-covered machines is always preserved. Finally, note that in each state, the weight of all the not 6-covered machines is less than 10, as required by the second invariant.

The final phase. It remains to describe and analyze the final phase of the algorithm.

If all the machines are 6-covered upon reaching the final phase, then schedule the remaining jobs on any of the machines.

If a single machine is not 6-covered, schedule all the remaining jobs on this machine. By the invariants, the 6-covered machines have total weight at most $10(m-1)$, the total weight of all jobs is $W \geq 10m$, thus after all jobs are scheduled, the last machine has weight at least 10 and thus it is 6-covered.

If two machines are not 6-covered upon reaching the final phase, then the new machine was requested for a huge job. Schedule this huge job on the machine with the smallest load and all the remaining jobs on the remaining not 6-covered machine. Inspecting the possible configurations, the huge job is scheduled on an active machine with load and weight at most 4. Consequently, similarly to the previous case, after all jobs are scheduled, the last machine has weight at least 6 and thus is 6-covered.

In all the cases, at the end all the machines are 6-covered by the semi-online algorithm, and we conclude that the algorithm is 11/6-competitive.

4 The Lower Bound

Theorem 4.1. *Any deterministic semi-online algorithm for machine cover has competitive ratio at least* 43/24.

Proof. Let ε be such that $1/\varepsilon$ is a large integer, let m be sufficiently large ($m = 44 + 6 \cdot 43/\varepsilon$ works). Without loss of generality, assume that $L^{\mathrm{OPT}} = 43$. Assume for a contradiction that there exists semi-online algorithm A with competitive ratio $43/(24+\varepsilon)$. We construct a counterexample, i.e., an instance for which $L^{\mathrm{A}} < 24 + \varepsilon$.

Table 2. The strategy of the adversary in phase 1. The machines marked by star are newly covered (and thus removed from the configuration).

Old configuration	New job	Possible new configurations
$\emptyset$	5	$\{5\}$
$\{5\}$	15	$\{5, 15\}$ $\{5\}, \{15\}$
$\{5, 15\}$	24	$\{5, 15, 24\}^*$, $\emptyset$ $\{5, 15\}, \{24\}$ – the adversary wins
$\{5\}, \{15\}$	9	$\{5\}, \{9, 15\}$ $\{5, 9\}, \{15\}$ – the adversary wins $\{5\}, \{9\}, \{15\}$ – the adversary wins
$\{5\}, \{9, 15\}$	19	$\{9, 15, 19\}^*, \{5\}$ $\{9, 15\}, \{5, 19\}$ – the adversary wins $\{9, 15\}, \{5\}, \{19\}$ – the adversary wins

We formulate the counterexample as a strategy for the adversary, based on how the algorithm A scheduled the jobs so far. The adversary wins the game when it is possible to modify the schedule produced by the algorithm A to get some (possibly suboptimal) schedule which has more 43-covered machines than is the number of $(24 + \varepsilon)$-covered machines of A. Strictly speaking, after this the adversary continues with jobs of size 43 until all the machines are covered.

Finally, we can assume without loss of generality that the algorithm A never schedules a job on any $(24 + \varepsilon)$-covered machine. (A new machine is always available as m is large.)

Throughout the proof, the content of a machine is written in braces as numbers denoting jobs of those sizes In addition, a number in square brackets denotes a set of jobs with this total size. Thus, for example, $\{9, 9, 10, [15]\}$ denotes a machine with total load 43 which contains two jobs of size 9, one job of size 10 and some other jobs.

Phase 0. The instance starts with a sequence of $2 \cdot 43/\varepsilon$ jobs of size 24. The optimum can create $43/\varepsilon$ of 43-covered machines, each containing two of the jobs. Thus at the end of the phase, the algorithm A also has $43/\varepsilon$ machines with two jobs, i.e., {24,24}, as otherwise the adversary wins.

Phase 1. The goal of phase 1 is to make the algorithm A to create $4 \cdot 43/\varepsilon$ machines of form $\{5, 15, 24\}$ or alternatively $2 \cdot 43$ machines $\{9, 15, 19\}$. Table 2 shows the strategy of the adversary for this phase. The table shows only nonempty machines that are not $(24 + \varepsilon)$-covered, or are newly covered (marked by a star). The first column describes possible configurations of the schedule of A in this phase. The second column gives the job submitted by the adversary for each configuration, and the last column describes all the possible configurations of A after the new job is scheduled.

It is easy to verify that all the machines $(24 + \varepsilon)$-covered by the algorithm A so far are also 43-covered.

The adversary stops in the situations marked in the table as winning. If the configuration is $\{5,15\},\{24\}$ or $\{9,15\},\{5,19\}$ or $\{9,15\},\{5\},\{19\}$ then the load on the uncovered machines is more than 43, and the adversary wins by reassigning these jobs on a single 43-covered machine (all the other machines stay as in the schedule of A). In configurations $\{15\},\{9,5\}$ and $\{15\},\{9\},\{5\}$ the adversary submits two additional jobs, one of size 5 and one of size 4. The algorithm A cannot cover another machine, but the adversary can convert the schedule using one $\{24,24\}$ machine to a schedule with two machines $\{24,15,4\}$ and $\{24,9,5,5\}$, so the adversary wins again.

If no such situation is encountered, then the adversary waits until the algorithm A covers $4 \cdot 43/\varepsilon$ machines by jobs $\{5,15,24\}$ or $2 \cdot 43$ machines by jobs $\{9,15,19\}$, and then continues with phase 2. Note that in the final configuration, either there is no non-empty (not covered) machine, or there is one machine $\{5\}$.

Phase 2. During this phase, let i_1 and i_2 be the indices of the two uncovered machines with the largest loads. I.e., L_{i_1} is the maximal load of an uncovered machine.

The phase proceeds in $43/\varepsilon$ rounds. The adversary maintains a rearranged schedule, starting with the schedule of the algorithm A after phase 1. After each round, if the adversary has not yet won, it rearranges some of the machines from the previous phases and the new jobs so that it has as many 43-covered machines as A has $(24+\varepsilon)$-covered. In addition, in each such rearrangement the adversary saves at least one job of size ε.

At the beginning of each round, we have some not covered machines with loads at most 14, containing jobs of size ε and possibly one job of size 5. The not covered machines in the rearranged schedule of the adversary may contain jobs different from the jobs on the machines of A, but the loads are the same.

Now we describe one round of phase 2. The adversary submits jobs of size ε until $L_{i_1} = 24$. If $L_{i_2} \geq 14$, the adversary converts the schedule using one machine $\{24,24\}$ to create two machines $\{24,[19]\}$ and wins. Otherwise $L_{i_2} \leq 14-\varepsilon$ and the adversary submits a job of size $X = 24 - L_{i_2} \geq 10+\varepsilon$. The algorithm A has to create a machine $\{X,[24]\}$, as otherwise the adversary uses the jobs from not covered machines to create one 43-covered machine and wins. Finally, the adversary submits jobs of size ε until $L_{i_1} \geq 5$.

Now we describe how the machines are rearranged. First, if the newly covered machine $\{X,[24]\}$ contains the job of size 5, then this job is exchanged with $5/\varepsilon$ jobs of size ε from machine i_1. At this point, the machine $\{X,[24]\}$ contains only X and jobs of size ε. Next, using this machine and some machines from the previous phases, the adversary uses one of following conversions (see Figure 1 for an illustration of the conversion (1)):

$$\begin{aligned}&\{24,24\}, 4\times\{5,15,24\}, \{X,[24]\}\\ &\qquad\to \{24,5,5,5,[4]\}, 4\times\{24,15,[4]\}, \{24,X,5,[4-\varepsilon]\}, \varepsilon\end{aligned} \tag{1}$$

$$\begin{aligned}&2\times\{24,24\}, 2\times\{9,15,19\}, \{X,[24]\}\\ &\qquad\to 2\times\{24,19\}, 2\times\{24,15,[4]\}, \{9,9,X,[15]\}, [1]\end{aligned} \tag{2}$$

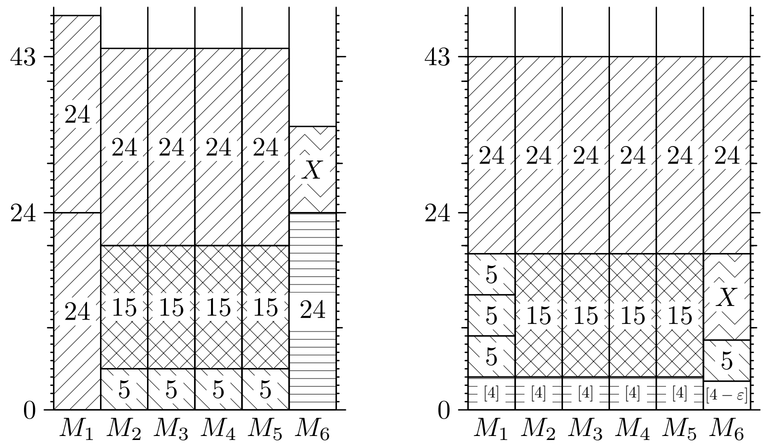


Fig. 1. Conversion (1) of the schedule of the semi-online algorithm (left) to a better schedule (right) with a saved job of size ε. Machine M_1 is from phase 0, machines $M_2, \ldots M_5$ from phase 1, and machine M_6 is created in phase 2.

A , i onv ion, n 43- ov , in in , , l o , v y i l o , n o $(24+\varepsilon)$- ov , in in , , l o A. So , v y y on in wi , no , o n o , , 2.

T n o , in ov in , 0 n 1 n , $43/\varepsilon$ on- v ion (1) o 43 onv ion (2) lw y o i l in , 2. W n , 2 i o l , , v y v l $43/\varepsilon$ jo o i ε. Now , v y , jo o n w 43- ov , in n win .

A , v y v n lly lw y win , w on l , , i no 43/24- o i iv l o i .

Acknowledgments

W l o nony o o ny l o n . T. E n- l n n J. S ll w i lly o y In i ion l R , Pl n No. AV0Z10190503, y In o T o Co S i., P (oj 1M0545 o MŠMT ČR), n n 201/05/0124 o GA ČR.

References

1. N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *J. Sched.*, 1:55–66, 1998.
2. Y. Azar and L. Epstein. Approximation schemes for covering and scheduling on related machines. In *APPROX*, volume 1444 of *Lecture Notes in Comput. Sci.*, pages 39–47. Springer, 1998.
3. Y. Azar and L. Epstein. On-line machine covering. *J. Sched.*, 1:67–77, 1998.

4. L. Epstein and J. Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. *Algorithmica*, 39:43–57, 2004.
5. D. Friesen and B. Deuermeyer. Analysis of greedy solutions for a replacement part sequencing problem. *Math. Oper. Res.*, 6:74–87, 1981.
6. G. J. Woeginger. A polynomial time approximation scheme for maximizing the minimum machine completion time. *Oper. Res. Lett.*, 20:149–154, 1997.

SONET ADMs Minimization with Divisible Paths

L . , E . in[1,⋆] . n A . L vin[2]

[1] Department of Mathematics, University of Haifa, 31905 Haifa, Israel
lea@math.haifa.ac.il

[2] Department of Statistics, The Hebrew University, Jerusalem, Israel
levinas@mscc.huji.ac.il

Abstract. We consider an optical routing problem. SONET add-drop multiplexers (ADMs) are the dominant cost factor in SONET /WDM rings. The number of SONET ADMs required by a set of traffic streams is determined by the routing and wavelength assignment of the traffic streams. In this paper we consider the version where a traffic stream may be divided into several parts and assigned different wavelengths. A specific division may increase or decrease the number of ADMs needed for a given input. Following previous work, we consider two versions. In the arc version, the route of each traffic stream is given as input, and we need to decide on divisions of streams, and then to assign wavelengths so as to minimize the total number of used SONET ADMs. In the chord version, the route is not prespecified, but is assigned by the algorithm, and only after this step the divisions are done and wavelengths are assigned. The previously best known approximation algorithm for the arc version has a performance guarantee of $\frac{5}{4} = 1.25$ whereas the previously best known approximation algorithm for the chord version has a performance guarantee of $\frac{3}{2} = 1.5$. We improve both these results. We present a $\frac{36}{29} \approx 1.24138$-approximation algorithm for the arc version and a $\frac{7}{5} = 1.4$-approximation algorithm for the chord version.

1 Introduction

WDM (W v l n Divi ion M l i l xin)/SONET (Syn ono O i l NET- wo) in o v y iv n wo i i in loy y owin n o l o i . In i i w v l n n- n l i i - SONET in . T y in in i n o i- l - o l i l x (OADM) n SONET - o l i l x (ADM). E v x i i wi x ly on OADM. T OADM n l iv ly o w v l n v x. T , i w v l n o no y ny ffi o o o v x, i OADM llow w v l n o o i lly y v x. T o , in SONET in SONET ADM i i v x i n only i in i o ffi in in i v x. In i - w y o l o ini i in o l o in y SONET ADM .

⋆ Research supported by Israel Science Foundation (grant no. 250/01).

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 119–132, 2006.

E li i o o l l o ini i in o l n o SONET ADM o on o w v l n on in i y. I. ., w v l n i on ll lin o li o ffi . S li i in l [6,5,8,2,4,3]. W l i o o i l . How v , G l l. [5] ill n o ADM n y llowin ffi o lo lly n o on ADM in w v l n o no ADM in iff n w v l n ny in i v x. S ffi i *divided.* C lin n W n [1] i i o l . T y ow llowin ivi ion o ffi y o o o i l ol ion y o o $\frac{1}{3}$, w o o ffi i i . T i jo o iv y o o l . T [1] l o ov NP- n n n oxi ion l o i o o v ion w o in lon in o ffi i i i o n o i . T v ion i ll *arc version* w on v ion i ll *chord version.* Fo v ion [1] n $\frac{5}{4}$- oxi ion l o i n o o v ion i n $\frac{3}{2}$- oxi ion l o i .

Fo i o v i a, b w no y (a, b) i onn a o b, n w no y $\{a, b\}$ n i w n wo v i .

T ARC VERSION OF MINIMIZING THE NUMBER OF ADMS WITH DIVISION (v ion) i n ollow . W iv n E o i l ov v i $0, 1, \ldots, n-1$, w v i o lo wi . A i o $(i, j), (k, l)$ i *non intersecting* i lo wi lon y l $0, 1, \ldots, n-1, 0$ onn i o j n lo wi onn k o l o no ny o y l . A S o i non in in i i o o S i non in in . A *division of an arc* (a, b) i n lo wi (o no in i l) w i onn a n b. A ivi ion o S o i n (l i) o l in o ivi ion o in S. A i l ol ion i i ion o ivi ion o E in o non in in o $E_1, E_2, \ldots, E_p$ (i. ., w n ivi ion o E n l o i ion o i in o non in in). T o o E_i i n o iff n v i o in n oin o o E_i. T o o ol ion i o o o E_i o ll i. T o l i o n ini o i l ol ion.

T CHORD VERSION OF MINIMIZING THE NUMBER OF ADMS WITH DIVISION (o v ion) i n ollow . W iv n o n i (o) E ov v i $0, 1, \ldots, n-1$, w v i o lo - wi . Fi , w n o o i n $\{a, b\}$, i. ., o n o i o i (a, b) o (b, a). L , w o in n in n o (ivi i l) v ion o l w olv .

T NON-DIVISIBLE ARC VERSION n NON-DIVISIBLE CHORD VERSION o o l v i n o v ion o l n o v ion o l , iv ly, w ivi ion o o o i no llow . T non- ivi i l o l w i o in [5,6,8,2,4,3,7]. C lin n W n [2] ovi $\frac{3}{2}$- oxi ion l o i o o non- ivi i l o l . In [4] w on i non- ivi i l v ion o l n o in $\frac{10}{7}$-

oxi ion l o i . A iff n oxi ion l o i o oxi ion io $\frac{10}{7}+\varepsilon$ w iv n y S lo n Z [7]. In [3] w on i non- ivi i l o v ion o l n o in $\frac{10}{7}$- oxi ion l o i w ll (in iff n o).

Fo n (i, j), w n i *length* $\ell(i,j) = j - i$ o n. Fo o ( ), l n o ( ) i o l l n o i . A *chain* i n o n i o l n o $n-1$, n *cycle* i lo i o l n x ly n. W.l.o. w n in E_i o onn o on n (i in o y l). T i i o i in E_i i onn n w n ion E_i o i onn o on n wi o in in i o l o . T o , w o i ion o E in o y l n (o n) in .

W n *the deficiency of a vertex v*, *def*(v), in v ion in ol- lowin w y. L *in*(v) n o in oin o v, n l *out*(v) n o o oin o V. T n, $def(v) = \frac{1}{2}|in(v) - out(v)|$. In o v ion i n y o v x v i i ly o i v, n v n , n $\frac{1}{2}$ o wi . T i n y o o (o) S, $def(S)$ i n o i n i ov ll v i o wi (o) S. T ni ion in [1]. No o o o o C, v l $|C| + def(C)$ i low o n on o i l o o i o in v ion o o l w ivi ion no llow , n l o in ivi i l o l w y in i .

Al o i E l i n Ro n in (ER) w in o in [1] w ll. Fo v ion ER wo ollow : L S o . A ini i o *fake* F w n i o v i wi non o i n y $def(S \cup F) = 0$. T i l in n E l i n (no n ily onn) . Now oo n E l i n o , n ivi i in o $def(S)$ (no n ily v li) in n o i ly o (no n ily v li) y l . In o o v li in n y l , w ly ollowin o . Con i in o y l w i v x x. All in in o y l , on in in v x x, ivi in o wo . S i lly, n (a, b) (x i on lo wi lon in o a o b) i ivi in o (a, x) n (x, b). T i onv E l i n o in o $def(S)$ v li in , n o v li y l .

Fo o v ion ER wo ollow . T S i o . In i F i in o o v i , n o $def(S \cup F) = 0$. W oo n E l i n o , n w o vio l o i in wo o i l i ion o o . On o i i , w o , n w ly ER o v ion. I w own in [1] nnin ER in o i ion , n oo in ol ion o ol ion o o $\frac{3}{2}|S| + def(S)$.

Our results. W iv i ov o n o o v ion o ivi i l o l . In o w o in wo l o i , o w i o w ll on iff n l o .

Fo v ion, w o in PIM n GP-ER in o A -Co in ion (AC). PIM w l y in o in [2] o non- ivi i l o l . I ov y l n n in o inin . I o no ivi . GP-ER, il o in , w ll o in n ily ov (v y ll ov o i lly). T n, i n ER on inin in n . W ov o n n o AC i o $\frac{36}{29}$ n l $\frac{11}{9}$.

Fo o v ion, w o in D-DAG n P-ER in o C o -Co in ion (CC). T l o i D-DAG w l y in o in [2] w i i ll E Avoi n Ro in l o i , n i n in [3]. T i l o i o i n o l in i y li , n n olv i in n o i lly. D-DAG o no ivi ny , in l in i in n o i l ol ion o no n o ivi , own in S ion 3.2. T l o i P-ER o in i o PIM n ER. I ov y l ily (PIM o), n ER (n no IM PIM o) on inin in n . W ov o n n o CC i x ly $\frac{7}{5} = 1.4$.

2 Algorithms for the Arc Version

In i ion w in o l o i G y P o -ER (GP-ER) o v ion n w o in i wi PIM o o l o i AC i own o $\frac{36}{29}$- oxi ion l o i . W o o i o o i l ol ion .

T ollowin l L 3 in [1]. (T oo in [1] i no o l i il o on i ll , in ll v ion o i w ovi o l oo .)

Lemma 1. *For an input arc set E', denote its optimal solution by $OPT_{E'}$. Let a, b, a be a two-arc cycle in E, then $OPT_E = OPT_{E \setminus \{(a,b),(b,a)\}} + 2$.*

By ov l , w i i o i l o o i l ol ion xi l n o wo- y l . T y l n o n o i lly o in o ny li l o i . W i o i l ol ion w wo l li o n ly .

A i l ol ion *SOL* in i ion o in o n E l i n - n o *mega-chains* ollow : W on i o y l n in y *SOL* o in i xili y ov $\{0, 1, \ldots, n-1\}$ w y l loo n in i i o i in v x o i n v x. In i i w n xi l in w i in- o v x l i o - . T inin n ini l o in in i i o v x w o o - i n i in- , ow v x w o in- i n i o - . E in in xili y o on o *mega-chain* in o i in l (y l in in xili y y i o on in in). T o , - in i o o o in . T inin in o i in l o *Eulerian subgraph.* No E l i n o no n o onn . No n o - in in SOL i in n n o SOL, n i o on o ll i l ol ion .

W o i , oo o ollowin l w i , i - in in OPT.

Lemma 2. *W.l.o.g. each mega-chain in OPT has length at most $n-1$.*

2.1 Algorithm GP-ER

W y o n l o i w . T l o i v l o in , n n i li E l i n Ro n in . I i no iffi l o ow i n in olyno i l i . To l o n ly l o i , w n o now x n o - in in OPT w i on i o in l . Sin w o no v i in o ion, w ly l o i o v y o i l v l (w n 0 n $|E|$), n oo ol ion w . T o , in n ly i , w n i n , w i w no MC_1, i nown. W wo $0 \leq \mu_2 \leq 1$ n $0 \leq \mu_3 \leq 1$ w i o i i l .

begin Algorithm GP-ER

Preprocessing phase:

1. R ov xi n o y l o vin x ly wo .
2. Con ollowin i i $B = (R_B, L_B, E_B)$: T i n i , R_B, on in o v i w o in- in (V,E) i n i o - , n l n i L_B on in o v i w o o - in (V,E) i n i in- . Fo n $(u,v) \in E$, o $u \in R_B$ n $v \in L_B$, w n o E_B w n wo o on in v i . T w i o n i i ly l n o o on in . A on ll o i l b- in o in li y MC_1, w n xi w i b- in in B w o n o v x u i wi i n y o i o on in v x in (V,E). Fo in o i l b- in , w ov w n i o on in v i .
3. A lon xi lo w l o (i. ., i i o , i i o l l n i n n i i y l ll l o *triangle*, n o wi i l n i $2n$ n w ll i *invalid triangle*), ov i n l o inv li i n l . I w ov n inv li i n l , w ivi on o i n o in wo y l , o wi wo .
4. A lon xi o y l , ov y l .
5. A lon xi wo- in o l n l $\mu_2 \cdot n$ onn v x w o o - i n i in- , o v x w o in- i n i o - , w ov in o wo .
6. A lon xi in o l n l $\mu_3 \cdot n$ onn v x w o o - i n i in- , o v x w o in- i n i o - , ov in.

Eulerian rounding phase: A ly Al o i ER on inin in n .

end of Algorithm GP-ER

2.2 Algorithm PIM ([2])

In i ion w in o wo l o i IM n PIM iv n in [2], on il on n l w will on l o i . Bo l o i o no ivi . In vio n ly i o l o i [2,4] li y o l in ol ion w o o n o i l ol ion o non- ivi i l v ion o l . T i n i n ly i o no ol o ivi i l o l .

W n l o i I iv M in (IM) ([2]). T l o i in in o v li in o $\mathcal{P}$ ov E o o i x ion. Ini i lly, $\mathcal{P}$ on i o in o w i i n in E. T $\mathcal{F}(\mathcal{P})$ i n ollow : i v x i $\mathcal{P}$, n wo o i v i onn y n i wo o on in in v o on n oin , n y n on n o o v li in. T l o i on $\mathcal{F}(\mathcal{P})$, n i i i no y, n i n xi in $\mathcal{M}$ in $\mathcal{F}(\mathcal{P})$. T n, i i o in o in $\mathcal{M}$ in o lon in. W n o $\mathcal{F}(\mathcal{P})$ i y, $\mathcal{P}$ i v li in n ion i n o . C lin n W n [2] ow oxi ion io o Al o i IM o non- ivi i l v ion o l i in in v l $[\frac{3}{2}, \frac{5}{3}]$. T o n w i ov o in v l $[\frac{8}{5}, \frac{5}{3})$ (i wo- y l no ov o in) n in v l $[\frac{14}{9}, \frac{5}{3})$ (i i o in i o) in [4].

C lin n W n on i v i n o Al o i IM: Al o i P o- I iv M in (PIM) n ollow :

1. P o in : ly ov y l on i in o inin n il no o y l n o in (wo- y l ov).
2. M in : ly Al o i IM o inin .

Fo non- ivi i l v ion o l , y ow Al o i PIM n oxi ion io o o $\frac{3}{2}$, n l $\frac{4}{3}$. T low o n w i ov o $\frac{3}{2}$ in [4]. W ly PIM x ly i i n in [2], n o i i olyno i l i l o i .

2.3 Analysis of PIM and GP-ER

W x n o i l ol ion OPT w i xi l n o wo y l (wi no ivi) n o i - in l n o $n-1$. No - in i in o OPT. S n o i l ol ion xi y L 1 n 2. Con i y l o OPT wi no ivi . Fo $i = 2, 3, 4$, l CY_i n o y l wi i n no ivi OPT , n l CY o l n o y l in OPT wi l v in n no ivi .

Con i n x wo- y l o OPT x ly on ivi . A i o y l , v wo o n i ivi in o x ly wo , i ll n *invalid triangle* o OPT. Cl ly, wo y l lon o E l i n . W no y ICY_3 n o inv li i n l in OPT.

Con i o ll w i lon o E l i n o OPT. F o i ov ll lon o y l o OPT wi no ivi n ll lon o inv li i n l o OPT. T o l l n o inin in i i no y L.

Con i n x o ll w i - in w i on i o in l . W no o l l n o ll y L_1. R ll MC_1 no n o - in wi in l n i i x n o - in wi in l ov y Al o i GP-ER.

In i ion, w ollowin no ion .

- MC - o l n o - in .
- MC_2^ℓ, MC_3^ℓ - n o - in o x ly wo n - iv ly, wi l n in in v l $[\mu_2 n, n-1]$ n $[\mu_3 n, n-1]$, iv ly.
- MC_2^s, MC_3^s - n o - in wi x ly wo n , iv ly, n l n l n $\mu_2 n$ n $\mu_3 n$, iv ly.
- MC_4 - n o - in wi l o .

T ℓ, s i n o lon n o , iv ly.

T l n o E i o $UB_L = (CY_2 + CY_3 + CY_4 + CY) \cdot n + ICY_3 \cdot 2n + L_1 + L + (MC_2^\ell + MC_3^\ell + MC_4) \cdot n + \mu_2 n MC_2^s + \mu_3 n MC_3^s$.

D no y A n o y l ov in 1, $B = MC_1$ - n o - in ov in 2, n L_B o l l n o - in . C n D n o v li i n l n o inv li i n l ov in 3, iv ly, F - n o y l ov in 4, G - n o - in ov in 5, n H - n o - in ov in 6.

T n, y o i li y o 1 in L 1, w $A = CY_2$. Mo ov , 1 i o o i lly, no only in o n o ov y l , in n ov y l x ly on OPT . T o i nno ff n x in ny w y. D o o i li y o 2 in o ov o l l n , w $L_B \geq L_1$. By ni ion, $B = MC_1$.

T y l ion l 3 i li $B + 3C + 3D \geq ICY_3 + CY_3$. T i ol in 3 i no ov n il no lo w l o l . E ov (v li o inv li) i n l oy o (v li o inv li) i n l o OPT. E in ov 2 n oy o on y l .

T y l ion l 4 i li $B + 3C + 3D + 4F \geq ICY_3 + CY_3 + CY_4$. A o ov in in S 2 oy o on on v li o inv li i n l n o - y l . A on o OPT l 2, ov 3 oy o o i i (v li o inv li) i n l o OPT o o y l . S 4 i no ov n il no v li o y l . E ov y l n oy o o o - y l o OPT.

In n x , lon - in o wo ov . T i , nli wo vio on , n o v n i ill xi in in lon - in o OPT wi wo . T on i i i n y o l on n oin o in o o (l o ov l in i o o), w o no ov i - in.

In 2, ov l o in l - in n oy o wo - in lon o MC_2^{ℓ}. Con i n (a, b). I (a, b) o no i - i in ny in o OPT in MC_2^{ℓ}, n i n ill ion o $\frac{1}{2}$ in i n i o a n b, n ion in o n o - in n o ov o o y 1 . O wi , wi o lo o n li y (a, b) i i in - in $a-b-c$ o OPT. In i ollowin wo in n. T - in $a-b-c$ i oy , n on o o , y o $\frac{1}{2}$ in i n y o b, w i n l o o 1 in n o - in o ov . In , i o l ill (a, b) oy l o y l o l ov in 3 o 4. T n in 5 i i il , ov - in n v n o wo o - in o in ov . No 3 n 4 nno n i n y o ny v x, o only w y y n oy - in i y ovin on o i n in ov . T o , ov o 3 y oy o - in , n o 4 o o . W on l $3B + 3C + 3D + 4F + 4G \geq ICY_3 + CY_3 + CY_4 + MC_2^{\ell}$.

In v y l , w ov lon - in o . W n ov 2 y oy o v n o ov l o - in n . T i in on o o in i n y o n oin o i , i n i l in - in o OPT. T i ion in y l ov l i o , in y o no n i n y. A ov in o vio n oy o wo in y in i , n wo i ion l in y in i n y. A ov in o n i i il ff , n oy o in y in i , n wo i ion l in y in i n y. W $3B+3C+3D+4F+4G+5H \geq ICY_3+CY_3+CY_4+MC_2^{\ell}+MC_3^{\ell}$.

W no o o oxi ol ion i o o $|E| + MC$ n o o ivi ion , i on n ni o l n . T i ol in in o y l o i , i only w n i o v i wi non o i n y. T n o in i no l n o i n i , w i i low o n on MC. L UB_L' n o n on o l l n o w i no ov in - o in .

T o , o o oxi ol ion o GP-ER i $GP\text{-}ER \leq |E| + MC + \frac{UB_L'}{n} + D$. To l o , i o n , w wo l li o n n o n on v l $UB_L' + Dn$. Sin w v n o n on UB_L, w n i o low o n on l n o ov . T i l n i l $An + L_B + Cn + 2Dn + Fn + \mu_2 Gn + \mu_3 Hn$, o w n low o n on $An + L_B + Cn + Dn + Fn + \mu_2 Gn + \mu_3 Hn$. Sin w . . v $A = CY_2$. n $L_B \geq L_1$, w o . on $Cn + Dn + Fn + \mu_2 Gn + \mu_3 Hn$. L α, β, γ . n δ non-n . . iv v. l . . W . , . , . . v , . i . n . $\alpha + \beta + \gamma + \delta \leq \frac{1}{3}$, $\beta + \gamma + \delta \leq \frac{1}{4}$, $\gamma + \delta \leq \frac{\mu_2}{4}$, $\delta \leq \frac{\mu_3}{5}$, $\mu_2 + \gamma + \delta \leq 1$, $\mu_3 + \delta \leq 1$. U in , . v. l . . . . l i li . o. , lin . . in . li i . . ov , w . , ollowin .

$$\begin{aligned}
&\alpha(B + 3C + 3D) + \beta(B + 3C + 3D + 4F) \\
&+ \gamma(3B + 3C + 3D + 4F + 4G) + \delta(3B + 3C + 3D + 4F + 4G + 5H) \\
&\geq (ICY_3 + CY_3)(\alpha + \beta + \gamma + \delta) + CY_4(\beta + \gamma + \delta) + MC_2^\ell(\gamma + \delta) + MC_3^\ell \delta.
\end{aligned}$$

T. l , . n . i o , . ov x . . . ion i

$$\begin{aligned}
&\alpha(B + 3C + 3D) + \beta(B + 3C + 3D + 4F) \\
&+ \gamma(3B + 3C + 3D + 4F + 4G) + \delta(3B + 3C + 3D + 4F + 4G + 5H) \\
&\leq MC_1(\alpha + \beta + 3\gamma + 3\delta) + C + D + F + \mu_2 G + \mu_3 H.
\end{aligned}$$

T. . o. , w . . v $An + L_B + Cn + Dn + Fn + \mu_2 Gn + \mu_3 Hn \geq CY_2 n + L_1 + (ICY_3 + CY_3)n(\alpha + \beta + \gamma + \delta) + CY_4 n(\beta + \gamma + \delta) + MC_2^\ell n(\gamma + \delta) + MC_3^\ell n\delta - MC_1(\alpha + \beta + 3\gamma + 3\delta)$.

Fin. lly, w . iv , ollowin o. oll . y.

Corollary 1.

$$\begin{aligned}
\text{GP-ER} \leq\ & |E| + MC + CY_3(1 - \alpha - \beta - \gamma - \delta) + ICY_3(2 - \alpha - \beta - \gamma - \delta) \\
&+ CY_4(1 - \beta - \gamma - \delta) + CY + MC_2^\ell(1 - \gamma - \delta) + MC_3^\ell(1 - \delta) + MC_4 \\
&+ \mu_2 MC_2^s + \mu_3 MC_3^s + MC_1(\alpha + \beta + 3\gamma + 3\delta) + \frac{L}{n}.
\end{aligned}$$

W . no y E' , . o . . . in OPT (. . ivi in . o o , . . . in , o. i in. l. E). T. n, $OPT = |E'| + MC$.

W . llo . , o o GP-ER. . on , ... , . o OPT (. y l wi , o. wi , o ivi . . . in , E l . i n. . . . . . , o. . . . . - . . in). To o . o, w n . *size* o . n . . in E'. W . . . . $s = \frac{2}{3}$. I , . . w. . l. . y in E (i. ., i , . . i no . . . l o ivi ion), , n i . . i i on . I i i . . . l o ivi ion o . n . . o E in o k . . . w n i . i o $\frac{1}{k}$. W will , . n , ni ion o . i o. . . . wi , . i $\frac{1}{2}$ in , ollowin w. y. Con i . . . i. o *twin arcs* $e, e' \in E'$, . . . l . o ivi in . o . on . . o E (. n . , ... i $\frac{1}{2}$). Con i . , ... , . o OPT , . on . in e . n e'. I . l . . on o , ... , . on . in . l o . no , . . . w. o . i i l . . , . n on , , n w i i , . e o. e' , . lon . o . , ... , . n w in . . . i . i o s w. , . . w . , . i o i . win . . o $1 - s$. W . ly , i . . lon . . , . xi . . , . i. wi , . l . . on o , . in . . . . . , . . on . in . no , . ivi . . . . I , . i . . i. o win . . . e, e' , , . e i on . in in . . . - , . in . n e' i on . in in , E l . i n. . . . . o OPT, , n w in . . . , . i o e o s . n . . . , . i o e' o $1 - s$. W . ly , i . o . o. . ll o. i l . i. . A , n o , i . o . , ... wi , . i $\frac{1}{2}$, . . i . *twin arc* wi , . i . l o $\frac{1}{2}$. n . , . . , ... , . o OPT , .

on in lon o E l i n n y o no v ny o ivi .

W llo o o GP-ER ollow : o o OPT w ini i li i llo o o l i o i (i will llo $|E|$ o o l o). Fo - in o OPT w in i llo o y on (in o o in o o n MC), n ly ollowin .

Fo y l o OPT w in i llo o y $1-\alpha-\beta-\gamma-\delta$. Fo o y l o OPT w in i llo o y $1-\beta-\gamma-\delta$. Fo on - - in o OPT w in i llo o y $\alpha+\beta+3\gamma+3\delta$. Fo wo- - in o OPT w in i llo o y $1-\gamma-\delta$. Fo - in o OPT w in i llo o y $1-\delta$. Fo - in o OPT wi l o , w in i llo o y 1. T inin o o on o o E l i n o no on i o y l . Fo y l in OPT (on in ivi), w i i i o i l o ivi ion o n lon n inv li i n l . I o w in llo o o y l y $\frac{2-\alpha-\beta-\gamma-\delta}{2}$ n o wi w in llo o o y l y 1.

T o , y Co oll y 1, o l llo o i l o o ol ion n y GP-ER. N x , w o i il llo ion o PIM. W o no llo l o o PIM o o ollowin l o i i in io o PIM (i. i o i l o o PIM). T i l o i ov y l in x w y PIM o N x , i on i y l n in o OPT. A v li in i on w i no ov in o in . T o o ov y l o wi o o i in i n o n on o o PIM. T i ol in PIM n xi i in on inin (n on in o o in in w) w w n o v li (no n ily xi) in on . T in in o in i ov l o ll ivi . W l v n . A ov l o y l n ivi o OPT, w l wi in . A y l w i w ov l in o i in . A in w i w ov l in o $i+1$ in . No o in o ni ion o PIM, y l l on ov . PIM o no ly IM n il in n o no on in y l (o o o non- ivi). I y l ivi n i i ov w ll. Cl ly, w l wi in only i i . N x , inin in i i ion in o i o on iv o in . E in y v on n (l on). T o ivi i 2 i i n . How v , in i i ivi , w llo ni o o v y o i , n i ov o o . T o i ov in o in i 1 w ll. T o o inin in o j i $\frac{3j+1}{2}$ i j i o , n $\frac{3j}{2}$ i j i v n. T o o o on n o OPT i i ly o llo o o i ov n o o in n on i inin . T o , llo o o y l o OPT wi k , w i w ov , i o $i+3(k-i)/2+i/2=3k/2$. T llo o o in o OPT wi k

. , . , w , i . , . w , , . ov , i . o $i+3(k-i)/2+(i+1)/2=(3k+1)/2$. W n o on i , ix i l . . w , , o . llo . o . y l o , , in i . . lly li , ly . . . ll , .

1. A y l o OPT wi , wo . , . , non o , in . ivi . , . T , y l . . , , . ov o i . lly, . n , , o , o . llo . o . , o , i 2.
2. A , . in o OPT w i , on i . o . in l ivi . , . W . llo . . ni o o , i , . in.
3. A y l o OPT wi , , , . , . , non o , in . ivi . , (, i w . on i , in [4]). I i , . . . in l , . ov . , , , , . . inin , . in , . . . n v n n . , o . , . (wo), . n , , o , . llo . o i 4. I i , . . wo , . ov . . , , , n , , . . inin , . in , . . o 2, . n , o . l o i . . . in 4.
4. A y l o OPT wi , v . , . , non o , in . ivi . , (, i w . on i , in [4]). I i , . . . in l , . ov . , , , , . . inin , . in , . . . n v n n . , o . , . (o ,), . n , , o , . llo . o i 7. I i , . . wo , . ov . . , , , , . . , . . , l , . n , , i . lw . y . n . j n . i , o . , . . . on , , , , , . , . , . . li in o wo , . in , o l n , . 1 . n 2, . n , o . l o i . . . in 7. I i , . . , , o . o , . . ov . . . , , n , o , , . . inin . , i no l , . , , . n 2, . n , . , o . l o i no l , . , , . n 7.
5. A , . in o OPT wi , wo . , . , non o , in . ivi . , . I no . , . . , , . ov , , . llo . o i 3 (, wo . , ... , x ly . . in OPT). I on o , i , . ov , , n , o o , o , , on i 2 . n in o . l 3.
6. An inv li y l o OPT wi , o . , . . . T , i i . . i , o y l . o OPT, . , o w i , on . in . . in l ivi . , , . n , wo ivi . , . . , , only wo . , . o , . . . o , i in l . , . Con i , in , , . l in y l . o OPT w wo y l . , on o , , ... n , o , , on o wo . , . . T , o o PIM o , , wo- . , y l i . o 3. In , lon , y l , i . l . . on . , i , . ov in , , - , o . . in , , n , o . l o i . o 4. O , , wi , . wo- . , , . in . n il , o , wo non- ivi . , . , . n , o o , lon , y l i 4 . . . in. T , i . iv . . o . l o . o 7.

2.4 Algorithm Arc-Combination

Al o , i , . AC i n . . . lyin o , PIM . n GP-ER . n , oo in , , . ol ion. T , . , . . , . o , GP-ER . , $s=\mu_2=\mu_3=\frac{2}{3}$, $\alpha=\frac{1}{12}$, $\beta=\frac{1}{8}$, $\gamma=0$. n $\delta=\frac{1}{8}$. T , i . l o , i , . , . . . , o , . . n . . , . n , . i , , . n $\frac{5}{4}$ . . w . . li , in , ollowin , o , . .

Theorem 1. *Algorithm AC has an approximation ratio of at most* $\frac{36}{29}\approx 1.24138$.

T , , oo i on y . . , l . n ly i o , o . . llo . y . , . l o , i , . o . , . , . , . , o OPT. In , ll v , ion o , . , , w . , ow . n x . l w , , , . . , oxi . ion , . io o AC i x ly $\frac{11}{9}$, , , o , , . , oxi . ion , . io o AC i . l . . $\frac{11}{9}\approx 1.22222$.

3 Algorithms for the Chord Version

In , i . ion w . y , , o v . ion o , . o l . . W v lo . $\frac{7}{5}$- . oxi . ion . l o i , . . i ov . , . . li . $\frac{3}{2}$- . oxi . ion . l o i , . o [1]. O . . l o i , . i o o o . . i o . l o i , . . : . n w . l o i , . n . P-ER . n . l o i , . D-DAG , . w . . i in [2,3] o , non- ivi i l , o v . ion . o l . .

3.1 Algorithm P-ER

In , i . . ion w in o . l o i , . P . o . . -ER w i , i . v . ion o . l o i , . ER o , , o v . ion (S ion 1). T . o . . in , . . wo . . . ollow . R . ov v li y l . on y on n il , in . n on . in no v li y l . . T n, o . . l o i , . ER.

T , o , , . l o i , . i o o o . . . o . . in . , . w . . . - . . in [2] o PIM, . n . . w . . . lyin . l o i , . ER.

L S , in . o . . . L S_1 , . o . . . , . w , . ov . in , . o . . in , . . , . n l $S_2 = S \setminus S_1$. Sin , . o . . in , . . ov . y l . only, w , . v $def(S) = def(S_1)$. T o o , y l . . ov in , . o . . in , . . i $|S_1|$. T o o , . ol ion . n y ER w n . li o S_2, i $\frac{3}{2}|S_2| + def(S_2)$. T o , , o . l o o , . ol ion . n y . l o i , . P-ER . . i $P\text{-}ER = |S_1| + \frac{3}{2}|S_2| + def(S)$.

Con i . . iv n o i . l . ol ion. L CH_i no , n . o , . in in , o i . l . ol ion w i , on . in i (o i in l o ivi) . . . L C_i no , n . o y l . in , o i . l . ol ion w i , on . in i (o i in l o ivi) . . . W . . in . . in , o . . n o P-ER . . . n ion o , v l . CH_i o $i \geq 1$. n C_i o $i \geq 2$.

W , . . , o o P-ER o , o on n . o OPT . o , . , o . l , ... o n i . l . . , o o P-ER.

E , . . o S_1 i , . . on ni , . n . , . . o S_2 i , . . 3/2 ni . . I . n . w . ivi in o k . . in OPT, . , . . i , . . y . n . l . . . (i. ., $1/k$) . o , . , o . l i , . . o n , . . o , . . . On o o , . , . , , . in o OPT i , . . y 1. In , i w y w ov . . l . . , o o $def(S)$, . w , . v (, n . o , . in in OPT i . l . . , i n .). W . . . . y o ov , ollowin l . . . .

Lemma 3. *The cost of P-ER is at most* $\sum_{i=1}^{n}\left(\frac{3i}{2}+1\right)CH_i + \sum_{i=2}^{n}\left(\frac{3i}{2}-\frac{1}{2}\right)C_i$.

Proof. Con i . . y l o OPT wi , p (o i in l o ivi) . . . , w . ow , . i . , . . o i . o $\frac{3p}{2} - \frac{1}{2}$. W . ow , . o . , . in o OPT wi , p (o i in l o ivi) . . . , i . , . . o i . o $\frac{3p}{2} + 1$.

Con i . . . . , . in o OPT wi , p . . . T , . . o o . , (o i in l o ivi) . . i . o 3/2. R . ll , . w , . . . n x . ni o . . , , . in, , o w , ... o $3p/2 + 1$ in o . l o , , . in.

I i l o on i . y l . o OPT. On on , . n i , y l on i . o o i in l . . . only, , n i , . . . l . . on . . in S_1, . in o , . wi i wo l , . v n ov o P-ER o l o in . T i i y 1, n o y o $\frac{3}{2}$. W on l o i o $3(p-1)/2+1 = 3p/2-1/2$. On o n , i y l l on ivi , n i o i o $3(p-1)/2+3/4 = 3p/2-3/4$, in l on o i i o o o $\frac{3}{4}$, n o v o o o $\frac{3}{2}$. T wo (i. wi i o) i $3p/2-1/2$.

3.2 Algorithm Directed-DAG

In i ion w o l o i Di -DAG (D-DAG) w vio ly on i y [2,3] o non- ivi i l o v ion o l .

Algorithm D-DAG. C oo n i y *e* o in $(n-1, 0)$. Di ll in o y o no v *e*. T l in in n i i y li , w o o olo i l o i $0, 1, \ldots, n-1$. Solv i in n o i lly (o non- ivi i l v ion) in G y Sw in l o i in [5]. Fo o l n , w i G y Sw in l o i . T i o wi o nin n w in o v y in 0. T n, o v y v x in $i = 2, \ldots, n-1$ (in i o), i xi in in w i n v x i wi in i. A n w in i o n o v y in v x i no n .

No D-DAG o no ivi ny o . W now ow iv n i y li w o o olo i l o i $0, 1, \ldots, n-1$, xi n o i l ol ion o o v ion o l o no ivi ny . Con i ol ion w n f i ivi in o l wo , wo o w i (a, b) n (b, c). Con i wo in in w i wo i i (i y lon o o on in, i no on o ivi ion). L x n y l n oin , n z n t i n oin iv ly. W i o n w in in ollowin w y. W - in o x o b, n on n - in o b o t o i . T o in w , on i o wo in , i. - in o y o b n - in o b o z. In in, w no lon n n ADM b, n o o o ol ion i y 1. T o , w o in on i ion o o i li y o o i in l ol ion. T o , w l i n o i l ol ion o no ivi ny in l in in n .

L *OPT* v l o o i l ol ion o o i in l in n . L *OPT′* v l o o i l ol ion o i in n . No y l n in o OPT on in o on v in *e*. T o , y i in , y l o OPT i i ion in o wo in . A in o OPT y i ion in o o in in i o .

T oo o ollowin l in ll v ion.

Lemma 4. *The cost of D-DAG is at most* $\sum_{i=1}^{n}(i+3)CH_i + \sum_{i=2}^{n}(i+2)C_i$.

3.3 Algorithm Chord-Combination

Al o i Co in ion (CC) n o P-ER n D-DAG, n oo ol ion.

Theorem 2. *The approximation ratio of CC is exactly* $\frac{7}{5} = 1.4$.

T o n oo i on n ly i in vio ion . T o l oo n o n in ll v ion o .

4 Conclusion

W n i ov l o i o ivi i l ADM ini i ion o l on in , o o v ion n o v ion. O inin ox- i ion l o i i l wo . An in in o n o l i w (ivi i l) v ion i lly i n (ivi i l) o v ion. In non- ivi i l o l , n nown o n l (o i v y iff n l o i in iff n o). T ion n w ll; w i v ion i o oxi ?

References

1. G. Călinescu and P.-J. Wan. Splittable traffic partition in WDM/SONET rings to minimize SONET ADMs. *Theoretical Computer Science*, 276(1–2):33–50, 2002.
2. G. Călinescu and P.-J. Wan. Traffic partition in WDM/SONET rings to minimize SONET ADMs. *Journal of Combinatorial Optimization*, 6(4):425–453, 2002.
3. L. Epstein and A. Levin. The chord version for SONET ADMs minimization. *Theoretical Computer Science*. To appear.
4. L. Epstein and A. Levin. Better bounds for minimizing SONET ADMs. In *Proc. of the 2nd Workshop on Approximation and online Algorithms (WAOA2004)*, pages 281–294, 2004.
5. O. Gerstel, P. Lin, and G. Sasaki. Wavelength assignment in a WDM ring to minimize cost of embedded SONET rings. In *INFOCOM1998*, volume 1, pages 94–101, 1998.
6. L. Liu, X.-Y. Li, P.-J. Wan, and O. Frieder. Wavelength assignment in WDM rings to minimize SONET ADMs. In *INFOCOM2000*, volume 2, pages 1020–1025, 2000.
7. M. Shalom and S. Zaks. A $10/7 + \varepsilon$ approximation for minimizing the number of adms in sonet rings. In *Proc. of the First International Conference on Broadband Networks (BROADNETS'04)*, pages 254–262, 2004.
8. P.-J. Wan, G. Călinescu, L. Liu, and O. Frieder. Grooming of arbitrary traffic in SONET/WDM BLSRs. *IEEE Journal on Selected Areas in Communications*, 18:1995–2003, 2000.

The Conference Call Search Problem in Wireless Networks

L E in[1,⋆] n A L vin[2]

[1] Department of Mathematics, University of Haifa, 31905 Haifa, Israel
lea@math.haifa.ac.il

[2] Department of Statistics, The Hebrew University, Jerusalem, Israel
levinas@mscc.huji.ac.il

Abstract. Cellular telephony systems, where locations of mobile users are unknown at some times, are becoming more common. Mobile users are roaming in a zone. A user reports its location only if it leaves the zone entirely. The Conference Call Search problem (CCS) deals with tracking a set of mobile users in order to establish a call. To find a single roaming user, the system may need to search each cell where the user may be located. The goal is to identify the location of all users, within bounded time, satisfying some additional constraints on the search scheme.

We consider cellular systems with n cells and m mobile users (cellular phones). The uncertain location of users is given by m probability distribution vectors. Whenever the system needs to find the users, it conducts a search operation lasting at most d rounds. A *request* for a single *search step* specifies a user and a cell. In this search step, the cell is asked whether the given user is located there. In each round the system may perform an arbitrary number of such requests. An integer number $B \geq 1$ bounds the number of distinct requests per cell in every round. The bounds d and B result from quality of service considerations. Every search step consumes expensive wireless links, which motivates search techniques minimizing the expected number of requests thus reducing the total search costs.

We distinguish between oblivious, semi-adaptive and adaptive search protocols. An oblivious search protocol decides on all requests in advance, and stops only when all users are found. A semi-adaptive search protocol decides on all the requests in advance, but it stops searching for a user once it is found. An adaptive search protocol stops searching for a user once it has been found (and its search strategy may depend on the subsets of users that were found in each previous round). We establish the difference between those three search models. We show that for oblivious "single query per cell" systems ($B = 1$), and a tight environment ($d = m$), it is NP-hard to compute an optimal solution (the case $d = m = 2$ was proven to be NP-hard already by Bar-Noy and Naor) and we develop a PTAS for these cases (for fixed values of $d = m$). However, we show that semi-adaptive systems allow polynomial time algorithms. This last result also shows that the case $B = 1$ and $d = m = 2$ is polynomially solvable also for adaptive search systems, answering an open question of Bar-Noy and Naor.

⋆ Research supported by Israel Science Foundation (grant no. 250/01).

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 133–146, 2006.

1 Introduction

C ll l , on y llow o on n l o o l , no i in in - in lo ion . In y w o i n w lo ion i i ov o n w ll, o n in i i l . M ny xi in y llow o o i lo ion o ly. F o , y l nn o v o n ll ll , w i i in i l o o o i i o o w n i o ll .

T Con n C ll S o l (CCS) l wi li in wi l on n ll n l y n n wi on in . T o l i o li on n ll w n m o in in ll l n wo on i in o n ll . T o l no in o o li n o on n ll. I. ., y n o n o o w i ll i onn o n . U in i o i l y n - io i ion o li li oo o o i in ll. T i i n y o ili y v o o i in o ili i o y o n in ll. W no y $p_{i,j}$ o ili y o n i in ll j. Followin vio wo [1], w $p_{i,j} > 0$ o ll v l o i, j. W i onn o x ly on ll in y n lo ion o iff n in n n n o v i l . T ool o n in *search requests.* Giv n o i n ll j, y ll j n w i i lo . D l y on in li i w ol o in o d yn ono o n ($1 \leq d \leq mn$). B n wi on in li i n o ll in o n o o iv n in n B $1 \leq B \leq m$. Bo l y n n wi on in o iv y li y o vi on i ion . W in in i nin o o ol w i ffi i n ly ili wi l lin , i. . iv n on in , ini i x o o , w in ni o o o 1.

No v n i o i i i l y l iv n lo in i ll, (i. ., i w in llo ll n w no lo), w ill n o i in ll w i i lo in o o l o ini i o ni ion lin .

W on i y o o o ol . An *oblivious search protocol* ll l n o o d o n , n o no n i . I o o l ly i ll o n . W n vi w i o o ol on w w no no i w n in l w lo , only i ll o w o n . A *semi-adaptive search protocol* ll l n o o d o n , n o no n i , ow v on i o n w o o i . An *adaptive search protocol* i on o n i i no i w i w o n in vio o n . I n v on in o l y n o n . A on n i in n o i l iv o o ol i o o l x n o i l o livio o o ol o o i l i- iv o o ol, i o n y o in o o w o n in o vio o n

W n tight instance o on n ll o l o n in n w $B = 1$ n $d = m$. To o iv o y o i in n w no o $B = 1$ i l n y w ll n o in l in o n . Cl ly i n o o n in ll y o m o n . In o o ini i wo l y, w i ll o n wi in x ly m o n (i. . $d = m$). No w n $B = 1$ n $d = m$ i ini n o o n n l i l ol ion o o l . So on y on i i ion o i in n o v i y o l o ini i in wo l y n ini i in xi lo on ll wi in i l o n n on y o l i o ini i on ion o wi l n wi n x n o .

Previous work. T [2] in o o l w o iff n o ll ly (i. ., w n no ll w i o n ly onn o i). T y ow $B = 1$, $d = m = 2$, i NP- o o livio o o ol . I w l o n o n o w i NP- o iv o o ol w ll. A i il o l w in o y B -Noy n M l wi [1]. In o l on ll i in o o n , i o no o in l (o li i n o), o li o ll in ll. T o on o livio ni . I i own in o ny on n n o $(m > 1, d > 1)$, n ny on n n o o n $1 \leq d \leq n$, o l i NP- . No o l o in l , w i i iv l n o o l i in i in i , i olyno i lly olv l in i l yn i o in [5,6]. B -Noy n M l wi [1] i l l o i w i o in n o l o i o $m = 1$. T i i $\frac{4}{3}$- oxi ion o $m = d = 2$ n $\frac{e}{e-1} \sim 1.581977$- oxi ion o o v l o d, m. In vio [3] w i n PTAS o l o l . T PTAS i n o o livio o l, n o i ily o wo o iv o l w ll.

Paper outline. In S ion 2 w ov n in n o i l o livio o o ol o i in n i NP- o ll x v l o $d \geq 2$. T i l l x n n li l o B -Noy n N o [2] o $d = 2$. W l o ow i d i o in n n in n o i l o livio o o ol o i in n o NP- in on n . In S ion 3 w n o PTAS o o livio o l i . W n l iv ly i l PTAS o $d = m = 2$ n w w n o o li PTAS o n i y on n $d = m$. Fin lly, in S ion 4 w ow o in n o i l i- iv o o ol o i in n w n o i on n , n on in olyno i l i . T i l l ow i in ili y o on n ll o l w n o livio n i- iv o o ol ; i NP- w on i olyno i lly olv l . T o i- iv o o ol wi $d = m = 2$ l o i li olyno i l i l o i o o in n o i l iv o o ol.

2 NP-Hardness for the Oblivious Problem

W ll B -Noy n N o [2] ov n in o i l o livio o o ol i NP- o $B = 1$ n $d = m = 2$. In i ion w x n i l o n l i .

Theorem 1. *Finding an optimal oblivious search protocol is NP-hard even when restricted to tight instances with $d = m$ rounds and $B = 1$ for all fixed values of $d \geq 2$.*

Proof. T l i o $d = m = 2$ i ov in [2]. W ov l i o $d \geq 3$ in ion o PARTITION o l (o l SP12 in [4]). In i o l w iv n N in n $a(1), \ldots, a(N)$, $\sum_{i=1}^{N} a(i) = 2S$ o o in $S \geq 2$, n ion i w xi $J \in \{1, \ldots, N\}$ $\sum_{i \in J} a(i) = S$. W n in n o o livio o l ollow . L $\delta > 0$ ll o i iv v l $\delta < \frac{1}{8S^2d^2}$. T $N + m - 2$ ll , $c_1, \ldots, c_{N+m-2}$. T (i n i l) o ili i o wo ollow . T o ili y o ll c_j, $j \leq N$ i $p_{1,j} = p_{2,j} = (1-\delta)\frac{a(j)}{2S}$. T o ili y o v y o ll $j > N$ i $p_{1,j} = p_{2,j} = \frac{\delta}{m-2}$. A o o $m - 2$, i $(3 \leq i \leq m)$ o ili y o $1 - \delta$ in ll $i + N - 2$ $(p_{i,i+N-2} = 1 - \delta)$ n o ili y $p_{i,j} = \frac{\delta}{N+m-3}$ o ll $j \neq i + N - 2$. T i o l i ion o ion.

W - o n o o n o i l o livio o o ol in xi n x i ion (i. ., PARTITION in n i YES in n). L J o $\{1, \ldots, N\}$ $\sum_{i \in J} a(i) = S$. In o n , ollow . T ll in J o n ll in $\{1, \ldots, N\} - J$ o on . No $\sum_{i \notin J} a(i) = S$ w ll. E o ll $N + k$ i o $k + 2$. R ll , o ili y o i n ll i $1 - \delta$. In on o n , ll in J o on . T ll in $\{1, \ldots, N\} - J$ o . All o in o i y o . T o ili y o n on o wo in o n i x ly $\frac{1-\delta}{2}$. T o ili y o n ny o in o n i x ly $1 - \delta$. T o , o ili y o n ll in o n i $\frac{(1-\delta)^m}{4}$, n o o ili y o v on o n i $1 - \frac{(1-\delta)^m}{4}$. Fo v y , o ili y o n i wi in wo o n i l $1 - \delta$. T o , o ili y o n ll wi in wo o n i l $(1-\delta)^m$, n o ili y will l l on n (n o m o n) i o $1-(1-\delta)^m$. W on l o l o i o

$$n + n\left(1 - \frac{(1-\delta)^m}{4}\right) + n(m-2)(1-(1-\delta)^m) \le$$
$$n + n\left(\frac{3}{4} + \frac{m\delta}{4}\right) + nm(m-2)\delta \le \frac{7n}{4} + nm^2\delta < \frac{7n}{4} + \frac{n}{8S^2}$$

w in li y ol in $(1-\delta)^m \ge 1 - m\delta$, on in li y ollow y i l l n l in li y ol $\delta < \frac{1}{8Sd^2} = \frac{1}{8Sm^2}$.

Con i now i ion w i no x i ion (i. ., PARTITION in n i NO in n). T o o v y $J' \subseteq \{1,\ldots,N\}$ i $\sum_{i\in J'} a(i) \le S-1$ o $\sum_{i\in J'} a(i) \ge S+1$. Fi no i on o ll $N+1,\ldots,N+m-2$ i no in o n o w o o ili y $1-\delta$ o in i ll, n o ili y o on o n i l $1-\delta$, n o i l $2n-n\delta$. O wi , on i ll $1,\ldots,N$. A $A_1 \subseteq \{1,\ldots,N\}$ o ll i o in o n n i join $A_2 \subseteq \{1,\ldots,N\}$ i o - on in o n . L $p(1)$ $(p(2))$ o o ili i o ll o (on) in o n . I. ., $p(1) = \sum_{j\in A_1} p_{1,j}$ n $p(2) = \sum_{j\in A_2} p_{2,j}$. Sin $A_1 \cap A_2 = \emptyset$ n $p_{1,j} = p_{2,j}$ o ll j, w on l $p(1) + p(2) \le 1-\delta$. D o ni ion o o ili i o wo in N ll , w now $p(i) = (1-\delta)\frac{X(i)}{2S}$, w $X(i)$ o $i = 1, 2$ in . Sin i no x i ion, w now $X(i) \neq S$. I $X(i) \le S-1$ o $i = 1,2$, n o ili y o on o n i l $1-(1-\delta)^2\cdot(\frac{S-1}{2S})^2 > 1-(1-\delta)^2\frac{S^2-1}{4S^2}$ w l in li y ol in $S \ge 1$. O wi , in $X(1)+X(2) \le 2S$, n non o v l n S, w v i o on o i, $X(i) = S+u \ge S+1$, n o o $3-i$ w v $X(3-i) \le S-u \le S-1$. In i o ili y o on o n i l $(1-(\frac{S+u}{2S})(\frac{S-u}{2S})(1-\delta)^2) \ge 1-(1-\delta)^2\frac{S^2-1}{4S^2}$ (in $u \ge 1$). T o in l wo i o l $n+n(1-\frac{S^2-1}{4S^2}(1-\delta)^2) \ge n+n(1-\frac{S^2-1}{4S^2}) = \frac{7n}{4}+\frac{n}{4S^2}$. No o w o in $(2n-n\delta)$ i no ll in $2n-n\delta \ge \frac{7n}{4}+\frac{n}{4S^2}$ i iv l n o $\delta + \frac{1}{4S^2} \le \frac{1}{4}$ w i ol in $\delta < \frac{1}{8Sd^2}$ n $S, d \ge 2$.

W o i i n x i ion, n o i l o i o $\frac{7n}{4}+\frac{n}{8S^2}$, w i i no x i ion, o i l o i l $\frac{7n}{4}+\frac{n}{4S^2}$. T o w o ion, w o i o $\frac{7n}{4}+\frac{3n}{16S^2}$, i NP- . □

In ll v ion o w ov ollowin o . W ow i d i no x , o in o l o on ly NP- .

Theorem 2. *Finding an optimal oblivious search protocol is strongly NP-hard even when restricted to tight instances with $d = m$ rounds and $B = 1$.*

3 A PTAS for the Oblivious Problem

Properties. R ll w non- o o ili i o i o n ll. In i , ll in x ly on o n . T o , ll n o i n ion o . R ll n o livio i n in v n , n l lon o i ill no lo . Sin w olv i in n , l y o n o n, n o $OPT \geq n$.

L ε v l $0 < \varepsilon < \frac{1}{(20m)^{m+1} \cdot m!}$. W ow olyno i l i oxi ion w nnin i i olyno i l in n, v l ε, n l o m n on n . T oxi ion io o l o i $1 + \Theta(\varepsilon)$.

O o o o v l in . In in w in in o ion o o OPT. E in n l vi n x iv n ion o ll o i ili i o i i o in o ion. O l o i n ll o i ili i , n on oo ol ion iv . In n ly i i i ffi i n o on i ol ion o in w n w o .

3.1 Two Users

W wi l iv ly i l PTAS o i . H on o wo o n . Fo iv n l o i , i o i i ly $2n - n(1-p)(1-q)$, w p n q o ili i o n in on (iv ly) in on o n . In i ion, l p n q no o ili i in n o i l ol ion.

L $p_j = p_{1,j}$ o ili y o o lo in ll j, n l $q_j = p_{2,j}$ o ili y o on o lo in ll.

D no o ili y in v l $I_0 = (0, \frac{\varepsilon}{n}]$, n o $1 \leq i \leq \lceil \text{lo}_{1+\varepsilon}\left(\frac{n}{\varepsilon}\right) \rceil$,

$$I_i = \left(\frac{\varepsilon}{n}(1+\varepsilon)^{i-1}, \frac{\varepsilon}{n}(1+\varepsilon)^{i}\right].$$

First guessing step. w k, w i i n o ll in on o n o . Mo ov , w o ili y p o n in in on o n . T i , w in x i $p \in I_i$.

Lemma 1. *The number of possibilities for the first guessing step is*

$$O\left(n\left[\text{lo}_{1+\varepsilon}\left(\frac{n}{\varepsilon}\right) + 2\right]\right).$$

Proof. Cl ly $0 < k < n$, in in ll ll o in o n lw y l in on o n , w i iv o $2n$, n i i - o i l. To on l oo , no n o in v l i o lo$_{1+\varepsilon}\left(\frac{n}{\varepsilon}\right) + 2$. □

By L . . . 1, . o. in . n x . . iv n . .. ion o . . .. . . in . . n on in olyno i l i . W on in o . n ly . i .. ion o . i in w i . w . .. . " o. . " v l . . . o. . . on o OPT. W no *the guess of* p y p' o . . o n o I_i; i. ., $p' = \frac{\varepsilon}{n}(1+\varepsilon)^i$.

T n x . i o . l . .o . ili i . o only . .. . ollow . Fo . ll j n $r_j = p_j/p'$ o . *scaled probability of cell* j *and the first user.* W on i . . v o $R = (r_j)$ o . . l .o . ili i . . . . . i in ll j. W . . ov . ll ll wi . . l .o . ili y l . . . . n 1. S . ll . nno . . o . . . . in . on . o n . . n . . o . . . . o . . . . in . . . o n .

W . . . . . i n . *type* o . . ll . o in o . ollowin w y. W n . . o in . v l $\mathcal{J}$ . . ollow : $J_0 = (0, \varepsilon]$, . n o . ll $\ell \geq 1$, $J_\ell = (\varepsilon \cdot (1+\varepsilon)^{\ell-1}, \varepsilon \cdot (1+\varepsilon)^\ell]$, . n $\mathcal{J} = \{J_0, J_1, \ldots\}$. Fo . . ll $1 \leq j \leq n$, w n . in . v l . o $\mathcal{J}$ . . on . in r_j. T . i , w o . v l t_j . . . . $r_j \in J_{t_j}$. T in x t_j i . y o ll j. Fo v l . o t_j . . . . $t_j > 0$, w . l r_j wi . r'_j w i . i . . o n o . in . v l J_{t_j}, i. ., $r'_j = \varepsilon(1+\varepsilon)^{t_j}$. O . wi . v l . . . in n . n , i. ., $r'_j = r_j$. No . . . n . o y i . o lo $_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right) + 2 = O\left(\text{lo }_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right)\right)$. W l S . . . o . l .o . ili i . o y 0 ll (. . in o n 2 o . . . .). L S' . o n o . i in . v l . . on . in S.

Second guessing step. W . . . . . o n o ll o . . y . . . . . . o . . . . in . on . o n . Mo ov . , w . . . v l o S'.

Lemma 2. *The number of possibilities in the second guessing step is*

$$O\left(n^{\log_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right)+2} \text{lo }_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right)\right) .$$

Proof. T n . . o ll . o . . y i . n in . . w n 0 . n $k \leq n-1$ (l . . ly, o n . o . ov y . n . o ll . . xi o . i y). T n . . o o ion o . . . in S' i . l . n . o in . v l in $\mathcal{J}$ . . i $O\left(\text{lo }_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right)\right)$. □

No . . . n . o o . i ili i . o . i . . . in . i olyno i l (o . . x v l o ε).

N x , o . . . iv n y o ll, $i > 0$, on i . . ll w i . lon o . i y . A . . o n in , . iff n w n . ll i . o . ili y o . on . o . . i in . i ll. Cl . . ly, . iv n . . s . . ll n o . . o . . . . in o n 2, i . n . . . . . o ll . o l . . o . on . in . . o n . W . o . . . ll wi . i . . o . ili y o . on . . . on . ll wi . o on y . A . i l x . n . . . n . ow . . on i in . i o ion only (o . o n in . n) . . y n v . in . . . o o . . ol ion. Fo ll o y 0, n . n i y o . ll j o q_j/p_j, . i i . .. io w n . o . ili i . o . wo . . . So . ll ll o y 0 y non-in . . in n i i . . A . w . . . , . . . ini . l . x o . o . ll , ... o . l . o . ili i o i l $S' = \varepsilon \cdot (1+\varepsilon)^{\ell}$. I o ll l o ili i o y 0 ll o no x S', n ll ll will o in o n 2. I $S' > \varepsilon$, n on wo l o o o l ll in o n 1. W llow li ly i o ili y in on o n , n i o o l ll ily. T o , w y only in o ili y o n in on in o n 1. I w o l no x S', in ll y 0 ll in o n 2 o , n i y li ly (il low), in y only in o ili y o n in on in o n 1.

Con i w i o ly v l k, o n o ll o y , n v l o S'. T in n ly i wo l o o n l ion w n o ili i o n in in in o i l ol ion n ol ion w n . L $\hat{p}$ n $\hat{q}$ o on in o ili i o p n q in l in ol ion. Sin o ili y o on o o n in o n y only in , w $1-\hat{q} \geq 1-q$, i. . $\hat{q} \leq q$.

To o n $\hat{p}$ in o p, no $p' \leq (1+\varepsilon)p + \frac{\varepsilon}{n}$. T o n in o ll w o o n o ili i no o y 0, l in o i l in o o ili i y l i li iv o o $1+\varepsilon$. Fo ll o y 0, $S' \in J_{\ell}$. I $\ell > 0$, w llow (o l o ili i) o o n ll o x v l $\varepsilon \cdot (1+\varepsilon)^{\ell}$. How v , in ll v l o ε, w n i iv o o o , o n , in i ion o l i li iv o n in o o $1+\varepsilon$. Fo y 0, wo wo l o l o ili i o l v n o, i ε n x i y o n . T o , $\hat{p} \leq p'(1+\varepsilon)+2\varepsilon p' = (1+3\varepsilon)p' \leq (1+3\varepsilon)(1+\varepsilon)p+(1+3\varepsilon)\frac{\varepsilon}{n} \leq (1+7\varepsilon)p + \frac{4\varepsilon}{n}$. T l in li y ol in $\varepsilon < 1$.

T o i o

$$
\begin{aligned}
APX = n(1+\hat{p}+\hat{q}-\hat{p}\hat{q}) &= n(1+\hat{p}+\hat{q}(1-\hat{p})) \leq n(1+\hat{p}+q(1-\hat{p})) \\
&= n(1+\hat{p}(1-q)+q) \leq n(1+(1+7\varepsilon)p(1-q)+\frac{4\varepsilon}{n}+q) \\
&\leq (1+7\varepsilon)n(1+p+q-pq)+4\varepsilon \leq (1+11\varepsilon)OPT = (1+\Theta(\varepsilon))OPT
\end{aligned}
$$

T l in li y ollow o $OPT \geq 1$ w i ol o ny in n o o l . T o , w v li ollowin o :

Theorem 3. *Problem CCS with two users, two rounds and $B = 1$ has a polynomial time approximation scheme.*

Remark 1. We can easily extend the scheme of this section to the case where there are also zero probabilities. To do so, we first guess the number of cells n_1 (n_2) to page the first (second) user in the first round such that the second (first) user has zero probability to be placed in this cell. Over the set of cells where both users have positive probability we apply the scheme of this section. Among the cells where the first (second) user has zero probability we will page for the second (first) user in the first round in the set of the n_2 (n_1) cells with the highest probability.

3.2 m Users

W on in wi PTAS o n l (on n) n o . W ov ollowin o .

Theorem 4. *Problem CCS with a constant $d = m$ and $B = 1$ has a polynomial time approximation scheme.*

T n o on i l 1 n o m. Sin lo ion o in in n n , w n o x ion o n o y l l in o r, o ili y o n in ll in o r on . Giv n n l o i ( ) l $q_{i,r}$ o ili y o n in i in on r y iv n ol ion, T n, o o i ol ion i

$$n\sum_{r=1}^{m}\left(1-\prod_{i=1}^{m}\left(\sum_{s=1}^{r-1}q_{i,s}\right)\right)=n\sum_{r=1}^{m}\left(1-\prod_{i=1}^{m}\left(1-\sum_{s=r}^{m}q_{i,s}\right)\right).$$

In i ion w $(q_{i,r})$ no ion o no v l in x o i l ol ion.

W wi ni o o n in o v l $p_{i,j}$. In i ion w ollowin o in v l o ll o n in o . W n $\mathcal{J}$ follow : $J_0 = (0, \varepsilon^{2m+5}]$, n o ll $k \geq 1$, $J_k = (\varepsilon^{2m+5} \cdot (1+\varepsilon)^{k-1}, \varepsilon^{2m+5} \cdot (1+\varepsilon)^k]$, n $\mathcal{J} = \{J_0, J_1, \ldots\}$. L s $1 \in J_s$. W l in v l J_s y $(\varepsilon^{2m+5} \cdot (1+\varepsilon)^{s-1}, 1]$, n only $s+1$ in v l . Fo i i, j w $1 \leq i \leq m$, $1 \leq j \leq n$, w n in v l o $\mathcal{J}$ on in $p_{i,j}$. T i , w o v l $t_{i,j}$ $p_{i,j} \in J_{t_{i,j}}$, n w n *type* o ll j o v o $(t_{1,j}, \ldots, t_{m,j})$. Fo v l o $p_{i,j}$ $t_{i,j} > 0$, w l $p_{i,j}$ wi $p'_{i,j}$ w i i o n o in v l $J_{t_{i,j}}$, i. ., $p'_{i,j} = \varepsilon^{2m+5} \cdot (1+\varepsilon)^{t_{i,j}}$. O wi , v l in n n , i. ., $p'_{i,j} = p_{i,j}$.

Corollary 1. *If $t_{i,j} > 0$ then $p_{i,j} \leq p'_{i,j} \leq (1+\varepsilon)p_{i,j}$.*

W i n *sub types* o ll , on (n n) v l o o ili i o y 0. I o ll $1 \leq i \leq m$, $t_{i,j} > 0$, i no i ion o y . O wi , l *weight* o ll j no w_j n $w_j = \max_{\{i|t_{i,j}=0\}} p_{i,j}$. Fo y v o o iv n ll j, ollowin v o a^j o l n m. T i- n y a^j_i i -1 i $t_{i,j} > 0$, n o wi $a^j_i = \frac{p_{i,j}}{w_j}$.

W i ion in o in v l in o o o n n l i y v o a^j. Fo n n y a^j_i, n in v l o $\mathcal{J}$ on in a^j_i. Co v l $t'_{i,j}$ $a^j_i \in J_{t'_{i,j}}$, n y o ll j i v o $(t'_{1,j}, \ldots, t'_{m,j})$ (w $t'_{i,j} = -1$ i $t_{i,j} > 0$). W v o a'^j, w a'^j_i i o n o in v l $J_{t'_{i,j}}$. I $a^j_i = -1$ n l o $a'^j_i = -1$. W l o ili i in in ollowin w y: i $t_{i,j} > 0$ n $p''_{i,j} = p'_{i,j}$ n o wi $p''_{i,j} = w_j a'^j_i$.

Corollary 2. *If* $t_{i,j} = 0$, $p''_{i,j} \leq w_j \left((1+\varepsilon)a_i^j + \varepsilon^{2m+5}\right) = (1+\varepsilon)p_{i,j} + w_j\varepsilon^{2m+5}$.

No l on n y in a'^j i 1, i n n y ℓ_j o w i $w_j = p_{\ell_j,j}$. W ll ℓ_j *the leader* o ll. No y o ni n i , in xi i no ni (in , ℓ_j i i o wi ini in x), o i o li ly ll o ili y, ill in l in v l.

A ll i i y i y y l n wi (x l in ll wi no y). Two ll j_1, j_2 v *general type* i y o v no y o i y v y y n l . T i w i w_{j_1} n w_{j_2} y o i y v l in $(0, \varepsilon^{2m+5}]$. T o n o n l y i o

$$m\left(2\,\mathrm{lo}\ {}_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right)^{2m+5} + 3\right)^m \leq \frac{m}{\varepsilon^{2m}}\ .$$

T i ollow o oi o $\varepsilon < \frac{1}{(20m)^{m+1}\cdot m!}$ n o $\ln\frac{1}{\varepsilon} \leq \frac{1}{\varepsilon}$, $\ln(1+\varepsilon) \geq \frac{\varepsilon}{2}$, n $m \geq 2$.

Giv n ll, in o o i y ol ion w n i o i ll, w n o iv ion o T wo l o in w i ll i o

Guessing step. Fo v y n l y n v y ion π (o o $m!$ o i l on), w n o ll o i n l y in o o ion π. No o n o l x ly o l n o ll o i n l y . Fo v y n l y t, x l in n l y wi no y w l o n in v l o o l o ili y $P(t,\pi)$ ll o n l y t, in ion π, in in o n w l o i n l y i (i. , o i w i lon o in v l).

Lemma 3. *The number of possibilities for the first guessing step is polynomial.*

Proof. T n o o in ion o n l y n ion i o $\frac{m!m}{\varepsilon^{2m}}$. T n o o i l o iv n ion n ll i o $n+1$ (i i n in w n 0 n n). T n o o i ili i o o ili y i $\left((2m+5)\,\mathrm{lo}\ {}_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right)+2\right) \leq \frac{1}{\varepsilon^2}$. T o n o o i ili i o in i o $\left(\frac{n+1}{\varepsilon^2}\right)^{\frac{m!m}{\varepsilon^{2m}}}$. □

Giv n , w i i ll o ion ollow . Fo n l y wi no y llo n o ll o i y o ion, i o i l . T x i i ion i no i o n . Fo o n l y (i. , wi y), iv n i n l y , l ℓ i l . D no ion y $\pi_1, \ldots, \pi_{m!}$. Giv n ion π_i, l t_i in x o o ili y in v l o i l , n l a_i n o ll o i . L n' n o ll n o i i . R -n ll o 1 o n' n no y w_j wi o i wi ll j. W n o i i n' ll o $m!$ ion, w o v y ion, n o n i iv n on o o ili i o ll llo o i w ll n on on n o ll. T i o on o ollowin in o . L $X_{i,j}$ n in i o v i l w o v l i 1 i ll j i llo o ion i. W ly o n o n n o ili i ollow. Fo $1 \leq i \leq m!$,

$$\sum_{i=1}^{n'} X_{i,j} \leq a_i \quad \text{n} \quad \sum_{i=1}^{n'} w_j \cdot X_{i,j} \leq \varepsilon^{2m+5}(1+\varepsilon)^t \ .$$

W l ly v $\sum_{i=1}^{m!} X_{i,j} \geq 1$ o ll $1 \leq j \leq n'$, in ll i i n o l on ion. I i i i n o o n on, on o i o n n ov wi o viol in o on in. T o l i o n i l in oin.

W l x in li y on in, n l i wi $X_{i,j} \geq 0$. W l wi lin o w i l ly ol ion i o i in l in o o. Solvin lin o w n n i ol ion. T i i ol ion o $2m! + n'$ non o v i l (n o on in). Cl ly, ll j, l on non o v i l $X_{i,j}$ n w n o ll no i n o l ly o ion (i. ., v o n on non o v i l o i wi) i o $2m!$. T ll ov n - i i o ion in o o i y o n o ll. In wo , ll. i ion l ll i n o on ion, in in i o l o ili y in o n o l (i. ., i o l w i) y n iv o o $2m!w_j\varepsilon^{2m+5}$, n v l w i no l n $2m!w_j\varepsilon^{2m+5}$ in o o n .

F o now on, w on i o o . W wo l li o o iff n w n v l $q_{i,r}$ y no i l l o i n on y o . L $q'_{i,r}$ v l y l o i . I. ., $q'_{i,r}$ i o l o ili y o n in i in o n r y .

Lemma 4. $q'_{i,r} \leq (1 + 7\varepsilon)q_{i,r} + \varepsilon^2$.

Proof. T wo y o n in v l $q_{i,r}$, l i li iv n n iv n . T wo o n in n o i o ll n . By Co oll y 1 n 2, w on l $p''_{i,j} \leq (1+\varepsilon)p_{i,j} + w_j\varepsilon^{2m+5}$. T o o iv n in ll i o ll n l i o n y ε^{2m+5} i o ll o ili i , w i i m. H n , $m\varepsilon^{2m+5}$ o n l in iv n in v l $q_{i,r}$.

T n x o n in i o $P(t, \pi)$. Ano l i li iv o o $1+\varepsilon$ i in o i i . Mo ov , o ili y o iv n ion π y in y n iv o o $(2m!+1)\varepsilon^{2m+5}$. In wo , i iv ow y n o v y i o n l y n ion. T $2m!\varepsilon^{2m+5}$ i o l w ion l ol ion o lin o i o n . An i ion l ε^{2m+5} i o o n in o $P(t, \pi)$ o i n oin o o ili y in v l .

R ll n o o in ion o n l y n ion i o $\left(\frac{m!m}{\varepsilon^{2m}}\right)$, i iv o i o $\left(\frac{m!m}{\varepsilon^{2m}}\right)(2m!+1)\varepsilon^{2m+5} \le \varepsilon^4$. S i in w ,

$$q'_{i,r} \le ((1+\varepsilon)q_{i,r} + \varepsilon^{2m+4})(1+\varepsilon) + \varepsilon^4 \le (1+3\varepsilon)q_{i,r} + \varepsilon^2 \ . \qquad \square$$

W o n o n o n in o l n ion v l .

$$n \sum_{r=1}^{m} \left(1 - \prod_{i=1}^{m} \left(1 - \sum_{s=r}^{m} q'_{i,s}\right)\right) \tag{1}$$

$$\le n \sum_{r=1}^{m} \left(1 - \prod_{i=1}^{m} \left(1 - \sum_{s=r}^{m} \left((1+3\varepsilon)q_{i,s} + \varepsilon^2\right)\right)\right) \tag{2}$$

$$\le n \sum_{r=1}^{m} \left(1 - \prod_{i=1}^{m} \left(1 - m\varepsilon^2 - (1+3\varepsilon)\sum_{s=r}^{m} q_{i,s}\right)\right) \tag{3}$$

$$\le n \sum_{r=1}^{m} \left(m^2\varepsilon^2 + (1+3\varepsilon)\left(1 - \prod_{i=1}^{m}\left(1 - \sum_{s=r}^{m} q_{i,s}\right)\right)\right) \tag{4}$$

$$\le n \left(m^3\varepsilon^2 + (1+3\varepsilon)\sum_{r=1}^{m}\left(1 - \prod_{i=1}^{m}\left(1 - \sum_{s=r}^{m} q_{i,s}\right)\right)\right) \tag{5}$$

$$\le \varepsilon OPT + (1+3\varepsilon)OPT = (1+4\varepsilon)OPT \ , \tag{6}$$

w (2) ollow y L 4, n (3),(5) ollow y i l l . N x , (4) ollow in iv n o m in n n n o v n , o ili y o i nion i l i li y o $(1+3\varepsilon)$ i w l i ly o ili y o v n in i y o n , n i w in o ili y o v n y n i iv o o $\rho = m\varepsilon^2$, n o ili y o nion in y o $m\rho = m^2\varepsilon^2$. Fin lly (6) ollow in $OPT \ge n$ n $\varepsilon < \frac{1}{m^3}$. T i o l oo o T o 4.

4 Polynomial Time Algorithms for Finding Optimal Semi-adaptive Search Protocols

In i ion w on i o l o o in n o i l i- iv o o ol o i in n o CCS. W ow olyno i l i l o i o olvin i o l . W i l o i o olv wo- wo- o n (i ol ion ol o iv y w ll). In ll v ion o w n yn i o in l o i o olv CCS o l wi on n n o .

Two users. W wo n wo o n n $B = 1$. B -Noy n N o [2] ow o in n o i l o livio o o ol o i i n NP- o l . T y l n o n ion o i i o - in n o i l iv o o ol i olyno i lly olv l . W no in i i- iv o o ol i iv l n o - iv o o ol. T o y o in n o i l i- iv o o ol in olyno i l i , w ovi o i iv n w o i ion.

O l o i , no y *Alg*, k i n n o ll n o i l ol ion o in o n . T i i i l n y n x iv n ion in k i n in in in v l $[0, n]$, n n nin ol ion o in in x iv n ion. W n x n ly i ion in w i i o .

D no y $I_i^k = p_{1,i} \cdot (n-k) - p_{2,i} \cdot k$ *index* o ll i in k- i ion. O l o i o in i o ll in non- in o n n i i k ll (in o li). T i ll o in o n , w o ll o on in o n .

Theorem 5. *Alg returns an optimal semi-adaptive search protocol.*

Proof. To ov o , i i ffi i n o ov ollowin l i : A xi i o ll i, j wi $I_i^k \geq I_j^k$ o i l ol ion j o in o n , n i i o on in o n . T n, l in ol o i n j (i. ., n w ol ion i o in o n , n i j o on in o n), l in no o i l ol ion.

To ov l i w in ol ion o - l in y i l n i $(n-k) \cdot (p_{1,i} - p_{1,j}) + k \cdot (p_{2,j} - p_{2,i})$. To i , no o ili y o n in in o n in y $p_{1,i} - p_{1,j}$, inin n x o o y $(n-k) \cdot (p_{1,i} - p_{1,j})$. Si il ly o on x in o i $k \cdot (p_{2,j} - p_{2,i})$.

How v , $(n-k) \cdot (p_{1,i} - p_{1,j}) + k \cdot (p_{2,j} - p_{2,i}) = p_{1,i} \cdot (n-k) - p_{2,i} \cdot k - [p_{1,j} \cdot (n-k) - p_{2,j} \cdot k] = I_i^k - I_j^k \geq 0$, w l in li y ollow y ion. T o l n o ol o i n j l in no o i l ol ion, w l i . □

T n x o oll y n w o n ion i li y [2].

Corollary 3. *Alg returns an optimal adaptive search protocol.*

5 Open Questions

W li v l o n ion l o :

– D in o l xi y o o in n o i l iv o o ol o i in n wi $d = m > 2$.
– Fin n FPTAS o ov i non- xi n (y owin o l i NP- in on n o x on n v l o $d = m$) o o in n o i l o livio o o ol o i in n wi x on n n o .

– Fin PTAS o ov i non- xi n (y owin o l i APX-) o o in n o i l o livio o o ol o n i- y i in n . T nnin i o PTAS o l olyno i l in n n in $d = m$.

References

1. A. Bar-Noy and G. Malewicz. Establishing wireless conference calls under delay constraints. *Journal of Algorithms*, 51(2):145–169, 2004.
2. A. Bar-Noy and Z. Naor. Establishing a mobile conference call under delay and bandwidth constraints. In *The 23rd Conference of the IEEE Communications Society (INFOCOM2004)*, volume 1, pages 310– 318, 2004.
3. L. Epstein and A. Levin. A PTAS for delay minimization in establishing wireless conference calls. In *Proc. of the 2nd Workshop on Approximation and Online Algorithms (WAOA2004)*, pages 36–47, 2004.
4. M. R. Garey and D. S. Johnson. *Computer and Intractability.* W. H. Freeman and Company, New York, 1979.
5. D. Goodman, P. Krishnan, and B. Sugla. Minimizing queuing delays and number of messages in mobile phone location. *Mobile Networks and Applications*, 1(1):39–48, 1996.
6. C. Rose and R. Yates. Minimizing the average cost of paging under delay constraints. *Wireless Networks*, 1(2):211–219, 1995.

Improvements for Truthful Mechanisms with Verifiable One-Parameter Selfish Agents

A. F n , G. P l o, F. So n ino, n C. V n *

Dipartimento di Informatica ed Applicazioni "R.M. Capocelli", Universitá di Salerno, via S. Allende 2, I-84081 Baronissi (SA), Italy
{ferrante, parlato, sorrentino, ventre}@dia.unisa.it

Abstract. In this paper we study optimization problems with verifiable one-parameter selfish agents introduced by Auletta et al. [ICALP 2004]. Our goal is to allocate load among the agents, provided that the secret data of each agent is a single positive rational number: the cost they incur per unit load. In such a setting the payment is given after the load completion, therefore if a positive load is assigned to an agent, we are able to verify if the agent declared to be faster than she actually is. We design truthful mechanisms when the agents' type sets are upper-bounded by a finite value. We provide a truthful mechanism that is $c \cdot (1 + \epsilon)$-approximate if the underlying algorithm is c-approximate and weakly-monotone. Moreover, if type sets are also *discrete*, we provide a truthful mechanism preserving the approximation ratio of the used algorithm. Our results improve the existing ones which provide truthful mechanisms dealing only with finite type sets and do not preserve the approximation ratio of the underlying algorithm. Finally we give a full characterization of the $Q||C_{max}$ problem by using only our results. Even if our payment schemes need upper-bounded type sets, every instance of $Q||C_{max}$ can be "mapped" into an instance with upper-bounded type sets preserving the approximation ratio.

1 Introduction

O i i ion o l lin wi o llo ion l i l l o i i o l n y v n i o in v l o l : n li v i i l o i , on-lin v off-lin l o i n o on. T n -lyin y o i n in i v il l o l o i (i o inni in off-lin l o i o in i x ion in on-lin l o- i). T i ion n o o n li i in on x o o n n wo li In n H v io o in own y *selfish* (*rational*) n o i iv in o ion (ll *type*) n o i i ion l o i will v o n o i y n n wo on *reported* y In i on x i i li i o

* Research supported by the European Project FP6-15964, Algorithmic Principles for Building Efficient Overlay Computers (AEOLUS). (Contract number 015964.)

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 147–160, 2006.

n n will li o y i i l o ol ion S ,
v n in i o S i no lo lly o i l.

T l o ni i n i n o G T o y n Mi o o-
no i i w y o in in , o y n , n o o
i y o o i i ion o l n olv on l in .
In i w y i n o l o i o olvin (o oxi ly
olvin) o in o i l o i i ion o l in n o l n .

Followin n no ion in y o oxi ion o o -
in o i l o i i ion o l (, ., [10]), w on i o l n
o - l $(\mathcal{I}, \mathsf{m}, \mathsf{sol}, \mathsf{goal})$, w $\mathcal{I}$ i o *instances* o o l ;
$\mathsf{sol}(I)$ i o *feasible solutions* o in n I; $\mathsf{m}(S, I)$ i *measure* o
i l ol ion S o in n I n goal i i in o x. T , o i-
i ion o l on i in n in i l ol ion S^* o in n I
$\mathsf{m}(S^*, I) = \mathsf{opt}(I) := \mathsf{goal}_{S \in \mathsf{sol}(I)} \mathsf{m}(S, I)$. A *c-approximation* l o i
A o $\Pi = (\mathcal{I}, \mathsf{m}, \mathsf{sol}, \mathsf{goal})$ i o ll $I \in \mathcal{I}$, x$\{\mathsf{m}(A(I), I)/\mathsf{opt}(I)$,
$\mathsf{opt}(I)/\mathsf{m}(A(I), I)\} \leq c$.

In n o i i ion o l Π wi *selfish agents*, m n w i
privately know o in . T v y in n $I \in \mathcal{I}$ on i o wo
$I = (T, \sigma)$, w v o $T = (t_1, t_2, \ldots, t_m)$ i *private part* o in
n σ i *public part* o in . In i l , w t_i i nown
only o n i, o $i = 1, 2, \ldots, m$ n w ll t_i *type* o n i. T *type
set* Θ_i o n i i o o i l y o n i. In i in ,
n will o o v l $b_i \in \Theta_i$ (w i n iff n o y
t_i). An l o i A o o i i ion o l Π wi l n iv
in v o o i $B = (b_1, b_2, \ldots, b_m)$, in o in n T i
i o o olv . E l n in o on y *cost*, $\mathsf{cost}_i(S, t_i)$,
n in on i l ol ion S n iv t_i. Sin v y n
i i l , i l $b_i \neq t_i$ o o in A o n ol ion
o n i. Un o n ly, v n o A i c- oxi in o in n T,
o $B \neq T$ ol ion n y A on in B i v , w.
in n T, -off o i $\mathsf{opt}(T)$.

In o o o in o ol ion, l o i A i i wi *payment
scheme* $P = (P_1, \ldots, P_m)$ in o o in v y n o o y .
A ol ion $S = A(B, \sigma)$ i o n i i w y n
$P_i(S, B, \sigma)$. W n i i *rational* in n i
y l ion b_i o o xi i *profit*.

Definition 1. *Let Π be an optimization problem with one-parameter selfish agents and A be an algorithm for Π, and P be a payment scheme. The profit function* profit *of agent i with respect to the pair (A, P) when B is the sequence of bids, σ is the public information, t_i is the true type of agent i, and $S = A(B, \sigma)$, is defined as* $\mathsf{profit}_i(S, B, \sigma, t_i) := P_i(S, B, \sigma) - \mathsf{cost}_i(S, t_i)$.

I i n l o on i ni in w i o o i- n i
xi i w n o $b_i = t_i$. W v ollowin l i l no-
ion o *truthful mechanism*. In ni ion o l ni (n

in o) ollowin no ion n o o l. L $X = (x_1, \ldots, x_k)$ v o. Fo ny $1 \leq i \leq k$, w i in X_{-i} no v o $X_{-i} := (x_1, \ldots, x_{i-1}, x_{i+1} \ldots, x_k)$ n w i in (y, X_{-i}) no v o $(y, X_{-i}) := (x_1, \ldots, x_{i-1}, y, x_{i+1} \ldots, x_k)$.

Definition 2. *The pair $\mathcal{M} = (A, P)$ is a* l ni o l n *if and only if for all σ, for all agents i, and all type declarations B, it holds* $\mathsf{profit}_i(A((t_i, B_{-i}), \sigma), (t_i, B_{-i}), \sigma, t_i) \geq \mathsf{profit}_i(A(B, \sigma), B, \sigma, t_i)$.

In w y, v y n xi i o w n i l. T , w in l ni v y n lw y o y , n l o i A lw y wo on in n T. A on n , w y l ni $\mathcal{M} = (A, P)$ i *c-approximating* o n o i i ion o l Π wi l n , i A i *c-* oxi in l o i o v y in n I o Π. Sin , in l ni , n no o v o i iv o , y wo l no i i in ni nl y w o . T i o iv ollowin ni ion.

Definition 3. *A truthful mechanism satisfies* vol n y i i ion *condition if agents who bid truthfully never incur a net loss, i.e. for all public information σ, for all agents i, and for all other agents' bids B_{-i},*

$$\mathsf{profit}_i(A((t_i, B_{-i}), \sigma), (t_i, B_{-i}), \sigma, t_i) \geq 0.$$

W now vi w on o o i i ion o l Π wi *one-parameter selfish agents* (i in [2]). H , n i, iv in o ion in l $t_i \in \mathbb{Q}$. Mo ov , i l ol ion S o n in n I o Π n , o n i, n o n $\mathsf{w}_i(S)$ o i n wo . W ll ol ion S *schedule.* No i in ni ion o on - o l ([2]) o l o n o wo o l n n on iv o in B. How v w i o lv , in wi o li , o in w i o n o wo i n o ll n n only on li in o ion (n no on n i). W no n o n o lo j $\mathcal{W} > 0$. T o n ion o n i, ollowin i l o .

Definition 4. *Let S be a feasible solution of Π. Then, the cost function* $\mathsf{cost}_i(S, t_i)$ *is defined as* $\mathsf{cost}_i(S, t_i) := \mathsf{w}_i(S) \cdot t_i$.

S lin o l y i l x l o o i i ion o l o on - l n . In *scheduling problem,* in on i o m in $s = (s_1, s_2, \ldots, s_m)$ n n jo w i $W = (w_1, w_2, \ldots, w_n)$. A l S i n i n n o jo o in . L $\mathsf{w}_i(S)$ o w i o jo i n o in i y l S. In lin o l on- i in o in l ini i in o n ion o i wi l . Fo in n , in $Q||C_{max}$ o l o o l S i *makespan* $MS(S)$ i xi o l ion i o in . Fo lly, $MS(S) = \max_{1 \leq j \leq m}\{\frac{\mathsf{w}_j(S)}{s_j}\}$. W on i in in w i in i i own y iff n n n s_i o in i i iv in o ion o n i. To in in o on - l n, w on i $t_i = 1/s_i$ y o n i. T li in o ion σ i n $W = (w_1, w_2, \ldots, w_n)$ o jo w i . W ll $Q||C_{max}$ o l i NP- . T o o w $Q||C_{max}$ o in x l .

A ni $\mathcal{M}$ o v i l on - l n i i $\mathcal{M} = (A, P)$ wo in ollow .

1. T llo ion l o i A in n o i $B = (b_1, b_2, \ldots, b_m)$ n li σ n o l $S = A(B, \sigma)$ o m n . W ll $\mathsf{w}_i(S)$ no o n o lo i n o n i y l S o y l o i A on in B n σ.
2. E n i i *observed* o o l i n lo in i $\mathcal{T}_i \geq \mathsf{w}_i(S) \cdot t_i$. No i n i o l lo $\mathsf{w}_i(S)$ i n o in i $\mathsf{w}_i(S) \cdot t_i$. A n i n ow v l y l o wo n o in l o v o l ion i n ni no w y o in i . How v , n i nno o v o ni lo o l o l ion i $\mathsf{w}_i(S) \cdot t_i$. Sin $\mathsf{w}_i(S) \cdot t_i$ i i o n i o o l lo $\mathsf{w}_i(S)$, w no wi $s_i = \frac{1}{t_i}$ *speed* o n i.
3. Fin lly, n i l i n wo , i w y n o y lyin n ion P_i on n S, B, σ, n *observed completion time* $\mathcal{T}_i$ o in i.

W in i in , y n ovi *after* x ion o lo n n (i lly) *verifiable* in ollowin n . I n i iv n o n o lo n 0, ni n n o w n i, l o n lly i (i , $b_i < t_i$). In , in i l i o l ion i $\mathsf{w}_i(S) \cdot b_i$ i ll n l o l ion i $\mathsf{w}_i(S) \cdot t_i$ n w v $\mathcal{T}_i \geq \mathsf{w}_i(S) \cdot t_i > \mathsf{w}_i(S) \cdot b_i$. Sin y n ovi *after* o l ion o lo , ni n i in onv ni n o l i . On o n , ni nno n o i n n l o low n lly i , in n n i o l y o o jo .

H n o w w o Π n o i i ion o l o v i l on - l n . L now in n i ni ion o o n l ni in i n w n io.

Definition 5. *Let A be an algorithm for Π, and P be a payment scheme. The profit function* profit *of agent i with respect to the pair (A, P), when B is the sequence of bids, σ is the public information, t_i is the true type of agent i, $S = A(B, \sigma)$, and $\mathcal{T}_i$ is the observed completion time of agent i for the load $\mathsf{w}_i(S)$, is defined as* $\mathsf{profit}_i(S, B, \sigma, t_i, \mathcal{T}_i) := P_i(S, B, \sigma, \mathcal{T}_i) - \mathsf{cost}_i(S, t_i)$.

Definition 6. *Let A be an algorithm for Π, and P be a payment scheme. A pair $\mathcal{M} = (A, P)$ is a truthful mechanism with respect to Π, if for all σ, for all i, for all bid vectors B, and for all observed completion times $\mathcal{T}_i \geq \mathsf{w}_i(A(B, \sigma)) \cdot t_i$, it holds that* $\mathsf{profit}_i(S, (t_i, B_{-i}), \sigma, t_i, \mathsf{w}_i(S) \cdot t_i) \geq \mathsf{profit}_i(A(B, \sigma), B, \sigma, t_i, \mathcal{T}_i)$ *where* $S = A((t_i, B_{-i}), \sigma)$.

No , , n w ni ion no n n , in in , i w , v o in o on , o v o l ion i l o.

Giv n , , l , ni $\mathcal{M} = (A, P)$ o Π, in [4] , , o iv n y on i ion , l o i A i y.

Definition 7 (weakly-monotone algorithm). *Let Π be an optimization problem for verifiable one-parameter selfish agents and A be an algorithm for Π. Algorithm A is* w ly- ono on *if and only if, for all σ, for all i, for all declared bid vectors B such that $\mathsf{w}_i(A(B, \sigma)) = 0$ and for all $b'_i \in \Theta_i$ with $b'_i > b_i$ it holds that $\mathsf{w}_i(A((b'_i, B_{-i}), \sigma)) = 0$.*

In o , wo w ly- ono on l o i , A , , ollowin o y. Fix o in (B, σ) o w i , l o i , A i n no lo o n i. I n i l o low (, i , , l $b'_i > b_i$) n , l i o , o , n in , , , n A i n no lo o n i.

Lemma 1 ([4]). *Let Π be an optimization problem for verifiable one-parameter selfish agents. If $\mathcal{M} = (A, P)$ is a truthful mechanism for Π, then A is a weakly-monotone algorithm.*

2 Previous Works and Our Contribution

T l VCG , ni [5,6,7,11] i , o in n , ni o iv , , l , ni o o i i ion o l . How v , , i , ni li only o *utilitarian* o l , , o l w , o j iv n ion i l o , o , o n ion o , n (., , o , , in- i nnin , , .). In , in l y Ni n n Ron n [8,9] i i oin o , VCG , ni o no o l ly in on x w , o ion l i l y i l ol in , y i i o i l o o n o i l ol ion o , o on in o i i ion o l (y NP- , o l). S , lin i l i l o i i ion o l , i no ili i n (in w i ini i in , xi ov ll , in o , i o l ion i) n i i NP- . Mo ov , , , lin o l i o n - o iff n llo ion n o in o l in o ni ion n wo . T , i , n , o l o w i , non VCG- , ni , v n in o .

Ni n n Ron n [8,9] iv n m- oxi ion , l , ni o , o l o , lin on m *unrelated* , in , w n , , in i own y iff n n , l , o in i o , - i n o , , in n , l o i , , o o , , lin on , v l l y , n . In [2], i on i , i l v i n o , , lin on *uniformly related* , in (in o $Q||C_{max}$), w , , , in i , s_i n , o in i o i iv n y , io w n , w i , o , n , o , , in . T y , i l o llo ion l o i , A o on - , o l , i y n , P o w i , $\mathcal{M} = (A, P)$ i , l , ni .

E n i lly, l ni o on - l n *mono-tone* l o i n , in i , y n i ni ly in (o n i iv o). In i iv ly, ono oni i y n in in , o x ly on , in o no l o i , wo i n o , in . T l o [2] , o l o i nin , l ni o $Q||C_{max}$ o l o i i o l o i nin oo l o i w i l o i i ion l ono oni i y i n . Effi i n ni o o in lin on l , in wi ll n (i l o on - n), v n ovi y A n T - o [2] n , n ly y A l l. [3] n y An l n, A n So ni [1].

A w , A l l [4] on i o i i ion o l o v i l on - o l . In i o l, y n iv n o n only n v o l lo i n . T i n o n iv o i iv lo , ni n *verify* i n l o lly i . T y ow , in o o v l ni o v i l on - l n , n y on i ion i l o i w ly- ono on .

Our Contribution. In i wo , w x n o l iv n in [4]. T - o w o y o i i ion o l o v i l on - l n . In i iv ly v i l n i n n y li in o in i y ni n v i y w n *i* n i (i. l $b_i < t_i$), ovi lo i n o i n i o i iv . Fo in n , o lin o l ni n *verify*, o o v o l ion i o n *i*, i l o n lly i , ovi l on jo n i n o . In [4] w ow i $\mathcal{M} = (A, P)$ i l ni o n o i i ion o l o v i l on - l n n *A* *weakly-monotone*. T y l o ovi y n *P* w i llow o v l ni , w n in li y o y i ni .

In S ion 3, w iv v y i l n ffi i n y n , l in o olyno i l- i l ni , o wi l o o i i ion o l wi v i l on - l n . In i l , w ovi y n $P^{(1)}$ wo o *discrete* n *upper-bounded* y (S ion 3). In i in , w n n i o in w i , i lw y w n inv o wo y . Con i in lin o l (w y inv o in '), o ion i i w n i i no o i l o v in x in *j* in ion on , o v y o i l $j \in \mathbb{Q}$. In , in only in o in (ffi i n ly) . Mo ov , w n n nno l o n ni v l . In lin o l , i n n in ni ly low , in o no xi . T o y o i li o ny l li li ion .

F o o i l oin o vi w, o l i ov on iv n in [4], ollow : (*i*) l o i n - o n y o ly in l l o ni y ; (*ii*) o ni v oxi-

Table 1. Comparing Results (c is the approximation of a given weakly monotone algorithm)

Problem Version	Payments Time Complexity	Apx Ratio
Θ_i finite and discrete [4]	poly$(\lvert\Theta_i\rvert, m, n)$	c
Smooth problems [4]	poly$(\log_{1+\epsilon} \lvert\Theta_i\rvert, m, n)$	$c \cdot (1+\epsilon)$
Θ_i upper bounded and discrete	poly(m, n)	c
Smooth problems with Θ_i upper bounded (continuous)	poly(m, n)	$c \cdot (1+\epsilon)$

ion io c o l o i i , w il ni iv n in [4] n o l i *smooth* (D . 11) in o o o in $c \cdot (1+\epsilon)$- oxi ion. (T i ion i i o o n in i in o o v y n o l in olyno i l i .)

In S ion 4, w iv y n $P^{(2)}$, l in o olyno i l- i l ni (T o 3), o n vin ion l y - o n (no i). In o o o in l ni w o n n ' i . U in i o n in ni , i l o i y - ni i c- oxi , n no in n i o oxi ion o l o i w n i n on o n i . How v , i o l i oo n ni i $c \cdot (1+\epsilon)$- oxi (T o 4). To o o nowl i i *first* l owin w ly- ono oni i y o l o i i ffi i n on i ion o xi n o l ni o o i i ion o l wi v i l on - l n wi on in- o y . I l o n w n y no - o n . In T l 1 w i o l o in o vio on .

Fin lly, in S ion 5, li ion o o l , w lly i $Q||C_{max}$ o l wi v i l on - l n in ny n- o n in n o o n on , o o inin olyno i l- i $c \cdot (1+$ $\epsilon)$- oxi l ni , iv n c- oxi w ly- ono on olyno i l- i l o i .

3 A Payment Scheme for Discrete Types

In i ion, w on i only y Θ_i vin ollowin o y.

Definition 8. *A set Θ_i is said* i *and* - o n *if: (i) there exists a value $\Delta_i \in \mathbb{R}^+$ such that, for all $b, \bar{b} \in \Theta_i$, $b \neq \bar{b}$, $|b^{-1} - \bar{b}^{-1}| \geq \Delta_i$* (i), *and (ii) there exists a finite value* $_i \in \mathbb{R}^+$ *such that* $_i \geq b$, $\forall b \in \Theta_i$ (- o n).

N x w n y n w i llow o on l - ni o Π, w n n v y i n - o n .

Definition 9. *Let S be a schedule, B be a bid vector, σ be the public part of the input, $\mathcal{T}_i$ be the observed completion time and $c_i^{(1)} \in \mathbb{R}^+$ be a constant (to be given). For each $i = 1, \ldots, m$, we define*

$$P_i^{(1)}(S, B, \sigma, \mathcal{T}_i) := \begin{cases} \frac{\mathcal{W}}{b_i} \cdot c_i^{(1)} \text{ i } \mathsf{w}_i(S) \neq 0 \text{ n } \mathcal{T}_i = \mathsf{w}_i(S) \cdot b_i; \\ 0 \text{ o wi .} \end{cases}$$

T i in y n $P_i^{(1)}$ i o iv n i i in n iv o l o low n lly i . On o n n i i l o i o o l o , i w v i ion n w ly- ono on l o i , own in n x o .

Theorem 1. *Let Π be an optimization problem for verifiable one-parameter selfish agents and A be a polynomial-time weakly-monotone algorithm for Π. If every Θ_i is upper-bounded by a finite value $_i$ and discrete w.r.t. a known value Δ_i, then for every $1 \leq i \leq m$ there exists a value for the constant $c_i^{(1)}$ such that $\mathcal{M} = (A, P^{(1)})$ is a polynomial-time truthful mechanism for Π. Moreover, $\mathcal{M}$ satisfies voluntary participation condition.*

Proof. L S_{t_i} l o y A w n in (t_i, B_{-i}), n S_{b_i} on on in (b_i, B_{-i}). To on $\mathcal{M}$ i l ni , w ow o ll $b_i \in \Theta_i$ n o ll $\mathcal{T}_i \geq \mathsf{w}_i(S_{b_i}) \cdot t_i$ ollowin l ion ol

$$\Lambda_i = \mathsf{profit}_i(S_{t_i}, (t_i, B_{-i}), \sigma, t_i, \mathsf{w}_i(S_{t_i}) \cdot t_i) - \mathsf{profit}_i(S_{b_i}, B, \sigma, t_i, \mathcal{T}_i) \geq 0.$$

Fo o ili y w no (t_i, B_{-i}) $\mathbf{T}$ n $\mathsf{w}_i(S_{t_i}) \cdot t_i$ $\mathcal{T}_i^*$. W on i $\mathsf{w}_i(S_{b_i}) = 0$. Sin $\mathsf{profit}_i(S_{b_i}, B, \sigma, t_i, \mathcal{T}_i) = 0$ w v

$$\Lambda_i = \frac{\mathcal{W}}{t_i} \cdot c_i^{(1)} - \mathsf{w}_i(S_{t_i}) \cdot t_i \geq \mathcal{W} \cdot \left(\frac{c_i^{(1)}}{\quad i} - \quad i \right) \geq 0 \tag{1}$$

o ll v l $c_i^{(1)} \geq \quad _i^2$. By ov l l ion , w l o v $\mathsf{profit}_i(S_{t_i}, \mathbf{T}, \sigma, t_i, \mathcal{T}_i^*) \geq 0$, n $\mathcal{M}$ i vol n y i i ion on i- ion. L $\mathsf{w}_i(S_{b_i}) > 0$. W i in i wo .

Case 1 ($b_i > t_i$). Sin A i w ly- ono on i ol $\mathsf{w}_i(S_{t_i}) > 0$. I $\mathsf{profit}_i(S_{b_i}, B, \sigma, t_i, \mathcal{T}_i) < 0$, o E . 1 w v $\Lambda_i > 0$ o $c_i^{(1)} \geq \quad _i^2$. L $\mathsf{profit}_i(S_{b_i}, B, \sigma, t_i, \mathcal{T}_i) \geq 0$. T n w v :

$$\begin{aligned} \Lambda_i &= \mathcal{W} \cdot \left(\frac{1}{t_i} - \frac{1}{b_i} \right) \cdot c_i^{(1)} - (\mathsf{w}_i(S_{t_i}) - \mathsf{w}_i(S_{b_i})) \cdot t_i \\ &\geq \mathcal{W} \cdot \Delta_i \cdot c_i^{(1)} - \mathcal{W} \cdot \quad _i \geq 0 \end{aligned}$$

o ll v l $c_i^{(1)} \geq \quad _i / \Delta_i$.

Case 2 ($b_i \leq t_i$). Sin $T > \mathsf{w}_i(S_{b_i}) \cdot b_i$, w v $P_i^{(1)}(S_{b_i}, B, \sigma, \mathcal{T}_i) = 0$ n $\mathsf{profit}_i(S_{b_i}, B, \sigma, t_i, \mathcal{T}_i) < 0$. T o o E . 1 w v $\Lambda_i > \mathsf{profit}_i(S_{t_i}, \mathbf{T}, \sigma, t_i, \mathcal{T}_i^*) \geq 0$ o $c_i^{(1)} \geq \quad _i^2$.

H n , o $c_i^{(1)} \geq$ x{ $_i^2, \frac{\quad i}{\Delta_i}$}, $\mathcal{M}$ i l. I i i o w y n $P^{(1)}$ i o l in olyno i l i . □

A in S ion 1, i A i l o i in l ni n i lw y wo on y in v y n lw y o y A on n i A i c- oxi n $\mathcal{M} = (A, P)$ i l ni n $\mathcal{M}$ i c- oxi w ll. T o T o 1 w v ollowin .

Theorem 2. *Let Π be an optimization problem for verifiable one-parameter selfish agents and A be a polynomial-time c-approximating weakly-monotone algorithm for Π. If every Θ_i is upper-bounded by a finite value $_i$ and is discrete w.r.t. a known value Δ_i, then $M = (A, P^{(1)})$ is a polynomial-time c-approximate truthful mechanism for Π, satisfying voluntary participation condition.*

No i i y i ni n i i i n ni ly - o n . Conv ly i y i i n ni ly - o n i o l on in in ni v l . Fo in n on i in w i o v y $i = 1, \ldots, m$, $\Theta_i \subseteq \{i^{-1} | i \in \mathbb{N}\}$. T i i i l o i n - o n y : $\Delta_i = 1$ n $_i = 1$, o v y y Θ_i.

4 A Payment Scheme for Rational Types

In i ion, w ow ow o x n o y n in o o l wi ion l y w i only - o n y ni v l $_i$. To o , w ly o n in ni on y . Giv n i v o B, w no y B^R v o o in y B y l in l n b_i wi o n v l b_i^R o b_i. I $\alpha^\gamma < b_i^{-1} \leq \alpha^{\gamma+1}$, n $b_i^R = 1/\alpha^{\gamma+1}$ o o $\gamma \in \mathbb{Z}$. T , i $B = (b_1, b_2, \ldots, b_m)$ n $B^R = (b_1^R, b_2^R, \ldots, b_m^R)$. Giv n n l o i A o Π, w n l o i A_α l o i , on in B n σ, i ly n l o i A on in B^R n σ.

Definition 10. *Let S be a schedule, B be a bid vector, σ be the public part of the input, $\mathcal{T}_i$ be the observed completion time and $c_i^{(2)} \in \mathbb{R}^+$ be a constant (to be given). For each $i = 1, \ldots, m$, we define*

$$P_i^{(2)}(S, B, \sigma, \mathcal{T}_i) := \begin{cases} \frac{\mathcal{W}}{b_i^R} \cdot c_i^{(2)} & \text{i } \mathsf{w}_i(S) \neq 0 \text{ n } \mathcal{T}_i = \mathsf{w}_i(S) \cdot b_i; \\ 0 & \text{o wi .} \end{cases}$$

T i in y n $P^{(2)}$ i i il o on o $P^{(1)}$. T iff - n i w on i o n i b_i^R in o l i b_i n on n $c_i^{(2)}$ i n i lly iff n o $c_i^{(1)}$. In n x o , w will l i y nin o on n $c_i^{(2)}$.

Theorem 3. *Let Π be an optimization problem for verifiable one-parameter selfish agents whose types are positive rational, and let A be a polynomial-time weakly-monotone algorithm for Π. If every Θ_i is upper-bounded by a finite value $_i$, then for every $1 \leq i \leq m$ there exists a value for the constant $c_i^{(2)}$, such that $\mathcal{M} = (A_\alpha, P^{(2)})$ is a polynomial-time truthful mechanism for Π. Moreover, $\mathcal{M}$ satisfies voluntary participation condition.*

Proof. Fi no , , i A i w ly- ono on , n A_α i w ly- ono on w ll. L B v o o i , n S_{t_i} , , l o y l o i , A_α w n i in (t_i, B_{-i}), n S_{b_i} , on on in (b_i, B_{-i}). To ow $\mathcal{M}$ i , l, w ov , o ll $b_i \in \Theta_i$ n $\mathcal{T}_i \geq \mathsf{w}_i(S_{b_i}) \cdot t_i$,

$$\Lambda_i = \mathsf{profit}_i(S_{t_i}, (t_i, B_{-i}), \sigma, t_i, \mathsf{w}_i(S_{t_i}) \cdot t_i) - \mathsf{profit}_i(S_{b_i}, B, \sigma, t_i, \mathcal{T}_i) \geq 0.$$

Fo , o ili y w no $\mathbf{T} = (t_i, B_{-i})$ n $\mathcal{T}_i^* = \mathsf{w}_i(S_{t_i}) \cdot t_i$. W on i $\mathsf{w}_i(S_{b_i}) = 0$. In , i w v :

$$\Lambda_i = \mathsf{profit}_i(S_{t_i}, \mathbf{T}, \sigma, t_i, \mathcal{T}_i^*) \geq \mathcal{W} \cdot \left(\alpha^{\gamma+1} \cdot c_i^{(2)} - \frac{1}{\alpha^\gamma} \right) \geq 0 \tag{2}$$

w n

$$\gamma \geq -\frac{\text{lo}\ _\alpha c_i^{(2)}}{2} - \frac{1}{2}. \tag{3}$$

A n o , , o , w i , ow o , oo $c_i^{(2)}$ in o $\mathcal{M}$ o l. F o E . 2, w l o v $\mathsf{profit}_i(S_{t_i}, \mathbf{T}, \sigma, t_i, \mathcal{T}_i^*) \geq 0$, $\mathcal{M}$ i vol n y i i ion on i ion.

I in o ow $\mathsf{w}_i(S_{b_i}) > 0$. W i in i wo :

Case 1 $(b_i > t_i)$. Sin A i w ly- ono on n $b_i^R \geq t_i^R$ i ol $\mathsf{w}_i(S_{t_i}) > 0$. W.l.o. w only on i in w i $\mathsf{profit}_i(S_{b_i}, B, \sigma, t_i, \mathcal{T}_i) \geq 0$. W n ly , $b_i^R = t_i^R$, i. b_i n t_i o n o ow o α.

$$\Lambda_i = \mathcal{W} \cdot \left(\frac{1}{t_i^R} - \frac{1}{b_i^R} \right) \cdot c_i^{(2)} - (\mathsf{w}_i(S_{t_i}) - \mathsf{w}_i(S_{b_i})) \cdot t_i = 0.$$

H , w n ly inin in w i $b_i^R > t_i^R$. T n, o o $\gamma \in \mathbb{Z}$, i ol :

$$\begin{aligned} \Lambda_i &= \mathcal{W} \cdot \left(\frac{1}{t_i^R} - \frac{1}{b_i^R} \right) \cdot c_i^{(2)} - (\mathsf{w}_i(S_{t_i}) - \mathsf{w}_i(S_{b_i})) \cdot t_i \geq \\ &\geq \mathcal{W} \cdot \left(\frac{1}{t_i^R} - \frac{1}{b_i^R} \right) \cdot c_i^{(2)} - \mathcal{W} \cdot t_i \geq \mathcal{W} \cdot \left((\alpha^{\gamma+1} - \alpha^\gamma) \cdot c_i^{(2)} - \frac{1}{\alpha^\gamma} \right) \end{aligned} \tag{4}$$

By i l l l ion w v E . 4 i o l o 0 w n:

$$\gamma \geq -\frac{\text{lo}\ _\alpha (c_i^{(2)})}{2} - \frac{\text{lo}\ _\alpha (\alpha - 1)}{2}. \tag{5}$$

A in vio , w o on i ion o oo in $c_i^{(2)}$ o n o o .

Case 2 $(b_i \leq t_i)$. Sin $T > \mathsf{w}_i(S_{b_i}) \cdot b_i$, w v $P_i^{(2)}(S_{b_i}, B, \sigma, \mathcal{T}_i) = 0$ n $\mathsf{profit}_i(S_{b_i}, B, \sigma, t_i, \mathcal{T}_i) < 0$, i lyin $\Lambda_i \geq \mathsf{profit}_i(S_{t_i}, \mathbf{T}, \sigma, t_i, \mathcal{T}_i^*) \geq 0$.

H w i ow o oo on n $c_i^{(2)}$ in o o i y o E . 3 n E . 5, o ny v l o γ. In i l , w j ow o in w i $\gamma = \gamma_{\min}$, w $\gamma_{\min}$ i ini l v l γ n v . Sin $\frac{1}{\sup_i} \leq (\frac{1}{\sup_i})^R = \alpha^{\gamma_{min}}$, y i l l l ion w v $\gamma_{\min} = \lceil$lo $_\alpha \frac{1}{\sup_i}\rceil$. Sin lo $_\alpha(c_i^{(2)})$ n v v l ny l n y v yin $c_i^{(2)}$, w n o - v l o $c_i^{(2)}$ o E . 3 n E . 5 i w n $\gamma = \gamma_{\min}$. H n , i i i o w y n $P^{(2)}$ i o l in olyno- i l i . □

No , i in D . 10 on n $c_i^{(2)}$ i no , n o E . 3 n E . 5 w y o v in o $\mathcal{M}$ o l, y Θ_i - o n y on n w i n on v l o x$\{-\frac{1}{2}, -\frac{\log_\alpha(\alpha-1)}{2}\}$. T , $c_i^{(2)}$ llow o l wi ny y Θ_i i - o n y ny on n $_i$.

In o o v l ni o o l n , involvin n vin y - o n y ni v l , w o n i . B w o oxi ion? I A i c- oxi ion l o i , n no in n i o oxi ion o A_α. N x , w n l o o l o w i o n in in oxi ion o A_α y n o wi o oxi ion n o A. H n o , w i o n ion only o ini i ion o l . W i il n n li o xi i ion o l w ll.

Definition 11. *Fix $\epsilon > 0$ and $\delta > 1$. A one-parameter minimization problem $\Pi = (\mathcal{I}, \mathsf{m}, \mathsf{sol},$ in$)$ is (δ, ϵ)- oo , if, for any pair of instances $I = (T, \sigma)$ and $\tilde{I} = (\tilde{T}, \sigma)$ such that $t_i \leq \tilde{t_i} \leq \delta \cdot t_i$ for $i = 1, 2, \ldots, m$, and for all $S \in \mathsf{sol}(\sigma)$, it holds that $\mathsf{m}(S, I) \leq \mathsf{m}(S, \tilde{I}) \leq (1 + \epsilon) \cdot \mathsf{m}(S, I)$.*

Fo in n , o v $Q||C_{max}$ i $(\alpha, \alpha-1)$- oo o ll $\alpha > 1$. F o ov ni ion, ollowin i i o w .

Remark 1. L Π (δ, ϵ)- oo on - ini i ion o l n l $I = (T, \sigma)$ n $\tilde{I} = (\tilde{T}, \sigma)$ wo in n o Π $t_i \leq \tilde{t_i} \leq \delta \cdot t_i$, o $i = 1, 2, \ldots, m$. T n, ny c- oxi ol ion S o I i $c\cdot(1+\epsilon)$- oxi o $\tilde{I}$ n ny c- oxi ol ion $\tilde{S}$ o $\tilde{I}$ i $c \cdot (1+\epsilon)$- oxi o I.

F o T o 3 n ov w v n x o .

Theorem 4. *Let Π be a $(\alpha, \alpha - 1)$-smooth optimization problem for verifiable one-parameter selfish agents whose types are positive rational, and let A be polynomial-time c-approximate weakly-monotone algorithm for Π. If every Θ_i is upper-bounded by a finite value $_i$, then $\mathcal{M} = (A_\alpha, P^{(2)})$ is $(\alpha \cdot c)$-approximate polynomial-time truthful mechanism for Π, satisfying voluntary participation condition.*

5 Applications to $Q||C_{max}$ Problem

In , i ion w iv non- ivi l. li ion o o l o w ll nown $Q||C_{max}$ o l . In in w i y i , n iv n c- oxi olyno i l- i w ly- ono on l o i o $Q||C_{max}$ o - l , w n on c- oxi olyno i l- i l ni o $Q||C_{max}$. On o n , w n w v no on in on y , n iv n c- oxi olyno i l- i w ly- ono on l o i o $Q||C_{max}$ o l , w n on $c\cdot(1+\epsilon)$- oxi olyno i l- i l - ni o $Q||C_{max}$, o v y $\epsilon > 0$.

W o S ion 1 o ni ion o o l . In $Q||C_{max}$ o l wi v i l on - l n [1], in own y v i l l n wi in o xi i i own o (i in ion 1) i in lo l n ini i ion. In i l , jo w i $W = (w_1, w_2, \ldots, w_n)$, li o in , n o in iv o in , i , n i iv ly now o in . A l, w y o n inv o .

A own in T o 1 n T o 3, o ly o y n , y - o n y ni v l . Fo $Q||C_{max}$ o l , in y inv o , i n o v y n (inv o l i) o low - o n y on n n 0, i o l no . T o , w ow o o l wi , . W ow i i lw y o i l o ny in n o $Q||C_{max}$ o on w v y y i - o n y ni v l vin o i o in n .

T i i o iv low o n on (n o n on l i) o n n in on l ion o o n . T , i n n l v l oo ll wi o o l , n n i . L o o o lly.

To o low o n o n w x ollowin l o i in in i v o B n w i o jo W. (W ll i l o i $BoundTypes(B, W)$.)

1. Fo ll $i \in \{1, \ldots, m\}$, i s_i i low - o n y on n $\hat{s_i} > 0$, n i v l low o n o in i, o wi x 2 – 3.
2. L k in in $\{1, \ldots, i-1\} \cup \{i+1, \ldots, m\}$ w i v o B (i in wi ll i wi o on i in in i); l i i n (on i in i b_k) o in k o x ll jo : i $_{\mathrm{i}} = \mathcal{W} \cdot {}_{\mathrm{k}}$.
3. L w_j ini w i jo ; v l $\hat{s}_i = \frac{w_j}{\mathrm{time_i}}$ low o n o o in i.

[1] In the rest of the paper, with an abuse of notation, we will simply call $Q||C_{max}$ problem the one with verifiable one-parameter selfish agents since here we deal only with the latter.

To n n o iv ion o i o w on i ollowin : i in i l $b_i > \frac{1}{\hat{s_i}}$, n o ny o i ol ion OPT w v $\mathsf{w}_i(OPT) = 0$, in xi in i in l i o x ll jo wi o i n o in i o o l jo vin ll w i . L A w ly- ono on l o i o $Q||C_{max}$ o l . Now, w i w ly- ono on l o i A' o $Q||C_{max}$ w i A o in . A' oxi ion io o A n n o l wi in vin n o n . I in i v o B n wi v o W n o l S.

1. L $(\hat{s}_1, \cdots, \hat{s}_m) = BoundTypes(B, W)$; l $\hat{B}$ i v o B wi o in i in $b_i > \frac{1}{\hat{s_i}}$; l $\hat{S}$ l n y A x on $\hat{B}$ n W;
2. L S l l o $\hat{S}$ o ll in l in $b_i < \frac{1}{\hat{s_i}}$ i nin 0 o ll o in ; n S l .

Now, w ow i l o i l o l ni (o wi o y n) n i oxi ion io l o i A.

Lemma 2. *If A is a weakly-monotone c-approximate algorithm for $Q||C_{max}$ problem, then A' is a weakly-monotone c-approximate algorithm for $Q||C_{max}$ problem.*

Proof. W ow A' i lin l o i . Sin A i lin l o i , w only v o ow l on in i iv n in o l o i A. Now, w ow w n v i in . L i in in $\{1, \ldots, m\}$ n k in in $\{1, \ldots, i-1\} \cup \{i+1, \ldots, m\}$. T n, o vio ly $s_k = \frac{1}{b_k} \leq \frac{1}{b_i} = s_i$. L T i n y in k o x ll jo n l w ll jo . F o ni ion o $\hat{s_i}$, w v $\hat{s_i} = \frac{w}{T} = \frac{w}{\mathcal{W}} \cdot s_k \leq s_k \leq s_i$. T i i li in ly no i . W now ov A' i w ly- ono on . Fix i v o B, n o $\mathsf{w}_i(A'(B, W)) = 0$. W ov $\mathsf{w}_i(A'((b', B_{-i}), W)) = 0$, o v y $b' \geq b_i$. In $b' > \frac{1}{\hat{s_i}}$, ivi lly $\mathsf{w}_i(A'((b', B_{-i}), W)) = 0$, in in i will i . I $b'_i \leq \frac{1}{\hat{s_i}}$, n in i no i n $\mathsf{w}_i(A'((b', B_{-i}), W)) = 0$, iv n l o i A i w ly- ono on . Fin lly, o ow l o i A' i c- oxi l o i , w only ov l ion o " low " in o no o i y o i . Mo i lly, l I ini i l in n o o l n OPT n o i ol ion o I. I $b_i > \frac{1}{\hat{s_i}}$ (i. in i i i in), n $\mathsf{w}_i(OPT) = 0$. In , i n y in i o o l ll jo w i n i n o in o o l ov ll jo . □

Al o i A', in l o i $BoundTypes$, ny (o n i lly n- o n) in n I o $Q||C_{max}$ o o n in n $\hat{I}$ o $Q||C_{max}$. T w n ly o y n . By L 2 n T o 2 w v :

Theorem 5. *Let A be a c-approximate polynomial-time weakly-monotone algorithm for $Q||C_{max}$ problem. If every Θ_i is discrete w.r.t. a known value Λ_i, then there exists a c-approximate polynomial-time truthful mechanism $\mathcal{M} = (A', P^{(1)})$ for $Q||C_{max}$, satisfying voluntary participation condition.*

By L 2, T o 4 n in $Q||C_{max}$ o l i $(1 + \epsilon, \epsilon)$- oo w v :

Theorem 6. *Let A be a c-approximate polynomial-time weakly-monotone algorithm for $Q||C_{max}$ problem. Then, for any $\epsilon > 0$, there exists a $c \cdot (1 + \epsilon)$-approximate polynomial-time truthful mechanism $\mathcal{M} = (A', P^{(2)})$ for $Q||C_{max}$, satisfying voluntary participation condition.*

Acknowledgments. W wi o n o o [4] o ovi in wi ll v ion o i .

References

1. N. Andelman, Y. Azar, and M.Sorani. Truthful approximation mechanisms for scheduling selfish related machines. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*. LNCS, 2005.
2. A. Archer and E. Tardos. Truthful mechanisms for one-parameter agents. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 482–491, 2001.
3. V. Auletta, R. De Prisco, P. Penna, and P. Persiano. Deterministic Truthful Mechanisms for Scheduling on Selfish Machines. In *Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2996, pages 608–619. LNCS, 2004.
4. V. Auletta, R. De Prisco, P. Penna, and P. Persiano. The Power of Verification for One-Parameter Agents. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3142, pages 171–182. LNCS, 2004.
5. E. H. Clarke. Multipart pricing of public goods. *Public Choice*, pages 17–33, 1971.
6. R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, (45):1563–1581, 1966.
7. R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematic*, 17(2), 1969.
8. N. Nisan and A. Ronen. Algorithmic Mechanism Design. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, pages 129–140, 1999.
9. N. Nisan and A. Ronen. Computationally Feasible VCG Mechanisms. In *Proceedings of the 2nd ACM Conference on Electronic Commerce (EC)*, pages 242–252, 2000.
10. V. Vazirani. *Approximation Algorithms*. Springer, 2001.
11. W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

Symmetry in Network Congestion Games: Pure Equilibria and Anarchy Cost*

Di i i Fo i [1], S y o Kon o i nni [2,3], n P l S i i [2]

[1] Dept. of Information and Communication Systems Engineering, University of the Aegean, 83200 Samos, Greece
fotakis@aegean.gr
[2] Research Academic Computer Technology Institute, 26500 Patras, Greece
{kontog, spirakis}@cti.gr
[3] Dept. of Computer Science, University of Ioannina, 45110 Ioannina, Greece

Abstract. We study computational and coordination efficiency issues of Nash equilibria in symmetric network congestion games. We first propose a simple and natural greedy method that computes a pure Nash equilibrium with respect to traffic congestion in a network. In this algorithm each user plays only once and allocates her traffic to a path selected via a shortest path computation. We then show that this algorithm works for series-parallel networks when users are identical or when users are of varying demands but have the same best response strategy for any initial network traffic. We also give constructions where the algorithm fails if either the above condition is violated (even for series-parallel networks) or the network is not series-parallel (even for identical users). Thus, we essentially indicate the limits of the applicability of this greedy approach.

We also study the price of anarchy for the objective of maximum latency. We prove that for any network of m uniformly related links and for identical users, the price of anarchy is $\Theta(\frac{\log m}{\log\log m})$.

1 Introduction

N wo on ion ovi o n o l o l o in o n li- l ffi n v nly n j o in n iv . T v il- in ion in n wo v o o wi o n ion o l o ' oo in ion (., [23,12,10,1,3]) n ffi i n o ion o N ili i (., [8,11,10]).

A n l y o o o in N ili i (PNE) i G y B R on (GBR). L on i yn i in wi n w ivin in n wo . T l y only on n i vo ly oo i y on iv l. E n w o ffi on ini l y iv n o n ly in n wo . Ho lly xi in

---

* This work was partially supported by the EU within the Future and Emerging Technologies Programme under contract IST-2001-33135 (CRESCCO) and within the 6th Framework Programme under contract 001907 (DELIS).

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 161–175, 2006.

no ff y n w on n n wo on ion in PNE wi o ny ion in l . T i o i no only in i iv n o ion lly ffi i n , l o l ow in wo in i . A n l ion i w ny in in l o n wo o w i G y B R on in in PNE.

G y B R on n n li ion o G ' LPT l o i [13]. T i ion o GBR o ll l-lin n wo i nown o in in PNE o i y non- in l n y n ion n w i ivin in non-in in o o w i [17,9]. In i wo , w ov GBR in in PNE o y i on ion in i - ll l n wo . T i l i x n o w i on ion wi in no ion o y y, n ly v on i o ny ini i l n wo ffi .

T on i o n i ion o o wi in ffi i n y o N ili i . T *coordination ratio* o *price of anarchy* w in o in [16] o in o n ion o l o ' oo in ion in o in . T i o n y i wo - io w n o o N ili i n o o n o i l ol ion. Fo n wo on ion , wo n l no ion o o o nin i o n y: *total* n *maximum* l n y. A i o n y o non- o i on ion o w ll- n oo (., [23,21] o o l l n y n [22,4] o xi l n y), in ov o o i in (. [18,12,10,1,3].) In o in o lin l n i i o in n n n o o o o vio wo .

In i , w y i o n y l iv o o j iv o x- i l n y o y i n wo on ion n l n y n ion $d_e(x) = a_e x, a_e \geq 0$. T i o on o ni o ly l lin , wi o ffi- i n a_e no in inv o lin e. W ow i o n y o ny n wo o m lin i $\Theta(\frac{\log m}{\log\log m})$.

Related Work. Ro n l [20] ini i y o on ion n ov i PNE o on o lo l o i o n l o n i l n ion. T o on yn i onv o PNE. On o n i i PLS- o l o n PNE in y i (no n ily n wo) n non- y i n wo on ion [8]. On o i iv i , [8] ow in y i *network* on ion , PNE n o n y in- o flow o ion. Fo w i on ion , [11] on i o i n i l ll l lin n i i n n n ow ow o o PNE in on ly- olyno i l i . [10] ow w i on ion wi lin l n i i w i o n i l n ion. T on yn i onv o PNE in o- olyno i l i .

In in l , Ko o i n P i i io [16] in o i o n y n on i o j iv o xi l n y o w i on- ion on m ni o ly l ll l lin . T i o n y o i $\Theta(\frac{\log m}{\log\log m})$ i i o lin i n i l [19,15,5] n $\Theta(\frac{\log m}{\log\log\log m})$ o wi [5]. Fo ni o ly l ll l lin , i n i l

. ., . n , o j iv o o . l l n y, , i o . n , , y i $2 - o(1)$ o, , . n . l . o ix ili i . n $4/3$ o, ili i [18,12].

Si il . l . . v n o . in n ly o n wo on ion . wi lin . l n y n ion . T i o . n , , y o o j iv o o . l l n y i $\frac{3+\sqrt{5}}{2}$ i w i , on ion . . n ix ili i . on i [1]. T i o . o $5/2$ o, i l . o i n i . l . n ili i ([1] . n in n n ly in [3]). T i o . n , , y o . xi l n y i . l o $5/2$ o N . ili i . n y i . (wi i n i . l .) . n o $\Theta(\sqrt{n})$ o non- y i . [3].

On o , , . n , i o . n , , y o m i n i . l lin . n o j - iv o . xi l n y i $\Omega(\frac{\log m}{\log \log m})$ i ix N ili i . on i [16,19]. [10] i w i , . in l - o o i y on ion . . in l y n - wo wi m i n i . l lin . n ow i o . n , , y o . xi l n y . . in $\Theta(\frac{\log m}{\log \log m})$ o n . l . o ix N ili i .

T o n . ov . ly o *atomic* in , wi on ollin . non- n . li i l . o n o ffi . n , . n on i o . n ix ili i (wi x ion o l in [3] on . xi l n y). I ov o n . n o . in in *non-atomic* in , w on ol . n li i l . o n o . n . n . n ix ili i iv l n . [23] ini i y o i o . n , , y o o j iv o o . l l n y in non- . o i in . n ow i o . n , , y o lin . l n i i $4/3$. [21] ov i o . n , , y n on l . o l n y n ion . n no on n wo o olo y . n iv . i o n o v y l . .

A o o j iv o . xi l n y in non- o i in , on o o . l l n y . l o . ly o . xi l n y in . in l - o o i y n wo [4]. Fo l i- o o i y n wo , i o . n , , y i $\Omega(|V|)$ v n o lin . l n i [4]. On o , , . n , i o . n , , y o . xi l n y i . o $|V| - 1$ in . in l - o o i y n wo . [22].

Contribution. I i n i . l, GBR . v . n onlin . l o i . Fo w i , GBR i o n . l y . l o i . in i in . x o in w i on i . n . n i vo . l y oi iv n oi o vio . .

In i , w n i lly i l . o n wo on ion . . o w i GBR . in . in . PNE. Mo i . lly, w ov . GBR . in- . in . PNE o y i on ion . in i - . . ll l n wo . T i i x n o w i on ion . wi *common best response* o y. T i o y i . v on i o . ny ini i l n wo ffi . In . i ion o y i n wo on ion . , i l . in l w i on ion . in l y n wo wi i n i . l (i. ., l y . iv n y . o on lin . l n y n ion). W . l o ov i ion o i - . . ll l n wo . n wi o on on o y i n i lly n . . y o GBR o . in . in . PNE.

Fo i o . n , , y, w o on o j iv o . xi l n y. W on i y i n wo on ion . . n lin . l n i wi no . - i iv , x n in o . i . y n wo wi ly- i in o i n i l n ni o ly l ll l lin (., [16,9,19,5]). W on- i n l o ix ili i n ow i o n y in $\Theta(\frac{\log m}{\log\log m})$ o i n i l n n wo o m lin . T in o i n i l n i y n wo i o o on l o in o [10] w n wo no ion o y y, n ly ll v l n n on i o i n i l n v iff n w i .

T l on i o n y w o in in n n ly o l o [1,3]. O o i n n lly iff n n y o in n n in . I i on n l o on n w n ix i n ion l $s-t$ flow (l o [10]). To o iv o , w ow o i l ol ion o i o o on o y i ix N ili i . W i o in li y n ow x o o ny in (o ix) N ili i i o 3 i o i l xi l n y. A C noff-Ho ff in o n yi l x xi l n y i $O(\frac{\log m}{\log\log m})$ i o i l xi l n y.

2 Definitions and Preliminaries

The Model. A *network congestion game* i l $(N, G, (d_e)_{e\in E})$, w $N = \{1,\ldots,n\}$ i o on ollin ni o ffi n , $G(V,E)$ i i n in o ni ion n wo , n d_e i l n y n ion o i wi $e \in E$. W d_e' non-n iv n non- in n ion o lo . I l y iv n y *common linear* l n y n ion, w y *identical*. Fo i n i l , w wlo l y iv n y i n i y n ion, i. $\forall e \in E, d_e(x) = x$. W i o n ion o *single-commodity* n wo on ion , w n wo G in l o s n in ion t n o ' i i o $s-t$, no $\mathcal{P}$. Wlo w G i onn n v y v x o G li on i $s-t$.

W l o on i *weighted* in l - o o i y n wo on ion , w i on ol w_i ni o ffi n [1]. T in x in non-in in o o w i , i. ., $w_1 \geq w_2 \geq \ldots \geq w_n$. Sin l - o o i y n wo on ion *symmetric*[2]. How v , w i non- y i in n l ' o n ion iff n n non- y i o iff n w i .

A v o $P = (p_1, \ldots, p_n)$ on i in o n $s-t$ p_i o i i *pure strategies profile.* L $\ell_e(P) \equiv \sum_{i:e\in p_i} w_i$ no lo o e in P. T o $\lambda^i_p(P)$ o i o o in n on p in o l P i

$$\lambda^i_p(P) \equiv \sum_{e\in p\cap p_i} d_e(\ell_e(P)) + \sum_{e\in p\setminus p_i} d_e(\ell_e(P) + w_i)$$

T o $\lambda^i(P)$ o i in P i $\lambda^i_{p_i}(P)$, n ly o l l y lon .

[1] In (unweighted) congestion games, $w_1 = w_2 = \ldots = w_n = 1$.

[2] A game is *symmetric* if all users have the same strategy set and the users' costs are given by identical symmetric functions of other users' strategies. In congestion games, the users are identical and a common strategy set implies symmetry.

A v o $Q = (q_1, \ldots, q_n)$ on i in o o ili y i i ion q_i ov $\mathcal{P}$ o i i *mixed strategies profile.* Fo p, $q_i(p)$ no o ili y i o n on p. L $\ell_p(Q) \equiv \sum_{j=1}^{n} q_j(p) w_j$ x lo o on p in Q, n l $\ell_p(Q^{-i}) \equiv \ell_p(Q) - q_i(p) w_i$ x lo on p x l in on i ion o i. Si il ly, l $\ell_e(Q) \equiv \sum_{p: e \in p} \ell_p(Q)$ n $\ell_e(Q^{-i}) \equiv \sum_{p: e \in p} \ell_p(Q^{-i})$ x lo on e wi n wi o i iv ly. T o $\lambda_p^i(Q)$ o i o o in n on p in ix i o l Q i x ion o in o Q^{-i} o $\lambda_p^i(P^{-i} \oplus p)$ ov ll i o l [3] P^{-i}. T o $\lambda^i(Q)$ o i in Q i x ion o in o Q o $\lambda^i(P)$ ov ll i o l P.

Fo i o l Q, l $\lambda^{\max}(Q) \equiv$ x$_{i \in N}\{\lambda^i(Q)\}$ xi o in Q.

In i , w on i ix i o l only o i n i l n lin l n y n ion $d_e(x) = a_e x$. T n, i ly $\lambda_p^i(Q) \equiv \sum_{e \in p} a_e(\ell_e(Q^{-i}) + 1)$ n $\lambda^i(Q) \equiv \sum_{p \in \mathcal{P}} q_i(p) \lambda_p^i(Q)$ y lin i y o x ion.

A ix (in n l) i o l Q i *Nash equilibrium* i o v y i n v y $p, p' \in \mathcal{P}$ wi $q_i(p) > 0$, $\lambda_p^i(Q) \leq \lambda_{p'}^i(Q)$. T o , i Q i N ili i , $\lambda_p^i(Q) = \lambda_{p'}^i(Q) = \lambda^i(Q)$ o v y i n v y $p, p' \in \mathcal{P}$ wi o $q_i(p), q_i(p') > 0$.

W v l i o l in o j iv o *maximum latency.* T xi l n y $L(P)$ o i o l P i xi o in P, $L(P) \equiv \lambda^{\max}(P)$. T xi l n y $L(Q)$ o ix i o l Q i x ion o in o Q o $L(P)$ ov ll i o l P, $L(Q) \equiv \sum_{P \in \mathcal{P}^n} \mathbb{P}(P, Q) L(P)$, w $\mathbb{P}(P, Q) = \prod_{i=1}^{n} q_i(p_i)$ i o n o ili y o P in Q. T o i l ol ion, no P^*, o on o i o l n o i l o i $L(P^*)$. T i o n y i n wo - io $L(Q)/L(P^*)$ ov ll N ili i Q.

Flows. A i l flow i n ion $f : \mathcal{P} \mapsto \mathbb{R}_{\geq 0}$ $\sum_{p \in \mathcal{P}} f_p = \sum_{i=1}^{n} w_i$. W l o f o no $|\mathcal{P}|$- i n ion l v o o on in o flow f. A flow i *unsplittable* i ' n i o on in l n *splittable* o wi . L $f_e \equiv \sum_{p: e \in p} f_p$ no flow on e.

Greedy Best Response. GBR on i on - y-on in non-in in o o w i . E o on y iv n i o vio . T oi i i vo l in no n n y in . In i l wo , l y only on n l i on y o n i on i y l o i .

Fo lly, l p_i o i, n l $P^i = (p_1, \ldots, p_i)$ i o l o 1, ..., i. T n, p_{i+1} o $i + 1$ i

$$p_{i+1} = \ldots \mathrm{in}_{p \in \mathcal{P}} \{\textstyle\sum_{e \in p} d_e(\ell_e(P^i) + w_{i+1})\} \tag{1}$$

W y GBR *succeeds* i v y o l P^i i N ili i .

---

[3] For a n-dimensional vector X, $X^{-i} \equiv (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ and $X^{-i} \oplus x \equiv (x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_n)$.

Common Best Response. T in l - o o i y n wo on ion $((w_i)_{i\in N}, G, (d_e)_{e\in E})$ *common best response* o y i o v y ini i l flow f (no n ily i l), ll v o on i w lo in y f. In o wo , i p i on w f o o n ollowin in li y ol o ll j n ll p':

$$\sum_{e\in p'} d_e(f_e + w_j) \geq \sum_{e\in p} d_e(f_e + w_j)$$

F o v y n π o on p i on o o in n o ny w n π' n oin . W o l i li in ni ion ov on o wi o in in o o n o y l y on i o ini i l flow f. T o on on o y i no ion o y y w n n ly ll o v o o n o ny ini i l ffi on i ion . T i no ion o y y i w n o y i ill on no o GBR wo in i - ll l n wo .

Layered and Series-Parallel Graphs. A i (l i) $G(V, E)$ wi i in i o s n in ion t i *layered* i ll i $s - t$ v x ly l n n v x li on i $s - t$. A l i i *series-parallel with terminals* (s, t) i i i i in l (s, t) o n o in o wo i - ll l wi in l (s_1, t_1) n (s_2, t_2) onn i in *series* o in *parallel*. In i onn ion, t_1 i i n i wi s_2, s_1 o s, n t_2 o t. In ll l onn ion, s_1 i i n i wi s_2 n o s, n t_1 i i n i wi t_2 n o t. A i wi in l (s, t) i i - ll l i n only i i o no on in θ- wi -2 in l o olo i l ino (Fi . 1.) [7].

Proposition 1. *Let $G(V, E)$ be a series-parallel graph with terminals (s, t), and let vertices u, v connected by two disjoint paths, denoted π and π', only sharing their endpoints. Every $s - t$ path having at least one edge in common with π' contains both u and v.*

3 Greedy Best Response in Series-Parallel Networks

W ow GBR i n wo i i - ll l n o on on o y.

Theorem 1. *If G is a series-parallel graph with terminals (s, t) and the game $((w_i)_{i\in N}, G, (d_e)_{e\in E})$ has the common best response property,* GBR *succeeds and computes a pure Nash equilibrium in time $O(nm \,\mathrm{lo}\ m)$.*

Proof. T oo i y in ion on n o on i y l o i . T l i ol o in o on y n i only in n wo . W in iv ly i n on i , $P^i = (p_1, \ldots, p_i)$ i N ili i . L p_{i+1}

o n y $i+1$ o in o (1). To on i ion, w $P^{i+1} = (p_1, \ldots, p_i, p_{i+1})$ i no N ili i .

Con n ly, i j, $j \leq i$, in no p o p_j. L u *split* oin w p o p_j (u y s). Any i o iff n l on li oin y v o on o . L v *merge* oin u w p join p_j in (v y t). E li oin i ollow y oin v o on in ion.

Fo i li i y o no ion, l π n π_j no n o p n p_j iv ly w n u n v. By ni ion o v, π n π_j i join n v only i n oin u n v in o on.

Sin j w n o o π_j in P^{i+1} no in P^i, i i p_{i+1} o wi π_j n i in io o π o j. Sin p_{i+1} n π_j v l on in o on, p_{i+1} on in o u n v y P o o i ion 1. L π_{i+1} n o p_{i+1} w n u n v (Fi . 1).

T p_{i+1} i on o $i+1$ w flow in y P^i. Sin o on on o y, p_{i+1} i l o on o j w flow in y P^i (i no in w_j l y on i o flow). T o n π_{i+1} i on w P^i o o in n o j o u o v:

$$\sum_{e \in \pi} d_e(\ell_e(P^i) + w_j) \geq \sum_{e \in \pi_{i+1}} d_e(\ell_e(P^i) + w_j) \tag{2}$$

Sin j π o π_j $i+1$ o in ffi on π_{i+1},

$$\sum_{e \in \pi_j \setminus \pi_{i+1}} d_e(\ell_e(P^i)) + \sum_{e \in \pi_j \cap \pi_{i+1}} d_e(\ell_e(P^i) + w_{i+1}) >$$

$$\sum_{e \in \pi} d_e(\ell_e(P^i) + w_j) \geq \sum_{e \in \pi_{i+1}} d_e(\ell_e(P^i) + w_j) \geq$$

$$\sum_{e \in \pi_{i+1} \setminus \pi_j} d_e(\ell_e(P^i) + w_j) + \sum_{e \in \pi_{i+1} \cap \pi_j} d_e(\ell_e(P^i) + w_{i+1})$$

T on in li y ollow o In . (2). T l in li y ol l n y n ion non- in n $w_j \geq w_{i+1}$.

I $\pi_j = \pi_{i+1}$, on i ion i i i . I $\pi_j \neq \pi_{i+1}$, j n $\pi_{i+1} \setminus \pi_j$ o n $\pi_j \setminus \pi_{i+1}$ v n in P^i:

$$\lambda^j_{\pi_j \setminus \pi_{i+1}}(P^i) = \sum_{e \in \pi_j \setminus \pi_{i+1}} d_e(\ell_e(P^i)) > \sum_{e \in \pi_{i+1} \setminus \pi_j} d_e(\ell_e(P^i) + w_j) = \lambda^j_{\pi_{i+1} \setminus \pi_j}(P^i)$$

T i on i o in iv y o i P^i i N ili i . T o , p n p_j o no v ny li oin n p oin i wi p_j. Con n ly, P^{i+1} i N ili i .

GBR o $n\,s-t$ o o ion in o m . T i n on in i $O(nm \log m)$ in Dij ' l o i . □

Sin l - o o i y n wo on ion wi i n i l v o - on on o y ' o n ion i n i l n - ion o lo . W l o w o l o w i in l - o o i y n wo on ion wi o on on o y.

Proposition 2. *A weighted single-commodity congestion game in a layered network with identical edges has the common best response property for any set of user weights.*

Corollary 1. GBR *succeeds for single-commodity congestion games in series-parallel networks:*

1. *if the users are* i n i l *(for arbitrary non-decreasing edge delays).*
2. *if the graph is* l y *and the edges are* i n i l *(for arbitrary user weights).*

GBR n l i i i l n ion on l l ion l- o i . T i o o on in o l y . W o now n wo n l n y n ion . W l o in y n n n lyin yn oni ion ni (lo i l i) llowin i i l o i o o in lo i l o n .

Ini i lly, ll o iv . In o n y n l l ion l o i n in iv o o l w i . T i o o i n on i on nno n i y o inin iv o n o iv . No i ll o n o i on lo lly. In offlin in l o i in oon no iv o . In onlin in n w / o y n y ny oin in i .

W on l y o GBR y ovi in o i l x l on- in GBR y no in in inin PNE i i n wo i no i - ll l o o no v o on on o y. H n o on i ion o T o 1. n y o GBR o .

I n wo i no i - ll l i l *symmetric game* o w i GBR il on i o wo i n i l n 3-l y iv l n o θ- wi i n i l (Fi . 1.). T N ili i i n on o π_1 n o o π_3. I GBR i n o π_2, i no y o on yi l N ili i . W n o GBR o i n o π_2 y li ly in l n y n ion o on o $(1-\epsilon)x$, w ϵ i ll o i iv on n .

T o on on o y i l o n y o i - ll l n - wo o n n o ll l-lin onn in i [4]. Fo x l l on i 2-l y i - ll l o Fi . 1. n o w i $w_1 = 100$, $w_2 = 10$, n $w_3 = 4$. T o on in on ion o no v o on on o y. GBR i n o π_1, on o π_2, n i o π_3, w il in v y N ili i wo i n o π_1.

[4] If the network consists of bunches of parallel-link connected in series, a pure Nash equilibrium can be computed by independently applying GBR to each bunch of parallel links.

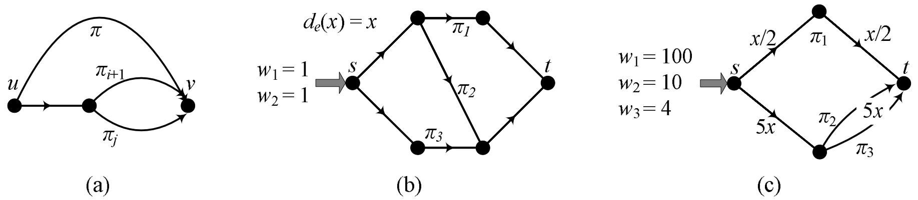


Fig. 1. (a) The graph in the proof of Theorem 1. GBR may fail if (b) the network is not series-parallel (even if the game is symmetric) and (c) the game does not have the common best response property (even if the network is series-parallel).

4 The Price of Anarchy in Networks of Uniformly Related Links

W o o o n i o n y in y i n wo on ion wi ni o ly l lin W on i n i n i l o in i (ni) ffi n on i $G(V, E)$ wi ni o s n in ion t, n $m \equiv |E|$ T i lin l n y n ion $d_e(x) = a_e x$, $a_e \geq 0$, o i wi e. W a_e inv o e. Fo $p \in \mathcal{P}$, l $a_p \equiv \sum_{e \in p} a_e$ no inv o p.

Flows and Mixed Strategies Profiles. A i l flow i n ion $f : \mathcal{P} \mapsto \mathbb{R}_{\geq 0}$ $\sum_{p \in \mathcal{P}} f_p = n$. L $\theta_p(f) \equiv \sum_{e \in p} a_e f_e$ no o l l y lon p w f. W ix (in n l) i o l $Q = (q_1, \ldots, q_n)$ o i l flow f^Q ollow : Fo $s-t$ $p \in \mathcal{P}$, $f_p^Q \equiv \ell_p(Q)$. In o wo , w n l x lo o on p in Q li l flow, w i o ion $q_i(p)$ o n on p. I Q i i o l , o on in flow i n li l .

W y i l flow f^Q o on in o i o l Q i N ili i wi n n in lly Q i N ili i Fo v y N ili i Q n o on in flow f^Q,

$$\lambda^{\max}(Q) \leq \min_{p \in \mathcal{P}}\{\theta_p(f^Q) + a_p\} \equiv \delta^{\min}(f^Q) \tag{3}$$

O wi o o $\lambda^{\max}(Q)$ in Q o l i ov o y wi in o ini i in $\theta_p(f^Q) + a_p$. F o , o ny $p \in \mathcal{P}$ wi $f_p^Q > 0$,

$$\max\{\theta_p(f^Q), a_p\} \leq \lambda^{\max}(Q) \leq \delta^{\min}(f^Q) \tag{4}$$

Fo i li i y, w o i o Q o i o on in flow f^Q w n i o l i l o on x .

Total Latency and Total Load. A flow f n v l y i *total latency* n

$$C(f) \equiv \sum_{p \in \mathcal{P}} f_p \theta_p(f) = \sum_{e \in E} a_e f_e^2$$

In . i ion, . flow f . n v. l . y i . *total load* n . .

$$W(f) \equiv \sum_{e\in E} a_e f_e = \sum_{e\in E} a_e \sum_{p:e\in p} f_p = \sum_{p\in \mathcal{P}} a_p f_p$$

W . o i . . $W(P^*)$ o no , o . l lo. o , flow o. . . on in o , o i . l . ol ion P^*.

Proposition 3. *Let f be a feasible flow at Nash equilibrium. Then $C(f) \le n\,\delta^{\min}(f)$.*

Proof. By In . (4), o. v . y . , $p \in \mathcal{P}$, $f_p\,\theta_p(f) \le f_p\,\delta^{\min}(f)$. S . . in ov . . ll . . . . , w on l , . $C(f) \le n\,\delta^{\min}(f)$. □

L M , $|\mathcal{P}| \times |\mathcal{P}|$ . . . . ix n . . $M[p,p'] \equiv \sum_{e\in p\cap p'} a_e$ o. . , . i. o . . , . $p, p' \in \mathcal{P}$. By ni ion, M i . . y. . i . . . ix. Fo. v . y flow f, Mf i , $|\mathcal{P}|$- i n ion. l v o. wi , oo. in . $\theta_p(f)$. T, . . , . o . l l. n y o f . n x . . . . . . $C(f) = f^T M f$. Sin $C(f) = f^T M f = \sum_{e\in E} a_e f_e^2$, . . . ix M i o. i iv . . i- ni [5].

L A , $|\mathcal{P}|$- i n ion. l v o. wi , $A[p] \equiv a_p$ o. . , . . , $p \in \mathcal{P}$. T, o . l lo. o v . y flow f . n x . . . . . $W(f) = A^T f$.

T, *maximum latency* o . n n li . l flow f i $L(f) \equiv$ . . $\mathrm{x}_{p:f_p>0}\{\theta_p(f)\}$. No i , . o. v . y . . . . . i . . . o l P . n i . o. . . on in n li . l flow f^P, $L(P) = L(f^P)$.

4.1 Computing a Symmetric Nash Equilibrium

W n x . ov , . , flow . ini i. in $\frac{n-1}{2n}C(f) + W(f)$ o. . . on . o . . y. . . i N. . , ili . i . . Fo. . . lly, l $\hat{f}$, o i . l . . ion. l . ol ion o , ollowin . . . . i . o . . . in$\{\frac{n-1}{2n} f^T M f + A^T f : \mathbf{1}^T f \ge n, f \ge \mathbf{0}\}$, w, . $\mathbf{1}$ (. . . $\mathbf{0}$) no . , $|\mathcal{P}|$- i n ion. l v o. wi , 1 (. . . 0) in . , oo. in . W o . . v , . $\hat{f}$ i . . li . l flow o v. l n.

Lemma 1. *Let Q be the mixed strategies profile where each user i routes its demand on every path p with probability $q_i(p) = \hat{f}(p)/n$. Then, Q is a symmetric Nash equilibrium.*

Proof. T, . ix . . . . . i . . . o l Q i . y. . . i y ni ion. W only , . v o . , ow , . Q i . N. . , ili . i . . By on , ion, o. v . y . . i . n v . y . . , p, $\ell_p(Q^{-i}) = \frac{n-1}{n}\hat{f}_p$. T, . o. , o. v . y . . i . n v . y . . e, $\ell_e(Q^{-i}) = \frac{n-1}{n}\hat{f}_e$. T, . . , , o. o . . . . i . o in , ... n on . . . , p in , . ix . . . . . i . . . o l Q i

$$\lambda_p^i(Q) = \sum_{e\in p} a_e(\ell_e(Q^{-i}) + 1) = \sum_{e\in p} a_e(\tfrac{n-1}{n}\hat{f}_e + 1) = \tfrac{n-1}{n}\theta_p(\hat{f}) + a_p$$

[5] A $n \times n$ matrix M is positive semi-definite if for every vector $x \in \mathbb{R}^n$, $x^T M x \ge 0$.

T flow $\hat{f}$ ini i , onv x n ion $\sum_{e\in E}(\frac{n-1}{2n}a_e f_e^2 + a_e f_e)$. T o , o v y $p, p' \in \mathcal{P}$ wi $\hat{f}_p > 0$, ollowin in li y ol (., [2], [23, L 2.5]):

$$\tfrac{n-1}{n}\theta_p(\hat{f}) + a_p = \sum_{e\in p}(\tfrac{n-1}{n}a_e\hat{f}_e + a_e) \le \sum_{e\in p'}(\tfrac{n-1}{n}a_e\hat{f}_e + a_e) = \tfrac{n-1}{n}\theta_{p'}(\hat{f}) + a_{p'}$$

Con n ly, o v y i n v y $p, p' \in \mathcal{P}$ wi $q_i(p) = \hat{f}_p/n > 0$,

$$\lambda_p^i(Q) = \tfrac{n-1}{n}\theta_p(\hat{f}) + a_p \le \tfrac{n-1}{n}\theta_{p'}(\hat{f}) + a_{p'} = \lambda_{p'}^i(Q)$$

n ix i o l Q i N ili i . □

Remark. I n wo on i o m ni o ly l ll l lin , i- li i o L 1 i i n i l o *generalized fully mixed* N ili i o [9, T o 5].

4.2 Bounding the Price of Anarchy

W ly C noff-Ho ff in o n n ov o v y N ili i Q wi $\lambda^{\max}(Q) \le \alpha\, L(P^*)$ o o on n $\alpha \ge 1$, $L(Q) = \alpha\, O(\frac{\log m}{\log\log m})\, L(P^*)$ (L 2). W n ov o v y N ili i Q, $\lambda^{\max}(Q) \le L(P^*) + \frac{2}{n}\, W(P^*)$ (T o 2). T oo i on Do n' T o [6] li in on li y in i o in . A n i i on n , w o in o v y N ili i Q, $\lambda^{\max}(Q) \le 3\, L(P^*)$ (Co oll y 2). T l in i ion n x n o y i (no n ily n wo) on ion wi i n i l , o E, y $\mathcal{P}$, n o o $d_e(x) = a_e x$, $a_e \ge 0$.

Lemma 2. *Let Q be any strategies profile at Nash equilibrium. If there exists some constant $\alpha \ge 1$ such that $\lambda^{\max}(Q) \le \alpha\, L(P^*)$, $L(Q) \le \alpha\, O(\frac{\log m}{\log\log m})\, L(P^*)$.*

Proof. Fo v y e n v y i, l $X_{e,i}$ n o v i l i in l lo o on e y i. T n o v i l $X_{e,i}$ i 1 i i o i n on on inin e n 0 o wi . T x ion o $X_{e,i}$ i $\mathbb{E}[X_{e,i}] = \sum_{p:e\in p} q_i(p)$. Sin l i in n ly, o v y e, n o v i l $\{X_{e,i},\ i \in N\}$ lly in n n .

Fo e, l $X_e = a_e \sum_{i=1}^n X_{e,i}$ n o v i l i l l y in y ny v in e. M l i lyin $X_{e,i}$ y a_e, w n X_e o n in n n n o v i l wi v l in $\{0, a_e\}$. By lin i y o x ion,

$$\mathbb{E}[X_e] = a_e \sum_{i=1}^{n} \mathbb{E}[X_{e,i}] = a_e \sum_{p:e\in p}\sum_{i=1}^{n} q_i(p) = a_e \sum_{p:e\in p} \ell_p(Q) = a_e \ell_e(Q)$$

T Ho ff in o n [6] o $w = a_e$ n $t = \kappa a_e$ x$\{\ell_e(Q), 1\}$, yi l o v y $\kappa \geq 1$,

$$\mathbb{P}[X_e \geq \kappa a_e \text{ x}\{\ell_e(Q), 1\}] \leq \kappa^{-\mathrm{e}\,\kappa}$$

A lyin nion o n , w on l

$$\mathbb{P}[\exists e \in E : X_e \geq \kappa a_e \text{ x}\{\ell_e(Q), 1\}] \leq m\kappa^{-\mathrm{e}\,\kappa} \tag{5}$$

Fo v y $p \in \mathcal{P}$ wi $\ell_p(Q) > 0$, w n n o v i l $X_p = \sum_{e \in p} X_e$ i in l l y lon p. T xi l n y o Q nno x x xi l y on p wi $\ell_p(Q) > 0$. Fo lly,

$$L(Q) \leq \mathbb{E}[\underset{p:\ell_p(Q)>0}{\text{x}}\{X_p\}]$$

L o ll $e \in E$, $X_e < \kappa a_e$ x$\{\ell_e(Q), 1\}$. L p ny wi $\ell_p(Q) > 0$, n l i ny wi $q_i(p) > 0$. T n,

$$\begin{aligned} X_p = \textstyle\sum_{e \in p} X_e < \kappa \textstyle\sum_{e \in p} a_e \text{ x}\{\ell_e(Q), 1\} &\leq \kappa \textstyle\sum_{e \in p} a_e(\ell_e(Q^{-i}) + 1) \\ = \kappa \lambda^i(Q) \leq \kappa \lambda^{\max}(Q) &\leq \kappa \alpha L(P^*) \end{aligned}$$

T on in li y ollow o x$\{\ell_e(Q), 1\} \leq \ell_e(Q^{-i}) + 1$ w i ol o v y $e \in p$ n v y i wi $q_i(p) > 0$. Sin $q_i(p) > 0$ n Q i N ili i , $\lambda^i(Q) = \sum_{e \in p} a_e(\ell_e(Q^{-i}) + 1)$ n n x li y ollow . T i in li y ollow o ni ion o $\lambda^{\max}(Q)$ n l in li y y y o i . T o in In . (5), w on l

$$\mathbb{P}[\underset{p:\ell_p(Q)>0}{\text{x}}\{X_p\} \geq \kappa \alpha L(P^*)] \leq m\kappa^{-\mathrm{e}\,\kappa}$$

In wo , o ili y l xi l y y Q x o i l xi l y y o n $\kappa \alpha$ i o $m\kappa^{-\mathrm{e}\,\kappa}$. T o , o v y $\kappa_0 \geq 2$,

$$\begin{aligned} L(Q) \leq \mathbb{E}[\underset{p:\ell_p(Q)>0}{\text{x}}\{X_p\}] &\leq \alpha L(P^*)\big(\kappa_0 + \textstyle\sum_{k=\kappa_0}^{\infty} m k^{-\mathrm{e}\,k}\big) \\ &\leq \alpha L(P^*)\big(\kappa_0 + 2m\kappa_0^{-\mathrm{e}\,\kappa_0}\big) \end{aligned}$$

Fo $\kappa_0 = \frac{2 \log m}{\log \log m}$, w o in $L(Q) \leq 2\ \alpha\,(\frac{\log m}{\log \log m} + 1)\, L(P^*)$. □

Theorem 2. *For every strategies profile Q at Nash equilibrium, $\lambda^{\max}(Q) \leq L(P^*) + \frac{2}{n} W(P^*)$.*

[6] We use the standard version of Hoeffding bound [14]: Let $X_1, X_2, \ldots, X_n$ be independent random variables with values in the interval $[0, w]$. Let $X = \sum_{i=1}^{n} X_i$ and let $\mathbb{E}[X]$ denote its expectation. Then, $\forall t > 0$, $\mathbb{P}[X \geq t] \leq (\frac{\mathrm{e}\,\mathbb{E}[X]}{t})^{t/w}$.

Proof. L $\overline{f}$ no o i l ion l ol ion o ollowin i o : QP $\equiv$ in$\{f^T(\frac{1}{2}M)f + A^T f : \mathbf{1}^T f \geq n, f \geq \mathbf{0}\}$. No i $\overline{f}$ i li l flow o v l n. W ov $n\lambda^{\max}(Q) \leq C(\overline{f}) + 2\,W(\overline{f})$.

W Do n' T o [6], w i li on li y in i o in [7], n ov o v y flow f,

$$n\,\delta^{\min}(f) - \tfrac{1}{2}C(f) \leq \tfrac{1}{2}C(\overline{f}) + W(\overline{f}) \tag{6}$$

T i o QP $\equiv$ in$\{f^T(\frac{1}{2}M)f + A^T f : \mathbf{1}^T f \geq n, f \geq \mathbf{0}\}$ i lw y i l n i o i l v l i $\frac{1}{2}C(\overline{f}) + W(\overline{f})$. T Do n' l o QP i DP $\equiv$ x$\{z \cdot n - f^T(\frac{1}{2}M)f : Mf + A \geq \mathbf{1}z, z \geq 0\}$ (., [6], [2, C 6]). Ev y flow f o i l ol ion o DP y in $z =$ in$_{p\in\mathcal{P}}\{\theta_p(f) + a_p\} \equiv \delta^{\min}(f)$. H n , o i l n l o i l . Sin ix M i y i n o i iv i- ni , y Do n' T o , o j iv v l o o i l l ol ion i x ly $\frac{1}{2}C(\overline{f}) + W(\overline{f})$[8].

Con n ly, o v y flow f, $(f, \delta^{\min}(f))$ i i l ol ion o DP n

$$n\,\delta^{\min}(f) - \tfrac{1}{2}C(f) \leq \tfrac{1}{2}C(\overline{f}) + W(\overline{f})$$

L f^Q flow o on in o i o l Q. Sin Q i N ili i , $C(f^Q) \leq n\,\delta^{\min}(f^Q)$ y P o o i ion 3. H n , $n\,\delta^{\min}(f^Q) \leq C(\overline{f}) + 2\,W(\overline{f})$. U in $\lambda^{\max}(Q) \leq \delta^{\min}(f^Q)$ y In . (3), w o in $n\lambda^{\max}(Q) \leq C(\overline{f}) + 2\,W(\overline{f})$.

To on l oo , l f^* n li l flow o on in o i o l P^*, n ly o i l ol ion w o j iv o xi l n y. T n,

$$\begin{aligned} n\lambda^{\max}(Q) &\leq 2\,[\tfrac{1}{2}C(\overline{f}) + W(\overline{f})] \leq 2\,[\tfrac{1}{2}C(f^*) + W(f^*)] \\ &\leq nL(f^*) + 2W(f^*) = nL(P^*) + 2W(P^*) \end{aligned}$$

T on in li y ol f^* i i l ol ion o QP. T i in li y ol v l n y o f^* nno x i xi l n y. Fo l li y, in P^* i i o l , i xi l n y n o l lo oin i wi o o f^*. □

[7] Let $\min\{x^T Mx + c^T x : Ax \geq b, x \geq \mathbf{0}\}$ be the primal quadratic program. The Dorn's dual of this program is $\max\{-y^T My + b^T u : A^T u - 2My \leq c, u \geq \mathbf{0}\}$. Dorn [6] proved strong duality when the matrix M is symmetric and positive semi-definite. Thus, if M is symmetric and positive semi-definite and both the primal and the dual programs are feasible, their optimal solutions have the same objective value.

[8] The optimal dual solution is obtained from $\overline{f}$ by setting $z = \delta^{\min}(\overline{f})$. Since $\overline{f}$ is an optimal solution to the primal program, we can use Karush-Kuhn-Tucker optimality conditions (e.g. [2]) and prove that for any $s-t$ path p with $\overline{f}_p > 0$, $\theta_p(\overline{f}) + a_p = \delta^{\min}(\overline{f})$. Multiplying this equality by $\overline{f}_p$ and summing over all $p \in \mathcal{P}$, we obtain that

$$z \cdot n = \delta^{\min}(\overline{f}) \textstyle\sum_{p\in\mathcal{P}} \overline{f}_p = \sum_{p\in\mathcal{P}} \overline{f}_p(\theta_p(f) + a_p) = C(\overline{f}) + W(\overline{f})$$

Therefore, the dual objective value of $(\overline{f}, \delta^{\min}(\overline{f}))$ is exactly $\frac{1}{2}C(\overline{f}) + W(\overline{f})$.

Corollary 2. *For every strategies profile Q at Nash equilibrium, $\lambda^{\max}(Q) \leq 3\,L(P^*)$.*

Proof. W o v $W(P^*) \leq n\,L(P^*)$ P^* i i o l. T o oll y ollow o T o 2. □

Theorem 3. *The price of anarchy for single-commodity network congestion games with identical users and latencies $d_e(x) = a_e x$ is at most* $6\ \ (\frac{\log m}{\log\log m}+1)$.

Acknowledgements. W wi o n B Moni n o in i ni n n o i ili y o o inin on l on ffi i n o - ion o PNE in i - ll l n wo .

References

1. B. Awerbuch, Y. Azar, and A. Epstein. The Price of Routing Unsplittable Flow. *STOC '05*, pp. 57–66, 2005.
2. M.S. Bazaraa, H.D. Sherali, and C.M. Shetty. *Nonlinear Programming: Theory and Algorithms (2nd edition).* John Wiley and Sons, Inc., 1993.
3. G. Christodoulou and E. Koutsoupias. The Price of Anarchy of Finite Congestion Games. *STOC '05*, pp. 67–73, 2005.
4. J.R. Correa, A.S. Schulz, and N.E. Stier Moses. Computational Complexity, Fairness, and the Price of Anarchy of the Maximum Latency Problem. *IPCO '04*, LNCS 3064, pp. 59–73, 2004.
5. A. Czumaj and B. Vöcking. Tight Bounds for Worst-case Equilibria. *SODA '02*, pp. 413–420, 2002.
6. W.S. Dorn. Duality in Quadratic Programming. *Quarterly of Applied Mathematics*, 18(2):155–162, 1960.
7. R.J. Duffin. Topology of Series-Parallel Networks. *J. of Mathematical Analysis and Applications*, 10:303–318, 1965.
8. A. Fabrikant, C. Papadimitriou, and K. Talwar. The Complexity of Pure Nash Equilibria. *STOC '04*, pp. 604–612, 2004.
9. D. Fotakis, S. Kontogiannis, E. Koutsoupias, M. Mavronicolas, and P. Spirakis. The Structure and Complexity of Nash Equilibria for a Selfish Routing Game. *ICALP '02*, LNCS 2380, pp. 123–134, 2002.
10. D. Fotakis, S. Kontogiannis, and P. Spirakis. Selfish Unsplittable Flows. *ICALP '04*, LNCS 3142, pp. 593–605, 2004.
11. M. Gairing, T. Lücking, M. Mavronicolas, and B. Monien. Computing Nash Equilibria for Scheduling on Restricted Parallel Links. *STOC '04*, pp. 613–622, 2004.
12. M. Gairing, T. Lücking, M. Mavronicolas, B. Monien, and M. Rode. Nash Equilibria in Discrete Routing Games with Convex Latency Functions. *ICALP '04*, LNCS 3142, pp. 645–657, 2004.
13. R. L. Graham. Bounds on Multiprocessing Timing Anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.
14. W. Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *J. of the American Statistical Association*, 58(301):13–30, 1963.
15. E. Koutsoupias, M. Mavronicolas, and P. Spirakis. Approximate Equilibria and Ball Fusion. *Theory of Computing Systems*, 36:683–693, 2003.

16. E. Koutsoupias and C. Papadimitriou. Worst-case Equilibria. *STACS '99*, LNCS 1563, pp. 404–413, 1999.
17. L. Libman and A. Orda. Atomic Resource Sharing in Noncooperative Networks. *Telecommunication Systems*, 17(4):385–409, 2001.
18. T. Lücking, M. Mavronicolas, B. Monien, and M. Rode. A New Model for Selfish Routing. *STACS '04*, LNCS 2996, pp. 547–558, 2004.
19. M. Mavronicolas and P. Spirakis. The Price of Selfish Routing. *STOC '01*, pp. 510–519, 2001.
20. R.W. Rosenthal. A Class of Games Possessing Pure-Strategy Nash Equilibria. *International Journal of Game Theory*, 2:65–67, 1973.
21. T. Roughgarden. The Price of Anarchy is Independent of the Network Topology. *STOC '02*, pp. 428–437, 2002.
22. T. Roughgarden. The Maximum Latency of Selfish Routing. *SODA '04*, pp. 980–981, 2004.
23. T. Roughgarden and É. Tardos. How Bad is Selfish Routing? *J. of the ACM*, 49(2):236–259, 2002.

A Better-Than-Greedy Algorithm for *k*-Set Multicover

To ı i o F ji o[1,⋆] n Hi K i[2,⋆⋆]

[1] Dept. of Inform. & Comp. Sciences, Toyohashi University of Technology, Toyohashi 441-8580, Japan
fujito@nuee.nagoya-u.ac.jp
[2] Graduate School of Information Science, Nagoya University, Furo, Chikusa, Nagoya 464-8603, Japan

Abstract. The set multicover (MC) problem is a natural extension of the set cover problem s.t. each element requires to be covered a prescribed number of times (instead of just once as in set cover). The k-set multicover (k-MC) problem is a variant in which every subset is of size at most k. Due to the multiple coverage requirement, two versions of MC have been studied; the one in which each subset can be chosen only once (*constrained MC*) and the other in which each subset can be chosen any number of times (*unconstrained MC*). For both versions the best approximation algorithm known so far is the classical greedy heuristic, whose performance ratio is $H(k)$, where $H(k) = \sum_{i=1}^{k}(1/i)$. It is no hard, however, to come up with a natural modification of the greedy algorithm such that the resulting performance is never worse, but could also be strictly better. This paper will verify that this is indeed the case by showing that such a modification leads to an improved performance ratio of $H(k) - 1/6$ for both versions of k-MC.

1 Introduction

Giv n U n ily $\mathcal{S}$ o o U, *set cover (SC)* o l o n ll ily $\mathcal{C} \subseteq \mathcal{S}$ ov ll l n o U (i. ., $\bigcup_{S\in\mathcal{C}} S = U$). W n v y in $\mathcal{S}$ i o i o n o ov y on n k, i i ll *k-set cover (k-SC)*. T ov o l , o v n k-SC o $k \geq 3$, i nown o *NP-* [13] w ll *APX-* [15]. T in j o i *set multicover (MC)* o l (o *multicover* o o), n l n li ion o SC o l , w l n $u \in U$ i o i wi n in r_u ll *coverage requirement*, n u o ov (l) r_u i . T k- l i ov (k-MC) o l i v i n in w i v y in $\mathcal{S}$ i o i o k.

T y yi i l y i l o in oxi in SC; i i iv ly o "o - ff iv " n il ll l n o U

⋆ Supported in part by a Grant in Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture of Japan. Also affiliated with Intelligent Sensing System Research Center, Toyohashi Univ. of Tech.

⋆⋆ Currently at Fuji Photo Film Co., Ltd.

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 176–189, 2006.

o ov (y o i). C ll n l n *alive* i i i no y ov y ny o i . T o ff iv n o S i n y v o w i S ov liv l n in i . In o wo , i $A(S)$ no n o liv l n in S n $\mathcal{C}$ ily o l y i , $A(S) = |S \setminus \bigcup_{T \in \mathcal{C}} T|$, n S n *newly* ov only o l n in $S \setminus \bigcup_{T \in \mathcal{C}} T$; n , o ff iv n o S wi o $\mathcal{C}$ i $1/A(S)$. I w own y Jo n on [12] o n io o y l o i i o n y n H oni n $H(n) = \sum_{i=1}^{n}(1/i)$ o SC,[1] o $H(k)$ o k-SC, n Lov o in l y in o *fractional covers* [14]. W il o n io w l own o ol v n o o n l o [2] y x n ion o n ly i vi lin o li y, Sl vı ov x o n i $\ln n - \ln \ln n + \Theta(1)$ [18]. I n o , o ov , y o n i li ly o n ly o i l on o SC in iv oo n l o F i ow SC i no oxi l wi in o o $(1-\epsilon)\ln n$ o ny x $\epsilon > 0$ nl $NP \subset \mathrm{DTIME}(n^{O(\log \log n)})$ [7].

A ov y l o i o k-SC ly i $S \in \mathcal{S}$ wi ini $1/A(S)$, n v y i S i i in o , S i ov o $\mathcal{S}$ n ll l n in S o U. A $A(S)$ i ono oni lly non-in in n o on in lon xi S wi $A(S) > 0$, $A(S)$ v n- lly o no l n 2 o ny $S \in \mathcal{S}$. W n i n , i n o v , y $(U, \mathcal{S})$ i o n k-SC o 2-SC, o o *edge cover* on i , w i i o l o o in ini ov in ll v i in . E ov i olv l in i o l xi y o xi in , n n , oon n in n i o on o ov (2-SC), w y ni n i o y o in n o i l ol ion o i . T i i x ly w l o i o Gol i l. [9] o , n y ov o i ion l o n i ov n on y o n o k-SC, n ly, $H(k) - 1/6$. A i ion lly lyin v io lo l ni o o in y y w o n l in low in o n io [10,11], n o n nown o i $H(k) - 1/2$ [6].

Ex n in ov i n o ni o ly l o 1 (in SC) o i y r_u $(u \in U)$ iv i n lly o wo v ion o o l in o MC; on in w i n o n only on (*constrained MC*) n o in w i n o n ny n o i (*unconstrained MC*). W ff iv n o y o in oxi in SC, n own x n i l o i v ion o MC (n in o o n l ov- in o l *multiset multicover* n *covering integer program*) [5,16], no l o i i y ov n o o o y l o i v n o k-MC.[2]

---

[1] Note: $\ln(n + 1) \leq H(n) \leq 1 + \ln n$.

[2] The only possible exception is the following recent result of Berman, DasGupta and Sontag; they presented a randomized algorithm for a variant of k-MC with uniform coverage requirement $r_e \equiv r$, and its expected performance ratio was shown better than $1 + \ln k$ for large r [1].

I i , , ow v , , o i l o "non- olyno i lly" , n on in k-MC o k-SC, n , o n n ly i o k-SC n lly i ov o o n on in k-MC. I , n o only n i o , ow o i l , SC, i i o yo , oi in olyno i l i in oxi in n on in MC, n i i l ly o l in o . To , o o nowl , on , o , , n , no , ion i nown o on in MC n i i no l in , i , ow o o o o v io l nown o SC. Fo , on w o i , , i ion on n on in MC, n on n , n o , on , on in on only.

R ll , ow , y , i i o k-SC w o i y Gol i l. [9], n i o n i l o , , y k-MC l o i w ll in i il ion. An l n u i n o liv n , n on x i , n o i ov in i i l , n r_u (n l $A(S)$ no , # o liv l n in S o). L wi , *greedy phase* in w i , o in y y o i iv ly i S wi ini $1/A(S)$ n il $A(S) \leq 2$ o ny S. A , i oin , o l i o on in 2-MC,[3] n o n ly, i i olyno i lly olv l 1) on in 2-MC n n o iv l n o *simple b-edge cover (SbEC)* on , , , o l o o in ll $F \subseteq E$, iv n , $G = (V, E)$ n $b \in \mathbb{Z}_+^V$, , # o in F on , o in i n o v i no l , n $b(v)$ o , $v \in V$, n 2) SbEC i nown olv l in i $O(mn \log n)$ [8]. W y , wi , o , *optimal phase* n olv 2-MC *optimally.* All , i in i , on i n l ol ion.

T , o , i vo o , n ly i o , i l o i , . I will on o ll , " l- in " , o (ollowin C v l' o , [2]), n , i i in on wi , n ly i o , - , n- y l o i , o k-SC, w i , ll on ly o in o i l n [9,10,11,6]. A il x o i ion o , l- in n ly i o , n y o k-MC i n in , oo o V i ni[19], w i , w n lly ollow in , o y . In , o , o i l , n LP , n ion o SbEC l y in ol , n w y in , , , l o i o n o - i l l ol ion o i . U in , o i , n o i l l ol ion o SbEC will in in o l ol ion o k-MC y " o n in " in v i l in i . I will own , , o i v ion o y o k-MC o i ly , n , o in y on , yi l in o n io o $H(k) - 1/6$. T i o n i l o i , i i own i , l y o k-SC wi , $k \geq 3$ [9].

2 Preliminaries

2.1 LP Relaxation

T on in MC o l o n in n o $(U, \mathcal{S}, r)$, w , $\mathcal{S} \subseteq 2^U$, $\bigcup_{S \in \mathcal{S}} S = U$ n $r \in \mathbb{Z}_+^U$, n o l y , ollowin i l in o :

[3] Well, not exactly as different sets in k-MC might correspond to a same set in 2-MC, but we don't have to care so much about it.

$$\begin{array}{llll} & \text{. in} & \sum_{S \in \mathcal{S}} x_S & \\ \text{(IP-MC)} & \text{. j o:} & \sum_{S: u \in S} x_S \geq r_u & \forall u \in U \\ & & x_S \in \{0, 1\} & \forall S \in \mathcal{S} \end{array}$$

w, , $x_S = 1$ iff S i , o n in. . ol ion. T, LP , l x ion o (IP-MC), no (P-MC), i , n o . in y , l in , in . ,. l on ,. in . $x_S \in \{0, 1\}$ in (IP-MC) y lin . , on ,. in . $0 \leq x_S \leq 1$ o, . ll $S \in \mathcal{S}$:

$$\begin{array}{llll} & \text{. in} & \sum_{S \in \mathcal{S}} x_S & \\ \text{(P-MC)} & \text{. j o:} & \sum_{S: u \in S} x_S \geq r_u & \forall u \in U \\ & & -x_S \geq -1 & \forall S \in \mathcal{S} \\ & & x_S \geq 0 & \forall S \in \mathcal{S} \end{array}$$

w, , $-x_S \geq -1$. , *not* , n . n on ,. in . nli in , . . o n on- . ,. in MC, . n i . . l i :

$$\begin{array}{llll} & \text{. . x} & \sum_{u \in U} r_u y_u - \sum_{S \in \mathcal{S}} z_S & \\ \text{(D-MC)} & \text{. j o:} & \sum_{u \in S} y_u - z_S \leq 1 & \forall S \in \mathcal{S} \\ & & y_u \geq 0 & \forall u \in U \\ & & z_S \geq 0 & \forall S \in \mathcal{S} \end{array}$$

L OPT . no , o i . l v. l o (P-MC), wi , w, i , , . i o o , . ol ion will o . . , . S o , . w now , . v . . l i ov , $\mathcal{C} \subseteq \mathcal{S}$. n . . l v. , i l . (y, z) . . i yin

1. $|\mathcal{C}| \leq \sum_{u \in U} r_u y_u - \sum_{S \in \mathcal{S}} z_S$, . n
2. $\sum_{u \in S} y_u - z_S \leq \alpha$ o, . , $S \in \mathcal{S}$,

o, . o $\alpha \in \mathbb{R}_+$. T, n, . in $(1/\alpha)(\sum_{u \in S} y_u - z_S) \leq 1$ ($\forall S \in \mathcal{S}$), $((1/\alpha)y, (1/\alpha)z)$ i . . i l o (D-MC), . n i . o j iv v. l i $(1/\alpha)$ $(\sum_{u \in U} r_u y_u - \sum_{S \in \mathcal{S}} z_S)$. T, LP . . li y , o, ... , . , . . n o j iv v. l o (D-MC) i . lw. y. . low , o n o, OPT, i. ., $\sum_{u \in U} r_u y_u - \sum_{S \in \mathcal{S}} z_S \leq \alpha \cdot \text{OPT}$. A OPT in , n low , o n . , o i . o (IP-MC),

Proposition 1. *If a multicover $\mathcal{C}$ and dual variables (y, z) satisfy the two conditions given above, $|\mathcal{C}| \leq \alpha \cdot OPT \leq \alpha \cdot |$optimal multicover$|$.*

(T, i i , . , o , , , n y C, v. . l [2] in . . li , in , . , y SC o n o $H(n)$.)

2.2 Simple b-Edge Cover

Fo n i , $G = (V, E), X \subseteq V$, n $v \in V$ l $E[X] = \{\{v, w\} \in E \mid \{v, w\} \subseteq X\}$, $\delta(X) = \{\{v, w\} \in E \mid |\{v, w\} \cap X| = 1\}$, n $\delta(v) = \delta(\{v\})$. Fo G n $b \in \mathbb{Z}_+^V$, $x \in \mathbb{Z}_+^E$ i ll *b-edge cover* o G i $x(\delta(v)) \geq b_v$ o ll $v \in V$, n i i ll *simple b-edge cover* o G i x i ion lly i $x_e \in \{0, 1\}, \forall e \in E$. T , ny i l b- ov n i n i wi o , n *simple b-edge cover problem* i o o n o ini i . T o l i nown no only o olyno i lly olv l , l o o v ollowin LP i ion.

Proposition 2 ([4,17]). *The simple b-edge cover problem can be formulated by the following LP:*

$$
\begin{array}{lll}
 & \text{in} \sum_{e \in E} x_e & \\
\textit{subject to:} & 0 \leq x_e \leq 1 & \forall e \in E \\
\textit{(P-SbEC)} & x(\delta(v)) \geq b(v) & \forall v \in V \\
 & x(E[X]) + x(\delta(X) \setminus F) \geq \left\lceil \dfrac{b(X) - |F|}{2} \right\rceil & \forall X \subseteq V, F \subseteq \delta(X)
\end{array}
$$

L $\Psi = \{(X, F) \mid X \subseteq V, F \subseteq \delta(X), E[X] \cup (\delta(X) \setminus F) \neq \emptyset\}$, n $\bar{\delta}_F(X) = E[X] \cup (\delta(X) \setminus F)$. T l LP o (P-S$b$EC) i iv n y:

$$
\begin{array}{lll}
 & \text{x} \sum_{v \in V} b(v) y_v - \sum_{e \in E} z_e + \sum_{(X,F) \in \Psi} \left\lceil \dfrac{b(X) - |F|}{2} \right\rceil \cdot w_{(X,F)} & \\
\text{j o:} & y_v \geq 0 & \forall v \in V \\
\text{(D-SbEC)} & z_e \geq 0 & \forall e \in E \\
 & w_{(X,F)} \geq 0 & \forall (X, F) \in \Psi \\
 & y_u + y_v - z_e + \sum_{(X,F) \in \Psi : e \in \bar{\delta}_F(X)} w_{(X,F)} \leq 1 & \forall e = \{u, v\} \in E
\end{array}
$$

To voi in o in in l on (l-loo) w n 2-MC on $(U, \mathcal{S}, r)$ i o SbEC on $G = (V, E)$, l v_0 n w v x (l n) no xi n in U, $V = U \cup \{v_0\}$, n $E = \{S \in \mathcal{S} \mid |S| = 2\} \cup \{\{u, v_0\} \mid \{u\} \in \mathcal{S}\}$. W l o l $b_v = r_v$ o $v \in U$ n $b_{v_0} = 0$.

3 Analysis

3.1 Structural Properties of the Optimal Dual for Simple b-Edge Cover

Fo l n ly i w n ollowin l on nin l o i o n o i l ol ion o (D-SbEC), LP l o i l b- ov .

Lemma 3. *If (D-SbEC) has an optimal solution (y, z, w), there exists one satisfying all the following properties:*

$$w_{(X,F)} = 0 \qquad \forall (X, F) \in \Psi \text{ with } |X| = 1 \tag{1}$$

$$y_v \in \{0, 1\} \qquad \forall v \in V \tag{2}$$

$$w_{(X,F)} \in \{0, 1\} \qquad \forall (X, F) \in \Psi \tag{3}$$

If $w_{(X_1,F_1)} > 0$ and $w_{(X_2,F_2)} > 0$ for different (X_1, F_1) and (X_2, F_2),

$$X_1 \cap X_2 = \emptyset \tag{4}$$

$$\text{and} \quad F_1 \cap F_2 = \emptyset \tag{5}$$

If $w_{(X,F)} > 0$,

$$y_v = 0 \qquad \forall v \in X \tag{6}$$

$$\text{and} \quad z_e = 0 \qquad \forall e \in F \tag{7}$$

By i l onin i n l o w.l.o. $y_{v_0} = 0$ n $w_{(X,F)} = 0$ i $v_0 \in X$. T oo o i l will iv n in S ion 3.4.

3.2 Rounding the Optimal Dual for Simple b-Edge Cover

R ll in P o o i ion 1 li y o ol ion o y l o i i l iv o v l o (in i l) ol ion o (D-MC). I i n o on l o n i o n o i l i l b- ov o in o i l y v l o i l in (D-SbEC) w ll. I n n, ow v , w n (D-MC) n (D-SbEC) o , y_v n z_e in l iv ly v i o n in o (i , y_u n z_S o $u = v$ n $S = e$), $w_{(X,F)}$ non . T i w n o o ow n n o i l SbEC ol ion (y, z, w) in o only y n z, n w o o y " o n in " ll w-v i l o o; i , no o low v l o ol ion o in , y n z will i o o n o i .

Fo n o i l ol ion (y, z, w) o (D-SbEC) i yin ll on i ion o L 3, l $\Psi_1 = \{(X, F) \in \Psi \mid w_{(X,F)} = 1\}$, $\bar{X} = \bigcup_{(X,F)\in\Psi_1} X$, n $\bar{F} = \bigcup_{(X,F)\in\Psi_1} F$. Fo $(X, F) \in \Psi_1$ w v

$$w_{(X,F)} = 1, \quad y_v = 0 \ (\forall v \in X) \ \text{ y } (6), \quad z_e = 0 \ (\forall e \in F) \ \text{ y } (7),$$

n w y' n z' ollow :

$$y_v = \frac{\lceil (b(X) - |F|)/2 \rceil + |F|/3}{b(X)} \qquad \forall v \in X$$

$$z_e = 1/3 \qquad \forall e \in F$$

w n v $w_{(X,F)}$ i o n o 0. No : D o (4) n (5), o $v \in \bar{X}$ n $e \in \bar{F}$ iv n w v l ov only on y ni ly o on in $(X, F) \in \Psi_1$.

Lemma 4. *The rounding of w given above causes no change in the objective value of (D-SbEC).*

Proof. R in $w_{(X,F)}$ o 0 o o j iv v l y $\lceil\frac{b(X)-|F|}{2}\rceil$ w in n w v l o y_v' n z_e' i i y

$$\sum_{v\in X}\frac{\lceil(b(X)-|F|)/2\rceil+|F|/3}{b(X)}\cdot b(v)-\sum_{e\in F}\frac{1}{3}=(\lceil\frac{b(X)-|F|}{2}\rceil+\frac{|F|}{3})-\frac{|F|}{3}$$

o $(X,F)\in\Psi_1$. □

T v l o o n ol ion (y,z) i no ll n o o i in l (o i l) ol ion (y,z,w), n i i w y i n in o n ly i in l o l ol ion. I will own in n x wo l o n ol ion i no o o l i ili y i .

Lemma 5. *After rounding w as above, we have*

$$y_v\le\begin{cases}1 & \text{if } v\notin\bar{X}\\ \frac{2}{3} & \text{if } v\in\bar{X}\end{cases}$$

for each $v\in V$.

Proof. Fo $v\notin\bar{X}$ y_v o no n i v l , n n , $y_v\in\{0,1\}$ (y (2)). Fo ny $v\in V$, $b(v)>0$ x o v_0. A $v_0\notin\bar{X}$, i $v\in X$ wi $(X,F)\in\Psi_1$, $b(v)>0$ n $|X|\ge 2$ (y (1)), w i i li $b(X)\ge 2$. I $b(X)=2$,

$$y_v=\frac{\lceil(2-|F|)/2\rceil+|F|/3}{2}=\frac{\lceil 1-|F|/2\rceil+|F|/3}{2}\le\frac{\lceil 1-1/2\rceil+1/3}{2}=\frac{2}{3}$$

n i $b(X)\ge 3$,

$$\begin{aligned}y_v&=\frac{\lceil(b(X)-|F|)/2\rceil+|F|/3}{b(X)}\\&\le\frac{(b(X)-|F|+1)/2+|F|/3}{b(X)}\\&=\frac{(b(X)+1)/2-|F|/6}{b(X)}\\&\le\frac{(b(X)+1)/2}{b(X)}=\frac{1}{2}+\frac{1}{2b(X)}\le\frac{2}{3}\end{aligned}$$

□

Lemma 6. *After rounding w as above, we have $y_u+y_v-z_e\le\frac{4}{3}$ for each $e=\{u,v\}\in E$.*

Proof. R ll , o i l (y,z,w), w

$$y_u+y_v-z_e+\sum_{(X,F)\in\Psi:e\in\bar{\delta}_F(X)}w_{(X,F)}\le 1 \tag{8}$$

o o n in o $e=\{u,v\}\in E$.

Case $u \notin \bar{X}, v \notin \bar{X}$: Sin $w_{(X,F)} = 0, \forall (X,F) \in \Psi$ wi $u \in X$ o $v \in X$ o o n in , $\sum_{(X,F):e\in\bar{\delta}_F(X)} w_{(X,F)} = 0$, n (8) o $y_u + y_v - z_e \leq 1$; i ill ol v n o n in y_u, y_v no z_e n i v l .

Case $u \in \bar{X}, v \notin \bar{X}, e \in \bar{F}$: By L 5, $y_u \leq 2/3$ n $y_v \leq 1$. Sin $z_e = 1/3$, $y_u + y_v - z_e \leq 2/3 + 1 - 1/3 = 4/3$.

Case $u \in \bar{X}, v \notin \bar{X}, e \notin \bar{F}$: B o o n in , $y_u = 0$, n xi (X, F) wi $w_{(X,F)} = 1$ $u \in X$ n $e \notin F$. T n, e i in $\bar{\delta}_F(X)$, n n , $y_v - z_e \leq 0$ y (8) o o n in . A o n in , w il y_v o no n i v l , $y_u \leq 2/3$ y L 5 in $u \in \bar{X}$, n w v $y_u + y_v - z_e \leq 2/3$.

Case $u \in \bar{X}, v \in \bar{X}$: Sin o y_u n y_v i $\leq 2/3$, $y_u + y_v - z_e \leq 4/3$. □

3.3 Performance Ratio

W ' y o v l o ol ion (y, z) o (D-MC), n o i in i o i on o in y o n in no i l l ol ion o SbEC will no y $(\tilde{y}, \tilde{z})$.

S o l n $u \in U$ i ov l_u i (y i) in y , w 0 $\leq l_u \leq r_u$. W n S i i , i o i i i on liv l n i ov , n i S ov u o j i , w i (u, j) ollow :

$$\text{i}\ (u,j) = \begin{cases} \frac{1}{A(S)} & \text{i } S \text{ i i in y (i. ., } 1 \leq j \leq l_u) \\ \tilde{y}_u & \text{i } S \text{ i i in o i l (i. ., } l_u < j \leq r_u) \end{cases}$$

F o w y l o i wo , i i l i $(u, 1) \leq$ i $(u, 2) \leq \ldots$ i $(u, l_u) \leq 1/3$ o $u \in U$, n # o i in y oin i wi $\sum_{u\in U} \sum_{j=1}^{l_u}$ i (u, j). L $\mathcal{S}_G = \{S$ i in y $\}$, n now v l o l v i l y n z ollow :

$$y_u = \max_{1\leq j\leq r_u} \{\text{ i } (u,j)\} = \max\{\text{ i } (u,l_u), \tilde{y}_u\}$$

$$z_S = \begin{cases} \sum_{u:\text{covered by } S} (y_u - \text{ i } (u, j_u)) & \text{i } S \in \mathcal{S}_G \\ \tilde{z}_e & \text{i } S \text{ n } e \text{ in o i l} \\ 0 & \text{o wi} \end{cases}$$

w j_u i o y o u i ov y S in y .

T ollowin wo l ow (y, z) i wo on i ion o in P o o i ion 1.

Lemma 7. *For* (y, z) *defined as above, # of sets picked by the algorithm* $\leq \sum_{u\in U} r_u y_u - \sum_{S\in\mathcal{S}} z_S$.

Proof. T n o i in y $= \sum_{u \in U} \sum_{j=1}^{l_u}$ i (u,j), w

$$\# \text{ o i in o i l } = \sum_{u \in V} b(u)\tilde{y}_u - \sum_{e \in E} z_e$$
$$= \sum_{u \in V} \sum_{j=l_u+1}^{r_u} \text{ i } (u,j) - \sum_{e \in E} z_e$$

y L 4. I ollow

$$\# \text{ o i in o l } = \sum_{u \in U} \sum_{j=1}^{r_u} \text{ i } (u,j) - \sum_{e \in E} z_e$$
$$= \sum_{u \in U} r_u y_u - \sum_{u \in U} \sum_{j=1}^{r_u} (y_u - \text{ i } (u,j)) - \sum_{e \in E} z_e$$
$$\leq \sum_{u \in U} r_u y_u - \sum_{e \in E} z_e - \sum_{S \in \mathcal{S}_G} \sum_{u \in U: \text{covered by } S} (y_u - \text{ i } (u, j_u))$$
$$= \sum_{u \in U} r_u y_u - \sum_{S \in \mathcal{S}} z_S \qquad \square$$

Lemma 8. *For (y,z) defined as above, $\sum_{u \in S} y_u - z_S \leq H(k) - \frac{1}{6}, \forall S \in \mathcal{S}$.*

Proof. A $S = \{u_1, u_2, \ldots, u_k\}$ n S i " lly" ov in i o (o $u_j, j = 1, 2, \ldots, k$). I l o y o u_i i ov in y S on in $k - i + 1$ liv l n i oin , $y_{u_i} =$ i $(u_i, r_{u_i}) \leq \frac{1}{k-i+1}$ in n l (o $i \leq k-2$), l o $y_{u_i} \leq 1/3$ v n o $i \geq k-1$ in ny i in y i o i l 3. I l o y o u i ov in o i l , $y_u \leq$ x$\{1/3, \tilde{y}_u\} \leq 1$ y L 5, n o ny $e = \{u, v\}$ inin no lly ov , $\tilde{y}_u + \tilde{y}_v - z_e \leq 4/3$ y L 6.

Case S i i in y : S o k' l n l y lly ov w n S i i (n n , k' l n), w i $0 \leq k' \leq k-3$. T n,

$$\sum_{i=1}^{k} y_{u_i} - z_S = \sum_{i=1}^{k} y_{u_i} - \sum_{i=k'+1}^{k} (y_{u_i} - \text{ i } (u_i, j_{u_i}))$$
$$= \sum_{i=1}^{k'} y_{u_i} + \sum_{i=k'+1}^{k} \text{ i } (u_i, j_{u_i})$$

$y_{u_i} =$ i $(u_i, r_{u_i}) \leq \frac{1}{k-i+1}$ o $i \in \{1, \ldots, k'\}$, n $\sum_{i=k'+1}^{k}$ i $(u_i, j_{u_i}) = 1$,

$$\leq (\sum_{i=1}^{k'} \frac{1}{k-i+1}) + 1$$
$$\leq (\frac{1}{k} + \frac{1}{k-1} + \ldots \frac{1}{4}) + 1$$
$$= H(k) - \frac{5}{6}$$

Case S i *not* i in y :

Subcase $A(S) = 0$ n o y : Sin S i no i ll y l o i , $z_S = 0$. Sin $y_u =$ i (u, l_u), $\forall u \in S$, $y_{u_i} \leq 1/(k-i+1)$ w ll $y_{u_i} \leq 1/3$. I ollow

$$\sum_{i=1}^{k} y_{u_i} - z_S \leq (\frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{3}) + \frac{1}{3} + \frac{1}{3} = H(k) - \frac{5}{6}$$

Subcase $A(S) = 1$ n o y : Sin u_k lly ov only in o i l , $y_{u_k} \leq 1$ y L 5, n n ,

$$\sum_{i=1}^{k} y_{u_i} - z_S \leq (\frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{3}) + \frac{1}{3} + y_{u_k} - z_e \quad \text{w} \quad e = \{u_k, v_0\}$$
$$\leq (\frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{3}) + \frac{1}{3} + 1$$
$$= H(k) - \frac{1}{6}$$

Subcase $A(S) = 2$ n o y :

$$\sum_{i=1}^{k} y_{u_i} - z_S = \sum_{i=1}^{k-2} \text{i} \ (u_i, l_{u_i}) + y_{u_{k-1}} + y_{u_k} - z_e \quad \text{w} \quad e = \{u_{k-1}, u_k\}$$
$$\leq (\frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{3}) + \frac{4}{3} \qquad \text{y L 6}$$
$$= H(k) - \frac{1}{6}$$

□

Now o o on i ion in P o o i ion 1 own i l , y L 7 n 8, wi $\alpha = H(k) - 1/6$,

Theorem 9. *The performance ratio of the modified greedy algorithm is at most* $H(k) - 1/6$.

Sin o n ly i i o o on LP li y, w i ion lly v

Corollary 10. *The integrality gap of (P-MC) is bounded above by* $H(k) - 1/6$ *when* $|S| \leq k$ $(\forall S \in \mathcal{S})$.

3.4 Proof of Lemma 3

D o o l l in li y o lin y in (P-SbEC) [3], i o i xi o (D-SbEC), i n i v y n in ol ion; w y oo wi n o i l l ol ion (y, z, w).

L $Y_e = y_u + y_v$ n $W_e = \sum_{(X,F)\in\Psi:e\in\bar{\delta}_F(X)}$ o $e = \{u, v\} \in E$. Al o l Δ_e n Δ_{obj} no iv ly, o n in in v l o $Y_e - z_e + W_e$ n in o j iv n ion v l in (D-SbEC). To ow in w ollow i ili y o n w ol ion, w v i y ll v i l in o o nonn iv v l n $\Delta_e \leq 0$ $(\forall e \in E)$, w w v i y $\Delta_{\text{obj}} \geq 0$ o ow o i li y o n w ol ion. To i in i v i l wi n w v l o v i l wi ol v l on wi n wly i n v l will no wi .

W 'll ow n o i l ol ion vin o i n o in y lyin n o o ion o ny o i l ol ion i in l.

(1) Fo ny $(\{u\}, F) \in \Psi$ wi $w_{(\{u\},F)} > 0$, $w'_{(\{u\},F)} = 0$, in y_u n z_e $(\forall e \in F)$ y $w_{(\{u\},F)}$. **Feasibility**: Y_e o (y $w_{(\{u\},F)}$) iff $e \in \delta(u)$. A z_e o y $w_{(\{u\},F)}$ i $e \in F$ w il W_e o own y $w_{(\{u\},F)}$ i $e \in \delta(u) \setminus F$, $\Delta_e \leq 0$ o ll $e \in \delta(u)$. **Optimality**: $\Delta_{\text{obj}} = b(u)a - |F|a - \lceil\frac{b(u)-|F|}{2}\rceil \cdot a = a(b(u) - |F| - \lceil\frac{b(u)-|F|}{2}\rceil) \geq 0$.

(2) Fo ny $v \in V$ wi $y_v > 1$, $y'_v = 1$, n z_e y $y_v - 1$ $(\forall e \in \delta(v))$. **Feasibility**: A $z_e \geq Y_e + W_e - 1 \geq y_v - 1$ o $e \in \delta(v)$, $z'_e \geq 0$. A $y'_v - z'_e = 1 - (z_e - (y_v - 1)) = y_v - z_e$ o $e \in \delta(v)$, v l o $Y_e - z_e$ i n n (n n , $\Delta_e = 0$) $e \in \delta(v)$. **Optimality**: $\Delta_{\text{obj}} = b(v)(1 - y_v) + |\delta(v)|(y_v - 1) = (y_v - 1)(|\delta(v)| - b(v)) \geq 0$.

(3) Fo ny $(X, F) \in \Psi$ wi $w_{(X,F)} > 1$, $w'_{(X,F)} = 1$ n z_e y $w_{(X,F)} - 1$ $(\forall e \in \bar{\delta}_F(X))$. **Feasibility**: $z'_e \geq 0$ o $e \in \bar{\delta}_F(X)$ $z_e \geq Y_e + W_e - 1 \geq w_{(X,F)} - 1$. Fo $e \in \bar{\delta}_F(X)$, z_e w ll W_e o own y $w_{(X,F)} - 1$, n n , $\Delta_e = 0$. **Optimality**: $\Delta_{\text{obj}} = (w_{(X,F)} - 1)|\bar{\delta}_F(X)| - \lceil\frac{b(X)-|F|}{2}\rceil(w_{(X,F)} - 1) = (w_{(X,F)} - 1)(|E[X] \cup (\delta(X) \setminus F)| - \lceil\frac{b(X)-|F|}{2}\rceil) \geq 0$.

A i oin w v n o i l ol ion vin P o i (1) o (3), n i will o i o i y o inin o i i in o iv n low. In oin o, ow v ol ion n y lo o o (1) o (3); i i o n oo , w n v i n o on in o ion iv n ov will li o ol ion n i o y will ov . Any o i ion y o ion ov no ff on o i o i x o ollowin n io. T o ion in (4) n (6) y l o viol ion o P o y (1), n i w y o x i i y n x l o viol ion o (6). W n w wo in on P o y (4), w n v P o y (1) i lo , w x i n o n w viol ion o (6) l on. W n w wo in on P o y (6) n i P o y (1) i lo , w will x i in, i will no l o no lo o P o y (6); P o y (4) i l y n o , v n i P o y (1) o n i l o no n o (6), y $v \in V$ wi o i iv $w_{(\{v\},F_1)}$, i no o $(X, F_2) \in \Psi$ $w_{(X,F_2)} > 0$ n $v \in X$. T o, v n i y_v in o x P o y 1 v, i nno in in on P o y (6).

(4) S o xi wo iff n (X_1, F_1) n (X_2, F_2) in Ψ $X_1 \cap X_2 \neq \emptyset$ n $w_{(X_1,F_1)} = w_{(X_2,F_2)} = 1$.

Claim. W y lw y oo (X_1, F_1) n (X_2, F_2) $(\delta(X_1) \cap \delta(X_2)) \cup (\delta(X_1) \cap E[X_2]) \cup (E[X_1] \cap \delta(X_2)) \subseteq F_1 \cup F_2$.

Proof. S o xi $e' \in (\delta(X_1) \cap (E[X_2] \cup \delta(X_2))) \setminus (F_1 \cup F_2)$ (in w i $\exists e' \in (\delta(X_2) \cap (E[X_1] \cup \delta(X_1))) \setminus (F_1 \cup F_2)$ i i il). W n ly ollowin o ion (n lon n xi); $w'_{(X_1,F_1)} = 0$, n $z_{e'}$ y 1, n in n $w_{(X_1,F_1 \cup e')}$ y 1. T l in ol ion n n i l o ollowin on : Sin $W_{e'} \geq 2$, $z_{e'} \geq Y_{e'} + W_{e'} - 1 \geq 1$, n n , $z'_{e'} \geq 0$. A $e = e'$, o z_e n W_e o own y 1, w n i n $e \neq e'$. T , $\Delta_e \leq 0$ in i . T o i li y l o n ily v i . □

A n o (X_1, F_1) n (X_2, F_2) i y l i o y, w i i li $\delta(X_3)$ i i ion o ollowin : $F'_1 = \delta(X_3) \cap (F_1 \setminus F_2)$, $F'_2 = \delta(X_3) \cap (F_2 \setminus F_1)$, n $F'_3 = \delta(X_3) \cap (F_1 \cap F_2)$. In n l i $2|E[X]| + |\delta(X)| \geq b(X)$ o ny $X \subseteq V$, i i l ol ion xi . W ivi in o wo :

Case $b(X_3) \leq 2|E[X_3]| + |\delta(X_3)| - 1$. L in $X_4 = X_1 \cup X_2$ n $F_4 = \delta(X_4) \cap ((F_1 \setminus F'_1) \cup (F_2 \setminus F'_2))$, $w'_{(X_1,F_1)} = w'_{(X_2,F_2)} = 0, w'_{(X_4,F_4)} = w_{(X_4,F_4)} + 1, z'_e = z_e - 1$ ($\forall e \in E[X_3]$), n $z'_e = z_e + 1$ ($\forall e \in F''_3$), w $F''_3 = (F_1 \cap F_2) \setminus \delta(X_3)$.

Feasibility: Sin $W_e \geq 2$ o ny $e \in E[X_3]$, $z_e \geq Y_e + W_e - 1 \geq 1$, n n , $z'_e \geq 0$. A o ff o in $w_{(X_4,F_4)}$-v l , no i $w_{(X_4,F_4)}$ o in W_e iff $e \in E[X_4] \cup (\delta(X_4) \setminus F_4)$.

Case $e \in E[X_4]$. I $e \in E[X_3]$, v l o o z_e n W_e o own y 1, w , i $e \in F''_3$, v l o o z_e n W_e o y 1. El (i. ., $e \in (E[X_4] \setminus E[X_3]) \setminus F''_3$) no n in v l o W_e (no z_e) e i in i $E[X_1] \cup (\delta(X_1) \setminus F_1)$ o $E[X_2] \cup (\delta(X_2) \setminus F_2)$, no in o (o ion o on (X_1, F_1) n (X_2, F_2)).

Case $e \in \delta(X_4) \setminus F_4$. By on iv n j ov , no n in W_e.

Optimality: T o j iv v l i own y $\lceil \frac{b(X_1)-|F_1|}{2} \rceil + \lceil \frac{b(X_2)-|F_2|}{2} \rceil + |F''_3|$ n y $\lceil \frac{b(X_4)-|F_4|}{2} \rceil + |E[X_3]|$, o w i i ollow $\Delta_{\text{obj}} \geq 0$.

Case $b(X_3) = 2|E[X_3]| + |\delta(X_3)|$. L in $X_5 = X_1 \setminus X_2, X_6 = X_2 \setminus X_1, F_5 = \delta(X_5) \cap (F_1 \cup F'_2), F_6 = \delta(X_6) \cap (F_2 \cup F'_1)$, $w'_{(X_1,F_1)} = w'_{(X_2,F_2)} = 0, w'_{(X_5,F_5)} = w_{(X_5,F_5)} + 1, w'_{(X_6,F_6)} = w_{(X_6,F_6)} + 1, y'_v = y_v + 1$ ($\forall v \in X_3$), $z'_e = z_e + 1$ ($\forall e \in F'_3$).

Feasibility:

Case $e \notin E[X_3] \cup \delta(X_3)$. N i Y_e no W_e n i v l .

Case $e \in E[X_3] \cup \delta(X_3)$. T v l o Y_e o . Mo i lly, Y_e y 2 W_e own y 2 i $e \in E[X_3]$. In w n $e \in \delta(X_3)$ Y_e i y 1, i z_e i l o y 1 (i $e \in F_3'$) o W_e i l o own y 1 (i $e \in F_1' \cup F_2'$).

Optimality: T o j iv v l i own y $\lceil \frac{b(X_1)-|F_1|}{2} \rceil + \lceil \frac{b(X_2)-|F_2|}{2} \rceil + |F_3'|$ n y $\lceil \frac{b(X_5)-|F_5|}{2} \rceil + \lceil \frac{b(X_6)-|F_6|}{2} \rceil + b(X_3)$, o w i i ollow $\Delta_{\rm obj} \geq 0$.

(5) S o xi $(X, F) \in \Psi$ n $v \in X$ $w_{(X,F)} = 1$ n $y_v = 1$. L in $X' = X \setminus v, \delta_F(v) = \delta(v) \cap F, \delta_{\bar{F}}(v) = \delta(v) \setminus F$, n $F' = \delta(X') \cap (F \cup \delta(v))$, $w'_{(X,F)} = 0, w'_{(X',F')} = w'_{(X',F')} + 1$, n $z'_e = z_e - 1$ $(\forall e \in \delta_{\bar{F}}(v))$.
Feasibility: In n l $z_e \geq Y_e + W_e - 1$, n $Y_e \geq 1$ n $W_e \geq 1$ o $e \in \delta_{\bar{F}}(v)$, w i i li $z_e \geq 1$, n n , $z_e \geq 0$. I z_e o (y 1), $e \in \delta_{\bar{F}}(v)$, n, W_e l o o y 1 e. On o n , W_e in iff e i in $E[X'] \cup (\delta(X') \setminus F')$ no in $E[X] \cup (\delta(X) \setminus F)$, i i i o i l in $E[X'] \cup (\delta(X') \setminus F') \subseteq E[X] \cup (\delta(X) \setminus F)$.
Optimality: $\Delta_{\rm obj} = |\delta_{\bar{F}}(v)| - \lceil \frac{b(X)-|F|}{2} \rceil + \lceil \frac{b(X')-|F'|}{2} \rceil \geq 0$

$$
\begin{aligned}
\lceil \frac{b(X')-|F'|}{2} \rceil &\geq \lceil \frac{1}{2}\{b(X) - b(v) - (|F| - |\delta_F(v)| + |\delta_{\bar{F}}(v)|)\} \rceil \\
&\geq \lceil \frac{1}{2}\{b(X) - |\delta(v)| - (|F| - |\delta_F(v)| + |\delta_{\bar{F}}(v)|)\} \rceil \\
&= \lceil \frac{b(X)-|F|}{2} - |\delta_{\bar{F}}(v)| \rceil \\
&= \lceil \frac{b(X)-|F|}{2} \rceil - |\delta_{\bar{F}}(v)|.
\end{aligned}
$$

(6) S o xi wo iff n (X_1, F_1) n (X_2, F_2) in Ψ $F_1 \cap F_2 \neq \emptyset$ n $w_{(X_1,F_1)} = w_{(X_2,F_2)} = 1$. Fo $e' \in F_1 \cap F_2$, $w'_{(X_1,F_1)} = 0$ n $w'_{(X_1,F_1 \setminus e')} = w_{(X_1,F_1 \setminus e')} + 1$, n n o on $(X_1, F_1 \setminus e')$ n (X_2, F_2) lon y o in n $|F_1 \setminus e'| > 1$ (in w n $|F_1 \setminus e'| = 1$, o ion o ov in P o y (3) will li).
Feasibility: A $e \in E[X_1] \cup (\delta(X_1) \setminus F_1)$, $w_{(X_1,F_1)}$ i own y 1 n $w_{(X_1,F_1 \setminus e')}$ y 1, W_e o no n i v l . On o n , $W_{e'}$ o y 1. A in P o y (4) i i , $w_{(X_1,F_1)} = w_{(X_2,F_2)} = 1$ n $e' \in F_1 \cap F_2$, $X_1 \cap X_2 = \emptyset$ n nno xi (X, F) wi $w_{(X,F)} = 1$ $e' \in E[X] \cup (\delta(X) \setminus F)$. T i in i $W_{e'}$ w l o 0 o i o . Mo ov , $y_u = y_v = 0$ in P o y (4) i i , w $e' = \{u, v\}$. I ollow $Y_{e'} - z_{e'} + W_{e'} \leq 0$ o n in w-v l , n n , $Y_{e'} - z_{e'} + W_{e'} \leq 1$ v n n in .

Optimality: Cl ly $\Delta_{\rm obj} \geq 0$.

(7) W l i P o y (7) i n l on n o vin n o i l o-l ion i yin ll vio o i i . Con i ny $(X', F') \in \Psi$ n $e' = \{u, v\} \in F$ wi $v \in X'$ $w_{(X',F')} = 1$. Sin $w_{(X,F)} = 0$ o ny (X, F) wi $v \in X$ i $(X, F) \neq (X', F')$, $\sum_{(X,F): v \in X, e' \in \bar{\delta}_F(X)} w_{(X,F)} = 0$

(y P o y (4)). By P o y (6) $y_v = 0$. I $y_u = 1$, $\sum_{(X,F):u\in X} w_{(X,F)} = 0$ (y P o y (6)) w $\sum_{(X,F):u\in X} w_{(X,F)} \leq 1$ v n i $y_u = 0$ (y P o - y (4)), w i i li $y_u + \sum_{(X,F):u\in X} w_{(X,F)} \leq 1$. T o , w v $Y_{e'} + W_{e'} \leq 1$, n n , $Y_{e'} - z_{e'} + W_{e'} \leq 1$ o ny nonn iv $z_{e'}$. I ol ion i o i l, $z_{e'}$ $= 0$.

References

1. Berman, P., DasGupta, B., Sontag, E., Randomized approximation algorithms for set multicover problems with applications to reverse engineering of protein and gene networks, 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), LNCS 3122 (2004) 39-50.
2. Chvátal, V., A greedy heuristic for the set-covering problem, Math. Oper. Res. 4(3) (1979) 233–235.
3. Cook, W.J., On some aspects of totally dual integral systems, PH.D. Thesis, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario (1983)
4. Cook, W., Pulleyblank, W.R., Linear systems for constrained matching problems, Math. Oper. Res. 12 (1987) 97–120.
5. Dobson, G., Worst-case analysis of greedy heuristics for integer programming with nonnegative data, Math. Oper. Res. 7(4) (1982) 515–531.
6. Duh, R., Fürer, M., Approximation of k-set cover by semi-local optimization, in: Proc. 29th Annual ACM Symp. Theory of Computing (1997) 256–264.
7. Feige, U., A threshold of $\ln n$ for approximating set cover, J. ACM 45(4) (1998) 634–652.
8. Gabow, H.N., An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems, in: Proc. 15th ACM Symp. Theory of Computing, (1983) 448–456.
9. Goldschmidt, O., Hochbaum, D.S., Yu, G., A modified greedy heuristic for the Set Covering problem with improved worst case bound, Inform. Process. Lett. 48 (1993) 305–310.
10. Halldórsson, M.M., Approximating discrete collections via local improvements, in: Proc. 6th Annual ACM-SIAM Symp. Discrete Algorithms (1995) 160–169.
11. Halldórsson, M.M., Approximating k-set cover and complementary graph coloring, in: Proc. IPCO V (1996) 118–131.
12. Johnson, D.S., Approximation algorithms for combinatorial problems, J. Comput. System Sci. 9 (1974) 256–278.
13. Karp, R.M., Reducibility among combinatorial problems, in: Miller, R.E., Thatcher, J.W. (eds.): Complexity of Computer Computations (Plenum Press, New York, 1972) 85–103.
14. Lovász, L., On the ratio of optimal integral and fractional covers, Discrete Math. 13 (1975) 383–390.
15. Papadimitriou, C., Yannakakis, M., Optimization, approximation and complexity classes, J. Comput. System Sci. 43 (1991) 425–440.
16. Rajagopalan, S., Vazirani, V.V., Primal-dual RNC approximation algorithms for set cover and covering integer programs, SIAM J. Comput. 28 (1999) 526–541.
17. Schrijver, A., Combinatorial Optimization (vol.A), Springer, Berlin (2003).
18. Slavík, P., A tight analysis of the greedy algorithm for set cover, J. Algorithms 25(2) (1997) 237-254.
19. V. Vazirani, Approximation Algorithms, Springer, Berlin (2001).

Deterministic Online Optical Call Admission Revisited*

Eli . , G. . . n [1] . n Sv n O. K . . [2]

[1] Technische Universität Graz, Institut für Mathematik B,
Steyrergasse 30, 8010 Graz, Austria
gassner@opt.math.tu-graz.ac.at

[2] University of Kaiserslautern, Department of Mathematics, P.O. Box 3049,
Paul-Ehrlich-Str. 14, 67653 Kaiserslautern, Germany
krumke@mathematik.uni-kl.de

Abstract. In the problem of *Online Call Admission in Optical Networks*, briefly called OCA, we are given a graph $G = (V, E)$ together with a set of wavelengths W ($\chi := |W|$) and a finite sequence $\sigma = r_1, r_2, \ldots$ of calls which arrive in an online fashion. Each call r_j specifies a pair of nodes to be connected. A lightpath is a path in G together with a wavelength $\lambda \in W$.

Upon arrival of a call, an online algorithm must decide immediately and irrevocably whether to accept or to reject the call without any knowledge of calls which appear later in the sequence. If the call is accepted, the algorithm must provide a lightpath to connect the specified nodes. The essential restriction is the wavelength conflict constraint: each wavelength is available only once per edge, which implies that two lightpaths sharing an edge must have different wavelengths. The objective in OCA is to maximize the overall profit, that is, the number of accepted calls.

A result by Awerbuch et al. states that a c-competitive algorithm for OCA with one wavelength, i.e., $\chi := |W| = 1$, implies a $(c+1)$-competitive algorithm for general numbers of wavelengths. However, for instance, for the line with $n + 1$ nodes, a lower bound of n for the competitive ratio of deterministic algorithms for $\chi = 1$ makes this result void in many cases. We provide a deterministic competitive algorithm for $\chi > 1$ wavelengths which achieves a competitive ratio of $\chi(\sqrt[\chi]{n} + 2)$ on the line with $n + 1$ nodes. As long as $\chi > 1$ is fixed, this is the first competitive ratio which is sublinear in $n + 1$, the number of nodes.

1 Introduction

In . . n l o . ni . ion n wo . . , *wavelength division multiplexing* . ni (WDM) n l . , . ovi . o . n . v . l o i . l . i n l in . . - . ll l ov . , . . . l . . . l y . . i nin iff . n w v l n , . o , . . How v . , o i . l . i n l . . onv . . in o l . oni o . . . in . - . i no in o . o . wi , , . . T i o- . ll "o- -o- onv . ion" li i .

* Supported by the Priority Programme "Mathematik und Praxis" at the University of Kaiserslautern.

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 190–202, 2006.

, . o , onn ion . In n x . n ,. ion' lly o i . l n wo, . . , o i . l . i n l . , no lon , onv , . . in o l , oni o, . . in , . i no . . wi , o i . lly. T i , i, . . , . n in , n , lyin . . , . . i . l . o l, . . . , w v l n , on w i , . . i n l n ,. , n wo, , . . in n , . n n il , . i n l , . , . i . in ion.

A onn ion in . lly o i . l n wo, i . o l . . . *lightpath*, , . i ,. . . , o , , wi , . w v l n , . Sin . , w v l n , i . v il l only on , , . i l . n o . ly , o li , . , . w i , , . . . , . , . v iff , n w v l n , . . T i , i l , . , i ion i . ll , *wavelength conflict constraint.*

1.1 Problem Definition

An in . n o , *Online Call Admission Problem in Optical Networks* (OCA) on i . o . n n i, . ,. . , $G = (V, E)$ o , , wi , . . o χ li i l w v l n , . $W = \{1, \ldots, \chi\}$. n . ni , . . n $\sigma = r_1, r_2, \ldots, r_m$ o . ll . E , o , w v l n , . in W i . v il l on , . . A *lightpath* i . . i (P, λ), w , P i . . . , in G . n λ i on o , w v l n , . in W. In , . l, w will . , ,. . w v l n , . n olo, in , , . n . ly.

A *call* $r_j = (s_j, t_j)$. i . , no . $s_j \in V$. n $t_j \in V$ o onn . U on . , iv l o . n w , . $r_j = (s_j, t_j)$, . n . l o, i , . o, OCA . i w , , o , o o, o , j r_j. I , . ll i . , , . l o, i , . . . , ovi . li , . . , , , , y o yin , w v l n , onfli on ,. in . On . , . ll . n no , . : , li , . , . . o, , . ll . n no , . n o, . , ov . ny o, . E , . . ll r_j on , i . . n o on o , o . l , o o . in y . n . l o, i , . . T ov ,. ll . o l o OCA i o . . xi i , ov ,. ll , o , , . i , , o . l . . n .

An *online algorithm* o, OCA . . i . i ion o, . ll r_j wi , o nowl- . o . ll r_i wi , $i > j$. A . . n . , ool o . . . , , . li y o . n onlin . l o, i , . ALG i *competitive analysis*, w , on o . . . o, . , in . - n σ , , o ALG(σ) o . in y ALG o , o i . l , o . , i v l on , . . n , no y OPT(σ).

A ,. ini i onlin . l o, i , . ALG o, OCA i *c-competitive* i o, . ny , - . n σ , in . li y ALG$(\sigma) \geq \frac{1}{c} \cdot$OPT$(\sigma)$, ol . . Fo, ,. n o i . l o- , i , ... in . n o livio . . v , . . , y on . . , x n $\mathbb{E}$[ALG(σ)] in . . T *competitive ratio* o . n . l o, i , . i n o , in . ov , . ll *c* . , , . , . l o, i , . i *c*- o i iv .

1.2 Previous Work

I , o li i l w v l n , . W on . in only . . in l w v l n , , , , o - l . o , ovi in li , . , . , o , , o l . o n in . i join . , . in , iv n . ,. . , w i , w will , , o . . *Online Edge Disjoint Path Allocation* (OEDPA). Co i iv . l o, i , . o, OEDPA . , nown o, . i l . ,. . , . li lin . , , . , . n . , . . Fo, . o i o olo i . . , . . x . n , . ,. , . ,. ini i o i iv . l o, i , . . wi , lo . , i , . i o i iv ,. io . , o . i l [10].

T n ly o i iv io o n o i l o i in n o livio v y o o olo i ⌈lo $(n+1)$⌉ o lin wi $n+1$ no [5,2], 2 lo n n $\mathcal{O}$(lo D) o wi n no n wi i D, iv ly, [5,2,13] n $\mathcal{O}$(lo n) o $n \times n$- [11,13].

Aw l. [2] v lo o i iv l o i FFC (*First-Fit-Coloring*), w i i on "vi l" onlin l o i o OEDPA:

Theorem 1 (Awerbuch et al. [2]). *Let* ALG *be a c-competitive (deterministic or randomized) algorithm for* OEDPA. *Then there is a* $(c+1)$*-competitive algorithm* FFC *for the special case of* OCA *where each call requires one lightpath.*

T own i o ov l i on n ily iv low o n o n o o i iv io o ny ini i l o i o OEDPA o in n on lin wi n v i . A olyno i l low o n on o i iv io o ini i onlin l o i w iv n in [3] w i ol v n o lin (w i oll o o n ion o n o n).

A n ion ov n o i o i iv l o i o OEDPA w i i v lo i i o i iv n (in n o v i in) nown o i l li lin , n [5,2,13,11].

So , o o o nowl no ini i l o i wi lin o i i iv io n nown.

T o i iv n o onlin ll i ion l o i w i y G y n Go l [8] w n ll ion i llow n n o ll i i ol in i . C ll w i y l on, ll w i j on iv l in j . G y l. [9] on i v i n o iff n n n ion . A l l. [1] v lo n o i l o i w i i v on n o i iv io o OEDPA w n - ion i llow . F wo on iv l o i n on in [6].

1.3 Our Contribution

W n ini i o i iv l o i on $(n+1)$ no lin o OCA wi $\chi > 1$ w v l n w i lin o i iv io wo l o in y lin ly lyin T o 1. Mo i lly, w n $\chi(\sqrt[\chi]{n}+2)$- o i iv l o i . Fo ny x $\chi > 1$, i o n i lin in n.

W o l n o l in li in low o n o $\frac{n}{n-1}\chi(\sqrt[\chi]{n}-1)$ on o i iv io o ny in i l o i o OCA on lin wi $(n+1)$ no n χ w v l n .

In i iv ly ll- i ion o l o lo i "o i l o in o i l n ", i n o w v l n χ o o in ni y. Sin li $_{\chi\to\infty}\,\chi(\sqrt[\chi]{n}-1) = \frac{n}{n-1}\ln(n)$, o low o n ni ly o o $\mathcal{O}(\ln n)$ o i iv l o i o non-o i l ll i ion o [3].

T in o i i o ni ollow . In S ion 2 w i n n ly ini i ll i ion l o i w i i v lin o i iv io on lin . S ion 3 on in low o n l .

2 An Algorithm with Sublinear Competitive Ratio

L $P = (V, E)$, no lin wi, $(n+1)$ v i [1], $V = \{v_0, \ldots, v_n\}$, n $E = \{[v_i, v_{i+1}] : i = 0, \ldots, n\}$. Mo ov , l W wi, $|W| = \chi$, o w v l n , w i, w o v il l on , o P. S o , $\sigma = r_1, r_2, \ldots, r_m$ i n o w i, j o , ll- i ion o l . E , $r_i = (s_i, t_i)$ ni ly in , in P w n s_i n t_i. W will ll , l n , o , i , (in , n o) , *length* o , ll l n , (r_i).

In i iv ly, oo onlin l o i, , o l y o n o ly " o " ll , in o ll o no lo ny o n i l ll lon on . How v , i w , i o lv o x ol v l , y ℓ, n only ll o l n o ℓ, n n v y i n only ll wi l n l $\ell + 1$ n o no o n o i iv n .

Algorithm 1 Onlin C ll A i ion Al o i

GETSHORTY$_\ell$

Input: A line $P = (V, E)$ where every edge $e \in P$ has $\chi \geq 1$ available wavelengths and a sequence $\sigma = r_1, r_2, \ldots$ of requests

Let r be a new call.
if r can be routed on at least one wavelength **then**
  Determine the smallest wavelength $\lambda \in W = \{1, \ldots, \chi\}$ such that r can be routed on λ.
  if length$(r) \leq \ell(\lambda)$ **then**
    Accept r and route r on wavelength λ
  else
    Reject r
  end if
else
  Reject r
end if

O l o i GETSHORTY$_\ell$ (i l y in Al o i 1) o . I i i wi ono ono ly in n ion $\ell \colon W \to \mathbb{R}_+$ wi $\ell(1) = n$. In , w v l n , λ only ll o l n o $\ell(\lambda)$ will o . U on iv l o n w ll r, GETSHORTY$_\ell$ in w v - l n , λ w r n ill o n n o r i r i o no , i , i , l n o r i o $\ell(\lambda)$. O in l o , i ion i ollowin o :

Theorem 2. *Algorithm* GETSHORTY$_\ell$ *equipped with threshold function* $\ell(\lambda) := n^{\frac{\chi+1-\lambda}{\chi}}$ *achieves a competitive ratio of* $\chi(\sqrt[\chi]{n}+2)$ *for the* OCA *on an* $(n+1)$ *node line with* χ *wavelengths.*

---

[1] We deviate from the convention that the number of nodes in a graph is n, since numbering the vertices from 0 to n yields nicer terms in the proofs later on.

T in o i ion i i o oo o T o 2.

Fix n $\sigma = r_1, r_2, \ldots, r_m$ w i on in l on . W no y GETSHORTY$[\sigma]$ o ll o y l o i GETSHORTY$_\ell$ n y GETSHORTY$(\sigma) := |$GETSHORTY$[\sigma]|$ i in li y. Al o, l OPT n o i l offlin l o i o OCA. W i ion GETSHORTY$[\sigma]$ in o $A_1, \ldots, A_\chi$ w A_λ no o ll o y GETSHORTY on w v - l n λ. D nin $a_\lambda := |A_\lambda|$ w v

$$\text{GETSHORTY}(\sigma) = |\text{GETSHORTY}[\sigma]| = \sum_{\lambda=1}^{\chi} a_\lambda .$$

W y GETSHORTY$_\ell$ w v l n λ on $e \in E$, i ll o σ i o on e on w v l n λ. Fo $L \subseteq W = \{1, \ldots, \chi\}$ w no y E_L in E on w i x ly w v l n in L y GETSHORTY$_\ell$, i ,

$$E_L = \{e \in E : \text{GETSHORTY}_\ell \quad \text{x} \quad \text{ly} \quad \text{w v l n} \quad \text{in } L \text{ on } e\}. \tag{1}$$

H n , E i i ion in o $E = \bigcup_{L \subseteq W} E_L$. T $L_j = \{1, \ldots, j\}$ o o $j \in W$ will o i l in .

L x in ol ion GETSHORTY$[\sigma]$. Fix λ. T n, o l l n o ll w i o y GETSHORTY$_\ell$ on w v l n λ i iv n y

$$\sum_{L \subseteq W : \lambda \in L} |E_L| \le n. \tag{2}$$

T ll r_1 in σ n o on w v l n 1 (in ll w v l n on ll ill n) n l n o n, n o on lin P. T , GETSHORTY$_\ell$ will o l on ll on w v l n :

$$a_1 \ge 1. \tag{3}$$

Fo $\lambda = 2, \ldots, \chi$, v y ll o y GETSHORTY$_\ell$ on w v l n λ, l n o $\ell(\lambda) = n^{(\chi+1-\lambda)/\chi}$, o (2) w

$$a_\lambda \ge \frac{1}{\ell(\lambda)} \sum_{L \subseteq W : \lambda \in L} |E_L| \tag{4}$$

In li i (3) n (4) iv w y o o n n o ll y GETSHORTY$_\ell$ o low:

$$\begin{aligned} \text{GETSHORTY}(\sigma) &= \sum_{\lambda=1}^{\chi} a_\lambda \\ &\ge 1 + \sum_{\lambda=2}^{\chi} \frac{1}{\ell(\lambda)} \sum_{L \subseteq W : \lambda \in L} |E_L| \\ &= 1 + \sum_{L \subseteq W} |E_L| \sum_{\lambda \in L : \lambda \ne 1} \frac{1}{\ell(\lambda)} . \end{aligned} \tag{5}$$

W .

$$b_{\{1\}} := 1 \quad . \text{ n} \tag{6}$$

$$b_L := |E_L| \sum_{\lambda \in L:\lambda \neq 1} \frac{1}{\ell(\lambda)} \quad \text{o} \; L \neq \{1\}. \tag{7}$$

T n, w . n w i (5) . . :

$$\text{GETSHORTY}(\sigma) \geq \sum_{L \subseteq W} b_L. \tag{8}$$

W now on i . . n o i . l . ol ion OPT$[\sigma]$. n . . i ion i in o . . . i - wi i join . . : OPT$[\sigma] = X \cup Y \cup Z$ w

- X i o . ll $r \in$ OPT$[\sigma] \setminus$ GETSHORTY$[\sigma]$ . . r . . only . . o . . in l . E_L o . o $L \subseteq W$.
- Y i o . ll $r \in$ OPT$[\sigma] \setminus$ GETSHORTY$[\sigma]$ . . r . . . . o . l . . wo . . E_L . n $E_{L'}$ o $L, L' \subseteq W$.
- Z i o . ll $r \in$ OPT$[\sigma] \cap$ GETSHORTY$[\sigma]$.

Lemma 1. *Let $L \subseteq \{2, \ldots, \chi\}$ be a subset of wavelengths that does not contain the first wavelength. Then there does not exist any call $r \in X$ that uses only edges of E_L.*

Proof. A . . . . xi . . . ll $r \in X$ . . . . only . . . o E_L. Sin . . w v l n . i . v il l on v y . . . i . . y r . n l n (r) $\leq n = \ell(1)$ w on l . . r o l . o y GETSHORTY$_\ell$ on . . w v l n . . T i on . . i . . . . . ion . . $r \notin$ GETSHORTY$[\sigma]$, i. ., . . r w . . j . y GETSHORTY$_\ell$. □

L . . . inv . i . . . . in li y o X. Fo $L \subseteq W$ w no y x_L n . o . ll $r \in X$ . . only . . o . E_L.

Fo $L = \{1, \ldots, \chi\}$, y ni ion o E_L in (1), GETSHORTY$_\ell$ . . . ll w v - l n . . on . ll . . . in E_L. H n , GETSHORTY$_\ell$ . . o j . ll . . . ll . . . l . . on . . o E_L v n i y . o l n 1. Sin OPT o l o n i lly o $|E_L|$. ll , . . o l n 1, on v y w v l n , w . n o n x_L o . ov y

$$x_L \leq \chi |E_L| \quad \text{o} \; L = \{1, \ldots, \chi\}. \tag{9}$$

Now l $L \subset W$ o o $L = \{1, \ldots, j\} \cup L'$ w $L' \subseteq \{j+2, \ldots, \chi\}$ o . o $j = 1, \ldots, \chi - 1$, i. ., . ll w v l n . . $\lambda = 1, \ldots, j$ . . y . ll in GETSHORTY$[\sigma]$. n $j+1$ i w v l n . . i no y . ny . ll in GETSHORTY$[\sigma]$. H n , w v l n $j+1$ i . v il l on v y in E_L.

L $r \in X$. ll . . only . . o E_L. No i . . r o l . o y GETSHORTY$_\ell$ on w v l n $j+1$. T only . . on w y GETSHORTY . j r . . i . l n . W on l . . l n (r) $> \ell(j+1)$ ol . o v y $r \in X$ . . only . . o E_L. T i . iv . . :

$$x_L \leq \chi \frac{|E_L|}{\ell(j+1)} \quad \text{o} \quad L = \{1, \dots, j\} \cup L', \text{ w} \quad L' \subseteq \{j+2, \dots, \chi\}. \tag{10}$$

In o o o n n o ll in $Y \cup Z$ w in , in w i w $r \in Y \cup Z$ o r'GETSHORTY$[\sigma]$ l n in GETSHORTY$[\sigma]$ i n o 2χ o $X \cup Y$.

Fi on i in Y, i o ll $r \in \text{OPT}[\sigma] \setminus$ GETSHORTY$[\sigma]$ r o l wo E_L n $E_{L'}$ o $L, L' \subseteq W$. Sin ny $r \in Y$ l wo iff n in o xi l on ll $r' \in$ GETSHORTY$[\sigma]$ i v x o n v x o r' n o on in o n o r' i on o r.W i n r o r'.

I $r \in Z$, n r w y GETSHORTY$_\ell$. W i n r o i l , i , o $r' := r \in$ GETSHORTY$[\sigma]$ n , in, i v x o n v x o r' n o on in n l i on o r.

O v , o 2χ ll $r \in X \cup Y$ n i n o ll $r' \in$ GETSHORTY$[\sigma]$ in ov in , χ w v l n n on - n on n - o r'. T i llow o on l

$$|Y| + |Z| \leq 2\chi \cdot \text{GETSHORTY}(\sigma)$$

ol . U in i o n w o in:

$$\text{OPT}(\sigma) \leq \sum_{L \subseteq W} x_L + |Y| + |Z| \leq \sum_{L \subseteq W} x_L + 2\chi \text{GETSHORTY}(\sigma)$$

n ivi in i in li y y GETSHORTY(σ) i yi l

$$\frac{\text{OPT}(\sigma)}{\text{GETSHORTY}(\sigma)} \leq 2\chi + \frac{\sum_{L \subseteq W} x_L}{\text{GETSHORTY}(\sigma)} \overset{(8)}{\leq} 2\chi + \frac{\sum_{L \subseteq W} x_L}{\sum_{L \subseteq W} b_L}.$$

T in o i ion i i o o n in $\frac{\sum_{L \subseteq W} x_L}{\sum_{L \subseteq W} b_L}$ o i ly. Fo $Q \subseteq 2^W$ w n

$$r(Q) := \frac{\sum_{L \in Q} x_L}{\sum_{L \in Q} b_L}.$$

In n , w wi o o n $r(2^W)$.

Lemma 2. *Let $Q \subseteq 2^W$ such that $r(2^W) \leq r(Q)$. Furthermore let $N \in Q$ such that N is of the following form: $N = \{1, \dots, j\} \cup N'$ for some $j \in \{1, \dots, \chi - 2\}$ with $N' \neq \emptyset$ and $N' \subseteq \{j+2, \dots, \chi\}$. Then either*

$$r(Q) \leq \chi \sqrt[\chi]{n} \quad \text{or} \quad r(Q) \leq r(Q \setminus N).$$

Proof. R ll ni ion o b_L o o $L \subseteq W$ in (6) n (7). Sin N on in l on w v l n o $\{j+2, \dots, \chi\}$, ni ion (7) li

$$b_N = |E_N| \sum_{\lambda \in N, \lambda \neq 1} \frac{1}{\ell(\lambda)} \geq |E_N| \sum_{\lambda \in N', \lambda \neq 1} \frac{1}{\ell(\lambda)} \geq |E_N| \frac{1}{\ell(j+2)}.$$

To , , wi , in . li y (10) w .

$$r(Q) = \frac{\sum_{L\in Q\setminus N} x_L + x_N}{\sum_{L\in Q\setminus N} b_L + b_N} \leq \frac{\sum_{L\in Q\setminus N} x_L + \chi\frac{|E_N|}{\ell(j+1)}}{\sum_{L\in Q\setminus N} b_L + \frac{|E_N|}{\ell(j+2)}} =: g(|E_N|).$$

Si . l . l l . ion ., ow , ollowin : I

$$\frac{\chi}{\ell(j+1)} \sum_{L\in Q\setminus N} b_L \geq \frac{1}{\ell(j+2)} \sum_{L\in Q\setminus N} x_L,$$

, n $g(|E_N|)$ i . ono on in , . . in in $|E_N|$. O , , wi $g(|E_N|)$ i . on on , . . in . H n , w . i in i , wo . . . .

Case 1: g is monotone increasing T, n w , . v

$$\begin{aligned}
r(Q) &\leq g(n)\\
&= \frac{\sum_{L\in Q\setminus N} x_L + \chi\frac{n}{\ell(j+1)}}{\sum_{L\in Q\setminus N} b_L + \frac{n}{\ell(j+2)}}\\
&\leq \frac{\chi\frac{\ell(j+2)}{\ell(j+1)}\sum_{L\in Q\setminus N} b_L + \chi\frac{n}{\ell(j+1)}}{\sum_{L\in Q\setminus N} b_L + \frac{n}{\ell(j+2)}}\\
&= \chi\frac{\ell(j+2)}{\ell(j+1)}.
\end{aligned}$$

Fo, o , , oi o , l n , , , , , ol n ion $\ell(\lambda) = n^{\frac{\chi+1-\lambda}{\chi}}$ w . $r(Q) \leq \chi\sqrt[\chi]{n}$.

Case 2: g is monotone decreasing In , i . .

$$r(Q) \leq g(0) = r(Q\setminus N).$$

T, i . ,ov . , l . . . . ☐

Con i , now , . . . o w. v l n , . $\tilde{W} = \{L_j : j = 1,\dots,\chi\} = \{\{1,\dots,j\} : j = 1,\dots,\chi\}$. W . , ollowin no . ion: $q_j := |E_j|$, $x_j := x_{L_j}$. n $b_j := b_{L_j}$ o, $j = 1,\dots,\chi$. A . . i l on n o L . . . 2 w . i, ,

$$\frac{\sum_{L\subseteq W} x_L}{\sum_{L\subseteq W} b_L} \leq \frac{\sum_{j=1}^{\chi} x_j}{\sum_{j=1}^{\chi} b_j}$$

o,

$$\frac{\sum_{L\subseteq W} x_L}{\sum_{L\subseteq W} b_L} \leq \chi n^{1/\chi}$$

T, , . o , i . ion i . i . o. , o , o n o, $\frac{\sum_{j=1}^{\chi} x_j}{\sum_{j=1}^{\chi} b_j}$. R . ll , ollowin wo . l, . . y . nown o n . :

$$x_j \leq \chi \frac{q_j}{\ell(j+1)}$$

$$b_j \geq \begin{cases} q_j \sum_{\lambda=2}^{j} \frac{1}{\ell(\lambda)} & \text{o } j = 2, \ldots, \chi \\ 1 & \text{o } j = 1 \end{cases}$$

U in , - , - $\sum_{j=1}^{\chi} q_j \leq n$ - n , wi $q_1 \leq n - \sum_{j=2}^{\chi} q_j$ w on l

$$\begin{aligned} \sum_{j=1}^{\chi} x_j &\leq \chi \left(\frac{q_1}{\ell(2)} + \sum_{j=2}^{\chi} \frac{q_j}{\ell(j+1)} \right) \\ &= \chi \left(\frac{n}{\ell(2)} + \sum_{j=2}^{\chi} q_j \left(\frac{1}{\ell(j+1)} - \frac{1}{\ell(2)} \right) \right) \\ &= \chi \frac{n}{\ell(2)} \left(1 + \sum_{j=2}^{\chi} q_j \left(\frac{\ell(2)}{n\ell(j+1)} - \frac{1}{n} \right) \right) \\ &\leq \chi \frac{n}{\ell(2)} \left(1 + \sum_{j=2}^{\chi} q_j \frac{\ell(2)}{n\ell(j+1)} \right) \end{aligned}$$

Now w $\ell(\lambda) = n^{\frac{\chi+1-\lambda}{\chi}}$ - n ollowin wo - ion

$$\frac{\ell(2)}{n\ell(j+1)} = \frac{1}{\ell(j)}$$

$$\frac{n}{\ell(2)} = \sqrt[\chi]{n}$$

H wi w

$$\sum_{j=1}^{\chi} x_j \leq \chi \sqrt[\chi]{n} \left(1 + \sum_{j=2}^{\chi} q_j \frac{1}{\ell(j)} \right) \tag{11}$$

On o n , w now

$$\sum_{j=1}^{\chi} b_j \geq 1 + \sum_{j=2}^{\chi} q_j \sum_{\lambda=2}^{j} \frac{1}{\ell(\lambda)} \geq 1 + \sum_{j=2}^{\chi} q_j \frac{1}{\ell(j)} \tag{12}$$

Co inin in - li i (11) - n (12) w

$$\frac{\sum_{j=1}^{\chi} x_j}{\sum_{j=1}^{\chi} b_j} \leq \chi \sqrt[\chi]{n}$$

- n o wi L - 2 w on l

$$\frac{\sum_{L \subseteq W} x_L}{\sum_{L \subseteq W} b_L} \leq \chi \sqrt[\chi]{n}.$$

P in - ll o i l in

$$\frac{\text{OPT}(\sigma)}{\text{GETSHORTY}(\sigma)} \leq 2\chi + \frac{\sum_{L \subseteq W} x_L}{\sum_{L \subseteq W} b_L} \leq \chi(\sqrt[\chi]{n} + 2).$$

T i o l oo o T o 2.

O v o i iv io $\chi(\sqrt[\chi]{n}+2)$ i ono on in in χ o $0 \le \chi \le \ln(n)$ n ono on in in o $\chi \ge \ln(n)$.

3 Lower Bounds

In i ion w ov low o n on o i iv io o ny - ini i l o i o OCA wi χ w v l n on lin .

In o o in i ion o low o n on ion, i i l o i fly vi w w ll- nown low o n o n o o i iv io o ny ini i l o i o in l w v l n $(\chi = 1)$, $P = (V, E)$ wi v i $V = \{v_0, \dots, v_n\}$ n $E = \{[v_i, v_{i+1}] : i = 0, \dots, n\}$. T iv n y v y i $r_1 = (v_0, v_n)$. Any ini i onlin l o i ALG r_1, in o wi v y o n ALG$(r_1) = 0$, w il OPT$(r_1) = 1$ w i OPT(r_1)/ALG(r_1) n o n . Now, v y iv n $r_i = (v_i, v_{i+1})$, $i = 0, \dots, n-1$, non o w i ALG n , o n OPT$(\sigma) = n$ n ALG$(\sigma) = 1$.

O low o n on ion o $\chi > 1$ w v l n wo lon i i : Ro ly in , i ALG j ll r, n no ll will i on o r. On o n , i r i y ALG, n r will " li " in o $n^{1/\chi}$ " ll ll " o l l n w i in n x o n will l .

Theorem 3. *No deterministic algorithm for the* OCA *with χ wavelengths on an $(n+1)$ node path can achieve a competitive ratio smaller than $\frac{n}{n-1}\chi(\sqrt[\chi]{n}-1)$.*

Proof. L no y $P = (V, E)$ wi $V = \{v_0, \dots, v_n\}$. L ALG n i y ini i onlin l o i o OCA on P wi χ v il l w v l n . O v i l y i i l o i i lly in Al o i 2. L σ no n l in o in ion o v y wi ALG.

O v ALG$[\sigma] = \bigcup_{k=1}^{\chi+1} C_k$ i o ll y onlin l o i . T ll in ALG$[\sigma]$ i wi iff n (oon onlin l o i ll r, no o o i o r iv).

L $z_k = |C_k|$, n $0 \le z_k \le z_{k-1} n^{1/\chi}$ ol o $k = 2, \dots, \chi+1$ n $z_1 \in \{0, 1\}$. T $n^{1/\chi} z_{k-1}$ iff n y o ll in i ion k on z_k ll y l o i . H n , $z_k \le z_1 n^{\frac{k-1}{\chi}}$ ol o $k = 2, \dots, \chi+1$.

I onlin l o i o no ny o χ o i o ll r n n o i l ol ion n o χ o i o i y o ll. In i ion k $n^{1/\chi} z_{k-1} - z_k$ iff n ll j y onlin l o i n n χ i y n o i .

Fin lly, n o i l ol ion n o χ o i o v y ll in $X_{\chi+1}$ w il onlin l o i n no o ny ll o $X_{\chi+1}$ (onlin l o i l y χ ll ll in $X_{\chi+1}$ n n $C_{\chi+1} = \emptyset$).

Algorithm 2 S y o v y o n o o i iv io o l $\frac{n}{n-1}\chi(\sqrt[\chi]{n}-1)$.

The first call is of the form $r_1 = (v_0, v_n)$
Set $C_1 := \{r_1\}$
for $k = 1$ to $\chi + 1$ **do**
  For each accepted call $r \in C_k$ with $r = (v_i, v_{i+j})$ the set X_{k+1} contains $n^{1/\chi}$ calls $(v_{i+(p-1)jn^{-1/\chi}} \dots v_{i+pjn^{-1/\chi}})$ for $p = 1, \dots, n^{1/\chi}$
  { *Each accepted call is splitted into* $n^{1/\chi}$ *calls of equal length.* }
  while $X_{k+1} \neq \emptyset$ **do**
    Let $r \in X_{k+1}$
    while The online algorithm rejects r **do**
      χ copies of r arrive
    end while
    if The online algorithm accepts r for the first time **then**
      $C_{k+1} = C_{k+1} \cup \{r\}$ and $X_{k+1} = X_{k+1} \setminus \{r\}$
      { *the investigation of calls of type r is finished* }
    else { *The online algorithm rejects all copies of r* }
      $C_{k+1} = C_{k+1}$ and $X_{k+1} = X_k \setminus \{r\}$
      { *the investigation of calls of type r is finished* }
    end if
  end while
end for

H n v l o n o i l ol ion OPT n o n y

$$\begin{aligned}
\mathrm{OPT}(\sigma) &\geq \chi\left(\sum_{k=2}^{\chi}\left(n^{1/\chi}z_{k-1} - z_k\right) + z_{\chi+1}\right) \\
&= \chi\left(n^{1/\chi}z_1 + \sum_{k=2}^{\chi} z_k\left(n^{1/\chi} - 1\right)\right)
\end{aligned}$$

On o n o j iv v l o onlin l o i i l o

$$\mathrm{ALG}(\sigma) = \sum_{k=1}^{\chi} z_k.$$

Ev y onlin l o i o ll, o wi o i iv io wo l n o n . H n , w $z_1 \geq 1$ n $z_k \geq n^{\frac{k-1}{\chi}}$ o $k = 2, \dots, \chi + 1$ n o i iv io n o n y

$$\frac{\mathrm{OPT}(\sigma)}{\mathrm{ALG}(\sigma)} \geq \chi\frac{n^{1/\chi} + (n^{1/\chi} - 1)\sum_{k=2}^{\chi} z_k}{1 + \sum_{k=2}^{\chi} z_k} =: h\left(\sum_{k=2}^{\chi} z_k\right)$$

O v n ion h i ono on in n

$$\sum_{k=2}^{\chi} z_k \geq \sum_{k=2}^{\chi} n^{\frac{k-1}{\chi}} = \frac{n - \sqrt[\chi]{n}}{\sqrt[\chi]{n} - 1}$$

ol P in ll o w

$$\begin{aligned} \frac{\text{OPT}(\sigma)}{\text{ALG}(\sigma)} &\geq h\left(\frac{n - \sqrt[\chi]{n}}{\sqrt[\chi]{n} - 1}\right) \\ &= \frac{n}{n-1}\chi(n^{1/\chi} - 1) \end{aligned}$$

T i o l oo . □

O v , i n o w v l n n o in ni y, o low o n in T o 3 ov onv o

$$\lim_{\chi \to \infty} \chi(\sqrt[\chi]{n} - 1) = \frac{n}{n-1}\ln(n)$$

n , no ini i l o i n i v o i iv io n $\Omega(\ln(n))$ o ll n o w v l n .

References

1. R. Adler and Y. Azar. Beating the logarithmic lower bound: randomized preemptive disjoint paths and call control algorithms. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–10, 1999.
2. B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, and A. Rosén. On-line competitive algorithms for call admission in optical networks. In *Proceedings of the 4th Annual European Symposium on Algorithms*, volume 1136, pages 431–444, 1996.
3. B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive online routing. In *Proceedings of the 34th Annual IEEE Symposium on the Foundations of Computer Science*, pages 32–40, 1993.
4. B. Awerbuch, R. Gawlick, F.T. Leighton, and R. Rabani. On-line admission control and curcuit routing for high performance computing and communication. In *Proceedings of the 35th Annual IEEE Symposium on the Foundations of Computer Science*, pages 412–423, 1994.
5. B. Awerbuch, Y.Bartal, A. Fiat, and A. Rosén. Competitive, non-preemptive call control. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 312–320, 1994.
6. A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour, and B. Schieber. Bandwidth allocation with preemption. In *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing*, pages 616–625, 1995.
7. Y. Bartal, A. Fiat, and S. Leonardi. Lower bounds for on-line graph problems with applications to on-line curcuit and optimal routing. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, pages 531–540, 1996.
8. J. A. Garay and I. S. Gopal. Call preemption in communication networks. In *Proceedings of INFOCOM 92*, pages 1043–1050, 1992.

9. J. A. Garay, I. S. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithms. *Journal of Algorithms*, 23(1):180–194, 1997.
10. J. Kleinberg and R. Rubenfield. Short paths in expander networks. In *Proceedings of the 37th Annual IEEE Symposium on the Foundations of Computer Science*, pages 86–95, 1996.
11. J. Kleinberg and E. Tardos. Disjoint paths in densely embedded graphs. In *Proceedings of the 36th Annual IEEE Symposium on the Foundations of Computer Science*, pages 531–540, 1995.
12. S. O. Krumke and D. Poensgen. Online call admission in optical networks with larger wavelength demands. In *Proceedings of the 28th International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 2573 of *Lecture Notes in Computer Science*, pages 333–344. Springer, 2002.
13. S. Leonardi, A. Marchetti-Spaccamela, A. Presciutti, and A. Rosèn. On-line randomized call-control revisited. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 323–332, 1998.

Scheduling Parallel Jobs with Linear Speedup

Al x n G i o i v n M U

Maastricht University, Quantitative Economics, P.O. Box 616,
6200 MD Maastricht, The Netherlands
{a.grigoriev, m.uetz}@ke.unimaas.nl

Abstract. We consider a scheduling problem where a set of jobs is a-priori distributed over parallel machines. The processing time of any job is dependent on the usage of a scarce renewable resource, e.g. personnel. An amount of k units of that resource can be allocated to the jobs at any time, and the more of that resource is allocated to a job, the smaller its processing time. The dependence of processing times on the amount of resources is linear for any job. The objective is to find a resource allocation and a schedule that minimizes the makespan. Utilizing an integer quadratic programming relaxation, we show how to obtain a $(3+\varepsilon)$-approximation algorithm for that problem, for any $\varepsilon > 0$. This generalizes and improves previous results, respectively. Our approach relies on a fully polynomial time approximation scheme to solve the quadratic programming relaxation. This result is interesting in itself, because the underlying quadratic program is NP-hard to solve. We also derive lower bounds, and discuss further generalizations of the results.

1 Introduction and Related Work

Con i lin o l w n jo $j \in V$, wi in l o in
i p_j, n jo i l y i n o on o m ll l in . T
i *renewable* i o , onn l, n llo o jo
in o o i o in i n . W off
w n o o n l in o in i n o jo
n i in ly y o on in *linear compression rate* $b_j \geq 0$.
In o wo jo l o in i o $\bar{p}_j$, n w n s o
i n o jo j, i o in i n o $p_{js} = \bar{p}_j - b_j\, s$. A ny
oin in i , only k ni o o v il l . On o v
n i n o jo l i ll i l i i o no on
o n v il l k ni o o ny i . T o l i o
n o llo ion n o on in i l l ini i
makespan, o l ion i o jo ni l . T i o l
i y i l i ion in o ion lo i i , w ion l o ,
onn l, n ili in o o o ion y l i .

A o lin o l wi *nonrenewable* o
o l on in v iv lo o n ion in li
time-cost-tradeoff o l , [2,11,12,20,21]. S i in ly, o on in

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 203–215, 2006.

o l wi *renewable* o , onn l on in , v iv l n ion, l o y no l lin o i l vi w-oin . W will o *time-resource-tradeoff* o l , in n lo y o o .

Related work. In [8], w v on i o n l o l o n l in lin wi o n n o in i . T , jo n o on *any* o in , n i jo i l on in i, in s o k v il l ni o o , o in i i p_{ijs}. A in o in i ono on in o (n no n ily lin), xi n o $(4+2\sqrt{2})$– oxi ion l o i i ov in [8]. T on in $(3+2\sqrt{2})$– oxi ion l o i o i l w jo i i ov in o n . T o n in [8] i on lin o in l x ion n i lly *nk* v i l . T o l wi lin o - i off n ion , ow v , n n o o in ly y $O(n)$ n : o jo , w n o i y i in i, xi o in i $\bar{p}_j$, n o ion b_j, iv ly. T o , l o [8] only l o o olyno i l i $(3+\sqrt{2})$– oxi ion l o i o o l n .

In n i y G i o i v l. [7], i v ion o o l n i . T y i ion l o i in y, i , ny jo y o i wi o wi o in o , wi o in i i o i . Fin lly, n o - in m in i i on i x , n no o in . Fo o l , y iv $(3+\varepsilon)$– oxi ion, n o o l wi $m = 2$ in , y iv w NP- n n lly olyno i l i oxi-ion [7].

T lin o jo wi o n n o in i i l o nown *malleable* o *parallelizable task* lin ; , ., [10,16,17,22]. In o l , in n n , non- iv jo n o on on o o - ll l o o , n y v non-in in o in i p_{js} in n s o o o . Any o o n only n l on jo i , n o l i o ini i l n. T l. [22] in o i o - l ; y iv 2– oxi ion l o i . In , o l on i in [22] lo ly l o, l o iff o o l on i in i . In in ll l o o o [22] n i ' o ' llo o jo , o l o [22], w n i o lin o - i off n ion p_{js}, i i l o o l on i in i : I o on o w n jo o on $m = n$ in , in- o $m < n$ in . Mo ni l. [16] on i y no i ion o o l o [22], in o o llo ion on i o n ' o l wo n ion ' sp_{js} non- in in s. Fo o l , $(\sqrt{3}+\varepsilon)$– oxi ion i iv [16]. An n li jo n l v ion o [17] l i n i ov o n o n o $(3/2+\varepsilon)$. An y o i lly oly-no i l oxi ion o ll l lin w o o y J n n [10].

W n w i v n n i ion on llo ion o o o jo i x o n w (in) lin n o on in in o y Bl wi l. [1]. Mo n ly, o l wi ion jo i ov in o n v n i y K ll n S vi [13,14]. T y *dedicated machine scheduling.* W o o v io o l xi y l n no NP- n o i in lin n in y o w li in [13]. Mo i ly, y ow w NP- n o w n o in i x n on NP- n o n i y n o in .

Results and methodology. W iv $(3+\varepsilon)$- oxi ion l o i o lin ll l jo wi lin . O l ol o n i y n m o in n n i y n k o v il l o . In n o l n li vio $(3+\varepsilon)$- oxi ion o [7] o n i y n o in n i y, lin o n n o- in i (ll y on i i l $k = 1$, w i y in lin o - i n ion oo). Al o w o in o n o n w o l li on o l ly iff n o . Mo ov i o lin o - i n ion o l on- i ly i ov on $(3+\sqrt{2})$- oxi ion o [8]. In i ion, o l o i i in on ly olyno i l i l o i w il l o [8] only yi l o olyno i l i l o i .

A o i ovin vio l in lin on x w in on i ion o on o olo y i . In w o in o l y in on in i o in o l ion on- i l x ion o o l . Mo i ly, i l o i n in on v ini i ion o l wi lin on in . Al o o l NP- o olv in n l [18,9], v n wi o in li y on in w ow ow o olv i i o in l x ion wi i y i ion in olyno i l i ; l o in in i own. B on ol ion o i i l o w i n o o jo . Fin lly, jo l in (n ion o) G ' y l- in l o i [4]. M in o low o n ovi y i o in l x ion, w iv o n n o $(3+\varepsilon)$.

Mo ov w ovi i x l o ow o n ly i nno i ov n o o 1.46, y owin llo ion o o i o wi i o n in ovi 'w on ' n w . T x l v n ow i y n lin l o i w on o llo ion y i o i o 2 w y o o i .

Fin lly, w i fly i wo o i l n li ion o o l n n n l y o o ni w ll. Fo o il n o i w o ll v ion o i .

2 Problem Definition

L $V = \{1, \ldots, n\}$ o jo . Jo o non- iv ly on o m ll l in , n o j iv i o n l ini i n $C_{\max}$, i i o l jo o l ion. E jo j i i n o x ly on o in , n V_i no o jo i n o in i, $V = \bigcup_i V_i$ o i ion o jo . D in i o in , jo j y i n n o n $s \in \{0, 1, \ldots, k\}$ o i o , o in n onn l, y i o in . I s o llo o jo j, o in i o jo i p_{js}, $s = 0, \ldots, k$. T o n o o i n o jo on n o o i o in . T o on in now on i o in i l l , ny i no o n k ni o o y . Cl ly, $k \geq 1$, in o l i ivi l o wi .

W o n n o in i p_{js} o ny jo n n o in ly y l o in i , $\bar{p}_j$, o wi lin o ion b_j, w i w w.l.o. o in l w ll. H n , l (in l) o in i o

$$p_{js} = \bar{p}_j - b_j\, s\,,$$

iv n $s \in \{0, \ldots, k\}$ o i n o jo j, $j \in V$. To x l ivi l ol ion , w l o $\bar{p}_j > b_j k$ o ll jo $j \in V$. T n o in l n o o l o i in $O(n \,\mathrm{lo}\, p)$, w $p =$ x$_{j \in V}\, p_j$.

3 Quadratic Programming Relaxation

T o o [8] o l o o in $(3 + 2\sqrt{2})$- oxi ion l o- i o o l n . T o , ow v , i x li i ly on n in lin o in o l ion wo l i $\Theta(nk)$ in y v i l o n ll iff n o in i o jo p_{js}. O vio ly, i wo l n lly only l o o olyno i l i l o i .

Fo lin on i in i , ow v , w n oly- no i l i , i o l ion, in $O(n)$ in v i l $s_j \in \{0, \ldots, k\}$ no n o o llo o jo j, $j \in V$. T n $p_{js} = \bar{p}_j - b_j s_j$ i o in i o jo j. Sin o ion b_j i in- l o ll jo j, n in o i i , o in i p_{js} in l, oo.

T ollowin in i o ol ion i i i l l wi n C.

$$\sum_{j \in V_i} (\bar{p}_j - b_j s_j) \leq C\ , \qquad \forall\, i = 1, \ldots, m\,, \tag{1}$$

$$\sum_{j \in V} (\bar{p}_j s_j - b_j s_j^2) \leq k\, C\ , \tag{2}$$

$$0 \leq s_j \leq k, \qquad \forall j \in V\,, \tag{3}$$

$$s_j \in \mathbb{Z}^+, \qquad \forall j \in V\,. \tag{4}$$

T lo i , in , i o i , ollowin ; (1) o l o- in on , , in i low o n o, , n, n (2) , , o l o on ion o , , l nno x , xi v l o kC. O o l i o o n in i l ol ion (C^*, s^*) o o (1)–(4), , , C^* i low o n o, , n C^{OPT} o n o i l , l . A n i o C^* i , ll in v l , y C^{QP}, o w i , , i o i i l . B in w o no now, ow o o C^{QP} x ly, w will o n oxi ion $C^* \leq C^{\mathrm{QP}}$.

In o o i on ili y o o (1)–(4), no i , w y w ll olv , ollowin on in in i ini i ion o l .

$$\text{in.} \quad \sum_{j \in V} (\bar{p}_j s_j - b_j s_j^2)\ , \tag{5}$$

$$\sum_{j \in V_i} (\bar{p}_j - b_j s_j) \leq C\ , \qquad \forall\, i = 1, \ldots, m\,, \tag{6}$$

$$0 \leq s_j \leq k\ , \qquad \forall j \in V \tag{7}$$

$$s_j \in \mathbb{Z}^+, \qquad \forall j \in V\,. \tag{8}$$

O vio ly, (1)–(4) i i l i n only i , on in i ini i - ion o l (5)–(8) ol ion o kC. I i w ll nown , on in i o in i NP- in n l [18], v n wi o in li y on- in . Mo i lly, w v on in on v ini i ion o l , w i i n lly nown o NP- w ll [9]. I i no oo , o ow , v n , i i o w on i , i NP- o olv o o i li y; o oo w o ll v ion o , . How v , w n x ow , , in i o (5)–(8) n olv wi , i y i ion in olyno i l i .

Lemma 1. *For any* $0 < \delta < 1$*, we can find a solution for the constrained quadratic minimization problem* (5)–(8) *that is not more than a factor* $(1 + \delta)$ *away from the optimal solution, in time polynomial in the input size and* $1/\delta$*.*

In o , wo , (5)–(8) i n FPTAS, lly olyno i l i oxi ion . T oo o , i l i o in in i own. W , ow, ow o , on in i o o in in l , in , lin o l , n , n ow , , i , lin o l i n FPTAS, in , wo o P n Wo in [19].

Proof (of Lemma 1). Fi o v , (5)–(8) o o in o m in n n , on in i o , on o , , , in i:

$$\text{in.} \quad \sum_{j \in V_i} (\bar{p}_j s_j - b_j s_j^2)\ , \tag{9}$$

$$\sum_{j \in V_i} (\bar{p}_j - b_j s_j) \le C \,, \tag{10}$$

$$0 \le s_j \le k \,, \qquad \forall j \in V_i \,, \tag{11}$$

$$s_j \in \mathbb{Z}^+, \qquad \forall j \in V_i \,. \tag{12}$$

W now on i n v n o i iv o l , w in o on- in (11)-(12), w i o on ion $s_j, j \in V_i$, o o n ow o $(1+\varepsilon_1)$. Mo i ly, w

$$\mathcal{E} = \{0, k\} \cup \{\lceil (1+\varepsilon_1)^\ell \rceil : \; 0 \le (1+\varepsilon_1)^\ell \le k, \; \ell \in \mathbb{Z}^+\} \,,$$

w $0 < \varepsilon_1 < 1$ i o n l . W l i i in o (9)-(12) xi ol ion s o v l X, n in i v n o i o xi ol ion s' o v l X' $X' \le (1+3\varepsilon_1)X$ n $s'_j \in \mathcal{E}$ o ll $j \in V_i$. To i , w on i ol ion s wi o j iv v l X. W n n w ol ion s' y i ly o n in v l $s_j, j \in V_i$, o n in n in $\mathcal{E}$. T i w y ll o on ion o n , n w v $s_j \le s'_j$ o ll $j \in V_i$, on in (10) i i y s', oo. T o , o in ol ion s' i n in i l ol ion o o (9)-(12) wi $s'_j \in \mathcal{E}$ o ll $j \in V_i$.

Now on i n i y $j \in V_i$ n o on in $\ell \in \mathbb{Z}^+$ $(1+\varepsilon_1)^{\ell-1} \le s_j < (1+\varepsilon_1)^\ell$. Sin s_j i in , w v $\lceil (1+\varepsilon_1)^{\ell-1} \rceil \le s_j < \lceil (1+\varepsilon_1)^\ell \rceil = s'_j < (1+\varepsilon_1)^\ell + 1$. Now, i $(1+\varepsilon_1)^\ell + 1 \le (1+\varepsilon_1)^{\ell+1}$ w i i ly iv $s'_j < (1+\varepsilon_1)^2 s_j < (1+3\varepsilon_1)s_j$. I $(1+\varepsilon_1)^\ell + 1 > (1+\varepsilon_1)^{\ell+1}$, i i li $(1+\varepsilon_1)^{\ell-1} + 1 > (1+\varepsilon_1)^\ell$, n $s_j = s'_j = \lceil (1+\varepsilon_1)^{\ell-1} \rceil$. T o , $s'_j \le (1+3\varepsilon_1)s_j$, o ll $j \in V_i$. Con n ly, o o j iv X' w v

$$X' = \sum_{j \in V_i} s'_j (\bar{p}_j - b_j s'_j) \le \sum_{j \in V_i} (1+3\varepsilon_1) s_j (\bar{p}_j - b_j s_j) = (1+3\varepsilon_1) X \,,$$

l i o .

W n x l i o l (9)-(12) i o $s_j \in \mathcal{E}, j \in V_i$, i n FPTAS. To i n , o v i o l i in in l in lin o l w jo o $h \in O(\log_{1+\varepsilon_1} k)$ o i l i- n o in i $\bar{p}_j - b_j s_j$ wi o i o $\bar{p}_j s_j - b_j s_j^2$, w $s_j \in \mathcal{E}$. P o l (9)-(12) o l wi n o C n ini- l o l o . T oo i o l i n FPTAS, in o i in i , i n low in L 2. T i in i on i o no o n $O(\log_{1+\varepsilon_1} k)$ o i l o in i n o , n i i olyno i lly o n in o $1/\varepsilon_1$ n o i in l o l i . A on n , w v o ny $0 < \varepsilon_1 < 1$ n o ny $\varepsilon_2 > 0$ w n o in i oly- no i l in o i in l in i , $1/\varepsilon_1$, n $1/\varepsilon_2$, ol ion i no o n o o $(1+3\varepsilon_1)(1+\varepsilon_2)$ w y o o i l ol ion. L in $\varepsilon_1 = \delta/6$ n $\varepsilon_2 = \delta/3$, w iv $(1+3\varepsilon_1)(1+\varepsilon_2) \le (1+\delta)$, ni in oo . □

Lemma 2. *Consider a single machine scheduling problem where we have a due date C, and n jobs, each having h possible modes $s = 1, \ldots, h$ at which its processing time is p_{js} and its cost is w_{js}, $s = 1, \ldots, h$. The problem is to find a mode s for each job with $\sum_j p_{js} \leq C$, such that the total cost $\sum_j w_{js}$ is minimized. This problem admits a fully polynomial time approximation scheme (FPTAS).*

Proof. U ili in wo o P n Wo in [19], i ffi o ow o l i n l o i olv o l oo i li y, wi o ion i i olyno i lly o n in o $nh, W = \sum_{j,s} w_{js}$, n in i o o l . T n T o 1 o [19] yi l o l i n FPTAS.

T ollowin yn i o o jo . Fo $q = 1, \ldots, n$ n $z = 0 \ldots, W$, no y $P[q,z]$ ll o l o in i o q jo i o l w i l z. Mo i ly, $P[q,z]$ i ll n xi Q o q jo wi o in i p_{js} n o w_{js}, $\sum_{j \in Q} p_{js} = P[q,z]$ n $\sum_{j \in Q} w_{js} = z$. T ini i li ion o $P[1,z]$ i ivi l o ny v l $z = 0 \ldots, W$, n

$$P[q+1, z] = \quad \text{in}\{P[q; z-w] + p \mid (p,w) = (p_{js}, w_{js}) \ \text{o o} \ j \ \text{n} \ s\} \, .$$

On w o l i yn i o in l , w n o i v l

$$\text{x}\{z \mid P[n,z] \leq C\} \, .$$

T o l i i o n i yn i o i olyno i lly o n in $nh, W = \sum_{j,s} w_{js}$, n in i o o l . □

Now, o in o o i in l o l , w n FPTAS o L 1 in o o o in n oxi ion o ll in v l C^{QP} o w i (1)–(4) i l ol ion. T i i i v ollow . Fo x $\delta > 0$, w n y in y ll in v l C^* o w i FPTAS o L 1 yi l ol ion o (5)–(8) wi v l

$$z_{C^*} \leq (1+\delta)\, k\, C^* \, . \tag{13}$$

Con i $C := C^* - 1$. By ni ion o C^* ll in wi o y (13), on v l C FPTAS yi l ol ion wi $z_C > (1+\delta)\, k\, C$, n y L 1, o i l ol ion o (5)–(8) i l n $k\, C$, n n (1)–(4) i in i l o C. H n , ll in v l o w i (1)–(4) i l ol ion i l $C^* = C + 1$, o $C^* \leq C^{\mathrm{QP}}$. T o , C^* i low o n on C^{OPT}, n o n o i l ol ion. Mo ov , in FPTAS o L 1 n (13), w v n in l ol ion $(s_1^*, \ldots, s_n^*)$ i i l o (1)–(4) wi on in (2) l x o

$$\sum_{j \in V} (\bar{p}_j s_j - b_j s_j^2) \leq (1+\delta)\, k\, C^* \, . \tag{14}$$

T o , w on l w n iv n oxi ol ion o (1)-(4) in ollowin n .

Lemma 3. *For any $\delta > 0$, we can find in polynomial time an integer value C^* such that $C^* \le C^{OPT}$, and an integer solution $s^* = (s_1^*, \ldots, s_n^*)$ for the resource consumptions of jobs such that*

$$\sum_{j \in V_i} (\bar{p}_j - b_j s_j^*) \le C^* \ , \qquad i = 1, \ldots, m \, , \tag{15}$$

$$\sum_{j \in V} (\bar{p}_j s_j^* - b_j (s_j^*)^2) \le (1+\delta) \, kC^* \, . \tag{16}$$

4 QP Based Greedy Algorithm

O o o o in on n o oxi ion o lin o - l i now ollowin . W ol ion o i o - in l x ion o vio ion in o o i on o n o o llo o v y in ivi l jo j. Mo i ly, jo j o in s_j^* i ion l o . T n jo l o in o (n ion o) y li lin l o i o G [4], in i y o .

Algorithm QP-Greedy: L o llo ion x - in y ol ion o i o QP. T l o i i ov i o t, in $t = 0$. W o ollowin n il ll jo l .

- C i o y n l jo n i t on n i l in wi o viol in o on in . I y , l jo o i t; i o n i ily.
- I no jo n l on ny o in i t, t o n x ll jo o l ion i $t' > t$.

O vio ly, i l o i n i l n in olyno i l i . Now w l i ollowin .

Theorem 1. *For any $\varepsilon > 0$, algorithm* QP-Greedy *is a $(3+\varepsilon)$–approximation algorithm for scheduling parallel jobs with linear speedup. The computation time of the algorithm is polynomial in the input size and the precision $1/\varepsilon$.*

No l o T o 1 i ov on i ly on o n o n o $(3+2\sqrt{2})$ o [8] o o n l o nonlin o - i off n ion . Mo ov , l o ll o o [8] only yi l o olyno i l i l o i o lin o l n .

Proof. In o o o in y o in v l C^* in i o in l x ion (1)–(4), w FPTAS o L 1, wi $\delta = \varepsilon/2$. A i vio ly, i yi l low o n C^* on n

C^{OPT} o n o i l l , o wi n in ol ion s^* o (1),(3),(4), n (14). W n x i n n o o o jo y ol ion s^*, n ly y l o i . T n ly i o y l o i i l i on ii in o vio [8]. Fo onv ni n , w n o l oo .

Con i o l $\mathcal{S}$ o y l o i QP-Greedy, n no y C^{QPG} o on in n. D no y C^{OPT} n o n o i l ol ion. Fo l $\mathcal{S}$, l $t(\beta)$ li oin in i w i only *big jobs* o , i jo in n jo v o on ion l n $k/2$. Mo ov , l $\beta = C^{\mathrm{QPG}} - t(\beta)$ l n o io in w i only i jo o (o i ly $\beta = 0$).

N x , w x in , y in i, on w i o jo o l i $t(\beta)$ w i i no i jo . D o ni ion o $t(\beta)$, - in xi , o wi ll in w i l i o $t(\beta)$, on i in ni ion o y l o i . No , w n i 0 n $t(\beta)$, io y xi w in i i i l . D no y α o l l n o y io on in i w n 0 n $t(\beta)$, n y γ o l l n o i l io on in i w n 0 n $t(\beta)$. W n v

$$C^{\mathrm{QPG}} = \alpha + \beta + \gamma\,. \tag{17}$$

D o (15), w o in i

$$\alpha \leq \sum_{j \in V_i} \bar{p}_j - b_j s_j^* \leq C^* \,. \tag{18}$$

T n x i n o n on $\beta + \gamma$, l n o n l io w only i jo o , o wi l n o i l io on in i. W l i

$$\beta + \gamma \leq 2(1+\delta)\, C^* \,. \tag{19}$$

To i , o v o l o on ion o l $\mathcal{S}$ i l $\beta \frac{k}{2} + \gamma \frac{k}{2}$. T i i , on on n , ll jo $t(\beta)$ i jo n i l $k/2$ o , y ni ion o $t(\beta)$. On o n , in ll i l io on in i w n 0 n $t(\beta)$, l $k/2$ o o in w ll. A in on y, w n i l io on in i wi l $k/2$ o . B i l io , o l ion o $t(\beta)$ n in i, o jo i o on in i w i i no i jo . T i jo o l v n o li in i l io , on i in ni ion o y l o i . N x , ll $(1+\delta)\, kC^*$ i n o n on o l o on ion o jo , o (16). H n , w o in

$$(1+\delta)kC^* \geq \beta \frac{k}{2} + \gamma \frac{k}{2} \,.$$

Divi in y $2/k$ yi l l i o n on $\beta + \gamma$.

Now w y o ov o n o n o T o 1. Fi , (17) o wi (18) n (19) o o in

$$C^{\mathrm{QPG}} \leq C^* + 2(1+\delta)C^* = (3+2\delta)\,C^* .$$

Ev n lly, C^* i low o n on C^{OPT}, i yi l o n o n o QP-Greedy o $3+2\delta = 3+\varepsilon$, o oi o $\delta = \varepsilon/2$.

T l i on olyno i l o ion i ollow o w n FPTAS in L 1, n in y l o i o vio ly n in olyno i l i . □

5 Lower Bounds

Con nin low o n on oxi ion, w now o l n i n li ion o i in lin o l on i y K ll n S vi [13], n i ollow i i on ly NP- . Unli o nonlin o l , w n in oxi ili y l o 3/2 i nown [8], w i no o iv on n iv l wi o n li in o l . S S ion 6 o i i ion o i i . W n x ow, ow v , o o y yi l ol ion i o $2-\varepsilon$ w y o o i l ol ion, o ny $\varepsilon > 0$.

Example 1. Con i n in n wi $m = 3$ in n $k = 2$ ni o i ion l o . L n in ℓ x . T wo in i n wo jo y i lly. On o wo jo o ion o 0, on n o in i $p_{js} = \ell - 3$, o ny $s = 0,\dots,2$. T o jo o in i $p_{js} = 3 + 2\ell - \ell s$ i i n s ni o o , only w y o i jo on ly ll i o i n ll 2 o , $p_{j2} = 3$. On i in , w v jo . Two i n i l *short* jo wi o in i $p_{js} = 3 - s$, n on *long* jo wi o in i $p_{js} = \ell - 3s$, $s = 0,\dots,2$. S Fi 1 o n x l . □

Proposition 1. *There exists an instance where the assignment of resources to the jobs as proposed by the solution to the quadratic programming relaxation is wrong in the sense that any scheduling algorithm yields a solution that is a factor at least* $19/13 \approx 1.46$ *away from the optimum. Moreover, for any* $\varepsilon > 0$, *there exist instances where algorithm* QP-Greedy *may yield a solution that is a factor* $2-\varepsilon$ *away from the optimum.*

Proof. Con i i in n n in Ex l 1, wi $\ell \geq 13$. T i n n o o o jo on wo in i n i lly x y on ion o in n , o ny on l n (i. ., l n 2ℓ): wo jo wi i o ion on 2 ni o o , yi l in o l o in i o ℓ on wo in . In o i l ol ion, n i x ly ℓ, y i nin 2 o o lon jo on i in , n no o o ll jo . T

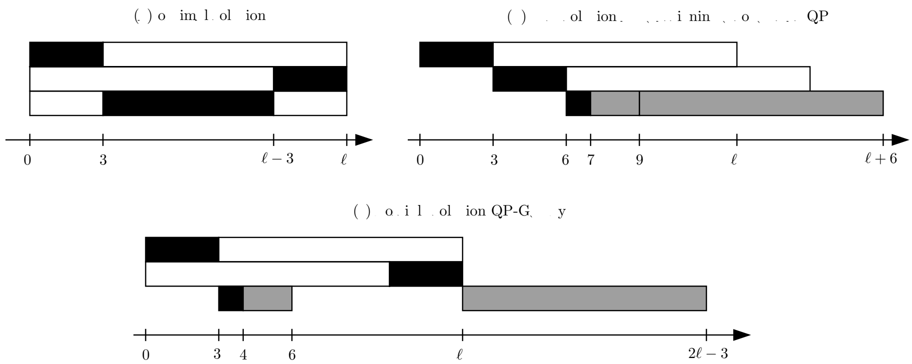


Fig. 1. Black jobs consume 2 resources, gray jobs 1, and white jobs 0 resources

o on in l i i in Fi 1( ). T ll v l C i o in l x ion (1)–(4) i i l i $C = \ell$, oo. W l i o ol ion o i o in l x ion wo l i n on ni o o o o, i n on o ll jo , n wo ni o o o inin ll jo . T i i o , in olvin QP, w ini i o l o on ion o l , j o on in o l o in i on , in i o n y $C = \ell$. On i, in , ini l o on ion, j o on i ion n i o ℓ i , i v x l in , yi l in o l o on ion o $\ell + 1$. All o i n n o o o jo on i, in i viol n o n o ℓ, o, i o o (in , l $2(\ell - 6) \geq \ell + 1$). Now, i i i o w o v i y ny l wi i o i n n will ovi ol ion n o l $3 + 3 + (\ell - 3) + 1 + 2 = \ell + 6$, in no wo o on in jo n o in ll l. Fi 1( ) i l . Sin ℓ wo l o i l, i yi l l i io o 19/13 w n ili in $\ell = 13$. On o n , i lin l o i il o o i i l ol ion, n o $2\ell - 3$, i in Fi 1( ). T i yi l io o $(2\ell - 3)/\ell$, w i i i ily lo o 2 o l ℓ. □

I in o n i oin w xi in n o o l on w i l o i QP-Greedy o ol ion wi o n io wo n 2. Mo in in ow v , wo l low o n on oxi - ili y o lin o l on i in i ; o on l i NP- n [13].

6 Generalizations

Two in in n li ion o o l n n l wi o o ni w ll. W i fly i ; o il n , w o ll v ion o i .

Firstly, consider the ore general case where each job has an *individual* upper bound on the axi al resource consu ption, so $p_{js} = \bar{p}_j - b_j s_j$, and $0 \leq s_j \leq k_j$ for each job j. The proble discussed in this paper then corresponds to the special case where $k_j = k$ for all jobs j. It is not hard to see that our approxi ation result holds for that generalized version of the proble , too. Moreover, this generalized version does not ad it an approxi ation algorith with a perfor ance ratio better than 3/2, which follows by a si ple adaption of the gap-reduction fro PARTITION in Theore 3 of [8].

Secondly, our results can be generalized to proble s where the functions that describe the resource-ti e tradeoff are not necessarily linear, but polyno ial. Whenever the axi u degree of these polyno ials is bounded, our proofs can be adapted to that case as well.

Acknowledgements. We thank Gerhard Woeginger for several helpful suggestions. In particular, Gerhard pointed us to the paper [19], and proposed the proof for the FPTAS for the single achine scheduling proble in Le a 2. We also thank Frits Spieks a for so e helpful re arks.

References

1. J. Blazewicz, J. K. Lenstra and A. H. G. Rinnooy Kan, Scheduling subject to resource constraints: Classification and complexity, *Discr. Appl. Math.* **5** (1983), pp. 11–24.
2. Z.-L. Chen, Simultaneous Job Scheduling and Resource Allocation on Parallel Machines, *Ann. Oper. Res.* **129** (2004), pp. 135-153.
3. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completenes*, W. H. Freeman, New York, 1979.
4. R. L. Graham, Bounds for certain multiprocessing anomalies, *Bell System Technical Journal* **45** (1966), pp. 1563–1581. See also [5].
5. R. L. Graham, Bounds on multiprocessing timing anomalies, SIAM *J. Applied Math.* **17** (1969), pp. 416–429.
6. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey, *Ann. Discr. Math.* **5** (1979), pp. 287–326.
7. A. Grigoriev, H. Kellerer, and V. A. Strusevich, Scheduling parallel dedicated machines with the speeding-up resource, manuscript (2003). Extended abstract in: *Proceedings of the 6th Workshop on Models and Algorithms for Planning and Scheduling Problems*, Aussois, France, 2003, pp. 131–132.
8. A. Grigoriev, M. Sviridenko, and M. Uetz, Unrelated Parallel Machine Scheduling with Resource Dependent Processing Times, *Proceedings of the 11th Conference on Integer Programming and Combinatorial Optimization*, M. Jünger and V. Kaibel (eds.), Lecture Notes in Computer Science 3509, 2005, pp. 182-195.
9. R. Horst and P. M. Pardalos, Editors, *Handbook of Global Optimization*, volume 2 of Nonconvex Optimization and Its Applications, Springer, 1995.
10. K. Jansen, Scheduling Malleable Parallel Tasks: An Asymptotic Fully Polynomial Time Approximation Scheme, *Algorithmica* **39** (2004), pp. 59-81.

11. K. Jansen, M. Mastrolilli and R. Solis-Oba, Approximation Schemes for Job Shop Scheduling Problems with Controllable Processing Times, *European Journal of Operational Research* **167** (2005), pp. 297-319.
12. J. E. Kelley and M. R. Walker, *Critical path planning and scheduling: An introduction*, Mauchly Associates, Ambler (PA), 1959.
13. H. Kellerer and V. A. Strusevich, Scheduling parallel dedicated machines under a single non-shared resource, *Europ. J. Oper. Res.* **147** (2003), pp. 345–364.
14. H. Kellerer and V. A. Strusevich, Scheduling problems for parallel dedicated machines under multiple resource constraints, *Discr. Appl. Math.* **133** (2004), pp. 45–68.
15. J. K. Lenstra, D. B. Shmoys and E. Tardos, Approximation algorithms for scheduling unrelated parallel machines, *Math. Prog.* **46** (1990), pp. 259–271.
16. G. Mounie, C. Rapine, and D. Trystram, Efficient Approximation Algorithms for Scheduling Malleable Tasks, *Proceedings of the 11th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1999, pp. 23–32.
17. G. Mounie, C. Rapine, and D. Trystram, A 3/2-Dual Approximation Algorithm for Scheduling Independent Monotonic Malleable Tasks, Manuscript, Retrieved from `http://citeseer.csail.mit.edu/558879.html`
18. P. M. Pardalos and G. Schnitger, Checking Local Optimality in Constrained Quadratic Programming is NP-hard, *Oper. Res. Lett.* **7** (1988), pp. 33–35.
19. K. Pruhs and G. J. Woeginger, Approximation Schemes for a Class of Subset Selection Problems, *Proceedings of the 6th Latin American Symposium on Theoretical Informatics*, M. Farach-Colton (ed.), Lecture Notes in Computer Science 2976, Springer, 2004, pp. 203–211.
20. D. B. Shmoys and E. Tardos, An approximation algorithm for the generalized assignment problem, *Math. Prog.* **62** (1993), pp. 461–474.
21. M. Skutella, Approximation algorithms for the discrete time-cost tradeoff problem, *Math. Oper. Res.* **23** (1998), pp. 909–929.
22. J. Turek, J. L. Wolf, and P. S. Yu, Approximate Algorithms for Scheduling Parallelizable Tasks, *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1992, pp. 323–332.

Online Removable Square Packing

Xin Han[1], Kazuo Iwa a[1], and Guochuan Zhang[2,⋆]

[1] School of Informatics, Kyoto University, Kyoto 606-8501, Japan
{hanxin, iwama}@kuis.kyoto-u.ac.jp
[2] Department of Mathematics, Zhejiang University, China
zgc@zju.edu.cn

Abstract. The online removable square packing problem is a two dimensional version of the online removable Knapsack problem. For a sequence of squares with side length at most 1, we are requested to pack a subset of them into a unit square in an online fashion where the online player can decide whether to take the current square or not and which squares currently in the unit square to remove. The goal is to maximize the total packed area. Our results include: (i) Any online algorithm cannot achieve a better competitive ratio than $(\sqrt{5}+3)/2 \approx 2.618$. (ii) The matching upper bound is achieved by a relatively simple online algorithm if repacking is allowed. (iii) Without repacking, we can achieve an upper bound of 3 by using the concept of *bricks* by Januszewski and Lassak [11]. (iv) The offline version of the problem admits a PTAS.

1 Introduction

The Bin Packing and Knapsack proble s are both very popular in the field of co binatorial opti ization. However, the situation is quite different in their online versions: Bin Packing has a long history of online algorith s where i -portant notions like co petitive analysis already appeared, due to [8], in the early stage of the literature. In contrast, the Knapsack proble   has an intrinsic hostility against an online algorith   which has to decide, for each ite Q sequentially given, whether it takes Q or not, i.e., whether it puts Q into the *knapsack* (or we will call it a *bin*) or not. This decision is irrevocable, which cannot cope with the following si ple instance: Suppose that the online player receives a sequence of s all ite s of size ε. If the player does not take any of the , then the co petitive ratio gradually worsens and if the player takes one, then the adversary i   ediately gives an ite   of size 1 which cannot be taken because of the s all ite   already in the bin. Thus one can easily see that there are no co petitive algorith s.

Recently, Iwa a and Taketo i [9] bypassed this difficulty by introducing "re ovability." Na ely, in each step, the online player can also re ove one or   ore ite s currently in the bin other than deciding whether or not it takes the current ite . They considered the one-bin and two- or   ore-bin cases and showed that there exist opti al online algorith s for both cases. This is one of the successful atte pts to relax the online condition, which has been popular

⋆ Research supported by NSFC (10231060).

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 216–229, 2006.

in many fields such as scheduling (see e.g.,[15]) and even towards more general algorithmic paradigms such as *priority algorithms* [2].

This paper discusses the online square packing problem which can be regarded as a two-dimensional version of this online Knapsack. Suppose that a sequence of squares (a_i, a_i) arrives one by one, where $0 < a_i \leq 1$ is the side length of the square. In each step i, the online player has to decide whether or not it packs (a_i, a_i) into the bin of size $(1, 1)$ before the next square comes. We allow removing just as the one-dimensional case, namely we allow to discard one or more items already in the bin but the items once discarded will never be considered again. It is easy to see as before that there is no competitive algorithm without this discarding rule. It should also be noted (see Sec. 2) that if each item may be a rectangle, then we cannot achieve a constant competitive ratio, either.

Our contribution. The basic difference between the one-dimensional and two-dimensional packing problems is that in the latter we need to assign the item into a specific position inside the unit square. This also means that it is important whether we allow *repacking* in each step or not, where repacking means after deciding whether the current item is taken or not and which item(s) are removed, we can once take out all the items that should be packed and can reassign them into the bin using some (off-line) algorithm. In the one dimensional case, since items are naturally packed from the bottom of the bin without space, repacking is implicitly allowed. However, it turns out from the algorithm in [9] that we actually do not need repacking to obtain the optimal competitive ratio in the one-bin case.

Our results are the following: (i) Any online algorithm for which both removing and repacking are allowed cannot achieve any competitive ratio better than $(\sqrt{5}+3)/2$ (≈ 2.618). (ii) The matching upper bound can be achieved by a relatively simple but a bit tricky algorithm if we allow repacking. (iii) We give an online algorithm which achieves a competitive ratio of at most 3 without repacking. This algorithm is borrowing the interesting notion of *brick* by Januszewski and Lassak [11] together with a couple of new ideas for packing and removing. In particular, our new partition of the bin improves the main result in [11] as a byproduct. (iv) We also consider the offline version of the problem, which is known to be strongly NP-hard [13]. And we prove the offline version admits a PTAS.

Related problems and previous work. Basically there are two categories of rectangle packing problems. Let B be the set of rectangular bins and L be the set of rectangles to be packed. In the *maximization* category, one is asked to pack a subset X of L, without any overlap, into the bins so that $f(X)$ is maximized, where $f(\cdot)$ is a function of rectangles. In the *minimization* category, all rectangles of L have to be packed, without any overlap, into a subset of B so that $g(Y)$ is minimized, where $g(\cdot)$ is a function of bins. In this category, the set of bins is assumed to be large enough (e.g., there are unlimited number of bins). In fact we can also name the two categories the *knapsack class* and the *bin packing class*, respectively. For each category of rectangle packing we can define the off-line version and the online version.

For the off-line version of the maximization category, Caprara and Monaci [3] first considered the problem to maximize the total area of packed rectangles. They mainly focused on exact algorithms. A polynomial $(3+\varepsilon)$-approximation algorithm was also derived. Then Jansen and Zhang [10] considered a problem to maximize the total profit of packed rectangles. In the problem the bin set contains exactly one bin and each rectangle R_i is associated with a profit p_i. The objective function is $f(X) = \sum_{R_i \in X} p_i$. The problem is to pack a subset X of L into the bin to maximize $f(X)$. Several approximation algorithms were presented, the best of which has a worst-case ratio of at most $2+\varepsilon$ for any given $\varepsilon > 0$. Our result (iv) claims that there exists a PTAS if input items are unweighted (i.e., $p_i =$ its area) squares.

Some online problems of the maximization category were also investigated. Januszewski and Lassak [11] proposed a novel concept *brick*. They partitioned the unit square into bricks and pack different squares into appropriate bricks. They showed that each list of squares with total area bounded above by 5/16 can be online packed into a unit square (In fact, in their paper, the main result dealt with the d-dimensional problem. They showed that every sequence of d-dimensional cubes of total volume $2(1/2)^d$ can be online packed into a unit cube, for $d \geq 5$). However, their algorithm is not competitive against general inputs given in an online manner. Caramia et al. [4] designed an online algorithm to maximize the total area of rectangles packed into a rectangular bin, but only the experimental analysis based on implementation was given.

For the minimization category there have been many results on the two-dimensional bin packing problem in which all rectangles have to be packed into a minimum number of square bins. Here we only mention some results on packing squares. For the off-line case, Ferreira et al. [7] gave an approximation algorithm with asymptotic worst-case ratio bounded above by 1.988. Kohayakawa et al. [12] and Seiden and van Stee [16] independently obtained approximation algorithms with asymptotic worst-case ratio of at most $14/9+\varepsilon$ (for any $\varepsilon > 0$). These results were recently improved by Correa and Kenyon [5], and Bansal and Sviridenko [1]. They independently proposed asymptotic PTASes for packing d-dimensional cubes into the minimum number of unit cubes. For the online case, if the number of bins is bounded, the best known asymptotic worst case ratio is 2.271 [6].

Competitive Ratio. To evaluate an online algorithm, we use the standard measure called *competitive ratio*. For any input sequence L, let $A(L)$ be the area packed in the bin by an online algorithm A and $OPT(L)$ be the packed area by an optimal off-line algorithm. The *competitive ratio* of algorithm A is then defined as $R_A = \sup_L \frac{OPT(L)}{A(L)}$.

2 Lower Bounds

We first mention the impossibility of competitive algorithms.

Fact 1. *If the removal is not allowed, then any online algorithm cannot achieve a constant competitive ratio.*

Fact 2. *No algorithm can achieve a constant ratio for packing rectangles.*

Now we prove our ain lower-bound result:

Theorem 1. *For any online algorithm with removing and repacking allowed, its competitive ratio is at least* $(\sqrt{5}+3)/2$ (≈ 2.618).

Proof. Let A be any online algorith . The adversary gives the first and second squares whose sizes are (q^2, q^2) and $(q+\varepsilon, q+\varepsilon)$, respectively. Here, $q = (\sqrt{5}-1)/2$ (i.e., $q + q^2 = 1$) and $\varepsilon > 0$, and hence these two squares cannot coexist in the bin. If A takes (q^2, q^2) (and gives up $(q+\varepsilon, q+\varepsilon)$), then the ga e is over because $OPT(L)/A(L)$ for this input sequence L is $(q+\varepsilon)^2/q^4 > q^2/q^4 = 1/q^2 = (\sqrt{5}+3)/2$. So, suppose that $(q+\varepsilon, q+\varepsilon)$ is now in the bin. Then the third and forth squares given by the adversary are both (q^2, q^2). For the sa e reason as above, the algorith A ust discard both. Then the adversary gives four identical squares of size $(1/2, 1/2)$. There are two cases:

Case 1. Algorith A takes one of those $(1/2, 1/2)$ squares and discards $(q+\varepsilon, q+\varepsilon)$. Then the adversary stops the ga e i ediately and we have $A(L) = 1/4$ and $OPT(L) \geq 3q^4 + 1/4 > 0.6875$ since OPT can pack three (q^2, q^2) and one $(1/2, 1/2)$. Thus $OPT(L)/A(L) > 2.75$.

Case 2. Algorith A gives up all four $(1/2, 1/2)$'s. Then $OPT(L)$ is obviously 1 and $OPT(L)/A(L) = 1/(q+\varepsilon)^2$ which tends to $(\sqrt{5}+3)/2$ as ε goes to zero.

Thus, we have $R_A = \sup_L OPT(L)/A(L) \geq (\sqrt{5}+3)/2$.

3 Optimal Algorithm with Repacking

In this section, we give a si ple online algorith called *RPK*, which achieves the opti al co petitive ratio given in Theore 1. *RPK* uses the well-known (off-line) square packing algorith called *NFDH* (Next Fit Decreasing Height) [14]. We o it the details of *NFDH*, but it is enough to see Fig 1 (a) to understand its basic idea. Na ely, we sort the squares by their sizes and then pack the fro the largest one using level-1 area. If level-1 beco es full then we use level-2 and so on. Here is a key property of *NFDH*.

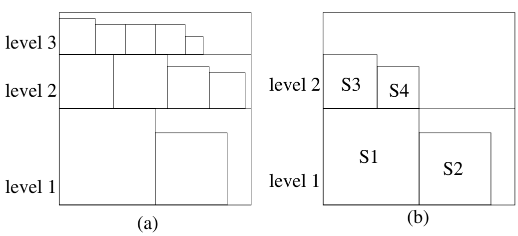


Fig. 1. NFDH packing

Lemma 1. [14], *Any set of squares with total area* $\leq 1/2$ *can be always packed into the unit square by NFDH.*

Now, a single round of our *RPK* can be described as follows. Note that $S_1, S_2, \cdots, S_n$ denotes the ite s currently in the bin whose side lengths are $x_1, x_2, \cdots, x_n$, respectively. W.l.o.g., we assu e that $x_1 \geq x_2 \geq \cdots \geq x_n$. Let Q be the current ite whose size is (x_{n+1}, x_{n+1}), and let $q = (\sqrt{5}-1)/2$.

1. If the packed area in the bin is at least q^2, then we discard Q.
2. Otherwise, if Q is large enough (that is, $x_{n+1} \geq q$), then we re ove everything in the bin and pack Q.
3. Else, if all of $S_1, S_2, \cdots, S_n$, and Q can be packed by *NFDH*, then do so.
4. Otherwise, if $(x_1 + x_{n+1}) > 1$ then we take the *smaller* one of S_1 and Q and pack it together with $S_2, \cdots, S_n$ by *NFDH.*
5. Otherwise, we find a axi u k such that $S_1, S_2, \cdots, S_k$ and Q can be packed by *NFDH* but $S_1, S_2, \cdots, S_k, S_{k+1}$ and Q cannot. Pack $S_1, S_2, \cdots, S_k$ and Q by *NFDH.*

Theorem 2. *The competitive ratio of RPK is at most* $(\sqrt{5}+3)/2$ (≈ 2.618).

Proof. Apparently the first square is taken by *RPK* and therefore the co - petitive ratio at the end of round 1 is 1. Suppose that the co petitive ratio OPT_{i-1}/RPK_{i-1} at the end of round $i-1$ ($i \geq 2$, or at the beginning of round i) is at ost $(3+\sqrt{5})/2$. Then we shall show that the co petitive ratio is also at ost $(3+\sqrt{5})/2$ at the end of round i. Recall that if $q = (\sqrt{5}-1)/2$, then $1/q^2 = (3+\sqrt{5})/2$.

In round i, one of the five steps $1 \sim 5$ above is executed.

Case 1. Step 1 or 2 is executed. This case is trivial since the total area is at least q^2.

Case 2. Step 3 is executed. The current ite Q is just added to the bin. So we have

$$\frac{OPT_i}{RPK_i} = \frac{OPT_{i-1} + |Q|}{RPK_{i-1} + |Q|} \leq \frac{OPT_{i-1}}{RPK_{i-1}} \leq (3+\sqrt{5})/2.$$

Case 3. Step 5 is executed (Step 4 will be considered next). We first show the following property of *NFDH.*

Lemma 2. *Suppose that a sorted sequence of squares* $S_1, S_2, \cdots, S_k$ *can be packed by* NFDH *but* $S_1, S_2, \cdots, S_k, S_{k+1}$ *cannot. Then* $k = 1$ *or* $k \geq 4$.

Proof. Suppose that $k \neq 1$. Then by assu ption we can pack at least S_1 and S_2. One can see that this packing needs only level 1. So, we ust have a space for S_3 above S_1 and a space for S_4 to the right of S_3, as shown in Fig. 1 (b). Thus we can pack at least up to S_4.

Now, suppose that $S_1, S_2, \cdots, S_k$ and Q can be packed but $S_1, S_2, \cdots, S_k, S_{k+1}$ and Q cannot. Since $S_1, S_2, \cdots, S_k$ and Q include at least two squares (recall that at least S_1 and Q can be packed), we have $k \geq 3$ by le a 2. Since the total a ount of area for $S_1, S_2, \cdots, S_n$ is less than q^2 (otherwise the packing should have been ended), the total area for $S_1, S_2, \cdots, S_k, S_{k+1}$ is also less than q^2. Since S_{k+1} is the s allest a ong $S_1, S_2, \cdots, S_k, S_{k+1}$ and $k \geq 3$, $|S_{k+1}| \leq q^2/4 < 0.1$. Moreover, because $S_1, S_2, \cdots, S_k, S_{k+1}$ and Q cannot be packed by *NFDH*, $|S_1| + \cdots + |S_{k+1}| + |Q| > 1/2$ by Le a 1. It then follows that $|S_1| + \cdots + |S_k| + |Q| \geq (1/2 - 0.1) > q^2$. Thus the packed area after this step is at least q^2.

Case 4. Step 4 is executed. Recall that if Step 1, 2, or 5 is once executed, then we have an enough packed area and we can stop packing. Also, in Step 3, nothing is discarded. Therefore, we can classify all the squares received so far into two groups G_1 and G_2 such that squares in G_1 are in the bin and squares in G_2 have been discarded only in Step 4. Thus we can prove the following facts:

(i) Let y be a side length of an arbitrary square Y in G_2. Then, $0.5 < y$ since it was not able to coexist with another single (s aller) square. Also $y < q$ since otherwise this single ite would be enough for the target ratio. Na ely $0.25 < |Y| < q^2$.

(ii) Let X be the largest ite in G_1. Then $|X| \geq q^4$. (The reason: Let Y be the square in G_2 which is discarded in this step. Then $|Y| < q^2$ by (i) and hence X and Y could coexist if $|X| < q^4$.)

(iii) This largest X cannot coexist with the s allest square, denoted by Y, in G_2. (The reason: Since Y was discarded, it was co pared with so e X' such that $|Y| \geq |X'|$. If $|X'| \leq |X|$ then we are done, so let us assu e $|X'| > |X|$. Since X' is not in G_1 now, it has to be oved to G_2 later. For this to occur, we eventually need another X'' such that $|X| \leq |X''| < |X'|$. If $|X| = |X''|$ then we are done since X'' (and also X) cannot coexist with X' or Y. Otherwise, X'' should be oved to G_2 but this violates the assu ption that Y is the s allest.)

Since any two squares in G_2 cannot coexist by (i). Also by (iii), OPT can take at ost one square say Y, in G_2 and $G_1 - \{X\}$, while RPK holds G_1. Therefore

$$\frac{OPT}{RPK} \leq \frac{G_1 - |X| + |Y|}{G_1 - |X| + |X|} \leq \frac{G_1 - |X| + q^2}{G_1 - |X| + q^4} \leq \frac{1}{q^2},$$

which was what we wanted to show.

4 Online Algorithm Without Repacking

In this section, we first review the on-line packing algorith by Januszewski and Lassak [11], called the JS algorith in this section, which uses a beautiful technique based on *bricks*. Unfortunately this algorith is not co petitive but it guarantees a certain a ount of total packed area. We then give our new ideas that ake this algorith both ore efficient and co petitive. A detailed description of our algorith and its analysis follow.

4.1 Packing by Using Bricks

A rectangle (w, h) is called a *brick* if $w/h = \sqrt{2}$ or $h/w = \sqrt{2}$. A brick has the following two i portant properties:

Fact 3. *If (w, h) is a brick, then either $(w/2, h)$ or $(w, h/2)$ is a brick. In other words, a brick can be partitioned into two congruent bricks.*

Fact 4. *If a square Q fits in a brick (w, h) $(w > h)$ but not in $(w/2, h)$, then $wh/2\sqrt{2} < |Q| \leq wh$.*

As shown in Fig 2, brick $B(w, h)$ can be continuously partitioned into s aller bricks, so eti es called subbricks $(w/2, h), (w/2, h/2), (w/4, h/2)$, and so on. For a square $Q = (a, a)$, we use $S(Q)$ to denote a brick which "just" fits for the square Q. More precisely, $S(Q) = (w/2^i, h/2^i)$ if $w/2^{i+1} < a \leq h/2^i$, or $S(Q) = (w/2^{i+1}, h/2^i)$ if $h/2^{i+1} < a \leq w/2^{i+1}$, for so e integer i. $|B|$ denotes the area of brick B. If a brick contains a square, then it is said to be *used*, otherwise *unused*.

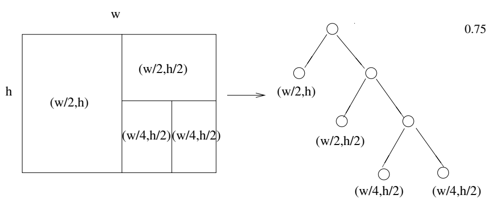


Fig. 2. partitioning bricks

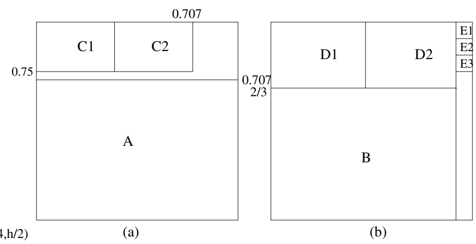


Fig. 3. Previous partition vs ours

Suppose that we are given a square Q and a brick B which ay be partitioned into subbricks and so e of the ay be already used. The JS algorith packs Q into the "right position" of B by the following subroutine:

Algorithm PB (Packing a Brick).

1. If all bricks in B are used or all unused bricks are s aller than $S(Q)$, then give up packing Q,
2. else pack Q into B depending on the following two cases.
   (a) If there is an unused brick congruent to $S(Q)$, then pack Q into it,
   (b) else find a s allest brick a ong all the unused ones whose area is larger than $|S(Q)|$. Denote such a brick by P, and partition P into a sequence of bricks whose areas are $\frac{|P|}{2}, \frac{|P|}{4}, \cdots, 2|S(Q)|, |S(Q)|, |S(Q)|$, respectively. Pack Q into an arbitrary one of the last two bricks whose area is $|S(Q)|$.

The entire JS algorith is quite si ple, which packs a sequence of ite s $Q_1, Q_2, \cdots, Q_n$ as follows: (i) Before packing Q_1, construct three bricks $A = (1, \sqrt{2}/2)$, $C_1 = (\sqrt{2}/4, 1/4)$ and $C_2 = (\sqrt{2}/4, 1/4)$ within the bin (a unit square) as shown in Fig. 3 (a). (ii) For packing current ite Q_i, pick up the bricks one by one in the order of C_1, C_2 and A and apply algorith PB, respectively. Once Q_i is packed, consider the next ite ; otherwise, if Q_i can be packed into neither of the , stop with failure. (iii) If we can pack all of $Q_1, \cdots, Q_n$, then stop with success.

Proposition 1 [11]. *If* $|Q_1|+\cdots+|Q_n| \leq 5/16$*, then the above algorithm always stops with success.*

4.2 New Ideas

(i) We construct six bricks in the bin as shown in Fig. 3 (b), $B = (2\sqrt{2}/3, 2/3)$, $D_1 = D_2 = (\sqrt{2}/3, 1/3)$ and $E_1 = E_2 = E_3 = (x, \sqrt{2}x)$, where $x = 1 - 2\sqrt{2}/3$. (ii) Recall that each item is a square but a brick is a rectangle. Hence a brick holding an item must have a space. The original JS algorithm never uses such a space, but our algorithm does, namely, two or more items may share a single brick.

The modification (i) is quite powerful; we can prove the following theorem.

Theorem 3. *By the modification (i), the JS algorithm can pack the items whose total area is up to 1/3.*

4.3 Competitive Algorithm

The basic strategy of our algorithm denoted by *RSP* (Removable Square Packing), is as follows: Suppose that the coming item Q is *large* (to be defined later). Then if it can be packed in the bin together with other *large* items, then we pack it without using the concept of *brick*, else we remove some *small* items to make a space for Q. On the other hand, suppose that the coming item is *small*. Then if there is a *large* item P in the bin, then we first try to append Q into the *brick* now being used by P. If there is no such a space in that *brick*, then we will remove some *small* items relatively smaller than Q.

A square (a, a) is called *small*, if $a \leq 1/3$, otherwise *large*. Moreover, if $a \leq 1 - 2\sqrt{2}/3$, we call it *tiny*. As mentioned before, the bin is divided into six bricks (Fig. 3 (b)) at the beginning. In the course of the algorithm, an item is packed into some brick and a brick is partitioned into smaller ones if necessary. At any stage there are two kinds of bricks, one which has not been partitioned yet is called t-brick, the other is called n-brick. For example, before packing any square, all bricks $B, D_1, D_2, E_1, E_2, E_3$ are t-bricks. In order to pack a square Q, we may divide brick B into two sub-bricks B_1 and B_2 and use one of them to pack Q. At this moment, B_1 and B_2 are t-bricks, one is a used t-brick, the other is an unused t-brick, but B became an n-brick.

In each round, algorithm RSP receives an input item Q and decides: (i) whether or not Q should be packed, (ii) if yes, which position Q should take, and (iii) to do so, which items should be removed. It should be noted that if the current amount of total packed area in the bin is 1/3 or more, then our target competitive ratio (= 3) is already achieved. So the algorithm can ignore (automatically discard) all the following items. For example, if $B \cup D_1 \cup D_2$ do not include any unused (sub)brick and if every used (sub)brick just fits its holding item then the packed area is greater than 1/3. (To see this, note that the total area of $B \cup D_1 \cup D_2$ is greater than $2\sqrt{2}/3$. Then by Fact 4, the packed area is greater than $(2\sqrt{2}/3)(1/2\sqrt{2})$=1/3.)

Now here is a detailed description of each round of *RSP*, which consists of three steps:

Step 1. The packing stops when one of the following two cases occurs.

(1.1) The total area of the squares already packed is at least $1/3$.
(1.2) The current ite Q is "very" large, Then e pty the bin and pack Q.

Step 2. Pick up the bricks one by one in the order of $E_1, E_2, E_3, D_1, D_2, B$ and apply algorith PB to pack ite Q, respectively. Otherwise (Q cannot be packed into any brick), goto step 3.

Step 3. There exist unused bricks and all of the are s aller than $S(Q)$.

- $|Q| \leq 1/9$.
  (3.1) There is one *t-brick* of area $2^i|S(Q)|$ in $D_1 \cup D_2 \cup B$ for so e $i \geq 2$. (Such a *t-brick* is used, but as shown later, there ust be roo for the square Q within that *t-brick*). Then pack Q into it and halt.
  (3.2) The largest *t-brick* in $D_1 \cup D_2 \cup B$ is of area $2|S(Q)|$. Then if Q can be packed into it, then pack Q; else select an *n-brick* whose packed area is the s allest a ong all *n-bricks* congruent to $S(Q)$ in $D_1 \cup D_2 \cup B$. E pty this brick and pack Q in it. Halt.
  (3.3) The largest *t-brick* in $D_1 \cup D_2 \cup B$ is not larger than $S(Q)$ Then find an *n-brick* P which is congruent to $S(Q)$ and contains the largest unused brick in $D_1 \cup D_2 \cup B$ (such a brick can be deter ined with the help of the binary tree as shown in Fig. 2). Re ove all squares fro P and pack Q into it.
- $2/9 < |Q| < 1/3$.
  (3.4) There is at least one s all square in B. Then re ove all squares fro B, except a s all one at a corner of B. Pack Q into B and halt.
  (3.5) There is exactly one large square and no s all ones in B. Then pack Q in the rest space of the unit square, if Q can be packed, and stop. Otherwise, re ove the larger one of Q and the large square already in B, and pack the s aller into B.
  (3.6) There are two large squares in B. Then keep the s aller one, re ove the larger as well as all s all squares if needed, then pack Q in the re aining space of the bin. Halt.
- $1/9 < |Q| \leq 2/9$
  (3.7) There are two large squares in B. Then re ove all squares fro $D_1 \cup D_2$ if needed, then pack Q in the re aining space of the bin. Halt.
  (3.8) There is only one square in B and its area is greater than $2/9$ (B is a t-brick). Then if Q can be packed to the re aining space of the square bin, then pack Q (Q ay go beyond the borders of bricks) and stop. Otherwise, re ove the square in B and pack Q (at this point, we also get a new *t-brick* $(\sqrt{2}/3, 2/3)$ since B is partitioned).
  (3.9) Else, there are two subcases.
    1. If there is one large square, then re ove all *small* squares fro B and pack Q.
    2. There are two *n-bricks* of $(\sqrt{2}/3, 2/3)$. E pty the one whose packed area is s aller then pack Q.

4.4 Analysis of Competitive Ratio

The packing ends either by the stopping rules of algorith *RSP* or by the instance itself. Note that in Step 2, we never re ove any squares, i.e, our packing is the sa e as opti al packing, so our analysis only focuses on Step 1,3. If *RSP* stops at Step 1 for so e round, the co petitive ratio is obviously achieved. Therefore, to analyze the algorith , we only need to consider Step 3. We first show that if *RSP* stops packing, the packed area of the bin is at least 1/3. Then we prove, for other cases, the co petitive ratio is at ost 3.

The following two le as 3 and 4 co e fro [11], which are useful for the analysis of our algorith . Let B be a brick. Recall that we use algoriht *PB* for packing a square Q into a brick B and $S(Q)$ denotes the brick which just fits Q.

Lemma 3. *If algorithm PB cannot pack a square Q, then all unused bricks in B are smaller than $S(Q)$, and there is at most one unused brick of area $|S(Q)|/2^i$ for each $i = 1, 2, \ldots$.*

Lemma 4. *If PB algorithm cannot pack Q, then the total area of used bricks is at least $|B| - |S(Q)|$.*

Lemma 5. *Given a brick A, any two squares with a total area of at most $|A|/\sqrt{2}$ can be packed together into A.*

Lemma 6. *If there is a tiny square in $D_1 \cup D_2 \cup B$, then the packed area in $E_1 \cup E_2 \cup E_3$ is greater than x^2, where $x = 1 - 2\sqrt{2}/3$.*

Proof. In this case, there is at least one *tiny* square in E_3. Otherwise the tiny square in $D_1 \cup D_2 \cup B$ would have been packed into E_3 by the algorith . Analogously, any square in E_3 can not be packed into $E_1 \cup E_2$. By Le a 4, the packed area in $E_1 \cup E_2 \cup E_3$ is greater than $2x\sqrt{2}x/(2\sqrt{2}) = x^2$.

Let m be the counter that the execution pass through Case (3.3),

Lemma 7. *In Case (3.3), the packed area in $D_1 \cup D_2 \cup B$ is greater than $1/3 - 2^{-m}/18$, the total removed area in Case (3.3) is at most $1/18 + 2^{-m}/36$. Moreover, if $m = 5$, the packed area in the unit square is greater than $1/3$.*

Proof. Since Q is a s all square, $S(Q)$ is not larger than $(\sqrt{2}/3, 1/3)$. In Case (3.3), the largest one of all t-bricks in $D_1 \cup D_2 \cup B$ is not larger than $S(Q)$. If there is no *n-brick* congruent to $S(Q)$ in $D_1 \cup D_2 \cup B$, then all t-bricks in $D_1 \cup D_2 \cup B$ are congruent to $S(Q)$. Further ore the area packed in $D_1 \cup D_2 \cup B$ is at least 1/3, which causes a contradiction. Hence, there is at least one *n-brick* congruent to $S(Q)$ in $D_1 \cup D_2 \cup B$.

By the algorith , we always pick the *n-brick* containing the largest unused brick, and further re ove all squares fro it (it beco es a *t-brick*) and pack square Q into it. Note that since Q is a s all square, the nu ber of unused bricks $S(Q)/2^i$ in $D_1 \cup D_2 \cup B$ is at ost 1 by Le a 3, for each $i \geq 1$, and the largest one of all unused bricks in $D_1 \cup D_2 \cup B$ is not larger than $(\sqrt{2}/6, 1/3)$

(whose area is $\sqrt{2}/18$), when $m = 0$. After Case (3.3) occurs m ti es, the area of the largest one of all unused bricks is not greater than $(\sqrt{2}/18) \cdot 2^{-m}$ and the total area of the unused bricks in $D_1 \cup D_2 \cup B$ is less than $(\sqrt{2}/9) \cdot 2^{-m}$, since the nu ber of every kind of unused brick is at ost one. Hence, the packed area in $D_1 \cup D_2 \cup B$ is greater than

$$(\frac{2\sqrt{2}}{3} - \frac{\sqrt{2}}{9} \cdot \frac{1}{2^m}) \cdot \frac{1}{2\sqrt{2}} = \frac{1}{3} - \frac{1}{18} \cdot 2^{-m}.$$

Next, we esti ate the total area which has been re oved fro the bin, when we pack the current square Q. Recall that the area of the largest one of unused bricks in $D_1 \cup D_2 \cup B$ is not greater than $(\sqrt{2}/18) \cdot 2^{-m}$. Except the largest one, the total area of unused bricks in $D_1 \cup D_2 \cup B$ is less than $(\sqrt{2}/18) \cdot 2^{-m}$. By the algorith , the selected *n-brick* is e ptied and its area is equal to $|S(Q)|$. By Le a 4 and Fact 4, except squares in that re oved *n-brick*, the total area of squares in $D_1 \cup D_2 \cup B$ is at least $(|B \cup D_1 \cup D_2| - |S(Q)| - (\sqrt{2}/18) \cdot 2^{-m})/2\sqrt{2} = 1/3 - |S(Q)|/2\sqrt{2} - 2^{-m}/36$. Note that before re oving the old squares, the packed area in the bin is less than 1/3 (otherwise the packing stops). Then the area packed in that *n-brick*, i.e., the area that we re oved is less than $|S(Q)|/2\sqrt{2} + 2^{-m}/36 \leq 1/18 + 2^{-m}/36$, since $|S(Q)| \leq \sqrt{2}/9$.

If $m = 5$, the largest one of all unused bricks in $D_1 \cup D_2 \cup B$ is not larger than brick $(\sqrt{2}/24, 1/24)$. Since $1/24 < 1 - 2\sqrt{2}/3$, there ust exist a tiny square in $D_1 \cup D_2 \cup B$. By Le a 6, the packed area in $E_1 \cup E_2 \cup E_3$ is greater than x^2. Therefore, the packed area in the square bin is greater than $1/3 - 2^{-5}/18 + x^2 > 1/3$.

Lemma 8. *If algorithm RSP stops packing, then the area packed in the unit square is at least* $\frac{1}{3}$.

Proof. Let Q be the last square i ediately before the packing is ter inated. We will consider all the cases in which the packing is stopped. It is not difficult to see that in Cases (3.5) and (3.8), when the packing stops, the total area of two large squares in the bin is greater than 1/3.

Case (3.1). Before packing Q, by Le a 4, the area of all packed bricks in $D_1 \cup D_2 \cup B$ is at least $2\sqrt{2}/3 - |S(Q)|$. Except the *t-brick* of area $2^i|S(Q)|$, the area of all packed bricks is at least $2\sqrt{2}/3 - |S(Q)| - 2^i|S(Q)|$. So, except the square S in that *t-brick*, the packed area in the bin is at least $1/3 - |S(Q)|/(2\sqrt{2}) - 2^i|S(Q)|/(2\sqrt{2})$, where $i \geq 2$. Since before packing Q, the packing does not stop, the packed area in the bin is less than 1/3, eaning $|S| < (|S(Q)| + 2^i|S(Q)|)/(2\sqrt{2})$. Since $|Q| \leq |S(Q)|/\sqrt{2}$, we have $|S| + |Q| < 2^i|S(Q)|/\sqrt{2}$ ($i \geq 2$). By Le a 5, S and Q can be packed together in the *t-brick* of area $2^i|S(Q)|$. After packing Q, the packed area is at least 1/3.

Case (3.2). Before packing Q, the packed area is at least $(2\sqrt{2}/3 - |S(Q)|)/(2\sqrt{2})$ by Le a 4 and Fact 4. If Q can be packed in the *t-brick* of area $2|S(Q)|$, then after packing Q, the area packed in the square bin is at least 1/3. Otherwise, by Le a 5, we have $|P| + |Q| > 2|S(Q)|/\sqrt{2}$, where P is the square in that *t-brick*. Except P and Q, by Lemma 4 and Fact 4, the area packed in the bin is at least $1/3 - |S(Q)|/(2\sqrt{2}) - 2|S(Q)|/(2\sqrt{2})$. If the packed area in any *n-brick* congruent to $S(Q)$ is not less than $|S(Q)|/(2\sqrt{2})$, then the total area packed in the bin is greater than $1/3$. Because we pick the one whose packed area is the smallest among all *n-bricks* congruent to $S(Q)$, the area packed in that *n-brick* is less than $|S(Q)|/(2\sqrt{2})$. So, after removing all squares in that *n-brick* and packing Q into it, the total area packed in the bin is greater than

$$(\frac{1}{3} - \frac{|S(Q)|}{2\sqrt{2}} - \frac{2|S(Q)|}{2\sqrt{2}}) + \frac{2|S(Q)|}{\sqrt{2}} - \frac{|S(Q)|}{2\sqrt{2}} = \frac{1}{3}.$$

Case (3.4). By using the techniques mentioned in Subsection 4.3, we can guarantee that at any moment, if there are small squares in brick B, then at least one of four corners of brick B is occupied by a small square. Moreover, it is possible to pack Q together with such a small square into B according to Lemma 5. By Lemma 4 and Fact 4, the total area of the squares in $D_1 \cup D_2$ and the small square in B is at least $1/9$. Hence, the packed area in the square bin is greater than $1/9 + 2/9 = 1/3$.
Case (3.6). The area of the smaller one of the two large squares must be smaller than $1/6$, and that smaller square is in the bottom of the bin. Since the side length of the current item Q is less than $\sqrt{3}/3 < (1 - \sqrt{6}/6)$, Q can be packed in the remaining space of the bin. After packing Q, the packed area in the bin is at least $1/3$.
Case (3.7). This case is similar with Case (3.6). After packing Q, the area in the bin is at least $1/3$.

Lemma 9. *If there are no more squares coming, the competitive ratio of RSP is at most* 3.

Proof. **Case** (3.5) and **Case** (3.8). In both cases, B contains exactly one large square before consider Q. It shows that Cases (3.1)-(3.3) and (3.9) have not yet occurred. In other words, no *small* squares have been removed so far. Note that the large square in B cannot be packed together with Q. Recall that the algorithm removes the larger one, whose area is greater than $1/4$ but smaller than $1/3$, while the area of the packed one is greater than $(1 - \sqrt{3}/3)^2 > 1/6$. If the packing stops after this step, the packed area is greater than $z + 1/6$, where z is the total area of all the *small* squares in the instance. Note that when Case (3.5) or Case (3.8) occurs, the removed one is always larger. It means that any two large squares cannot be packed together in the bin and the larger square has an area less than $1/3$. Then the packed area by an optimal algorithm is less than $z + 1/3$. It follows that the competitive ratio of algorithm RSP is less than 2 in this case.
Case (3.9). In this case, there are some *small* squares in B, before considering Q. By Lemma 4 and Fact 4, the total area of the squares in $D_1 \cup D_2$ is greater than $1/18$. And, there are two subcases. Let z be the total area of the squares in the bin and let y be the total area of all squares removed in Case (3.3).

First consider the case of one large square in B. If Case (3.5) or Case (3.8) has occurred before, the area of the large square in B is at least $1/6$. After packing Q, the packed area will exceed $1/3$ $(= 1/18+1/6+1/9)$. If neither Case (3.5) nor Case (3.8) has occurred, we have not re oved any large square so far. After packing Q, the packed area in the bin is at least $1/3-2^{-m}/18$. Obviously, $z \geq 1/3 - 2^{-m}/18$. By Le a 7, if $m > 0$, then $y \leq \sum_{i=0}^{m-1}(\frac{1}{18} + \frac{1}{36} \cdot 2^{-i})$. Otherwise $y = 0$. We can assu e that $m \leq 4$. The reason is that if $m \geq 5$, by Le a 7, the packed area is greater than $1/3$ and the packing would have stopped at Case (1.1).

Note that each subcase of Case (3.9) occurs at ost once. Recall that the packed area in $D_1 \cup D_2$ is at least $1/18$. In the first subcase of Case (3.9), the area of the squares re oved fro the *n-brick* is less than $1/6$, since otherwise the packed area is over $1/3$. In the second subcase, the total area packed in brick B is less than $5/18$. Then the area of the squares re oved is less than $5/36$. With calculations for each $0 < m \leq 4$, the co petitive ratio is less than $(z + y + 5/36 + 1/6)/z < 3$.

Now let us consider the latter case of Case (3.9), i.e., there are only *small* squares in B. Then neither of Cases (3.5), (3.8) and the first subcase of Case (3.9) has occurred. Analogously as the first subcase, it can be shown that the co petitive ratio is at ost $(z + y + 5/36)/z \leq 3$.

Case (3.3). In this case, we clai that neither of Cases (3.5), (3.8) and (3.9) has occurred so far. If it is not the case, then there is at least one *large* square in the bin, and the packing would have stopped in Case (3.1) or Case (3.2). Therefore, no *large* squares have been considered so far. As in the proof for Case (3.9), let z be the total area of the squares in the bin and y be the total area of all squares re oved. Then $z \geq 1/3 - 2^{-m}/18$ and $y \leq \sum_{i=0}^{m-1}(1/18 + 2^{-i}/36)$, for $m \leq 4$. The co petitive ratio is at ost $(z + y)/z < 3$.

The following theore i ediately follows fro le as 8 and 9.

Theorem 4. *The competitive ratio of RSP is* 3.

5 A PTAS for Offline Packing

The proble is, given a set S of n squares of side length at ost one, how to pack a subset of S into a fixed rectangle of size $(1 \times h)$ so that the packed area beco es axi u (previously h was 1, which can be generalized). The algorith is based on the sa e idea as [5]: (i) Select an integer k such that $k > \delta(1 + h)$, which is associated with the error bound of the PTAS. (ii) The region (0,1] is divided into $k + 2$ sub-intervals, $R_0, R_1, \cdots, R_{k+1}$, where $R_i = (P_i, P_{i-1}]$, $R_0 = (P_0, 1]$ and $R_{k+1} = (0, P_k]$ and $P_i = k^{-3^i}$ and $P_0 = 1/k$, $1 \leq i \leq k$. Deco pose S into $S_0, \cdots, S_{k+1}$ such that $Q \in S_i$ if and only if its side length is in R_i. $|S_i|$ denotes the total area of squares in S_i. (iii) Pack all squares in $S - S_0$ by *NFDH*. If there re ain one or ore squares unpacked, then output that packing. (iv) Otherwise, i.e., if all squares in $S - S_0$ are packed, then find an index i such that $|S_i| = min\{|S_1|, \cdots, |S_{k+1}|\}$. Let $X_l = S_0 \cup \cdots \cup S_{i-1}$ and $X_s = S_{i+1} \cup \cdots \cup S_{k+1}$. (Here it is i portant to re ove S_i, which akes a "gap" between large and s all ite s.) (v) Obtain an opti al packing for the "large" ite s in X_l using the exhaustive ethod [5]. The unpacked region in the bin can be deco posed into a li ited nu ber of rectangles. Then the "s all" ite s in X_s are packed into those rectangles (in an arbitrary order) by *NFDH*.

Theorem 5. *The worst case ratio of the algorithm is* $(1 + 6(1 + h)/k)$. *(The proof is similar to [5] and may be omitted.)*

6 Concluding Remarks

As entioned earlier, it is still open if we can achieve the bound of Theore 1 without using repacking. Our algorith RSP borrows the concept of brick. If we stay on this line, then it see s hard to obtain a better bound than $2\sqrt{2}$. Extending to online rectangle packing with reasonable restrictions should also be nice future work.

References

1. N. Bansal and M. Sviridenko, New approximability and inapproximability results for 2-dimensional bin packing, SODA 2004, 189-196.
2. A. Borodin, M. N. Nielsen, C. Rackoff: (Incremental) Priority agorithms. Algorithmica 2003,37(4): 295-326.
3. A. Caprara and M. Monaci, On the two-dimensional knapsack problem, Operations Research Letters 2004, 32: 5-14.
4. M. Caramia, S. Giordan, and A. Iovanella, An on-line algorithm for the rectangle packing problem with rejection, Proc. WEA 2003, LNCS 2647, 2003, pp. 59-69.
5. J.R. Correa and C. Kenyon, Approximation schemes for multidimensional packing, SODA, 179-188, 2004.
6. L. Epstein and R. van Stee, Optimal online bounded space multidimensional packing, SODA 2004, 207-216.
7. C.E. Ferreira, E.K. Miyazawa, and Y. Wakabayashi, Packing squares into squares, Pesquisa Operacional, 1999, 19: 223-237.
8. A.Fiat and G.J.Woeginger, Online Algorithms, LNCS 1442 1998.
9. K. Iwama and S. Taketomi, Removable online knapsack problems, Proc. ICALP2002, LNCS 2380, pp. 293-305.
10. K. Jansen and G. Zhang, On rectangle packing: maximizing benefits, SODA 2004, 197-206.
11. J. Januszewski and M. Lassak, On-line packing sequences of cubes in the unit cube, Geometriae Dedicata 1997, 67: 285-293.
12. Y. Kohayakawa, F.K. Miyazawa, P. Raghavan, and Y. Wakabayashi, Multidimensional cube packing, Algorithmica, 40(3) 173-187,2004.
13. J.Y.-T. Leung, T.W. Tam, C.S. Wong, G.H. Young, and F.Y.L. Chin, Packing squares into a square, J. Parallel Distrib. Comput. 1990, 10: 271-275.
14. A.Meir and L. Moser, On packing of squares and cubes, Journal of combinatorial theory, 1968, 5: 126-134.
15. P. Sanders, N. Sivadasan, M. Skutella, Online Scheduling with Bounded Migration, ICALP 2004,1111-1122 .
16. S.S. Seiden and R. van Stee, New bounds for multidimensional packing, Algorithmica 2003, 36: 261-293.

The Online Target Date Assignment Problem*

S. Heinz[1], S.O. Kru ke[2], N. Megow[3], J. Ra bau[4],
A. Tuchscherer[1], and T. Vredeveld[5]

[1] Konrad-Zuse-Zentrum für Informationstechnik Berlin, Department Optimization,
Takustr. 7, 14195 Berlin, Germany
{heinz, tuchscherer}@zib.de

[2] University of Kaiserslautern, Department of Mathematics,
P.O. Box 3049, Paul-Ehrlich-Str. 14, 67653 Kaiserslautern, Germany
krumke@mathematik.uni-kl.de

[3] Technische Universität Berlin, Institut für Mathematik,
Strasse des 17. Juni 136, 10623 Berlin, Germany
nmegow@math.tu-berlin.de

[4] Universität Bayreuth, Lehrstuhl für Wirtschaftsmathematik,
95440 Bayreuth, Germany
Joerg.Rambau@uni-bayreuth.de

[5] Maastricht University, Department of Quantitative Economics,
P.O. Box 616, 6200 MD Maastricht, The Netherlands
T.Vredeveld@KE.unimaas.nl

Abstract. Many online problems encountered in real-life involve a two-stage decision process: upon arrival of a new request, an irrevocable first-stage decision (the assignment of a specific resource to the request) must be made immediately, while in a second stage process, certain "subinstances" (that is, the instances of all requests assigned to a particular resource) can be solved to optimality (offline) later.

We introduce the novel concept of an *Online Target Date Assignment Problem* (OnlineTDAP) as a general framework for online problems with this nature. Requests for the OnlineTDAP become known at certain dates. An online algorithm has to assign a target date to each request, specifying on which date the request should be processed (e. g., an appointment with a customer for a washing machine repair). The cost at a target date is given by the *downstream cost*, the optimal cost of processing all requests at that date w. r. t. some fixed downstream offline optimization problem (e. g., the cost of an optimal dispatch for service technicians). We provide general competitive algorithms for the OnlineTDAP independently of the particular downstream problem, when the overall objective is to minimize either the sum or the maximum of all downstream costs. As the first basic examples, we analyze the competitive ratios of our algorithms for the particular academic downstream problems of bin-packing, nonpreemptive scheduling on identical parallel machines, and routing a traveling salesman.

* Supported by the DFG Research Center Matheon *Mathematics for key technologies* in Berlin.

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 230–243, 2006.

1 Introduction

Many real-world online problems exhibit a two-stage structure. In a first stage, an immediate online action has to be taken, while in a second stage "certain offline subproblems" (which we will refer to as downstream optimization problems) can be solved to optimality offline. In this paper we provide a general framework for online problems of this type, the *Online Target Date Assignment Problem* (OnlineTDAP).

As an illustration, consider the following scenario arising in the dispatching of service technicians. When a customer calls in, requesting a maintenance service for his washing machine, one of the service technicians has to visit the customer at its location and fix the problem. This service can be done within a certain time frame, say within a week. The customer must be given the day (and possibly a more narrow time window) when the technician will arrive, while he is on the phone and without knowledge of future service requests, that is, it must be given online. However, until the promised service day arrives, the decision which service technician to send and in which order the customers should be visited can be safely deferred. In other words, the exact scheduling and routing of service technicians for a fixed day can be done optimally offline at the night before.

In this paper, we introduce structures that account for the following dichotomy in many day-to-day resource dispatching problems: First, a resource has to be assigned to a request (e. g., assign a service vehicle to a repair request) and then the processing of all requests assigned to a certain resource can be optimized (find an optimal tour for each service vehicle). The assignment decisions influence the overall cost because they determine the input and thus the optimal costs of the single resource dispatching problems, the *downstream optimization problems.*

Offline, both stages can be integrated to obtain an overall optimal solution, even in many practical applications. However, if for each request the first decision, i.e., the assignment decision, has to be made online, the situation changes: the resulting problem is not offline anymore, but it is neither just the online version of the integrated dispatching problem; it is something in between. In stochastic programming the optimal decisions of a second stage optimization are called a *recourse.* In a way, in this paper we introduce *competitive analysis with recourse.*

Our object of study can be seen as the most extreme distinction between the online requirement of the first decision and the downstream optimization: We present a model where the first decision has to be made immediately and irrevocably before the next request is revealed (no knowledge about the input), while the downstream optimization can be carried out offline (complete knowledge about the input). The resource that has to be assigned to requests in our main actor, the OnlineTDAP, is a target date, a date at which the service should take place.

There are many variants conceivable of this concept: if the current day is allowed as a target date then the downstream optimization becomes an online problem as well, although a large portion of the data is known before the target date. It is also possible to relax the online requirement of the assignment decision: all requests on a single day might be collected, and the target dates are chosen and communicated at the end of the day. And there are, of course, variants where resources other than dates have to be assigned online (machines, vehicles) before a single resource offline problem has to be solved.

Problems of this type are abundant in reality, and very often the first decision is online. There is, however, almost no theoretical background published on this topic for the case where no stochastic information about future requests is available. And many of the stochastic models, e. g., Markov Decision Processes [5], cannot be solved for practical problem sizes. Therefore we feel that the investigation of the most basic structures in such problems seems adequate. Thus, we get started in this paper by investigating competitive online algorithms for the OnlineTDAP w. r. t. to classical downstream problems.

We think that the introduction of the OnlineTDAP will foster various lines of research, e. g., dealing with competitive analysis for OnlineTDAP w. r. t. various other, maybe more sophisticated downstream problems, with variants of the OnlineTDAP itself, but also with decision support methods for variants of the OnlineTDAP outside competitive analysis.

Problem description. An instance of the OnlineTDAP consists of a sequence of requests $\sigma = r_1, r_2, \ldots$ and a *downstream problem* Π, an offline optimization problem for which arbitrary subsets of σ are feasible inputs.

Each request r_i has an integral release date $t(r_i)$ and must be assigned immediately and irrevocably to a target date in the time period $t(r_i)+1, \ldots, t(r_i)+\delta(r_i)$, where $\delta(r_i)$ is the allowed time for deferring the service of request r_i (one week in our service technician scenario), which is also revealed upon arrival of the request. In this paper we consider only the case of uniform deferral times, that is, $\delta(r_i) = \delta$ for all requests r_i, where $1 \leq \delta < +\infty$. For an algorithm Alg we denote the particular date to which request r_i is assigned by $\text{Alg}[r_i] \in \{t(r_i)+1, \ldots, t(r_i)+\delta\}$.

A solution for an OnlineTDAP w. r. t. to downstream problem Π is feasible if

- each request is assigned to a feasible target date, and
- for each single target date, the corresponding instance of Π is feasible, too.

Let σ_d be the subset of requests assigned to date d by an online algorithm Alg. The optimal cost of Π on σ_d is called *downstream cost of* Alg *at date d*, and we denote it by $\text{downcost}(\sigma_d)$.

The overall online cost $\text{Alg}(\sigma)$ of an online algorithm Alg is defined as either the sum of the incurred downstream costs over all dates (min-total problems), or the maximum of the incurred downstream costs over all dates (min-max problems). The goal is to find online algorithms whose *competitive ratios* are as small as possible. An online algorithm Alg is called *c-competitive* if the cost of Alg is never larger than c times the cost of an optimal offline solution. The *competitive ratio* of Alg is the infimum over all $c \geq 1$ such Alg is c-competitive [2].

Our results. The OnlineTDAP provides a general framework for a large class of online problems and gives a novel view on online optimization. We provide

Table 1. Main bounds on the competitive ratio of best possible deterministic online algorithms for the OnlineTDAP with a certain downstream problem minimizing the total or maximum downstream cost

downstream problem	lower bound	upper bound
bin-packing	3/2	2
scheduling	$\sqrt{2}$	2
traveling salesman	$\sqrt{2}$	2

Minimizing the total downstream cost (min-total objective).

downstream problem	lower bound	upper bound
bin-packing	2	$\min\{4, \delta\}$
scheduling	3/2	$3 - 1/\delta$
traveling salesman	2	$2\delta - 1$

Minimizing the maximum downstream cost (min-max objective).

general co petitive online algorith s for the OnlineTDAP and analyze the in greater detail w. r. t. classical co binatorial downstrea proble s such as bin-packing [4, SR1], nonpree ptive parallel achine scheduling [4, SS8] and the traveling sales an proble [4, ND22]. The algorith s we propose do not depend on the downstrea proble (although the analysis does). We e phasize that the particular downstrea proble s discussed in this paper should be seen ainly as illustrating exa ples for the general fra ework. Concerning standard online investigations on these proble s, [3] gives surveys on online bin-packing and scheduling; the online traveling sales an proble has been considered in [1].

Within the OnlineTDAP fra ework, our results are online algorith s and lower and upper bounds on their perfor ance guarantees, the co petitive ratio, obtained by classical co petitive analysis for online algorith s (see, e. g. [2]). In Section 2 we present a 2-co petitive algorith for the in-total objective, i. e., the objective to ini ize the total cost su ed over all target dates.

In Section 3 we consider in- ax proble s for which the objective is to ini ize the axi u downstrea cost that occurs on a target date. Here, we give a general online assign ent algorith that we prove to be 4-co petitive for the OnlineTDAP with the bin-packing downstrea proble and which is 3-co petitive for the scheduling setting. Our ain results are su arized in Table 1. Finally, we observe for both objective functions that special profiles for the downstrea proble , as e. g., (un-) bounded nu ber of achines or bins per target date, lead to trivial proble s or prevent any deter inistic online algorith fro achieving a constant co petitive ratio.

2 Minimizing Total Downstream Cost

In this section, we consider the OnlineTDAP with the objective to ini ize the total downstrea cost su ed up over all target dates (in-total objective). Particular downstrea proble s we deal with are bin-packing, scheduling on parallel achines, and the traveling sales an proble .

We first present our ain co petitiveness result which is an online algorith for ulated independently of the downstrea proble . Let us say that a target date is *used*, if a request has been assigned to it.

Algorithm PackTogetherOrDelay (PTD) Assign a request r to the earliest date in the feasible range $t(r)+1,\dots,t(r)+\delta$ which is already used. If no used target date is feasible for request r, then assign it to the latest feasible target date, that is, to $t(r)+\delta$.

The above algorithm always finds a feasible solution under the assumption that the amount of requests that can be assigned to the same target date is not restricted (we call this the case of *unlimited resources*). Under this assumption at any moment in time at most one feasible target date is used by PTD.

Theorem 1. *Consider the* ONLINETDAP *w. r. t. downstream problem Π with the min-total objective. Assume that there are unlimited resources in Π and suppose that the following properties hold for any subinstance $\bar{\sigma}$ of σ:*

i. *The optimal offline cost for the downstream problem Π is a monotonously increasing function, that is,* $\mathrm{OPT}(\bar{\sigma}) \le \mathrm{OPT}(\sigma)$ *(i. e., Π is monotone).*
ii. *For each disjoint partition $\sigma^{(1)},\dots,\sigma^{(k)}$ of the subsequence $\bar{\sigma}$ the inequality* $\mathrm{downcost}(\bar{\sigma}) \le \sum_{i=1}^{k} \mathrm{downcost}(\sigma^{(i)})$ *holds (i. e., Π allows for synergy).*

Then, algorithm PTD *is* 2*-competitive.*

Proof. For a given sequence of requests σ consider the target dates $d_1 < d_2 < \dots < d_k$ that PTD chooses. Denote by σ_{odd} (and σ_{even}) the subsequence of requests that the algorithm assigns to target dates d_i with odd (respective even) index i.

Observe that, if the input to PTD were solely σ_{odd} or σ_{even}, then each request would still be assigned to the same target date as when operating on σ. Therefore,

$$\mathrm{PTD}(\sigma) = \mathrm{PTD}(\sigma_{\mathrm{odd}}) + \mathrm{PTD}(\sigma_{\mathrm{even}}). \tag{1}$$

Moreover, we know by definition of the algorithm that the difference between any two used target dates is at least δ. Thus, the distance between any two different target dates designated for two requests of the subsequence σ_{odd} (or σ_{even}, respectively) is at least 2δ. This implies that no two requests of the same subsequence σ_{odd} (or σ_{even}, respectively) that have not been assigned to the same target date share a single feasible target date. Therefore, no algorithm can assign such two requests to the same target date. With property (ii) we conclude that

$$\mathrm{PTD}(\sigma_{\mathrm{odd}}) = \mathrm{OPT}(\sigma_{\mathrm{odd}}) \quad \text{and} \quad \mathrm{PTD}(\sigma_{\mathrm{even}}) = \mathrm{OPT}(\sigma_{\mathrm{even}}).$$

It follows with (1) and the monotonicity condition (i) that we have online cost

$$\mathrm{PTD}(\sigma) \;=\; \mathrm{OPT}(\sigma_{\mathrm{odd}}) + \mathrm{OPT}(\sigma_{\mathrm{even}}) \;\le\; 2\,\mathrm{OPT}(\sigma). \qquad \square$$

Note, that in the case that property (ii) only holds in a relaxed version with a factor α, i. e., $\mathrm{downcost}(\bar{\sigma}) \le \alpha \sum_i^k \mathrm{downcost}(\sigma^{(i)})$, PTD is 2α-competitive.

We will now demonstrate the power of Theorem 1 by applying it to various instantiations of the ONLINETDAP.

2.1 Downstream Bin-Packing

In bin-packing n items with sizes $s_1, \ldots, s_n$ need to be packed in unit sized bins. The objective is to find a packing such that the total size of the items packed in one bin does not exceed the bin's capacity and the total number of bins needed to pack the items is minimized. In OnlineTDAP w. r. t. bin-packing, a request $r = (t(r), s(r))$ is given by its release date $t(r)$ and its size $0 < s(r) \leq 1$. We assume that the number of available bins per day is not bounded because this would disable any deterministic online algorithm to guarantee a feasible solution. The objective is to find an assignment of requests to feasible target dates that minimizes the total sum of used bins of all target dates.

The following theorem gives a lower bound on the competitive ratio of any deterministic online algorithm.

Theorem 2. *No deterministic online algorithm for* OnlineTDAP *w. r. t. bin-packing minimizing the min-total objective has a competitive ratio less than* $3/2$.

Proof. The adversarial sequence starts with a request r_1 released at time 0 with size $s(r_1) < 1/2$. Consider an online algorithm, Alg, that does not assign this request to its deadline δ. Then at time Alg$[r_1]$ a second request is released with size $s(r_2) = 1 - s(r_1)$. Alg cannot assign this request to the same date as the first request and therefore it needs two bins, whereas the optimum needs only one.

Now consider an online algorithm Alg that assigns the first request to its deadline δ. Then at time $t(r_2) = 1$ a second request with size $s(r_2) = s(r_1)$ is released. If the algorithm does not pack this item with the first request, then it needs two bins and the optimum needs only one. Otherwise, at time $t(r_3) = \delta - 1$ and $t(r_4) = \delta$ two requests are released both with size $s(r_3) = s(r_4) = 1 - s(r_1)$. To pack these items, Alg needs two extra bins, thus in total three bins, whereas the optimum would pack request r_1 and r_3 to date δ and item r_2 and r_4 to $\delta + 1$, needing only two bins. $\square$

Since the properties of Theorem 1 are met, we immediately have the following result.

Theorem 3. *The competitive ratio of* PTD *minimizing the total number of used bins for* OnlineTDAP *w. r. t. bin-packing is* 2.

That PTD cannot achieve a better competitive ratio than 2, can be shown by the following instance. For given $k \in \mathbb{N}, k \geq 3$, let $\varepsilon < 1/(2k - 4)$ and $\sigma = \sigma^{(1)} \cup \ldots \cup \sigma^{(k)}$. $\sigma^{(1)}$ consists of the following three requests:

$$r_1 = (0, 1), \quad r_2 = (1, 1/2 - \varepsilon), \quad r_3 = (\delta, 1/2 + \varepsilon).$$

For $i = 2, \ldots, k$, the subsequence $\sigma^{(i)}$ is defined by

$$\sigma^{(i)} = ((i\delta - 1, 1/2 + (i - 2)\varepsilon), (i\delta, 1/2 - (i - 2)\varepsilon)).$$

The cost of PTD on this sequence is $\text{PTD}(\sigma) = 2k + 1$. On the other hand, the number of required bins of the optimal offline algorithm is $\text{OPT}(\sigma) = k + 1$. By letting $k \to \infty$, the lower bound follows.

We conjecture that the following online algorithm, PACKFIRSTORDELAY, has a better performance guarantee than PTD although the analysis for the general problem seems more difficult.

Algorithm PackFirstOrDelay (PFD) If there is a used target date to which the current request r can be assigned without increasing the number of necessary bins, then the earliest of these dates is chosen. Otherwise, assign the latest possible date, $t(r) + \delta$.

This algorithm achieves a better solution on the lower bound instance for PTD from above. However, there exist instances for which PFD performs worse than PTD, as for example: $r_1 = (0, 2/5)$, $r_2 = (0, 1/5)$, $r_3 = (0, 1/5)$, $r_4 = (\delta - 1, 2/5)$, $r_5 = (\delta - 1, 2/5)$, and $r_6 = (\delta - 1, 2/5)$.

If all items have identical size the problem becomes much easier.

Theorem 4. *Consider the* ONLINETDAP *w. r. t. bin-packing with the min-total objective. Then,* PFD *is optimal if all item sizes are equal.*

Proof. Assume that the bin-packing instance at each date is solved in such a way that at most one bin is partially filled. Given a sequence σ, let $\text{PFD}(\sigma) = f + p$, where f is the number of full bins and p is the number of partially filled bins. Let $d_0 < d_1 < \ldots < d_p$ be the dates on which PFD has partially filled bins. Let σ' be the subsequence consisting of all requests that are packed in a full bin and for each partially filled bin the request that opened this bin. Note that $\text{PFD}(\sigma') = \text{PFD}(\sigma)$.

We partition σ' into subsequences σ_ℓ consisting of all requests $r \in \sigma'$ with $d_{\ell-1} \leq t(r) < d_\ell$. As the last request in σ_ℓ and the first request in $\sigma_{\ell+1}$ are both assigned using the delay tactic of PFD, we know that there is no overlap in the feasible target dates of requests of different subsequences. Hence, $\text{OPT}(\sigma') = \sum_\ell \text{OPT}(\sigma_\ell)$. Moreover, PFD packs the items of a subsequence in all but one fully filled bins and thus $\text{PFD}(\sigma_\ell) = \text{OPT}(\sigma_\ell)$. Combining these equalities, we get

$$\text{PFD}(\sigma) = \text{PFD}(\sigma') = \sum_\ell \text{PFD}(\sigma_\ell) = \sum_\ell \text{OPT}(\sigma_\ell) = \text{OPT}(\sigma') \leq \text{OPT}(\sigma).$$

$\square$

2.2 Downstream Parallel-Machine Scheduling

In this section, we consider the ONLINETDAP w. r. t. nonpreemptive machine scheduling of jobs on identical parallel machines to minimize the makespan, i. e., the latest completion time of all jobs on all machines of one date. The overall objective is now to minimize the sum of makespans over all target dates. For convenience we will use standard scheduling terminology, i. e., a request r is a job that has a processing ti e denoted by $p(r)$. We denote a request by an ordered pair of release date and processing ti e, $r = (t(r), p(r))$. The nu ber of achines available per date is denoted by m. Note, that in case $m = 1$, the proble is trivial since any target date assign ent yields a total downstrea cost of $\sum_{r \in \sigma} p(r)$, for any sequence σ. Therefore, we assu e for the re ainder of this section that ore than one achine are available each date.

Consider the general online algorith PTD. Also for this setting with the scheduling downstrea proble , Theore 1 applies and PTD is 2-co petitive. The analysis is tight as the following sequence shows:

$$r_1 = (0, \varepsilon), \quad r_2 = (\text{PTD}[r_1] - 1, 1), \quad r_3 = (\text{PTD}[r_1], 1),$$

where $\varepsilon < 1$. The costs incurred by the algorith are $\text{PTD}(\sigma) = 2$, whereas opti al offline costs are $\text{OPT}(\sigma) = 1 + \varepsilon$. Thus, we have shown:

Theorem 5. *The deterministic online algorithm* PTD *has a competitive ratio of* 2 *for the* ONLINETDAP *for downstream scheduling on identical parallel machines ($m > 1$) subject to minimize the sum of makespans induced on all target dates.*

Moreover, we obtain the following general lower bound result for this proble setting.

Theorem 6. *No deterministic online algorithm can achieve a competitive ratio less than $\sqrt{2}$ for the* ONLINETDAP *minimizing the total downstream cost caused by nonpreemptive scheduling on more than one machine.*

Proof. In order to obtain this bound consider for a given online algorith ALG the following sequence:

$$r_1 = (0, 1), \quad r_2 = (\text{ALG}[r_1] - 1, 1 + \sqrt{2}).$$

If ALG assigns a target date different fro $\text{ALG}[r_1]$ to request r_2, then no further requests are given. Thus, ALG's cost is $\text{ALG}(r_1, r_2) = 2 + \sqrt{2}$, whereas an offline opti u yields a solution with cost $\text{OPT}(r_1, r_2) = 1 + \sqrt{2}$, which gives a ratio of $\sqrt{2}$.

Assu e that ALG assigns request r_2 to the sa e date as r_1, and a third request $r_3 = (\text{ALG}[r_1], 1 + \sqrt{2})$ is given. Then the cost of the online algorith is $2 + 2\sqrt{2}$, whereas the opti al offline costs are $2 + \sqrt{2}$. Again, the ratio of the incurred costs of ALG and OPT is $\sqrt{2}$. □

Note that the lower bound construction heavily depends on different processing ti es of jobs. Let us briefly consider the restricted setting where we assu e that all requests have equal processing ti e. In this case, we can easily transfor the ONLINETDAP w. r. t. parallel achine scheduling into an ONLINETDAP w. r. t. bin-packing: Each request $(t(r_j), p(r_j))$ is transfor ed into a request $(t(r_j), s(r_j) = 1/m)$, i. e., to each job corresponds an ite of size $1/m$, where m is the nu ber of achines in the scheduling proble and we assu e unit bin capacity in the bin-packing problem. Both problems are equivalent; therefore the results from the previous section carry over, and thus, we have with PFD an optimal online algorithm.

Corollary 1. PFD *is an optimal algorithm for the* ONLINETDAP *with downstream problem scheduling of jobs with equal processing times for the min-total objective.*

2.3 Traveling Salesman Problem

In this section, we consider the ONLINETDAP with the downstream problem of finding a minimal tour of a traveling salesman problem, i. e., for a given set of points in a metric space (request set) a tour has to be found, from the origin through all points ending in the origin. The overall objective is now to minimize the sum of the optimal tour lengths on all target dates.

For this problem setting we provide the following general lower bound.

Theorem 7. *No deterministic online algorithm has a competitive ratio less than* $\sqrt{2}$ *for the* ONLINETDAP *w. r. t. a traveling salesman problem on* $\mathbb{R}_+$ *as the downstream problem minimizing the total downstream cost.*

Proof. Consider the following simple instance: At time 0, request r_1 with distance 1 from the origin is given. In order to be better than 2-competitive an algorithm has to assign the request to target date δ, because otherwise an identical request would be given at the chosen target date. Now, a second request r_2 appears at time 1 with distance $1+\sqrt{2}$ to the origin. If the algorithm assigns it to some target date different from δ, then no more requests are released and the ratio of costs of an online algorithm to those of the optimum is $\sqrt{2}$. Otherwise, a third request at the same location of request r_2 is released at time δ. In this case the ratio of costs is $\sqrt{2}$. □

As before, the conditions in Theorem 1 are also met for the traveling salesman problem as the downstream problem.

Theorem 8. PTD *has a competitive ratio of* 2 *for the* ONLINETDAP *w. r. t. minimizing the tour length in a traveling salesman problem as a downstream problem for the min-total objective.*

In order to show that this result is tight, consider two requests released at time 0 and 1, with distances ε and 1 from the origin, respectively. Let the distance between r_1 and r_2 be equal to the sum of their distances to the origin, $1+\varepsilon$. If a third request is released at time δ in exactly the same position as r_2, then the ratio of total sum of route length for PTD to OPT tends to 2 for $\varepsilon \to 0$.

3 Minimizing Maximum Downstream Cost

In this section, we consider ONLINETDAP subject to minimize the maximum downstream cost over all target dates for the downstream problems bin-packing, scheduling on parallel machines, and the traveling salesman problem.

As in the previous section, we firstly present a general online algorithm that is independent of the specific downstream problem.

Algorithm Balance (Bal) Assign a given request to the earliest feasible target date such that the increase in the objective value, i.e., the maximum downstream cost over all dates, is minimal.

Notice that processing each request requires Bal to solve several instances of the downstream problem optimally. However, computing optimal solutions may not be feasible under real-time aspects because of the complexity of the downstream problem. But in the analysis of our algorithm we only use such upper bounds on the offline optimum that are also satisfied by simple approximation algorithms. Therefore, all results presented in this section still hold true if the optimization is done approximately and all algorithmic computations can be accomplished in polynomial time.

3.1 Downstream Bin-Packing

We analyze the OnlineTDAP with bin-packing as downstream problem subject to minimizing the maximum number of used bins over all target dates. The notation and downstream problem definition is similar as in Section 2.1.

Our first result is a general lower bound on the competitive ratio of any online algorithm.

Theorem 9. *For the* OnlineTDAP *with min-max objective for downstream bin-packing no deterministic online algorithm has a competitive ratio of less than* 2*.*

Proof. In order to obtain this bound we consider a sequence σ with the following two first requests: $r_1 = (0, \varepsilon)$ and $r_2 = (0, \varepsilon)$ for some $\varepsilon < 1/2$.

If the considered online algorithm Alg assigns the same target date to both requests, then sequence σ is completed by the requests:

$$r_3 = (0, 1-\varepsilon), \quad r_4 = (0, 1-\varepsilon), \quad r_j = (0, 1) \quad 5 \leq j \leq \delta + 2.$$

Obviously, we have $\text{Alg}(\sigma) \geq 2$ and $\text{Opt}(\sigma) = 1$.

Suppose now that the online algorithm assigns different target dates to the requests r_1 and r_2, then the following additional requests are given:

$$r_3 = (0, 1-2\varepsilon), \quad r_j = (0, 1) \quad 4 \leq j \leq \delta + 2.$$

Again, any deterministic online algorithm is forced to open at least two bins on some date, i.e., $\text{Alg}(\sigma) \geq 2$, whereas the optimum has only cost $\text{Opt}(\sigma) = 1$. □

Next we analyze the algorithm Bal for the OnlineTDAP with downstream bin-packing.

Theorem 10. *The algorithm* Bal *is* 4*-competitive for the* OnlineTDAP *with downstream bin-packing subject to minimizing the maximum number of used bins over all target dates.*

Proof. The crucial observation is the following: Given a request r, the total size of all items assigned by BAL within the time frame $t(r)+1, \ldots, t(r)+\delta$ is bounded from below by half the number of bins required, whenever more than one bin is used in this period of dates.

This claim can be shown by induction on the number of requests assigned to any of the considered dates. Obviously, the claim holds when none of the considered dates has yet been used. Assume that the claim is true after k requests have been assigned to the dates $t(r)+1, \ldots, t(r)+\delta$ and let r_{k+1} be another request. If $s(r_{k+1}) \geq 1/2$, the claim obviously also holds after assigning r_{k+1}. So assume that $s(r_{k+1}) < 1/2$. If BAL can assign r_{k+1} to some date without increasing the number of used bins at that date, we are also done. But if BAL needs to use a new bin at the assigned date, we know that previously the load of each bin at the dates $t(r)+1, \ldots, t(r)+\delta$ was at least $1 - s(r_{k+1}) > 1/2$, which proves the claim.

Now we can prove that BAL is 4-competitive. Let r_k be the first request in a given sequence σ such that the maximum downstream cost is attained, i. e., $\textsc{Bal}(r_1, \ldots, r_k) = \textsc{Bal}(\sigma)$. Notice that the assigned target date for r_k is $\textsc{Bal}[r_k] = t(r_k) + 1$. Let $\bar{\sigma}$ be the subsequence of all requests from σ up to r_k that have been assigned a target date $d \geq t(r_k) + 1$. On the one hand, we have:

$$\textsc{Opt}(\sigma) \geq \frac{1}{2\delta - 1} \sum_{r \in \bar{\sigma}} s(r) > \frac{1}{2\delta} \sum_{r \in \bar{\sigma}} s(r). \tag{2}$$

On the other hand, BAL uses in total $\delta(\textsc{Bal}(\sigma) - 1) + 1$ bins in the time period from $t(r_k) + 1$ to $t(r_k) + \delta$. Since we may assume $\textsc{Bal}(\sigma) > 1$ (otherwise there is nothing to show), the sum of all item sizes assigned to theses dates is at least half the number of bins required by BAL. This implies,

$$\frac{1}{2} \left(\delta(\textsc{Bal}(\sigma) - 1) + 1\right) \leq \sum_{r \in \bar{\sigma}} s(r).$$

Together with (2), we can bound the cost of BAL by

$$\textsc{Bal}(\sigma) \leq \frac{2}{\delta} \sum_{r \in \bar{\sigma}} s(r) + 1 - \frac{1}{\delta} < 4\,\textsc{Opt}(\sigma) + 1 - \frac{1}{\delta}.$$

Finally, the integrality of $\textsc{Bal}(\sigma)$ and $\textsc{Opt}(\sigma)$ gives $\textsc{Bal}(\sigma) \leq 4\,\textsc{Opt}(\sigma)$. □

For small values δ the FIRSTFIT Algorithm that assigns a given request r to its earliest feasible target date $t(r)+1$, improves the competitiveness result of Theorem 10. It is easy to see that FIRSTFIT has a competitive ratio of δ.

As in Section 2.1, the situation improves significantly for equal item sizes.

Theorem 11. *The algorithm* BAL *is* 2*-competitive for the* ONLINETDAP *with downstream bin-packing subject to minimizing the maximum number of used bins over all target dates if all requests have equal sizes.*

Proof. Let r_k be the first request in a given sequence σ such that the maximum downstream cost is attained, i. e., $\text{BAL}(r_1, \ldots, r_k) = \text{BAL}(\sigma)$. Moreover consider on date $t(r_k) + 1$ an optimal packing which only uses one bin partially. With respect to such an optimal packing all bins at the dates $d > t(r_k)$ except one on the date $t(r_k) + 1$ are filled with a maximum number of items, because of equal item sizes. Since OPT requires the same number of bins distributed onto at most $2\delta - 1$ dates, we have

$$\text{OPT}(\sigma) \geq \frac{1}{2\delta - 1}\delta(\text{BAL}(\sigma) - 1) > \frac{1}{2}(\text{BAL}(\sigma) - 1).$$

This implies $\text{BAL}(\sigma) < 2\,\text{OPT}(\sigma) + 1$, which gives the theorem by the integrality of $\text{BAL}(\sigma)$ and $\text{OPT}(\sigma)$. □

Theorem 12. *For the* ONLINETDAP *with min-max objective for downstream bin-packing where all requests have equal sizes, no deterministic online algorithm has a competitive ratio of less than* $3/2$.

Proof. Let s denote the size of all requests, and consider an arbitrary online algorithm ALG and the following sequence σ of requests. $\delta\lfloor 1/s \rfloor$ requests are given at date 0. In order to achieve a competitive ratio better than 2, ALG must not use more than one bin each date. Next, at date 1 additionally $(\delta + 2)\lfloor 1/s \rfloor$ requests are given, which gives $\text{ALG}(\sigma) \geq 3$ and $\text{OPT}(\sigma) = 2$. □

3.2 Downstream Parallel-Machine Scheduling

In this section we consider the ONLINETDAP w. r. t. nonpreemptive machine scheduling on parallel machines subject to minimize the maximum makespan over all target dates. Notations and the exact downstream problem definition is used as in Section 2.2. Note, that if an infinite number of machines is available at each date, i. e., $m = \infty$, then the problem becomes trivial since any feasible solution yields a downstream cost of $\max_{r\in\sigma} p(r)$, for any sequence σ. In the following we assume a bounded number of machines.

In this problem setting where the number of available machines per date is bounded ($m < \infty$) the following instance shows a lower bound of $3/2$ on the competitive ratio of any deterministic online algorithm. Given $m\delta$ requests with release date 0 and processing time 1, only an algorithm ALG that assigns m jobs to each date can be better than 2-competitive. However, at date 1 are given $m(\delta + 2)$ more requests with processing time 1, then ALG has a makespan of at least 3 whereas the optimum makespan over all dates equals 2. Note that this request sequence contains only requests with equal processing time. Thus, we have shown the following:

Theorem 13. *No deterministic online algorithm can achieve a competitive ratio less than* $3/2$ *for the* ONLINETDAP *w. r. t. scheduling to minimize the maximum makespan over all target dates, where the number of available machines per date is bounded and the processing times for all requests are equal.*

We next prove that the general algorith Bal for the OnlineTDAP w. r. t. scheduling on parallel achines is $(3 - 1/\delta)$-co petitive.

Theorem 14. Bal *is* $(3 - 1/\delta)$*-competitive for the* OnlineTDAP *with downstream scheduling to minimize the maximum makespan over all target dates for a bounded number of available machines per date.*

Proof. Consider a request sequence σ served by Bal and let r denote the first request that causes the axi u akespan. Consider the schedule obtained by Bal before r is released with respect to the offline opti u and let w denote the load of a least loaded achine over all feasible target dates.

Then, the Bal's akespan is at ost $w + p(r)$. Since all feasible target dates for r have load of at least wm, the total load in that ti e period is at least $wm\delta + p(r)$.

Any of the corresponding requests in that ti e period could not be issued earlier than δ dates before the release date of request r. Hence, even an opti al offline algorith Opt obeying feasibility conditions has at least the following cost on sequence σ:

$$\mathrm{Opt}(\sigma) \geq \frac{wm\delta + p(r)}{(2\delta - 1)m} > \frac{w\delta}{2\delta - 1}.$$

Hence, we have:

$$w < \left(2 - \frac{1}{\delta}\right) \mathrm{Opt}(\sigma).$$

Since $\mathrm{Opt}(\sigma)$ is bounded fro below by $p(r)$, we conclude

$$\mathrm{Bal}(\sigma) \leq w + p(r) < \left(2 - \frac{1}{\delta}\right) \mathrm{Opt}(\sigma) + \mathrm{Opt}(\sigma) = \left(3 - \frac{1}{\delta}\right) \mathrm{Opt}(\sigma).$$

□

The following sequence σ shows for $\varepsilon \to 0$ that Bal is not better than 2-co petitive:

$$r_i = \begin{cases} (0, 1/2 + \varepsilon) & \text{if } i \in \{1, \ldots, m(\delta - 1)\}, \\ (0, 1) & \text{if } i \in \{m(\delta - 1) + 1, \ldots, m\delta\}, \\ (\delta - 1, 1) & \text{if } i \in \{m\delta + 1, \ldots, m(2\delta - 1) + 1\}. \end{cases}$$

Note, that this lower bound construction is based on jobs with different processing ti es. Now, let us briefly consider the restricted setting where we assu e that all requests have equal processing ti e. Then, the downstrea proble scheduling is equivalent to the bin-packing proble of unifor ite s as we described in Section 2.2. Hence, the results fro the previous section carry over.

Corollary 2. *The algorithm* Bal *is* 2*-competitive for the* OnlineTDAP *with min-max objective for downstream scheduling if all jobs have identical processing times. Furthermore, no deterministic online algorithm can achieve a competitive ratio of less than* 3/2 *in this setting.*

3.3 Traveling Salesman Problem

In this section we analyze the traveling salesman problem as downstream problem for the OnlineTDAP with objective to minimize the maximum downstream cost. Similar to the downstream problems considered before, the algorithm Bal is trivially $(2\delta - 1)$-competitive since the requests assigned to the date at which the maximum tour length is attained can at most be spread over $2\delta - 1$ dates. On the other hand, we have the following lower bound on the competitive ratio of any online algorithm.

Theorem 15. *No deterministic online algorithm for the* OnlineTDAP *w. r. t. the traveling salesman problem as downstream problem minimizing the maximum tour length achieves a competitive ratio less than* 2*.*

Proof. Consider a metric space induced by the unweighted star graph with at least $\delta + 1$ leaves. First, δ requests in δ different leaves are given at date 0. In case an algorithm Alg assigns more than one request to one date, it cannot be better than 2-competitive. Otherwise, let r be the request with $\text{Alg}[r] = 1$. At date 1 another request associated with the point not yet used is released as well as a request for the point of request r, yielding $\text{Alg}(\sigma) \geq 2$. In contrast, $\text{Opt}(\sigma) = 1$ since Opt is able to assign both requests for the same point to the same target date. $\square$

References

1. Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo, *Algorithms for the on-line travelling salesman.*, Algorithmica **29** (2001), no. 4, 560–581.
2. Allan Borodin and Ran El-Yaniv, *Online computation and competitive analysis*, Cambridge University Press, 1998.
3. Amos Fiat and Gerhard J. Woeginger (eds.), *Online algorithms, the state of the art*, Lecture Notes in Computer Science, vol. 1442, Springer, 1998.
4. Michael R. Garey and David S. Johnson, *Computers and intractability (a guide to the theory of NP-completeness)*, W.H. Freeman and Company, New York, 1979.
5. Martin L. Putermann, *Markov decision processes*, Wiley Interscience, 2005.

Approximation and Complexity of k–Splittable Flows

Ronald Koch[1], Martin Skutella[1], and Ines Spenke[2]

[1] Universität Dortmund, Fachbereich Mathematik, D – 44221 Dortmund, Germany
{ronald.koch, martin.skutella}@math.uni−dortmund.de
http://www.mathematik.uni-dortmund.de/lsv/

[2] Technische Universität Berlin, Institut für Mathematik, D – 10623 Berlin, Germany
spenke@math.tu−berlin.de
http://www.math.tu-berlin.de/∼spenke/

Abstract. Given a graph with a source and a sink node, the NP–hard maximum k–splittable flow (MkSF) problem is to find a flow of maximum value with a flow decomposition using at most k paths [6]. The multicommodity variant of this problem is a natural generalization of disjoint paths and unsplittable flow problems.

Constructing a k–splittable flow requires two interdepending decisions. One has to decide on k paths (routing) and on the flow values on these paths (packing). We give efficient algorithms for computing exact and approximate solutions by decoupling the two decisions into a first packing step and a second routing step. Our main contributions are as follows:

(i) We show that for constant k a polynomial number of packing alternatives containing at least one packing used by an optimal MkSF solution can be constructed in polynomial time. If k is part of the input, we obtain a slightly weaker result. In this case we can guarantee that, for any fixed $\epsilon > 0$, the computed set of alternatives contains a packing used by a $(1-\epsilon)$–approximate solution. The latter result is based on the observation that $(1-\epsilon)$–approximate flows only require constantly many different flow values. We believe that this observation is of interest in its own right.

(ii) Based on (i), we prove that, for constant k, the MkSF problem can be solved in polynomial time on graphs of bounded treewidth. If k is part of the input, this problem is still NP–hard and we present a polynomial time approximation scheme for it.

(iii) Finally, we provide a comprehensive overview of the complexity and approximability landscape of MkSF for different values of k.

1 Introduction

Many applications in transport, teleco unication, production or traffic are odelled as flow proble s. In classic flow theory, flow is sent through a network fro sources to sinks respecting edge capacities. It does not atter on how any paths the flow is sent. It can split into s all flow portions along a large

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 244–257, 2006.

nu ber of paths. But any applications do not allow an arbitrarily large nu ber of paths. For exa ple, in logistics co odities are usually transported with a given nu ber of vehicles. This bounds the nu ber of paths that can be used si ultaneously. Another exa ple is data transport in co unication networks. Co unication syste s often split data into packages. These packages traverse the network along different paths. Every package has to carry full infor ation about source and target of the data, about the position of this package a ong other packages, and so on. It is therefore not efficient to split data into too any packages. As a consequence, various applications require that the flow does not use too any paths. Classical flow algorith s do not take such restrictions into account.

Problem description. Let $G = (V, E)$ be a connected undirected or directed graph with n nodes and m edges with capacities $u : E \to \mathbb{Q}_{\geq 0}$. Moreover, there is a source and a sink node $s, t \in V$. Baier, Köhler, and Skutella [6] introduce the concept of *k–splittable flows*. For a given nu ber k, a feasible s, t–flow is called *k–splittable* if it can be deco posed into flows along at ost k paths leading fro s to t. We do not require the paths to be disjoint, not even different. The Maxi u k–Splittable Flow proble (MkSF) is to find a k–splittable s, t–flow of axi u value. Of course, k–splittability can also be considered in the ore general ulti–co odity setting. Then the nu ber of s_i, t_i–paths is restricted for each co odity i. In this paper, however, we concentrate on the single–co odity case.

Results from the literature. Since the se inal work of Ford and Fulkerson [10], there has been a vast a ount of literature on classical s, t–flows with no restriction on the nu ber of paths used. It is well known that a axi u s, t–flow can be co puted in polyno ial ti e, for exa ple, by aug enting path algorith s. Another classical result states that any s, t–flow can be deco posed into flow on at ost m paths and cycles. For further details we refer to the book by Ahuja, Magnanti, and Orlin [1].

Kleinberg [12] introduces *unsplittable flows*. These ultico odity flows route the total de and of each co odity along one single path. They generalize edge–disjoint paths. Kleinberg analyses co plexity and approxi ation algorith s for different unsplittable flow proble s, e.g. for ini izing the congestion on edges or equivalently axi izing the throughput, for the proble of ini izing the nu ber of rounds needed to satisfy all de ands and for the proble of axi izing the total de and which can be routed si ultaneously. In the ultico odity setting, k–splittable flows constitute a generalization of unsplittable flows.

Baier, Köhler, and Skutella [6] (see also [5]) investigate k–splittable flows in the single- and in the ulti–co odity setting. They prove NP–hardness of MkSF in directed graphs for all constant $k \geq 2$. For the special case of the *uniform* MkSF, where all k paths ust carry the sa e a ount of flow, they give a axflow– incut type result as well as an $O(km \log n)$ algorith that co putes an opti u solution. Based on these insights, they present $\frac{1}{2}$–approxi ation algorith s for the general MkSF proble . Bagchi, Chaudhary, Scheideler, and Kolman [4] consider fault tolerant routings in networks and define notions similar to k–splittable flows. To ensure connection for each commodity for up to $k-1$ edge failures in the network, they require edge disjoint flow–paths per commodity. Martens and Skutella [14] consider a new variant of k–splittable multi–commodity flows with upper bounds on the amount of flow sent along each path. The objective is to minimize the congestion of arcs. They prove that any ρ–approximation for the unsplittable flow problem gives a 2ρ–approximation for two different variants of the considered problem.

Krysta, Sanders, and Vöcking [13] consider related problems in the area of machine scheduling problems by imposing a bound on the number of preemptions of each task. In their k–splittable scheduling problem, each task can be split into at most $k \geq 2$ pieces that are assigned to different machines. They describe a polynomial time algorithm for finding an exact solution for the k–splittable scheduling problem and a slightly more general problem. This algorithm has a running time which is exponential in the number of machines but linear in the number of tasks.

Many NP–hard problems on graphs become easy when restricted to special graph classes. In this context, graphs of bounded treewidth have turned out to be a particularly successful concept. Originally introduced by Robertson and Seymour [15] in the context of graph minors, these graphs are also relevant in several practical applications. Bodlaender [9] presents a general framework for obtaining polynomial algorithms for problems in graphs of bounded treewidth that are NP–hard in general graphs. Bodlaender [7] and Arnborg, Lagergren, and Seese [3] give general characterizations of problems that can be solved in polynomial time on graphs of bounded treewidth. The MkSF problem does not fall into one of these classes of problems. For a more detailed account of concepts and results in this area we refer to the survey paper by Bodlaender [9].

The only paper we are aware of that considers flows in graphs of bounded treewidth is the one by Hagerup et al. [11]. Given a graph with a constant number of terminals and with arc capacities, they show that all realizable demand/supply patterns at the terminals can be found efficiently in graphs of bounded treewidth.

Our paper. Constructing k–splittable flows requires to decide which paths should be used and what flow values should be sent. Of course, these two decisions cannot be made independently of each other but are coupled by the requirement to obey arc capacities. A natural approach is to first choose a collection of paths $(P_1, \ldots, P_k)$. Arc capacities then bound possible tuples of flow values $(f_1, \ldots, f_k)$ on these paths. In this paper we take the reverse approach. We first fix flow values $(f_1, \ldots, f_k)$ that we wish to send (packing). Then, in the second step (routing), we try to find a collection of paths $(P_1, \ldots, P_k)$ on which these flow values can be routed without violating arc capacities.

In Section 2 we consider the packing step, first for fixed k, then for k being part of the input. The number of possibilities for flow values $(f_1, \ldots, f_k)$ in an optimal solution of MkSF is a priori not bounded. For fixed k, we describe how to determine a polynomial number of alternatives for $(f_1, \ldots, f_k)$ containing the flow value pattern of at least one optimal solution to MkSF. These alternatives are determined in polynomial time by solving certain linear equation systems. We do not know whether all of these alternatives can be routed in G without violating capacities. But we know that at least one alternative can be routed yielding an optimal solution to MkSF.

Not surprisingly, the situation gets more difficult when k is no longer constant but part of the input. We prove that, for any fixed $\epsilon > 0$, there exists a $(1-\epsilon)$–approximate solution to MkSF that only uses constantly many flow values on paths. To be more precise, $|\{f_1, \ldots, f_k\}| \in O(\log(1/\epsilon)/\epsilon^2)$ for this solution. We believe that this result is also interesting for other flow problems (e.g., multicommodity flows etc.). As a result of this observation, we can "guess" the flow values used by a $(1-\epsilon)$–approximate solution to the MkSF problem while only increasing the running time of the subsequent routing procedure by a polynomial factor.

In Section 3 we consider the routing step on graphs of bounded treewidth. For constant k, the problem can be solved to optimality in polynomial time. Surprisingly, however, if k is part of the input, the MkSF problem is NP–hard on graphs of bounded treewidth. Based on our results from Section 2 and standard dynamic programming techniques, we obtain a polynomial–time approximation scheme (PTAS) in this case.

Finally, in Section 4 we classify the complexity and approximability of the MkSF problem for different values of $k \geq 2$ on directed and undirected graphs. In particular, we prove that the problem on undirected graphs is already NP–hard for $k = 2$. So far, NP–hardness was only known for the case of directed graphs. Moreover, we show that, for arbitrary constant k, the problem cannot be approximated with performance ratio better than $5/6$. The question whether MkSF is also NP–hard for "large" values of k, like for example $k = m/2$, has so far been open. We prove that the problem is NP–hard for all values of k within the range from 2 to $m - n + 1$ (for $n \geq 3$). For $k \geq m - n + 2$ the problem can be solved optimally in polynomial time.

Due to space limitations, we omit some proofs in this extended abstract. More details are given in the full version of the paper which can be found on the authors' homepages.

2 The Packing Stage

As mentioned in the introduction, we want to solve MkSF as a two–stage problem with a packing and a routing stage. Here, we consider the packing stage. Lemma 1 shows that, in order to solve the MkSF problem to optimality, it is not necessary to take all rational valued k–tuples $(f_1, ..., f_k)$ into account. It suffices to consider only $O(m^k)$ candidates of such tuples.

Lemma 1. *If k is constant, it is sufficient to consider $O(m^k)$ candidates to obtain a packing $(f_1, ..., f_k)$ of an optimal solution to* MkSF. *An appropriate set of candidates can be determined in $O(m^k)$ time.*

Proof. If we knew paths $P_1, \ldots, P_k$ used in an opti u solution to MkSF, then corresponding opti al path flow values $(f_1, \ldots, f_k)$ could be obtained by solving the following linear progra :

$$\begin{array}{lll} \text{ax} & f_1 + f_2 + ... + f_k & \\ \text{s.t.} & \displaystyle\sum_{i \in \{1,\ldots,k\}:\ e \in P_i} f_i \le u_e & \text{for all } e \in E(G), \\ & f_i \ge 0 & \text{for all } i \in \{1, ..., k\}. \end{array}$$

There exists an opti u solution to this linear progra which corresponds to a vertex of the underlying polytope defined by the $m+k$ inequalities. Every vertex of this polytope is defined by a subsyste consisting of k linearly independent inequalities which ust be tight for this vertex. The resulting syste of k linear equations is given by a regular $\{0,1\}$– atrix of size $k \times k$ and a right hand side vector consisting of edge capacity values and zeros. Since the nu ber of atrices in $\{0,1\}^{k \times k}$ is 2^{k^2} and the nu ber of possible right hand side vectors is at ost $(m+1)^k$, there are only $O(m^k)$ possible solutions to such equation syste s. This yields $O(m^k)$ candidates for flow values $(f_1, ..., f_k)$ in an opti u solution to MkSF. Notice that each candidate can be co puted in constant ti e by solving a syste of linear equations of size $k \times k$. □

For constant k only a polyno ial nu ber of candidates has to be considered. If k is not constant but part of the input, however, the latter insight is not useful in obtaining efficient algorith s for MkSF since the nu ber of candidate solutions is exponential in k and thus in the input size.

We can overco e this proble if, instead of looking for flow values $(f_1, ..., f_k)$ in an opti u solution, we settle for a near–opti u solution.

Assu e that an opti u solution to MkSF assigns flow values $x_1, \ldots, x_k$ to paths $P_1, \ldots, P_k$. The following packing le a shows that, for arbitrary $\epsilon > 0$, there exists a k–splittable flow of value at least $1 - \epsilon$ ti es the value of an opti u flow which uses only a constant nu ber (depending on ϵ) of different flow values on paths.

Lemma 2. *Let $\epsilon > 0$ be sufficiently small. Consider an arbitrary collection of k bins with capacities $x_1, \ldots, x_k$. Then, there exist k items with sizes $y_1, \ldots, y_k$ such that*

(i) the items can be packed into the given bins without violating capacities,
(ii) there are at most $3\log(1/\epsilon)/\epsilon^2$ different item sizes, and
(iii) the total item size is close to the total bin capacity, that is,

$$\sum_{i=1}^{k} y_i \ \ge\ (1 - 4\epsilon) \sum_{i=1}^{k} x_i \ .$$

Interpret the ite sizes $y_1, \ldots, y_k$ as flow values. Then, for each $j = 1, \ldots, k$, we can route flow of value y_j along path P_i where i is the bin which ite j has been assigned to. The resulting k–splittable flow does not violate capacities due to (i) and its flow value is al ost opti al due to (iii).

Proof. Let $X := \sum_{i=1}^{k} x_i$ denote the total bin capacity. We recursively define a partition of the set of bins into subsets $B_1, B_2, \ldots, B_\ell$ as follows. Consider the bins in order of non–increasing capacities. Add the first bin to B_1. Keep adding bins to B_1 as long as the total capacity of bins in B_1 is at most ϵX. The first bin which cannot be added to B_1 due to this restriction goes to B_2. The following bins are added to B_2 as long as the total capacity of bins in B_2 is at most ϵX and so on. Since, except for the last subset, the total capacity of bins in each subset is at least $\epsilon X/2$, the number of subsets obtained in this way is $\ell \le 2/\epsilon$. Notice that the first few subsets may contain a single bin of size greater than ϵX. All further subsets contain bins whose total capacity is at most ϵX.
For all but at most three subsets of size at most ϵX, we will fill all bins i contained in these subsets with items of total volume at least $(1-\epsilon)x_i$. We shortly argue that such a packing fulfills property (iii): The total capacity of bins contained in the three neglected subsets is at most $3\epsilon X$. The remaining capacity of at least $(1-3\epsilon)X$ is filled up to at least a $(1-\epsilon)$–fraction. Thus, the total size of all items packed is at least $(1-\epsilon)(1-3\epsilon)X \ge (1-4\epsilon)X$, for $\epsilon > 0$.

Packing Phase I: For all subsets B_p whose largest bin capacity is within a factor $1/\epsilon$ of its smallest bin capacity, we pack one item into each bin in B_p using at most $1 + \log_{1+\epsilon}(1/\epsilon)$ different item sizes: Take the smallest bin in B_p and denote its capacity by z; pack an item of size z into all bins of capacity at most $(1+\epsilon)z$ in B_p. Remove all packed bins and continue recursively.

Packing Phase II: In order to simplify notation, the subsets that were not treated in phase I are re–indexed and denoted by $B'_1, \ldots, B'_{\ell'}$; the smallest bin in B'_j is at least as large as the largest bin in B'_{j+1}, for $j = 1, \ldots, \ell'-1$. The largest bin capacity in B'_j is denoted by z_j. We ignore all bins in $B'_{\ell'-2} \cup B'_{\ell'-1} \cup B'_{\ell'}$. For $j = 1, \ldots, \ell' - 3$, greedily pack all bins in B'_j using at most $|B'_{j+2}|$ items of size z_{j+2}.

It remains to prove that each packed bin is filled up to at least a fraction $1-\epsilon$ of its capacity. First notice that the capacity x_i of each bin $i \in B'_j$ is greater than z_{j+2}/ϵ. This is due to the fact that the ratio of the largest and smallest capacity of bins in B'_{j+1} is greater than $1/\epsilon$ (otherwise, subset B'_{j+1} would have been treated in phase I). Thus, if enough items of size z_{j+2} are available, each bin $i \in B'_j$ can be filled leaving a slack smaller than $z_{j+2} < \epsilon x_i$. In order to prove that enough items are available, it suffices to show that the total volume of $|B'_{j+2}| + 1$ items of size z_{j+2} exceeds the total capacity of all bins in B'_j:

$$\sum_{i \in B'_j} x_i \;\le\; \epsilon X \;<\; \sum_{i \in B'_{j+2}} x_i + z_{j+2} \;\le\; (|B'_{j+2}| + 1) \cdot z_{j+2} \enspace .$$

The number of different item sizes used in phase I and II is bounded by $\ell(1 + \log_{1+\epsilon}(1/\epsilon)) \le 3\log(1/\epsilon)/\epsilon^2$ for ϵ small enough. Moreover, at most k items are used and the sizes of the remaining items can be set to zero. □

Corollary 1. *If the value of an optimum solution to* MkSF *is known, flow values together with multiplicities denoting the number of paths which carry these* *flow values used by a* $(1-\epsilon)$*–approximate solution can be obtained by testing* $(\frac{k}{\epsilon})^{O(\log(1/\epsilon)/\epsilon^2)}$ *candidates.*

Proof. We denote the value of an optimum solution by OPT. As discussed above, it follows from Lemma 2 that there exists a $(1-\epsilon)$–approximate solution which uses $O(\log(1/\epsilon)/\epsilon^2)$ different flow values. If we round down all flow values to multiples of $\epsilon OPT/k$, we lose another factor of at most $1-\epsilon$ in the flow value. The resulting flow is therefore still $(1-2\epsilon)$–approximate and uses $O(\log(1/\epsilon)/\epsilon^2)$ out of k/ϵ possible flow values. These flow values can therefore be guessed by trying all $(\frac{k}{\epsilon})^{O(\log(1/\epsilon)/\epsilon^2)}$ possible alternatives. For each fixed alternative, we have to assign a number to each flow value of this alternative which determines the number of paths carrying this flow value. For each alternative, the number of different assignments is bounded by $k^{O(\log(1/\epsilon)/\epsilon^2)}$. Thus, we have to test $(\frac{k}{\epsilon})^{O(\log(1/\epsilon)/\epsilon^2)}$ candidates. □

Notice that one can get rid of the assumption that the value of an optimum solution is known. Using standard binary search, OPT can be determined within a factor of $1-\epsilon$ while increasing the running time of the embedded algorithm only by a polynomial factor.

3 The Routing Stage in Graphs of Bounded Treewidth

In this section we consider the MkSF problem on graphs of bounded treewidth, a graphclass introduced by Robertson and Seymour [15]. For constant k, we present a polynomial time algorithm for MkSF. For arbitrary k, the problem remains NP–hard even if restricted to graphs of bounded treewidth (with only three nodes and two sets of parallel arcs). We give a polynomial time approximation scheme for the general MkSF problem on graphs of bounded treewidth.

Theorem 1. *On graphs of bounded treewidth, the* MkSF *problem can be solved in polynomial time if* k *is constant. For arbitrary* k*, the problem is* NP*–hard and there exists a polynomial time approximation scheme.*

3.1 Preliminaries on Graphs of Bounded Treewidth

Given a graph $G=(V,E)$ (directed or undirected), a *tree decomposition* is a pair (T,χ) where T is a tree and $\chi=\{X_i | X_i \subseteq V, i \in V(T)\}$ is a family of subsets of V associated with the nodes of T such that the following conditions hold: (i) Each node of G is contained in a subset X_i for some $i \in V(T)$. (ii) For each edge in G there exists a node $i \in V(T)$ such that X_i contains both endpoints of that edge. (iii) For each node $u \in V(G)$, the vertices $i \in V(T)$ with $u \in X_i$ span a subtree of T.

The width of a tree decomposition (T,χ) is $\max_{i\in V(T)} |X_i|-1$. The treewidth of a graph G is the minimum width over all tree decompositions of G. Given as input a graph G and an integer ω, it is NP–complete to decide if G has treewidth at ost ω; see [2]. On the other hand, if the treewidth of G is bounded by a fixed constant, a deco position tree can be constructed in linear ti e [8].

We can restrict to tree deco positions featuring a special structure: A tree deco position (T, χ) of G is called *nice* if T is a rooted binary tree and if the nodes partition into four types: A *join node* $i \in V(T)$ has two children $j, h \in V(T)$ fulfilling $X_i = X_j = X_h$. An *introduce node* $i \in V(T)$ has only one child j and that child fulfills $X_j \subset X_i$. A *forget node* $i \in V(T)$ has only one child $j \in V(T)$ and that child fulfills $X_j = X_i \cup \{u\}$ for so e $u \in V(G) \setminus X_i$. Finally, for a *leaf node* $i \in V(T)$, the set X_i consists of so e node $u \in V(G)$ together with a subset of its neighborhood. Further ore, in a nice tree deco position, there is a leaf containing u and v, for each edge $(u, v) \in E(G)$. For a given graph G, a tree deco position can be transfor ed into a nice tree deco position of the sa e width in linear ti e with tree size $O(|V(G)|)$; see, e.g., [16].

3.2 The Algorithm

In the following description of the algorith we restrict to the case of si ple (directed) graphs without parallel edges. Without going into further details, we re ark that Theore 1 also holds for non–si ple graphs. As a result of Section 2, a polyno ial ti e algorith for the following proble on graphs of bounded treewidth will prove Theore 1.

Given: Directed or undirected graph $G = (V, E)$ with edge capacities $u : E \to \mathbb{Q}_{>0}$, a source $s \in V$, and a sink $t \in V$; a constant nu ber of flow values $f_1, \ldots, f_\ell \in \mathbb{Q}_{>0}$ together with ultiplicities $q_1, \ldots, q_\ell \in \mathbb{N}$ which are polyno ially bounded in the size of G.

Task: For $j = 1, \ldots, \ell$, find q_j paths (not necessarily distinct) fro s to t such that sending f_j flow units along each path (si ultaneously for all j) does not violate edge capacities; alternatively, decide that no such flow exists.

Theorem 2. *On graphs of constantly bounded treewidth, the problem stated above can be solved in polynomial time. Moreover, if the multiplicities q_j, $j = 1, \ldots, \ell$, are all constant, it can be solved in linear time.*

For the sake of si plicity, we reduce the flow proble to a circulation proble by introducing a new edge fro t to s with sufficiently large capacity (notice that adding an edge to a graph increases its treewidth by at ost one). Now the proble can be refor ulated as follows: Find a feasible circulation in the extended graph which fulfills the additional require ent that, for $j = 1, \ldots, \ell$, exactly q_j cycles (not necessarily distinct) each carrying flow value f_j have to traverse the special edge fro t to s.

Algorith s exploiting bounded treewidth of the input graph are usually based on a dyna ic progra ing approach that proceeds botto –up in the deco position tree. Our algorith follows along the sa e line. For a general description of this approach we refer to [9].

The rough idea of the algorithm is as follows. Each edge of graph G is associated with exactly one leaf of T containing its two endpoints. For a tree node $i \in V(T)$ we denote by $G_i = (V_i, E_i)$ the subgraph of G given by

$$V_i := \{v \in V(G) \mid v \in X_h \text{ with } h = i \text{ or } h \text{ is descendant of } i \text{ in } T\} \qquad \text{and}$$
$$E_i := \{e \in E(G) \mid e \text{ is associated with } i \text{ or a descendant of } i \text{ in } T\} \ .$$

For every tree node $i \in V(T)$, we determine all possible ways of sending flow in graph G_i on X_i–paths. (A path is called an *X_i–path* if its ends are distinct vertices in X_i and no internal vertex belongs to X_i.) To be more precise, each possible *state* of node i is specified by the following information: For every ordered pair of distinct vertices $(u, v) \in X_i \times X_i$ and for every $j \in \{1, \ldots, \ell\}$, we give the number $\pi(u, v, j) \le q_j$ of (not necessarily different) X_i–paths between u and v carrying f_j units of flow.

Notice that the number of possible states at node i is at most $\prod_{j=1}^{\ell}(1+q_j)^{|X_i|^2}$ and thus polynomially bounded. Of course, we are only interested in states/flows that can be realized without violating edge capacities. Moreover, if the special edge from t to s is contained in G_i, we only consider flows where the number of X_i–paths of flow value f_j using that edge is exactly q_j, for $j = 1, \ldots, \ell$. These two requirements are taken care of when computing the set of feasible states at the leaf nodes of T. In the following we give an overview of how the required information can be computed at the nodes i of T. Since we basically follow a standard approach, further details are omitted.

If i is a *leaf node*, G_i contains only a constant number of edges. For each such edge $(u, v) \in E_i$, we generate all possible configurations $\pi(u, v, j)$, $j = 1, \ldots, \ell$, that do not violate the capacity of that edge. Of course, if the edge happens to be the special one from t to s, only the unique feasible configuration is generated. By taking all possible combinations of configurations at the edges, we get the set of all states of node i.

If i is an *introduce node*, the set of all states of i is identical to the set of all states of its only child i'. Notice that no flow can be sent from or received by terminals in $X_i \setminus X_{i'}$ since no edge in G_i is incident with one of these terminals.

If i is a *forget node*, the set of all states of i can be obtained from the set of all states of its only child i' as follows: Delete all states of i' that do not fulfill flow conservation at the unique node $u \in X_{i'} \setminus X_i$ separately for every $j = 1, \ldots, \ell$. For the remaining states, generate all possible matchings of incoming and outgoing flow paths of the same flow value at node u. This yields all possible flow patterns between terminals $X_i = X_{i'} \setminus \{u\}$.

Finally, if i is a *join node*, every feasible state of i can be generated by adding two states, one from each child node. Of course, we only consider sums for which $\pi(u, v, j) \le q_j$, for all $u, v \in X_i$ and $j = 1, \ldots, \ell$.

As it is always the case with this approach, the answer to the problem which we like to solve can be found at the root node r of tree T: There exists a feasible solution if and only if r has a state where flow conservation is fulfilled at all nodes in X_r, separately for every $j = 1, \ldots, \ell$. A solution (circulation) can be obtained by traversing the tree forwards to the leafs beginning with a feasible state at r. We o it all further details.

We conclude this section with a generalization of the obtained result. Notice that the described approach can also be applied if, instead of only one source s and one sink t, there is a constant nu ber of source–sink pairs (co . odities).

Corollary 2. *For a constant number of commodities and constant k, the k–splittable multicommodity problem can be solved in polynomial time on graphs of bounded treewidth. If we drop the requirement on k, we still obtain a polynomial time approximation scheme for the maximum k–splittable multicommodity problem with a fixed number of commodities.*

4 Complexity and Approximability for General Graphs

In this Section we return to general graphs. We analyze the hardness of MkSF proble s and their approxi ability. In the first part, we consider constant values of $k \geq 2$. MkSF is shown to be strongly NP–hard. We show that there is no approxi ation algorith with perfor ance ratio better than $\frac{5}{6}$. This is the first constant bound given for this proble . In the second part, k is a function of the nu ber of vertices n and edges m. We classify NP–hard and polyno ially solvable cases. For the sake of si plicity we restrict ourselves to undirected graphs, but any result in this section can be applied to the directed case by . inor . odifications in the proofs.

4.1 Constant k

In [6] the NP–hardness of MkSF is proven for constant $k \geq 2$ in directed graphs. The construction given there does not apply to undirected graphs. Theore 3 shows that the NP–hardness also holds for the undirected case. Further ore, a new construction in the proof enables us to derive two bounds on the approxi ability. To si plify notation, we denote the proble MkSF with $k = 2$ by M2SF, as well for other values of k.

Theorem 3. *For all constant $k \geq 2$,* MkSF *is strongly NP–hard and cannot be approximated with performance guarantee better than $\frac{k}{k+1}$, unless $P = NP$.*

Proof. First we give a reduction fro 3SAT to M2SF and show that a satisfiable instance of 3SAT yields an opti u solution of value 3 whereas a nonsatisfiable instance yields an opti u solution of value 2 for the corresponding M2SF–instance. Later we extend the reduction to any constant $k \geq 2$.

Consider a 3SAT–instance with variables $x_1, ..., x_r$ and clauses $C_1, ..., C_q$. In the following we construct the corresponding M2SF–instance in two steps illustrated in Figures 1 and 2.

Step 1, Figure 1 (top): The graph constructed in this step represents the clauses of the 3SAT–instance. Introduce two nodes s and t and two nodes a_j, b_j for every clause C_j. For every litaral of C_j we construct an a_j, b_j–path, which we initialize with one edge $\{a_j, b_j\}$. These paths will be expanded in Step 2. Connect the clause representations by the $q+1$ edges $\{s, a_1\}, \{b_1, a_2\}, \{b_2, a_3\}, ..., \{b_q, t\}$. All edges created in this step get capacity 1. The construction so far allows s,t–paths traversing each clause along one path representing one literal of the clause. To control that such s,t–paths do not use paths that belong to contrary literals we introduce a blocking construction in Step 2.

Step 2, Figure 1 (bottom): Assume that there are h pairs of contrary literals x_i and $\bar{x}_i$, which belong to different clauses. Consider the l-th pair and assume that x_i appears in a clause C and $\bar{x}_i$ in a clause C'. Insert one edge $\{y_l, z_l\}$ into an edge $\{u, v\}$ of unit capacity of the path representing x_i. The new edges $\{u, y_l\}$ and $\{z_l, v\}$ get a capacity of 1 and the edge $\{y_l, z_l\}$ gets a capacity of 2. Analogously, insert an edge $\{y'_l, z'_l\}$ into an edge $\{u', v'\}$ of unit capacity of the path representing $\bar{x}_i$. Introduce two nodes c_l and d_l and edges $\{c_l, y_l\}, \{d_l, z_l\}, \{c_l, y'_l), \{d_l, z'_l\}$ with capacities 2 to get a blocking construction for the l-th pair of contrary literals. To complete the construction we add edges $\{s, c_1\}, \{d_1, c_2\}, \{d_2, c_3\}, ..., \{d_{h-1}, c_h\}, \{d_h, t\}$, also with capacities 2.

Figure 2 shows the entire construction for an example instance. Notice, that this reduction is of polynomial size because the number of nodes is at most quadratic in the number q of clauses and the maximum degree of a node is 4. Furthermore, any s,t–flow has a value less than or equal to 3, because the edges incident to s have together a capacity of 3. Next we show that any 2–splittable flow with a value greater than 2 implies the satisfiability of the 3SAT–instance.

Let us consider two s,t–paths which together carry a flow of value greater than 2. So there is a path P_1 with flow value greater than 1, which therefore can only use edges of capacity 2. Such edges only occure in the blocking construction of contrary literals and because of the structure of the graph P_1 must traverse all these constructions. The second path P_2 must be disjoint from P_1 because all edge capacities are bounded by 2. So it has to traverse all clause representations constructed in Step 1. While traversing the clauses, P_1 never sends flow along paths representing contrary literals simultaneously because P_2 blocks at least one of them. Refering to the 3SAT instance, set $x_i := 1$ if P_2 traverses an a_j, b_j–path representing x_i in one arbritrary clause C_j and otherwise set $x_i := 0$. Then, every variable is set to 0 or 1 and every clause contains one true literal. So we have described a satisfying assignment for the 3SAT–instance.

On the other hand, every satisfiable 3SAT instance implies a 2–splittable flow of value 3. Choose one satisfied literal for each clause in a satisfying assignment. Route one unit of flow along a path P_2 traversing the representations of the clauses always along the path of the chosen literal. Afterwards, we send two units of flow along the blocking constructions using one s,t–path P_1. This is possible because P_2 never traverses paths of contrary literals simultaneously. We get a 2–splittable s,t–flow of value 3.

Thus, 3SAT can be reduced to M2SF and a 3SAT–instance is satisfiable if and only if a maximum 2–splittable flow has value 3 and is not satisfiable if and only if the maximum value is 2.

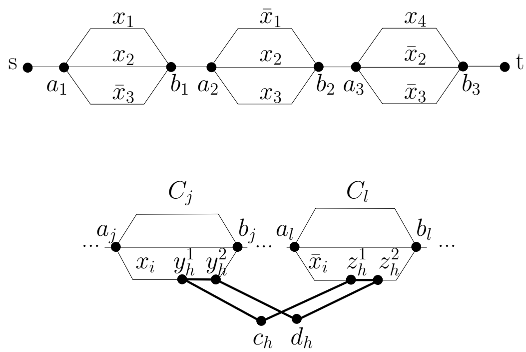


Fig. 1. Step 1 (top) and step 2 (bottom) for the 3SAT instance $x_1 \vee x_2 \vee \bar{x}_3, \quad \bar{x}_1 \vee x_2 \vee x_3, \quad \bar{x}_2 \vee \bar{x}_3 \vee x_4$

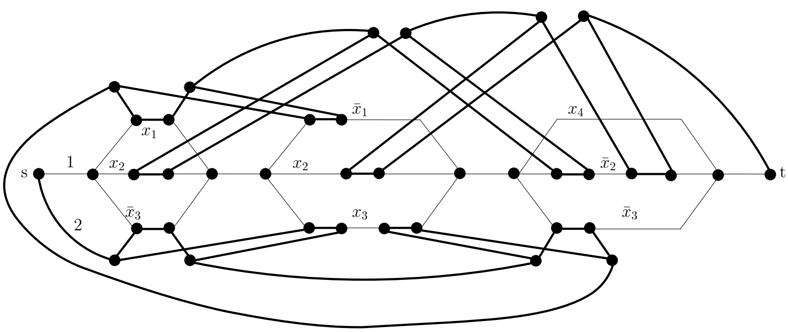


Fig. 2. Entire construction for the 3SAT instance $x_1 \vee x_2 \vee \bar{x}_3, \quad \bar{x}_1 \vee x_2 \vee x_3, \quad \bar{x}_2 \vee \bar{x}_3 \vee x_4$

To extend the reduction to all constant $k \geq 2$ we add $k-2$ s,t–edges with capacity 1. Then a 3SAT instance is satisfiable if and only if a k–splittable flow has a axi u value of $k+1$ and is not satisfiable if and only if the axi u value is k. So MkSF is strongly NP–hard (because of the NP–hardness of 3SAT an the constantly bounded capacities in the reduction) and cannot be approxi ated with guarantee better than $\frac{k}{k+1}$, unless $P = NP$. □

Corollary 3. MkSF, $k \geq 2$, *cannot be approximated with performance guarantee better than* $\frac{5}{6}$, *unless* $P = NP$. *(Proof omitted.)*

4.2 k as a Function of m and n

Here, we consider k as a function of the nu ber of edges m and of the nu ber of nodes n of a graph G. Note, that k is not seen as a part of the input, but a property of the proble MkSF. Thus, for different functions k we consider different proble s. So e functions result in polyno ially solvable proble s.

Lemma 3. *For a graph G,* MkSF *with $m - n + 2 \leq k$ is polynomial solvable.*

Proof. We show, that any maximum s,t–flow f in G can be decomposed into at most $m - n + 2$ path and cycles in polynomial time. Consider an orientation of the edges of G such that f is still a feasible flow and add an edge (t, s) of infinite capacity to obtain a directed graph G'. Setting the flow on the edge (t, s) to value(f) results in a circulation f' in G'. Each decomposition of f' in cycles easily yields a decomposition of f in paths and cycles with the same number of elements.

We compute a decomposition of f' with the standard decomposition algorithm of Fulkerson. That means, start with any flow carrying edge and go through G' only using edges with a positive amount of flow until a cycle is closed. Assign the maximal possible flow value to this cycle with respect to f' and reduce f' by this cycle flow. Repeat this procedure until $f' = 0$. Since in any iteration the flow value of at least one edge is decreased to 0 the incidence vectors of these cycles are linearly independent. Furthermore, the cycle space of G' has a dimension of $m + 1 - n + 1 = m - n + 2$ such that the computed decomposition of f' contains no more than $m - n + 2$ cycles. □

In the following, we show that MkSF is NP–hard for all k with $2 \leq k \leq m - n + 1$. This is done in two steps. We prove the NP–hardness for $2 \leq k \leq m - m^{\epsilon}$ by a reduction from 3SAT and then for $m^{\epsilon} \leq k \leq m - n + 1$ by a reduction from SubsetSum. In both cases $\epsilon \in (0, 1)$.

Theorem 4. *For all constant $\epsilon \in (0, 1)$* MkSF *with $2 \leq k \leq m - m^{\epsilon}$ is strongly NP–hard and cannot be approximated with a guarantee better than $\frac{k}{k+1}$, unless $P = NP$.*

Proof. Given $\epsilon \in (0, 1)$ we reduce 3SAT to MkSF with a k arbitrary in the range $2 \leq k \leq m - m^{\epsilon}$.

According to Theorem 3 it suffice to show that the graph G consisting of the graph G' shown in Figure 2 together with $k - 2$ additional s,t–edges from is of polynomial size in relation to the size of the considered 3SAT instance. Let m' be the number of edges of G'. Then we have $m = k - 2 + m'$ and it follows:

$$m \leq m - m^{\epsilon} - 2 + m' \quad \Rightarrow \quad m^{\epsilon} \leq m' - 2 \quad \Rightarrow \quad m \leq (m' - 2)^{\frac{1}{\epsilon}}.$$

Thus, m is polynomial in m' and because m' is polynomially bounded it holds also for G. □

Theorem 5. MkSF *with $k = m - n + 1$ is NP–hard for every given $n > 2$. (Proof omitted.)*

Corollary 4. *Given $\epsilon \in (0, 1)$* MkSF *with $m^{\epsilon} \leq k \leq m - n + 1$ is NP–hard for every given $n > 2$. (Proof omitted.)*

Corollary 5. MkSF *with $2 \leq k \leq m - n + 1$ is NP–hard for all graphs with $n > 2$.*

Proof. Choose $\epsilon := \frac{1}{3}$. Theore 4 proves the NP–hardness of MkSF for $2 \leq k \leq m - m^{1/3}$. Corollary 4 shows the NP–hardness for $m^{1/3} \leq k \leq m - n + 1$. Since $m^{1/3} \leq m - m^{1/3}$ for $m \geq 3$ the bounds overlap what proves the corollary. ($m \leq 2$ does not allow any k here) □

References

1. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows. Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.
2. S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8:277–284, 1987.
3. S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.
4. A. Bagchi, A. Chaudhary, P. Kolman, and C. Scheideler. Algorithms for faulttolerant routing in circuit-switched networks. In *Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 265–274. ACM Press, 2002.
5. G. Baier. *Flows with Path Restrictions*. PhD thesis, TU Berlin, 2003.
6. G. Baier, E. Köhler, and M. Skutella. On the k-splittable flow problem. *Algorithmica*, 42:231–248, 2005.
7. H. L. Bodlaender. Dynamic programming on graphs of bounded treewidth. In *Proceedings 15th International Colloquium on Automata, Languages and Programming*, volume 317 of *Lecture Notes in Computer Science*, pages 105–118. Springer, Berlin, 1988.
8. H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.
9. H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In I. Privara and P. Ruzicka, editors, *Proceedings 22nd International Symposium on Mathematical Foundations of Computer Science*, volume 1295 of *Lecture Notes in Computer Science*, pages 19–36. Springer, Berlin, 1997.
10. L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.
11. Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizations of k-terminal flow networks and computing network flows in partial k-trees. In *Proceedings 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 641–649, 1995.
12. J. M. Kleinberg. *Approximation algorithms for disjoint paths problems*. PhD thesis, M.I.T., 1996.
13. P. Krysta, P. Sanders, and B. Vöcking. Scheduling and traffic allocation for tasks with bounded splittability. In *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science*, volume 2747, pages 500–510. Springer, Berlin, 2003.
14. M. Martens and M. Skutella. Flows on few paths: Algorithms and lower bounds. In S. Albers and T. Radzik, editors, *Algorithms — ESA '04*, volume 3221 of *Lecture Notes in Computer Science*, pages 520–531. Springer, Berlin, 2004.
15. N. Robertson and P. D. Seymour. Graph minors. ii: algorithmic aspects of treewidth. *Journal of Algorithms*, 7:309–322, 1986.
16. P. Scheffler. A practical linear time algorithm for disjoint paths in graphs with bounded tree–width. Technical Report 396, FU Berlin, Fachbereich 3 Mathematik, 1994.

On Minimizing the Maximum Flow Time in the Online Dial-a-Ride Problem

Sven O. Kru ke[1], Wille E. de Paepe[2], Diana Poensgen[3],
Maarten Lip ann[2], Alberto Marchetti-Spacca ela[4], and Leen Stougie[2]

[1] University of Kaiserslautern, Department of Mathematics, P.O. Box 3049, Paul-Ehrlich-Str. 14, 67653 Kaiserslautern, Germany
krumke@mathematik.uni-kl.de
[2] Combinatorial Optimization Group, Technical University of Eindhoven, P.O. Box 513, 5600MB Eindhoven, The Netherlands
{leen, w.e.d.paepe}@win.tue.nl
[3] Konrad-Zuse-Zentrum für Informationstechnik Berlin, Department Optimization, Takustr. 7, 14195 Berlin, Germany
poensgen@zib.de
[4] Dipartimento di Informatica e Sistemistica, Università di Roma “La Sapienza”, Via Salaria 113, 00198 Rome, Italy
alberto@dis.uniroma1.it

Abstract. In the online dial-a-ride problem (OlDarp), objects must be transported by a server between points in a metric space. Transportation requests (“rides”) arrive online, specifying the objects to be transported and the corresponding source and destination.

We investigate the OlDarp for the objective of minimizing the maximum flow time. It has been well known that there can be no strictly competitive online algorithm for this objective and no competitive algorithm at all on unbounded metric spaces. However, the question whether on metric spaces with bounded diameter there are competitive algorithms if one allows an additive constant in the definition competitive ratio, had been open for quite a while. We provide a negative answer to this question already on the uniform metric space with three points. Our negative result is complemented by a strictly 2-competitive algorithm for the Online Traveling Salesman Problem on the uniform metric space, a special case of the problem.

1 Introduction

In the Dial-a-Ride Proble (Darp), a server of unit capacity has to transport objects through a given etric space. The server starts at a designated point of the etric space, its *origin*. Once the server has picked up an object, it can only drop it at its destination. A special case of the Darp is the Traveling Sales an Proble (Tsp) in which the server erely has to visit points in that etric space.

In the online version of the proble , *online Dial-a-Ride problem* requests (also called *rides*) arise while the server is already oving. Each request r_j specifies a

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 258–269, 2006.

release time $t_j \geq 0$, a source u_j, and a destination v_j. An online algorithm only learns about r_j at its release time. The objective we consider is to minimize the *maximum flow time*. If request r_j is served at time t, its flow time is $t - t_j$. We abbreviate the resulting problem by $F_{\max}$-OlDarp. The maximum flow time can be identified with the maximal dissatisfaction of customers who are waiting to be transported or to receive a desired good. Natural applications include elevator control, delivery services, and craftsmen on duty.

Given a sequence $\sigma = r_1, \ldots, r_m$ of requests, we denote by ALG(σ) the maximum flow time in the solution provided by algorithm ALG. We evaluate the quality of online algorithms by competitive analysis [4]: an online algorithm ALG is *c-competitive*, if there exists $b \geq 0$ such that

$$\text{ALG}(\sigma) \leq c \cdot \text{OPT}(\sigma) + b \tag{1}$$

holds for all request sequences σ, where OPT denotes an optimal offline algorithm, which has complete knowledge about the input at time 0. If (1) holds with $b = 0$, then ALG is termed *strictly c-competitive*. Competitive analysis can be imagined as a game between an online player and a malicious *adversary* who tries select a worst-case request sequence which maximizes the ratio between the online and the offline cost.

Online Dial-a-Ride problems have been previously investigated with various objective functions and for different metric spaces [2, 1, 6, 12, 5, 9, 11, 3, 10]. It is well known that for the $F_{\max}$-OlDarp in general metric spaces no strictly competitive algorithms can exist, see e.g. [12, 7, 8]. For the special case of the $F_{\max}$-OlTsp, where source and destination for each ride coincide ($u_j = v_j$ for all j), a restriction on the adversary allows for a strictly competitive algorithm on the real line [12].

In this paper we study we study the $F_{\max}$-OlDarp and $F_{\max}$-OlTsp on the *uniform metric space* with n points, where any two distinct points have distance one. This can be envisioned as a complete graph K_n with unit length edges. Only the n points can occur as source or destination. We allow servers to move continuously at unit speed along the edges.

Observe that with $b = n$ in (1), a simple online algorithm that visits the n nodes of the uniform metric space in a round-robin manner is 1-competitive for the $F_{\max}$-OlTsp. For the $F_{\max}$-OlDarp, however, it had been an open question whether allowing an additive constant $b > 0$ allows to prove (positive) competitiveness results. We resolve this question.

1.1 Contribution and Paper Outline

In this paper we show that on the uniform metric space with $n = 3$ points neither an arbitrary additive constant b nor restricting the adversary to be fair in the sense of [12, 3] allows for competitive algorithms. On the uniform metric space an adversary is *fair*, if at any moment $t > 0$, her server is located between two points, each of which is the origin or has occurred as source or destination of a request with release time at most t (this definition extends the notion of fairness given in [3] for the real line).

We also investigate the $F_{\max}$-OLTSP on the unifor etric space against the fair adversary and prove that a si ple first-co e-first-serve strategy is strictly 2-co petitive, which we prove to be best possible for online algorith s.

2 A Negative Result for the $F_{\max}$-OlDarp

In this section we shall prove the following theore :

Theorem 1. *For the $F_{\max}$-OLDARP on the uniform metric space, no deterministic online algorithm can be competitive even against a fair adversary.*

Proof. Assu e for the sake of a contradiction that ALG is a c-co petitive deter- inistic algorith with additive constant b as in (1). W. l. o. g., we ay assu e that c is integral. In the sequel we will also assu e that at integral points in ti e, ALG's server is located in one of the nodes. Note that any online algorith can be tranfor ed into another online algorith with this property at the cost of an additive constant of one. So, this assu ption is without loss of generality for the proof.

We show that for any $k \in \mathbb{N}$ we can construct a finite sequence $\sigma = \sigma(k)$ such that $\text{OPT}(\sigma) \leq 3$, whereas $\text{ALG}(\sigma) \geq k$. This contradicts the fact that ALG is c-co petitive and also rules out any additive constant (one just has to choose k appropriately depending on c and b).

Our construction just uses the subgraph of K_n induced by the origin and two additional points. The resulting three points and edges are denoted by x, y, z and X, Y, Z, see Figure 1. Our sequence has the following properties: (i) Requests are only given at integer ti es, (ii) at any ti e unit, at ost one request is given, and, (iii) in any two consecutive ti e units, at least one request is given.

A crucial ingredient for our construction is that of an *empty move*, in which a server oves fro a node u to so e other node v without serving a request (t, u, v). The ain idea is to enforce an e pty ove for the online algorith which can be avoided by the adversary. This way, work piles up for ALG, while the nu ber of unserved requests for the adversary re ains bounded by a constant (na ely, three).

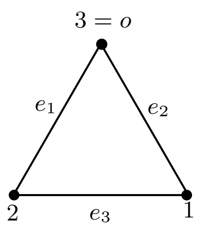


Fig. 1. Metric space and notation for the lower bound construction

In the sequel we describe a request by the corresponding edge and a direction, which depends on the direction of the previous request for that edge. Two requests for the sa e edge with different directions are called *opposed*. We use $\sigma_{\leq t}$ to denote the requests in σ with release ti e at ost t.

The sequence is constructed in phases. In each phase, the nu ber of unserved requests for the online algorith increases by one. More precisely, the kth phase starts at ti e T_k when the following property P_k is satisfied:

Property P_k: There exists $i \in \{1, 2, 3\}$ such that:

(i) ALG has k unserved requests for e_i, none for all the other edges, and is located in one of e_i's end points, $\lceil k/2 \rceil$ requests directed away fro ALG's position, and $\lfloor k/2 \rfloor$ requests directed the other way.
(ii) $\text{ADV}(\sigma_{\leq T_k}) \leq 3$, and ADV's server is located in one of e_i's end points with exactly two opposed requests for e_i pending, one of which has been released at T_k, the other one at ti e $T_k - 1$.

Figure 2 displays the scenario described by Property P_k.

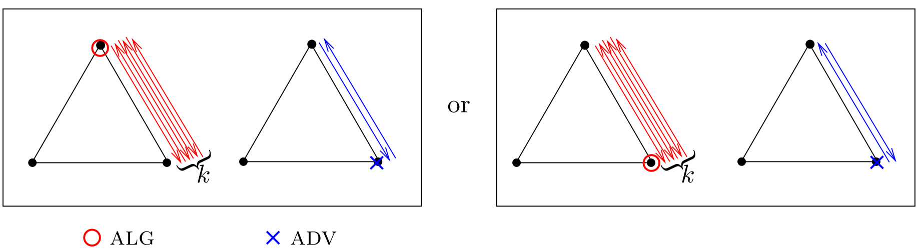


Fig. 2. Property P_k

Claim 1. *Assume that P_k holds at time T_k. Then, the adversary can release further requests such that, at some time $T_{k+1} \geq T_k$ Property P_{k+1} holds.*

Proof. Proof of Clai 1 Assu e that Property P_k holds at ti e T_k with $i = 2$ and that the adversary is positioned at node 1 (the other cases are sy etric). We have to distinguish two cases depending on the parity of k.

Case 1: If ALG served all pending e_2-requests in a row, it would finish in node 3 (left part of Figure 2 if k is even; right part of the figure if k is odd).

The sequence continues with a request for e_3 (direction arbitrary) at ti e $T_k + 1$. The adversary then releases one request for e_3 at each integer ti e (directions alternating), until at so e ti e T ALG has served the last pending e_2-request and is located in one of e_3's end points, either node 1 or node 2. Observe that such a ti e ust exist since otherwise ALG can not be co petitive at all). Let $T' \leq T$ be the earliest ti e when ALG has served all pending e_2-requests. At ti e T', ALG is either in node 1 or in node 3. In the for er case, ALG ust have ade an e pty ove: it ust have either oved along e_2 without serving a request or it has oved around the triangle and ade an e pty ove along e_1 (this uses the assu ption of Case 1). We proceed to show that Property P_{k+1} holds at ti e $T_{k+1} := T$.

Starting at ti e T_k, ADV serves the two pending requests for e_2 and then serves one request for e_3 in each ti e unit. Clearly, at ti e T, she is located either in 1 or 2 and has two pending requests for e_3, one released at ti e $T - 1$ and the other one at T. This ensures part (ii) of Property P_{k+1}.

To prove (i), we co pute how any unserved e_3-requests have piled up for ALG by ti e T. ALG has at least k requests for e_2 pending at ti e T_k. Since it can serve at ost one request per ti e unit and ust serve k pending requests on e_2, at ti e T' at least k requests for e_3 have piled up (no atter whether so e of the e_3-requests have been served before ti e T').

If $p^{\text{ALG}}(T') = 1$, then $T = T'$ and as we have seen above, ALG ust have ade an e pty ove which results in an extra unserved request for e_3. If $p^{\text{ALG}}(T') = 3$, ALG needs at least one e pty ove in order to reach node 1 or 2 at ti e $T \geq T' + 1$. In any case, at ti e T ALG has at least $k + 1$ pending requests for e_3. Hence, part (i) of Property P_k follows.

Case 2: If ALG served all pending e_2-requests in a row, it would finish in node 1.

In this case the sequence is continued until the prerequisites of Case 1 are et and we can continue as described above (with a suitable cyclic exchange of indices)

At ti e $T_k + 1$, no request is given. Starting at ti e $T_k + 2$, the adversary gives one request for edge e_1 in each ti e unit (directions alternating) until ti e τ, the earliest ti e at which ALG has finished serving the last a ong the pending e_2-requests and is located in one of e_1's end nodes, either 2 or 3. Let $\tau' \leq \tau$ be the earliest ti e at which ALG has finished serving the pending e_2-requests. Again, the existence of $\tau \in \mathbb{N}$ is ensured by the assu ption that ALG is co petitive.

ALG can serve at ost one of the k pending e_2-requests in each ti e unit. Since one new request for e_1 is given in each ti e unit except for $T_k + 1$, ALG has at least $k - 1$ unserved e_1-requests at ti e τ. Moreover, by sa e reasoning as before, the assu ption in Case 2 yields that ALG ust have ade an e pty ove if it ends up in node 3 at ti e τ', resulting in one extra request for e_1 at ti e τ' and in $\tau = \tau'$. Otherwise, $p^{\text{ALG}}(\tau') = 1$ and ALG ust ake an e pty ove to reach one of e_1's end points at ti e $\tau > \tau'$. In either case, ALG has at least k unserved requests for e_1 at ti e τ.

The adversary serves her pending requests as follows: first, she handles the two pending e_2-requests, ending up in node 1 at ti e T_{k+2}. Now, and this is the ain difference to Case 1, she ust also ake an e pty ove before she can start to serve the e_1-requests at ti e T_{k+3}. This e pty ove can be either to node 2 or node 3 and deter ines whether $p^{\text{ADV}}(\tau) = 2$ or $p^{\text{ADV}}(\tau) = 4$. No atter how the e pty ove is done, at ti e τ there will be exactly two pending requests for the adversary at e_1. The choice is ade according to the following rule: Let v denote the position of ALG's server if starting at ti e τ it would serve all pending e_1-requests in a row; then the adversary akes the e pty ove such that $p^{\text{ADV}}(\tau) \neq v$. Thus, at time τ we are in the situation of Case 1 (with a proper shift in indices). □

In order to prove Theorem 1 by applying Claim 1, we construct the beginning of the sequence in such a way that Property P_k is satisfied for some $k \geq 1$.

To this end starting at time 0, the adversary issues one request for edge e_3 in each time unit (directions alternating), until the online server is located either in node 1 or 2 for the first time. Let this time be $t \in \mathbb{N}$. At time 0, the fair adversary can move her server to the source of the first request issued, either 1 or 2, arriving there at time 1 with two pending requests for edge e_3. Then, she continues to serve one request for e_3 in every time unit until time t.

When ALG reaches 1 or 2 at time t, it must hold that $t \geq 1$, and $t+1$ unserved requests for e_3 have piled up in the meantime. Thus, Property P_{t+1} holds at time t for some $t \geq 1$.

This completes the proof of the theorem. □

The construction used in the proof of Theorem 1 also works for even stronger restrictions on the adversary. It can be seen that the adversary used above is even non-abusive in the sense of [12]: besides serving requests she only moves to sources of unserved requests. Finally, Theorem 1 also holds for servers of larger, but finite capacity $K > 1$: we simply multiply each request in the sequence by K, that is, we give each request K times.

3 The $F_{\max}$-OlTsp Against the Fair Adversary: An Easy 2-Competitive Algorithm

We now consider the special case of the $F_{\max}$-OLDARP where source and destination of each ride coincide: the $F_{\max}$-OLTSP. The main difference to the previous section is that a server can serve an unlimited number of requests simultaneously if these requests specify all the same node to be visited. It is easy to see that even on the uniform metric space with at least two points the standard unrestricted adversary can construct sequences where it achieves a zero maximum flow time whereas any deterministic online algorithm has a positive flow time for some request, see also [12, 7, 8]. Consequently, there can not be any strictly competitive algorithm. As already mentioned in the introduction allowing an additive constant $b = n$, that is, equal to the number of nodes in the space, allows for a trivial 1-competitive algorithm.

We therefore consider the $F_{\max}$-OLTSP against the fair adversary.

Theorem 2. *The algorithm first-come first-serve (*FCFS*) which always serves an oldest unserved request next is strictly* 2*-competitive against the fair adversary for the* $F_{\max}$-OLTSP *on the uniform metric space.*

Before we can prove Theorem 2, we need to establish an elementary lemma.

Lemma 1. *Given a sequence* $\sigma = r_1, \ldots, r_m$*, let* σ_i *denote the subsequence of* σ *that contains the first* i *requests, i.e.,* $\sigma_i = r_1, \ldots, r_i$*. Then* $\text{OPT}(\sigma_i) \leq \text{OPT}(\sigma)$ *for any* $i \in \{1, \ldots, m\}$*, where* OPT *refers to a fair adversary.*

Proof. Note that, for the standard adversary, the claim is trivial and holds for any subsequence of σ in place of σ_i. For the fair adversary, however, we must be more careful. Removing some request from a sequence can in fact lead to an increased maximum flow time of the fair adversary, as that request might enlarge the space where the adversary is allowed to move to. To see this, assume that the request r that is removed from σ is the first request for node v. When serving the whole sequence σ, the adversary can benefit from r, if another request q for node v is given later on: he can already be waiting in the "allowed" node v when q request is released, thus incurring a smaller flow time for q and thereby possibly also for other requests to follow.

The described construction, however, is the only way how the fair adversary's maximum flow time can increase by removing a request, and it shows that the removal of a request can only affect the flow times of requests given later. Hence, removing the tail from a sequence cannot increase the fair adversary's flow time on the preceding requests. □

3.1 Competitiveness of fcfs

This subsection is dedicated to the proof of Theorem 2 on the competitiveness of FCFS. We first give an intuitive description of the proof. It is based on the following two ideas. First, in order to increase FCFS' flow time while keeping OPT's maximum flow time stable, the adversary must issue a second request for some node v shortly after the online server has left that node. The offline server in turn must be able to serve some other requests in the meantime and only arrive in v when it is requested the second time. Then, the second request for v requires the online server to move to v once more while the offline server can serve both requests for v simultaneously. The second useful idea is the following: If F^* denotes OPT's maximum flow time on a given sequence, the optimal offline algorithm must serve request r_i within the time window $[t_i, t_i + F^*]$. Consequently, once FCFS lags behind by F^* time units, meaning that there exists an unserved request for some node w that is older than F^* time units, the optimal offline server must serve that request before the online server, and can therefore not use another request for w to further increase FCFS' flow time while keeping OPT's stable.

For the formal proof, we need some further notation. We say that FCFS serves a request r_i for node v *for free*, if it serves an older request r_j for v together with r_i. By $F^{\text{FCFS}}(r_i)$, we denote the flow time of request r_i in FCFS' schedule. Note that F^* must be greater than 0 for any meaningful request sequence, since the adversary is fair.

Assume the claim is false and FCFS is not 2-competitive. Then there exists a request sequence on which FCFS' maximum flow time is more than twice as large as the maximum flow time of OPT on that sequence. Among all request sequences with this property, let $\sigma = r_1, \ldots, r_m$ be a shortest one with respect to the number of requests. We call this sequence σ a *shortest counterexample*.

Lemma 2. FCFS *does not serve any request in a shortest counterexample* σ *for free.*

Proof. If there is a request $r \in \sigma$ that FCFS serves for free, then $\tilde{\sigma} := \sigma \setminus \{r\}$ is a shorter sequence on which FCFS incurs the same flow time as on σ. Since r is served for free by FCFS, there must be an older request for the same node. Thus, r does not open up new space for the adversary and cannot be used to incur a smaller flow time on a later request (cf. the proof of Lemma 1). Therefore, the maximum flow time of the fair adversary cannot become larger by removing r from σ.

This contradicts our definition of σ as a shortest sequence on which FCFS has a maximum flow time that is more than twice as large as that of OPT. □

Our choice of $\sigma = r_1, \ldots, r_m$ as a shortest sequence on which FCFS incurs a maximum flow time more than twice as large as OPT's maximum flow time gives rise to another observation.

Lemma 3. *Let* F^* *denote the maximum flow time of* OPT *on the shortest counterexample* σ*. Only the last request* r_m *is served by* FCFS *with a flow time of more than* $2F^*$*.*

Proof. If FCFS served a request r_l for some $l < m$ with a flow time of more than $2F^*$, it would achieve a maximum flow time of more than $2F^*$ also on the cut sequence $\sigma_l = r_1, \ldots, r_l$. By Lemma 1, the maximum flow time of OPT on σ_l is at most F^*. Again, this contradicts the definition of σ. □

Consider the schedule of FCFS on σ. If FCFS serves some request r_i at time t with flow time $F^{\text{FCFS}}(r_i)$, then it incurs a flow time of at most $F^{\text{FCFS}}(r_i) + 1$ on the request it serves at time $t + 1$, since this request cannot be older than r_i. Moreover, since FCFS serves the last request in the sequence with a flow time of more than $2F^*$, there must be a time after which FCFS serves all requests with flow time at least F^*.

Define T to be the earliest time such that all requests served by FCFS at or after that time are served with a flow time of at least F^*. By the reasoning above, FCFS must be serving some request at each time $t \geq T$, until it serves r_m, since its flow time can increase by at most one from one request to the next one it serves.

Claim 2. *Let* r_i *be the request in the shortest counterexample* σ *that* FCFS *serves at time* T*. Then,* $F^{\text{FCFS}}(r_i) = F^*$*, and no request served by* FCFS *after time* T *is released before time* $T - F^*$*.*

Proof. By definition of T, we have that $F^{\text{FCFS}}(r_i) \geq F^*$. Assume that $F^{\text{FCFS}}(r_i) \geq F^* + 1$. Then, the release time of r_i satisfies $t_i \leq T - (F^* + 1)$. Let r_j be the request served by FCFS at time $T - 1$. Since FCFS does not serve any request in σ for free, and as FCFS serves r_j before r_i, it must hold that $t_j \leq t_i$, and we deduce that

$$F^{\text{FCFS}}(r_j) = T - 1 - t_j \geq T - 1 - t_i \geq T - 1 - (T - (F^* + 1)) = F^*,$$

which contradicts our choice of T. Consequently, $F^{\text{FCFS}}(r_i) = F^*$. This in turn i plies that $t_i = T - F^*$. If r_k is a request that FCFS serves after ti e T, that is, after it has served request r_i, then r_k cannot have been released earlier than r_i by construction of FCFS. Hence, $t_k \geq T - F^*$, as clai ed. □

Let L be the ti e at which OPT finishes serving the shortest counterexa ple σ. For the final proof of Theore 2, we represent the requests in σ by an $L \times L$- atrix M as follows:

$$M_{ij} := \begin{cases} 1, & \text{if so e request released at ti e } i \text{ is served at ti e } j \text{ by OPT}, \\ 0, & \text{otherwise.} \end{cases}$$

Figure 3 displays the structure of the atrix M. Since FCFS serves no request in σ for free, the sa e node cannot have been requested twice at the sa e ti e. Hence, each request is represented by exactly one non-zero entry of the atrix, and the non-zero entries of colu n $(\cdot, j)$ of M stand for requests specifying the sa e node (otherwise, OPT could not serve the si ultaneously at ti e j). Moreover, since OPT's axi u flow ti e on σ equals F^*, we know that $M_{ij} = 1$ only if $i \leq j \leq i + F^*$. We use M to prove the following clai , which is the key ingredient for the proof of Theore 2.

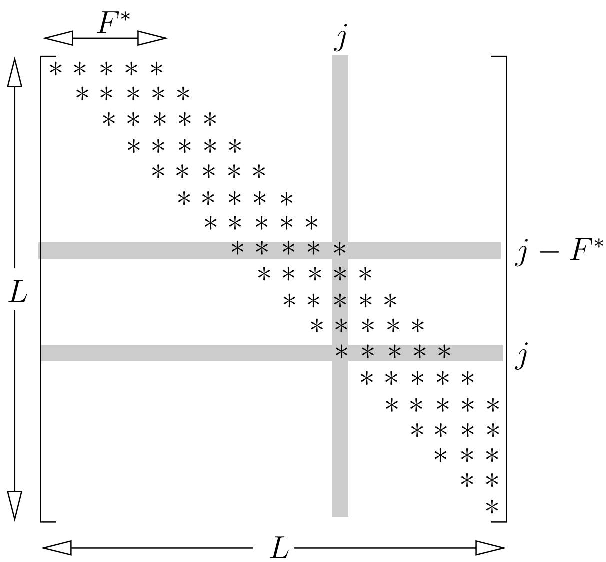


Fig. 3. The structure of the matrix M: entries $* \in \{0, 1\}$, all other entries are zero

Claim 3. FCFS *finishes serving the shortest counterexample* σ *no later than at time* $L + F^*$.

Proof. Let R denote the set of requests served by FCFS at or after time T. By Claim 2, the requests in R are all released at or after time $T - F^*$ and thus served by OPT not earlier than time $T - F^*$. Therefore, by construction of the matrix M, the non-zero entries representing the requests in R are all contained in columns $(\cdot, T - F^*), \ldots, (\cdot, L)$. As mentioned above, all requests belonging to the same column specify the same node. Hence, if we are able to show that FCFS serves all requests in R that are contained in the same column simultaneously, then the claim follows: there are $L - (T - F^*) + 1$ columns, and FCFS starts serving the requests in R at time T. Hence, it will be finished at time $L + F^*$ if it serves one column at a time.

To see that FCFS serves all requests in R represented in the same column simultaneously, recall that all requests corresponding to a non-zero entry of column $(\cdot, j)$ must have been released between time $j - F^*$ and j.

By definition of T, the requests in R are served with a flow time of at least F^*. In particular, that request from R in column $(\cdot, j)$ which is served first by FCFS can be served earliest at time $j - F^* + F^* = j$, since its release time is at least $j - F^*$. But at time j, all requests in column $(\cdot, j)$ have already been released, and can therefore be served simultaneously by FCFS. Thus, FCFS indeed serves all requests in R belonging to the same column simultaneously. □

We are now ready to derive the necessary contradiction that proves Theorem 2. Our assumption was that FCFS has a maximum flow time of at least $2F^* + 1$ on σ. By Claim 3, we know that FCFS incurs its maximum flow time on the last request in the sequence, r_m. Since OPT finishes serving at time L, the last request r_m cannot have been released before time $L - F^*$. Otherwise, OPT would incur a flow time of more than F^* on the request it serves at time L, since that request is at least as old as r_m. Thus, by Claim 3, the flow time that FCFS incurs on r_m is at most $L + F^* - (L - F^*) = 2F^*$, contradicting our initial assumption. Consequently, FCFS is 2-competitive. This proves Theorem 2. □

Remark 1. Observe that in the above proof of Theorem 2 we actually use the fairness of the adversary only to show that the optimal flowtime F^* is at least 1 on any non-trivial problem instance. This in turn is only used in the proof of Claim 2 where the existence of the request r_j is only guaranteed, if F^* is greater than 0.

The above results show that the fairness condition restricts the adversary's power sufficiently if the server only needs to visit points. Moreover, note that fairness is not required anymore if we do not ask for strict competitiveness, i.e., if we allow an additive constant $b \geq 1$ in the definition of competitiveness. In fact, the proof remains valid in this case, as can be easily checked.

3.2 A General Lower Bound

Theorem 3. *For the $F_{\max}$-OLTSP on a uniform metric space with at least five nodes, no deterministic online algorithm can be strictly c-competitive against the fair adversary with $c < 2$.*

Proof. Let ALG be an arbitrary deter inistic online algorith . The origin is assu ed to be in node v_0. Consider the following instance. First, the adversary gives a request for v_1 at ti e 0, and a request for v_2 at ti e 1. Clearly, at ti e 3, ALG has distance at least 1 to at least one of the nodes in $\{v_0, v_1, v_2\}$. Let $y \in \{v_0, v_1, v_2\}$ be that node. Si ilarly, there exists a node $z \in \{v_3, v_4, v_5\}$ such that $d(p^{\mathrm{ALG}}(3), z) \geq 1$. Then, at ti e 3, the adversary issues two ore requests: one for y and one for z. Thus, we have that $\sigma = r_1, r_2, r_3, r_4 := (0, v_1), (1, v_2), (3, y), (3, z)$. Figure 4 shows the requests of σ as points of a ti e-space diagra for $y = v_1$ and $z = v_3$.

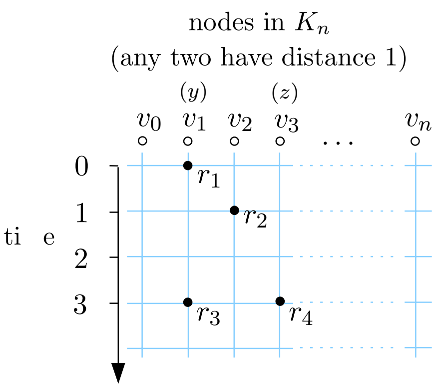


Fig. 4. The lower bound construction for the $F_{\max}$-OLTSP against a fair adversary

By construction, ALG has distance at least 1 to both nodes y and z at ti e 3. Therefore, it can have finished serving the last of those two requests earliest at ti e 5. Since both y and z have been released at ti e 3, ALG's axi u flow ti e is at least 2. On the other hand, the adversary can serve request r_1 at ti e 1, request r_2 at ti e 2, and then ove i ediately to y, serving r_3 at ti e 3 and r_4 at ti e 4. This gives OPT$(\sigma) = 1$, which eans that ALG is not better than 2-co petitive. Note that the adversary is indeed fair: at ti e 2, the "allowed" subgraph is induced by v_0 (the origin), v_1 and v_2. Hence, OPT ay ove to y i ediately after having served r_2. □

References

1. N. Ascheuer, S. O. Krumke, and J. Rambau. Online dial-a-ride problems: Minimizing the completion time. In *Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*, pages 639–650. Springer, 2000.
2. G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Algorithms for the on-line traveling salesman. *Algorithmica*, 29(4):560–581, 2001.

3. M. Blom, S. O. Krumke, W. E. de Paepe, and L. Stougie. The online-TSP against fair adversaries. *Informs Journal on Computing*, 13(2):138–148, 2001. A preliminary version appeared in the Proceedings of the 4th Italian Conference on Algorithms and Complexity, 2000, vol. 1767 of Lecture Notes in Computer Science.
4. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
5. W. E. de Paepe, J. K. Lenstra, J. Sgall, R. A. Sitters, and L. Stougie. Computer-aided complexity classification of dial-a-ride problems. *Informs Journal on Computing*, 2003. To appear.
6. E. Feuerstein and L. Stougie. On-line single server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001.
7. M. Grötschel, S. O. Krumke, and J. Rambau, editors. *Online Optimization of Large Scale Systems*. Springer, Berlin Heidelberg New York, 2001.
8. D. Hauptmeier, S. O. Krumke, and J. Rambau. The online dial-a-ride problem under reasonable load. In *Proceedings of the 4th Italian Conference on Algorithms and Complexity*, volume 1767 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2000.
9. S. Irani, X. Lu, and A. Regan. On-line algorithms for the dynamic traveling repair problem. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 517–524, 2002.
10. S. O. Krumke. *Online Optimization: Competitive Analysis and Beyond*. Habilitationsschrift, Technische Universität Berlin, 2002.
11. S. O. Krumke, W. E. de Paepe, D. Poensgen, and L. Stougie. News from the online traveling repairman. *Theoretical Computer Science*, 295(1–3):279–294, 2003. A preliminary version appeared in the Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science, 2001, vol. 2136 of Lecture Notes in Computer Science.
12. S. O. Krumke, L. Laura, M. Lipmann, A. Marchetti-Spaccamela, W. E. de Paepe, D. Poensgen, and L. Stougie. Non-abusiveness helps: An $O(1)$-competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, volume 2462 of *Lecture Notes in Computer Science*, pages 200–214. Springer, 2002.

Tighter Approximations for Maximum Induced Matchings in Regular Graphs

Zvi Gotthilf and Moshe Lewenstein

Bar-Ilan University

Abstract. An induced matching is a matching in which each two edges of the matching are not connected by a joint edge. Induced matchings are well-studied combinatorial objects and a lot of consideration has been given to finding maximum induced matchings, which is an NP-complete problem. Specifically, finding maximum induced matchings in regular graphs is well-known to be NP-complete. A couple of papers lately showed a couple of simple greedy algorithm that approximate a maximum induced matching with a factor of $d - \frac{1}{2}$ and $d - 1$ (different papers - different factors), where d is the degree of regularity. We show here a simple algorithm with an $0.75d + 0.15$ approximation factor. The algorithm is simple - the analysis is not.

1 Introduction

Let $G = (V, E)$ be a graph. A set of edges $M \subseteq E$ is an *induced matching* if it is a matching such that no two edges in the matching have a third edge connecting them. Equivalently, the subgraph of G induced by M consists of exactly M itself. Stockmeyer and Vazirani [13] introduced maximum induced matching as a variant of the maximum matching problem, and motivated it as the "risk free" marriage problem: find the maximum number of pairs such that each married person is compatible with no married person other than the one he or she is married to. Methods to find strong edge-colorings in a graph are based on finding large induced matchings (see - Erdös [4], Faudree, Gyarfas, Schelp and Tuza [5], Steger and Yu [12]). There is also an immediate connection between the size of an induced matching and the irredundancy number of a graph[8] (in [8] they were called *strong matchings*). On the practical side, induced matchings have the following applications for secure communication channels. Consider a bipartite graph $G = (X, Y, E)$ where edges represent communication capabilities between broadcaster nodes in X and receiver nodes in Y. We want to select k edges $e_i (i = 1, ..., k)$ such that messages on channel i will be passed from broadcaster $X(e_i)$ to receiver $Y(e_i)$ so that it is impossible for a message broadcast on channel i to be leaked or intercepted. Similar applications exist for VLSI and network flow problems.

Maximum induced matching is NP-complete even for bipartite graphs bounded by degree 4 [13]. Furthermore, Zito [14] shows that maximum induced matching is NP-complete for $4k$-regular graphs for each $k \geq 1$. Ko and Shepherd [11] found a close relationship between a maximum induced matching and

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 270–281, 2006.

ini u do inating set. For certain special classes axi u induced atchings can be found in polyno ial ti e. For chordal graphs Ca eron[1] found a polyno ial ti e algorith . Golu bic and Laskar[8] give a polyno ial-ti e algorith for axi u induced atching in circular-arc graphs. Golu bic and Lewenstein [9] give polyno ial-ti e algorith s in interval di ension graphs, trapezoid graphs and coco parability graphs by transferring the proble into a axi u independent set proble . In addition, they present a linear-ti e algorith in interval graphs. Also Ca eron [2] shows polyno ial-ti e algorith s for polygon-circle graphs and asteroidal triple-free graphs. For trees there are several algorith s running in linear-ti e presented by Fricke and Lasker[6], Zito [14] and Golu bic and Lewenstein [9].

1.1 Approximation Results

Regarding the approxi ability of axi u induced atching, which is our ain interest, Zito [14] has shown that for every $k \geq 1$ there is a constant $c \geq 1$ such that approxi ating axi u induced atching within a factor c on $4k$-regular graphs is NP-hard, so axi u induced atching is APX-co plete for 4k-regular graphs. We will focus on the case of d-regular graph. Zito [14] show that for d-regular graphs axi u induced atching is approxi able within a factor of d - 1/2. Moreover Duckworth, Manlove and Zito [3] i proved this bound by presenting an approxi ation algorith for axi u induced atching in d-regular graphs, which has an asy ptotic perfor ance ratio of $d - 1$ for each $d \geq 3$. Both use a si ple greedy ethod. We present another si ple variant of greedy which achieves an $0.75d + 0.15$ approxi ation factor for the proble . The analysis is quite intricate and has several subtle points.

2 Definitions

We give a couple of definitions that are necessary to understand the algorith . Obviously, not all edges can be together in an induced atching. The following definitions categorize what kind of conflicts there are between edges relative to an induced atching.

Definition 1. *Let $G = (V, E)$ be a graph and $e = \{u, v\} \in E$ an edge. An edge $f = \{x, y\}$ is a* conflict edge *of e if either x or y is within distance 1 of u or v. A conflict edge is of* first-degree *if x or $y = u$ or v and is of* second-degree *if $\{x, y\} \cap \{u, v\} = \emptyset$.*

An edge is defined to be a conflict edge of itself (but is neither first-degree nor second-degree).

Definition 2. *Let $G = (V, E)$ be a graph and M an induced matching on G. We denote with $\mathcal{C}(e)$ the set of conflict edges of e and define the* conflict degree *of e to be the number of conflict edges of e, i.e. the conflict degree of e is equal to $|\mathcal{C}(e)|$.*

Fig. 1. First and Second Degree Edges

3 The Algorithm

The algorithm is a 2 stage greedy algorithm which works as follows. In the first stage we choose an arbitrary edge e whose conflict degree is $\leq 1.5d^2 - 0.5d$ (this number is instrumental in achieving the desired approximation factor). We move e into our matching M and remove e and all edges that conflict with e from the graph. We then search for another edge whose conflict degree is $\leq 1.5d^2 - 0.5d$ (in the graph with the removed edges). We repeat this process as long as we can find such edges. (See Appendix A for a detailed algorithm.)

It follows from the method that M is induced and we are left with a graph in which every edge has conflict degree $> 1.5d^2 - 0.5d$. This brings us to the second stage which is based on the following two simple rules.

Rule 1(e, M)	Comment: $e \notin M$
True,	if $M \cup \{e\}$ is an induced matching.
False,	otherwise.

Rule 2(e, e', e'', M)	Comment: $e \in M$, $e', e'' \notin M$
True,	if $M \cup \{e', e''\} - \{e\}$ is an induced matching.
False,	otherwise.

In stage 2 we iteratively apply Rule 1 until there are no more edges fulfilling Rule 1. In each application of Rule 1 we add the edge to our second stage matching M'. When there are no edges satisfying Rule 1 we seek a triplet of edges $e \in M'$, $e', e'' \notin M'$ that satisfy Rule 2. If we find one we update M' accordingly and return to Rule 1. We do this procedure until Rule 1 and Rule 2 cannot be applied anymore. See Appendix B for the detailed algorithm of Stage 2.

The final induced matching is $M \cup M'$.

Time and Correctness Analysis: First of all, it is easy to see that $M \cup M'$ is an induced matching. This follows since, in the first stage we remove all the conflicting edges in each round and in the second stage we follow the rules that make sure that an induced matching is maintained. The time is clearly polynomial as each the size of the induced matching increases in each round.

Note that stage 2 of the algorithm returns M'. The following observation refers to properties of M' that immediately follow from the applications of the rules.

Definition 3. *Let $G = (V, E)$ be a graph and M an induced matching on G. The M-*conflict degree *of an edge e is the number of conflict edges of e that belong to M. The M-conflict degree of e is equal to $|\mathcal{C}(e) \cap M|$.*

Observation 1. *The matching M' returned by stage 2 of the algorithm satisfies:*
1. There are no edges with M'-conflict degree equal to 0.
2. There is no pair of edges $e', e'' \in E'$ fulfilling the following criteria:
- e' and e'' do not conflict.
- Both e', e'' have M'-conflict degree equal to 1 and are in conflict with the same $e \in M'$.

4 Algorithm Analysis

To get a better understanding of the analysis before delving into the details note the following obvious observation.

Observation 2. *Let $G = (V, E)$ be a d-bounded graph (i.e. the degree of each vertex is $\leq d$). The conflict degree of any edge is at most: $2d^2 - 2d + 1$.*

This observation led Zito [14] to his analysis of a $d - \frac{1}{2}$ approximation. Simply reiterate on Rule 1 only. Since the graph is d-regular there are $nd/2$ edges. By Observation 2 the induced matching Zito finds is $\geq \frac{nd}{2(2d^2-2d+1)}$. Moreover, the size of the optimal induced matching is no more than $\frac{nd}{2(2d-1)}$. The ratio between the two is $d - \frac{1}{2}$ yielding the desired result.

Of course, if one could produce a tighter upper bound on the conflict degree of the matched edges on average then this would immediately yield a better bound on the approximation ratio. This is where the first stage comes into play trying to maximize profit from lightweight conflict degree edges. In general one can deduce the following regarding the first stage.

Lemma 1. *Let M be the resulting induced matching of the first stage. The size of M will be at least $|E1|/(1.5d^2 - 0.5d)$, where $E1$ is the group of edges extracted from G in the first stage.*

Proof: On each cycle of the first stage we remove at most $1.5d^2 - 0.5d$ edges which is the maximum number of conflict edges an edge can have if it was removed on this stage. On every cycle we enlarge M by one. The number of iterations will be at least $|E1|/(1.5d^2 - 0.5d)$. Therefore M will be at least $|E1|/(1.5d^2 - 0.5d)$. □

Hence, the main part of the analysis will be the analysis of the second stage. Our analysis works as follows. Every edge potentially has a conflict with $2d^2 - 2d + 1$ edges but, since it is stage 2, it must have a conflict with at least $1.5d^2 - 0.5d$ edges. We show that this constrains the form of the graph allowing to achieve a better approximation. To be formal we define for each edge e a potential conflict degree, i.e. the number of conflicting edges, and show that the more lax we are with the form in the immediate neighborhood of e the lower the potential.

Definition 4. *Let e be an edge in E. The* potential conflict degree *of an edge e is an upper bound on the number of conflicting edges that e may have.*

As mentioned, every edge has a potential conflict degree of $2d^2 - 2d + 1$ which can be lowered for certain graph structures that we detail below.

To gain from this we consider edges with M-conflict degree greater than one. When considering the ratio described above used to achieve an approximation factor, these edges are counted only by one of the edges of M' when evaluating the size of the induced matching. From arguments on the potential conflict degree it turns out that the number of edges with M'-conflict degree 1 chosen in stage 2 is not that large. This in turn forces the number of edges with M'-conflict degree ≥ 2 to be large. This yields the better approximation factor.

4.1 One-Neighborhoods

Before we begin analyzing the performance of the above-suggested algorithm, we need some terminology on conflict edges that relate specifically to the matching.

Following the discussion from the previous section, we will focus on edges that have M-conflict degree of one and in the final analysis we will cash in on those that have higher M-conflict degree.

Definition 5. *Let $G = (V, E)$ be a graph, $e = \{u, v\} \in E$ an edge. Define the* 1-neighbourhood(e) *to be the group of edges which have a conflict with e and have M-conflict degree* 1*.*

Our goal is to find out what is the maximum size of the $1 - neighbourhood(e)$ for each $e \in M'$. The following lemma, which we use later on, gives us our first structural limitation on the graph based on the M-conflict degrees of the edges.

Lemma 2. *Let M be a maximal (perhaps not maximum) induced matching. Let $e \in M$ and $e', e'' \in 1 - neighbourhood(e)$. Then there is no edge $e''' \notin 1 - neighbourhood(e)$ that connects between e' and e''.*

Proof: First we will prove that e''' cannot have an M-conflict degree > 1. Assume, by contradiction, that $e''' = \{u, v\}$ has an M-conflict degree ≥ 2, say with $e1$ and $e2$ from M. Since e''' connects between e' and e'', u and v also belong to e' and e''. Hence, $e1$ must conflict with either e' or e'' and, likewise, $e2$ must conflict with either e' or e''. So, at least one of e' and e'' conflicts with an edge from M other than e, a contradiction.

Similarly, if e''' has M-conflict degree 1, where $e1 \neq e$ is the edge from M conflicting e''', then either e' or e'' must conflict with $e1$ which is impossible. □

Conversely to the previous Lemma, we can show a relationship between edges of any given one-neighborhood of the second stage solution M'.

Lemma 3. *Let M' be the resulting induced matching from the second stage of the algorithm and let $e \in M'$. Every pair of edges $e', e'' \in 1 - neighbourhood(e)$ must conflict.*

Proof: Let us assu e that there is such a pair of edges $e', e'' \in 1-neighbourhood$ (e) such that e' and e'' do not conflict. We show a contradiction. Both e' and e'' have an M'-conflict degree equal to 1 and are in conflict with $e \in M'$. If e' and e'' do not have a conflict this contradicts the second part of Observation 1. Therefore e', e'' ust conflict. □

4.2 Choosing Sides

We are interested in bounding the size of the one-neighborhood of every edge of the atching which will give a tighter analysis than the one entioned i - ediately after Observation 2. To this end we will have it easier to deal with the first-degree conflicting edges and the second-degree conflicting edges separately. In fact, the first-degree edges is no ore than $2(d-1)$ whereas the set of second-degree edges can be as large as $2(d-1)^2$. Hence, we will ainly focus on bounding the second degree edges. In this subsection we partition the into three categories. The first of the three is trivial to handle and the other two co prise the next two subsections.

Notation 1. *Let $G = (V, E)$ be a graph, $e \in E$ an edge and e_i a conflict edge of e of first-degree. Denote $sec_i(e)$ to be the group of edges which are second-degree conflict edges of e and share a vertex with e_i.*

Definition 6 (Shared Edges). *Let $G = (V, E)$ be a graph and $e \in E$ an edge. An edge e' is said to be a* shared-edge *(relative to e) if (1) e' is a second-degree conflict edge of e and (2) there are two distinct edges e_i, e_j that are first-degree conflict edges of e such that e' shares a vertex with both. (see Figure 2).*

When it is clear fro context we drop the ention of "relative to e" and si ply call an edge a shared-edge.

Definition 7 (Diagonal Edges). *Let $G = (V, E)$ be a graph, $e = \{u, v\} \in E$ an edge, $e' \in sec_i(e)$, $e'' \in sec_j(e)$ (where $i \neq j$). Let e' and e'' be* diagonal-edges(e) *if they share a common vertex and e', e'' are not shared edges (relative to e).*

Let $e = \{u, v\} \in M'$, we will divide the conflict edges of e of second-degree into two group according to the generate vertex of e - the first group will be $sideU = \{\cup sec_i(e)\}$ such that e_i (a conflict edge of e of first-degree) contains

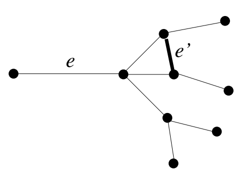


Fig. 2. Shared Edge

u as a vertex. The second group will be $sideV = \{\cup sec_j(e)\}$ such that e_j (a conflict edge of e of first-degree) contains v as a vertex. Note that there ay be so e edges that belong to both groups.

Recall that we desire to upper bound the size of the one-neighborhoods. We will bound the one-neighborhoods of $sideU$ and then conclude fro sy etric reasoning that $sideV$ has the sa e bound. In the last section of the paper, we erge all bounds together.

Let $e \in M'$ and $e', e'' \in \{1 - neighbourhood(e) \cap \{sideU - \{\text{shared edges}\}\}$ then e', e'' can have a conflict in only one of the following scenarios - depicted in Figure 3:

- Both $e' \in sec_i(e)$ and $e'' \in sec_i(e)$ - in this case e', e'' have a conflict of first degree.
- $e' \in sec_i(e)$ and $e'' \in sec_j(e)$ while $i \neq j$ and there is no shared edge between $sec_i(e)$ and $sec_j(e)$ - in this case e', e'' can have a conflict only if e' have a first-degree conflict (*diagonal-edges*) with at least one of the edges in $sec_j(e)$ or e'' have a first-degree conflict with at least one of the edges in $sec_i(e)$.
- $e' \in sec_i(e)$ and $e'' \in sec_j(e)$ while $i \neq j$ and there is an edge e''' such that $e''' \in \{sec_i(e) \cap sec_j(e)\}$ - e''' is a shared edge - in this case all the edges fro $\{sec_i(e) \cup sec_j(e)\}$ have a conflict.

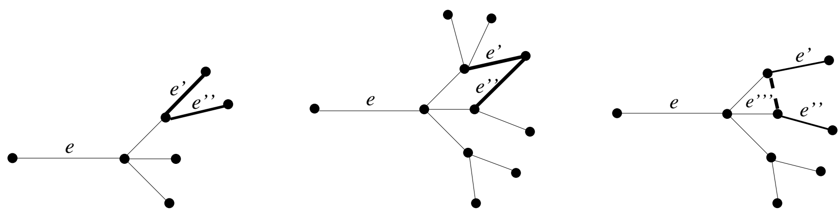


Fig. 3. The Three Cases

We point out that the second case is slightly ore inclusive than is depicted in the figure. Also, there is one ore case that we disregarded, na ely when e' and e'' share an edge $e''' = (u, v)$ where u is the endpoint of e' farther away fro e and v is the endpoint of e'' farther away fro e. This case cannot happen, see Le a 2.

4.3 Shared Edges

We now consider the case of shared edges.

Lemma 4. *A shared edge that connect between $sec_i(e)$ and $sec_j(e)$ such that both groups belongs to $sideU$ will decrease the potential conflict degree of e_i and e_j by at least d.*

Proof: First we will prove a simple property of d-bounded graphs. Let $e \in E$ be an edge which is part of a triangle then we claim that the potential conflict degree of e will decrease by at least d. The existence of a triangle does not affect the number of first degree conflict edges of e. However instead of having up to $2(d-1)$ second degree conflict edges of e (each of the edges e_i and e_j have at most $d-1$ second-degree neighbors) only $d-2$ second degree conflict edges of e can be produced by those two edges. Therefore the potential conflict degree of e will decrease by at least d.

Now let $e' = \{k, l\}$ be a shared edge that connect between $sec_i(e)$ and $sec_j(e)$ such that both groups belongs to *sideU* then $e_i = \{u, k\}, e_j = \{u, l\}$ and $e' = \{k, l\}$ create a triangle and therefore, the potential conflict degree of e_i and e_j will decrease by at least d. □

Lemma 5. *A shared edge that connects between* $sec_i(e)$ *and* $sec_j(e)$ *such that both groups belongs to sideU will increase the number of edges (from* $sec_i(e)$ *or* $sec_j(e)$*) counted to the 1-neighborhood(e) by at most d-2.*

Proof: Let $e' = \{k, l\}$ be a shared edge that connect between $sec_i(e)$ and $sec_j(e)$ such that both groups belongs to *sideU* then all edges of $sec_i(e)$ are connected to all edges from $sec_j(e)$. Both $|sec_i(e)| \leq$ d-1 and $|sec_j(e)| \leq$ d-1. Moreover e' is a shared edge. Therefore every edge from $sec_i(e)$ is now connected to at most d-2 edges from $sec_j(e)$ that do not belong to $sec_i(e)$ and vice versa. □

4.4 Diagonal Edges

Now let us analyze the case of *diagonal-edges*.

Lemma 6. *Let* $e' \in sec_i(e)$*,* $e'' \in sec_j(e)$ *be* diagonal-edges *such that both* $sec_i(e)$ *and* $sec_j(e)$ *belongs to sideU then the potential conflict degree of* e_i *and* e_j *will decrease by at least 1.*

Proof: Let $e \in E$ be an edge which is part of a square. We claim that the potential conflict degree of e will be decreased by at least 1. The existence of a square does not affect the number of first-degree conflict edges of e. However instead of having up to $2(d-1)$ second-degree conflict edges of e (produced by its two square neighbors) only $2d-3$ second-degree conflict edges of e can be produced by those two neighbor edges. Therefore the potential conflict degree of e will decrease by at least 1.

It is obvious, e', e'', e_i and e_j creates a square and therefore, the potential conflict degree of all four edges will decrease by at least 1. □

Lemma 7. *Let* $e' \in sec_i(e)$*,* $e'' \in sec_j(e)$ *be* diagonal-edges *such that both* $sec_i(e)$ *and* $sec_j(e)$ *belong to sideU then the size of the 1-neighborhood(e) will increase the number of edges (from* $sec_i(e)$ *or* $sec_j(e)$*) counted to the 1-neighborhood(e) by at most 1.*

Proof: Let $e_1 = \{k, l\} \in sec_i(e)$, $e_2 = \{l, m\} \in sec_j(e)$ be *diagonal-edges* that connect between $sec_i(e)$ and $sec_j(e)$ such that both groups belongs to *sideU* then e_1, e_2, e_i and e_j are four edges that create a "square". Every edge fro $sec_i(e)$ is now connected to e_2 and every edge fro $sec_j(e)$ is now connected to e_1 but this pair of edges doesn't connect between other edges fro both groups. Therefore every edge fro $sec_i(e)$ is now connected to only 1 edge (through this pair of diagonal edges) fro $sec_j(e)$ that doesn't belong to $sec_i(e)$ and vice versa. $\square$

4.5 Bounding the One-Neighborhood on One Side

Lemma 8. *Let $sec_i(e)$ be the group, among all the groups $sec_j(e)$ on $sideU$, with the maximum number of edges that have M-conflict degree 1 (not including shared edges).*

Then the number of edges with M-conflict degree 1 belonging to $\{sideU - \{\{$ shared edges$\}$ $\cup$ $sec_i(e)\}\}$ is at most $0.5d^2 - 1.5d + 1$.

Proof: There are two possible ways to "create" a conflict between all edges fro $sec_i(e)$ to other edges fro $\{sideU - \{\{$ shared edges$\}$ $\cup$ $sec_i(e)\}\}$:

- *shared edges* - in this case, by Le a 5, each shared edge adds at ost d-2 edges to the solution but, by Le a 4, the potential conflict degree of e_i decreases by d.
- *diagonal-edges* - by Le a 7, each pair of *diagonal-edges*(one of the edges ust be fro $sec_i(e)$) add exactly 1 edge to the solution but, by Le a 7, the conflict degree of e_i decreases by 1.

Therefore in order to increase by 1 the group of edges with M-conflict degree 1 belonging to $\{sideU - \{\{$shared edges$\} \cup sec_i(e)\}\}$ we will decrease the conflict degree of e_i by at least 1. If the nu ber of edges with M-conflict degree 1 belongs to $\{sideU - \{\{$shared edges$\} \cup sec_i(e)\}\} > 0.5d^2 - 1.5d + 1$ then the conflict degree of $e_i \leq 2d^2 - 2d + 1 - (0.5d^2 - 1.5d + 1) = 1.5d^2 - 0.5d$ but this is a contradiction, because after stage 1 there are no edges with conflict degree $\leq 1.5d^2 - 0.5d$. $\square$

Note that sy etrically the bound holds for *sideV*.

4.6 Bounding the Shared Edges

Lemma 9. *Each shared edge that connects between $sec_i(e)$ and $sec_j(e)$ will decrease the potential conflict degree of e by at least 1.*

Proof: Let $e' = \{k, l\}$ be a shared edge we will divide the proof into two cases:

- $sec_i(e) \in sideU$ and $sec_j(e) \in sideV$ or vice versa: $e_i = \{u, k\}$, $e_j = \{v, l\}$, $e' = \{k, l\}$ and $e = \{u, v\}$ create a "square" then according to the proof of Le a 6 the potential conflict degree of e will decrease by 1.
- $sec_i(e)$, $sec_j(e) \in sideU$ or $sec_i(e)$, $sec_j(e) \in sideV$. Let us assu e w.l.o.g $sec_i(e)$, $sec_j(e) \in sideU$: Let $e_i = \{u, k\}$, $e_j = \{u, l\}$ and $e' = \{k, l\}$ creates a "triangle". The existence of a triangle does not affect the nu ber of first-degree conflict edges of e. However instead of create up to $2(d-1)$ second-degree conflict edges of e (both e_i and e_j can produce at ost $d-1$ edges)

only $2d-3$ second-degree conflict edges of e can be produced by those edges because e' is "counted twice" as a second-degree conflict edge.

Therefore the potential conflict degree of e will decrease by at least 1. □

4.7 Putting the Approximation Bound Together

By Lemma 9, if the number of shared edges $> 0.5d^2 - 1.5d + 1$ then the conflict degree of $e \leq 2d^2 - 2d + 1 - (0.5d^2 - 1.5d + 1) = 1.5d^2 - 0.5d$ but this is a contradiction to the algorithm - after stage 1 there are no edges with conflict degree $\leq 1.5d^2 - 0.5d$. Therefore the number of shared edges can be at most $0.5d^2 - 1.5d + 1$. Thus, the number of edges with M-conflict degree 1 that are shared edges is at most $0.5d^2 - 1.5d + 1$.

The number of edges in $sec_i(e)$ can be at most $d-1$ and the number of edges that are first-degree to e together with e itself can be at most $2d-1$.

The average number of shared edges over all *1-neighbourhood(e)* such that $e \in M'$ is denoted with *shared*. By Lemma 8, the average size of the *1-neighbourhood(e)* can be at most: $2(0.5d^2 - 1.5d + 1) + 2(d-1) + 2d - 1 + shared = d^2 + d - 1 + shared$. Hence, the sum of M-conflict degrees, over all edges, is at least: $2|E'| - (d^2 + d - 1 + shared)|M'|$. We call the sum of M-conflict degrees, over all edges, the *M-conflict degree sum.*

Lemma 10. *Let $G' = (V', E')$ be a d-bounded graph and M' a maximum induced matching such the M'-conflict degree sum (of E') $\geq 2|E'| - (d^2 + d - 1 +$ shared$)|M'|$ then the size of M' is at least: $2|E'|/(3d^2 - d)$.*

Proof: The sum M-conflict degree of all edges is at least $2|E'| - (d^2 + d - 1 + shared)|M'|$. Now, choose an edge ($e$, from the solution) and for each conflict edge e' of e decrease the M-conflict degree of e' by one. If the M-conflict degree of an edge became 0, remove the edge from G'. At the beginning, the sum of the M'-conflict degree is at least $2|E'| - (d^2 + d - 1 + shared)|M'|$. Adding an edge to M' will, on average, decrease the sum of the M'-conflict degree by at most $2d^2 - 2d + 1 - shared$. Therefore, the size of M' will be at least: $[2|E'| - (d^2 + d - 1 + shared)|M'|] \;/\; [(2d^2 - 2d + 1 - shared)] \Rightarrow |M'|(2d^2 - 2d + 1 - shared) \geq 2|E'| - (d^2 + d - 1 + shared)|M'| \Rightarrow |M'|(3d^2 - d) \geq 2|E'| \Rightarrow |M'| \geq 2|E'|/(3d^2 - d)$. □

Theorem 1. *The approximation ratio of our algorithm is $0.75d + 0.15$.*

Proof: According to 1 the size M is at least: $|E1|/(1.5d^2 - 0.5d)$ and according to Lemma 10 the size of M' is at least: $2|E'|/(3d^2 - d)$. Therefore size of $M \cup M'$ is at least $2|E'|/(3d^2 - d) + |E1|/(1.5d^2 - 0.5d)$. Moreover we know $\{E' \cup E1\} = E \Rightarrow |E' \cup E1| = |E| = nd/2$. $|M'| + |M| \geq [2|E'|]/[(3d^2 - d)] + [nd/2 - |E'|]/[(1.5d^2 - 0.5d)] = nd/[(3d^2 - d)]$.

Therefore the approximation ratio will be: $[nd/2(2d-1)]/[nd/(3d^2 - d)] = 0.75d + 0.15$. (This is true for $d \geq 3$. If $d < 3$ then it is simple to solve exactly.) □

References

1. K. Cameron. Induced matchings. *Discrete Applied Mathematics*, 24:97-102, 1989.
2. K. Cameron. Induced matchings in intersection graphs. *Proceedings of the Sixth International Conference on Graph Theory, Marseille, France, August*, 2000.
3. W. Duckworth, D. Manlove and M. Zito. On the approximability of the maximum induced matching problem. *Journal of Discrete Algorithms*, 3(1):79-91, 2005.
4. P. Erdos. Problems and results in combinatorial analysis and graph theory. *Discrete Mathematics*, 72:81-92, 1988.
5. R.J. Faudree, A. Gyarfas, R.H. Schelp and Z. Tuza. Induced matchings in bipartite graphs. *Discrete Mathematics*, 78:83-87, 1989.
6. G. Fricke and R. Lasker. Strong matchings in trees. *Congressus Numerantium*, 89:239-244, 1992.
7. M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, 1980.
8. M.C. Golumbic and R.C. Laskar. Irredundancy in circular arc graphs. *Discrete Applied Mathematics*, 44:79-89, 1993.
9. M.C. Golumbic and and M. Lewenstein. New results on induced matchings. *Discrete Applied Mathematics*, 101:157-165, 2000.
10. P. Horak, H.Qing and W.T. Trotter. Induced matchings in cubic graphs. *Journal of Graph Theory*, 17(2):151-160, 1993.
11. C.W. Ko and F.B. Shepherd. Adding an identity to a totally unimodular matrix. *Working paper LSEOR.94.14, London School of Economics, Operational Research Group*, 1994.
12. A. Steger and M. Yu. On induced matchings. *Discrete Mathematics*, 120:291-295, 1993.
13. L.J. Stockmeyer and V.V. Vazirani. NP-completeness of some generalization of the maximum matching problem. *Information Processing Letters*, 15(1):14-19, 1982.
14. M. Zito. Induced matchings in regular graphs and trees. *Proc. of WG 1999, LNCS 1665*, 89-100, 1999.

A Algorithm - Stage 1

(1.) $M \leftarrow \emptyset$
(2.) Stay $\leftarrow$ True
(3.) While(Stay) $\{$
  (3.a) for every $e \in E$ compute the conflict degree of e.
  (3.b) choose an edge e with the minimum conflict degree.
  (3.c) if the conflict degree of $e \leq 1.5d^2 - 0.5d$
    (3.c.1) $M \leftarrow M \cup \{e\}$.
    (3.c.2) $E \leftarrow E - \mathcal{C}(e)$.
  (3.c') else
    (3.c'.1) Stay $\leftarrow$ False.
$\}$

B Algorithm - Stage 2

$M \leftarrow$.
Perform *Init* on $G(V, E)$ and M to receive $G'(V', E')$.
$M' \leftarrow$.
Perform *Rule1* on $G'(V', E')$ and M'.
Perform *Rule2* on $G'(V', E')$ and M' until switch was made or no such switch exist.
If a switch was made during *Rule2* return to *Rule1*.
Else, stop and return $\{M \cup M'\}$ as the induced matching.

On Approximating Restricted Cycle Covers*

Bodo Manthey**

Universität zu Lübeck, Institut für Theoretische Informatik,
Ratzeburger Allee 160, 23538 Lübeck, Germany
manthey@tcs.uni-luebeck.de

Abstract. A cycle cover of a graph is a set of cycles such that every vertex is part of exactly one cycle. An L-cycle cover is a cycle cover in which the length of every cycle is in the set L. A special case of L-cycle covers are k-cycle covers for $k \in \mathbb{N}$, where the length of each cycle must be at least k. The weight of a cycle cover of an edge-weighted graph is the sum of the weights of its edges.

We come close to settling the complexity and approximability of computing L-cycle covers. On the one hand, we show that for almost all L, computing L-cycle covers of maximum weight in directed and undirected graphs is APX-hard and NP-hard. Most of our hardness results hold even if the edge weights are restricted to zero and one. On the other hand, we show that the problem of computing L-cycle covers of maximum weight can be approximated with factor 2.5 for undirected graphs and with factor 3 in the case of directed graphs. Finally, we show that 4-cycle covers of maximum weight in graphs with edge weights zero and one can be computed in polynomial time.

As a by-product, we show that the problem of computing minimum vertex covers in λ-regular graphs is APX-complete for every $\lambda \geq 3$.

1 Introduction

The *travelling salesman problem* (TSP) is perhaps the best-known combinatorial optimisation problem. An instance of the TSP is a complete graph with edge weights, and the aim is to find a minimum or maximum weight cycle that visits every vertex exactly once. Such a cycle is called a *Hamiltonian cycle*. Since the TSP is NP-hard [10, ND22+23], we cannot hope to always find an optimal cycle efficiently. For practical purposes, however, it is often sufficient to obtain a cycle that is close to optimal. In such cases, we require approximation algorithms, i.e. polynomial-time algorithms that compute such near-optimal cycles.

The problem of computing *cycle covers* is a relaxation of the TSP: A cycle cover of a graph is a spanning subgraph such that every vertex is part of exactly one simple cycle. Thus, a solution to the TSP is a cycle cover consisting of a single cycle. In analogy to the TSP, the weight of a cycle cover in an edge-weighted graph is the sum of the weights of its edges.

* A full version of this work is available at http://arxiv.org/abs/cs/0504038.

** Supported by DFG research grant RE 672/3.

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 282–295, 2006.

In contrast to the TSP, cycle covers of maximum weight can be computed efficiently. This fact is exploited in approximation algorithms for the TSP; the computation of cycle covers forms the basis for the currently best known approximation algorithms for many variations of the TSP. These algorithms usually start by computing an initial cycle cover and then join cycles to obtain a Hamiltonian cycle.

Short cycles in a cycle cover limit the approximation ratios achieved by such algorithms. In general, the longer the cycles in the initial cover are, the better the approximation ratio. Thus, we are interested in computing cycle covers without short cycles. Moreover, there are approximation algorithms that behave particularly well if the cycle covers that are computed do not contain cycles of odd length [6]. Finally, some so-called vehicle routing problems (cf. e.g. Hassin and Rubinstein [12]) require covering vertices with cycles of bounded length.

Therefore, we consider *restricted cycle covers*, where cycles of certain lengths are ruled out a priori: Let $L \subseteq \mathbb{N}$, then an *L-cycle cover* is a cycle cover in which the length of each cycle is in L. To fathom the possibility of designing approximation algorithms based on computing cycle covers, we aim to characterise the sets L for which L-cycle covers of maximum weight can be computed efficiently.

1.1 Preliminaries

A **cycle cover** of a graph $G = (V, E)$ is a subgraph of G that consists solely of cycles such that all vertices in V are part of exactly one cycle. The length of a cycle is the number of edges it consists of. We are concerned with simple graphs, i.e. the graphs do not contain multiple edges or loops. Thus, the shortest cycles of undirected and directed graphs have length three and two, respectively.

An ***L*-cycle cover** is a cycle cover in which the length of every cycle is in the set $L \subseteq \mathbb{N}$. For undirected graphs, we have $L \subseteq \mathcal{U} = \{3, 4, 5, \ldots\}$, while $L \subseteq \mathcal{D} = \{2, 3, 4, \ldots\}$ in case of directed graphs. A ***k*-cycle cover** is a $\{k, k+1, \ldots\}$-cycle cover. Let $\overline{L} = \mathcal{U} \setminus L$ in the case of undirected graphs and $\overline{L} = \mathcal{D} \setminus L$ in the case of directed graphs (this will be clear from the context).

Given an edge weight function $w : E \to \mathbb{N}$, the **weight $w(C)$** of a subset $C \subseteq E$ of the edges of G is $w(C) = \sum_{e \in C} w(e)$. This particularly defines the weight of a cycle cover since we view cycle covers as sets of edges. Let $U \subseteq V$ be any subset of the vertices of G. The **internal edges of U** are all edges of G that have both vertices in U. We denote by $\boldsymbol{w_U(C)}$ the sum of the weights of all internal edges of U in C. The **external edges at U** are all edges of G with exactly one vertex in U.

For $L \subseteq \mathcal{U}$, the set ***L*-UCC** contains all undirected graphs that have an L-cycle cover as spanning subgraph.

Max-*L*-UCC is the following optimisation problem: Given a complete undirected graph with edge weights zero and one, find an L-cycle cover of maximum weight. We can also consider the graph as being not complete and without edge weights. Then we try to find an L-cycle cover with a minimum number of "non-edges" ("non-edges" correspond to weight zero edges, edges to weight one edges). Thus, Max-L-UCC can be viewed as a generalisation of L-UCC.

Max-W-$\boldsymbol{L}$-UCC is the problem of finding maximum-weight L-cycle covers in graphs with arbitrary non-negative edge weights.

For $k \geq 3$, **$\boldsymbol{k}$-UCC**, **Max-$\boldsymbol{k}$-UCC**, and **Max-W-$\boldsymbol{k}$-UCC** are defined like L-UCC, Max-L-UCC and Max-W-L-UCC except that k-cycle covers instead of L-cycle covers are sought.

$\boldsymbol{L}$-DCC, **Max-$\boldsymbol{L}$-DCC**, **Max-W-$\boldsymbol{L}$-DCC**, **$\boldsymbol{k}$-DCC**, **Max-$\boldsymbol{k}$-DCC**, and **Max-W-$\boldsymbol{k}$-DCC** are defined for directed graphs like L-UCC, Max-L-UCC, Max-W-L-UCC, k-UCC, Max-k-UCC, and Max-W-k-UCC for undirected graphs except that $L \subseteq \mathcal{D}$ and $k \geq 2$.

An instance of **Min-Vertex-Cover** is an undirected graph $H = (X, F)$. A vertex cover of H is a subset $\tilde{X} \subseteq X$ such that at least one vertex of every edge in F is in $\tilde{X}$. The aim is to find a vertex cover of minimum cardinality. **Min-Vertex-Cover($\boldsymbol{\lambda}$)** is Min-Vertex-Cover restricted to λ-regular graphs, i.e. to simple graphs in which every vertex is incident to exactly λ edges. Already Min-Vertex-Cover(3) is APX-complete [2].

We refer to Ausiello et al. [3] for a survey on NP optimisation problems.

1.2 Existing Results

Undirected Graphs. $\mathcal{U}$-UCC, Max-$\mathcal{U}$-UCC, and Max-W-$\mathcal{U}$-UCC can be solved in polynomial time via reduction to the classical perfect matching problem, which can be solved in polynomial time [1, Chap. 12]. Hartvigsen presented a polynomial-time algorithm for computing a *maximum-cardinality triangle-free two-matching* [11] (see also Sect. 5). His algorithm can be used to decide 4-UCC in polynomial time. Furthermore, it can be used to approximate Max-4-UCC within an additive error of one according to Bläser [4].

Max-W-k-UCC admits a simple factor 3/2 approximation for all k: Compute a maximum weight cycle cover, break the lightest edge of each cycle, and join the cycles to obtain a Hamiltonian cycle, which is sufficiently long if the graph contains at least k vertices. Unfortunately, this algorithm cannot be generalised to work for Max-W-L-UCC with arbitrary L. For the problem of computing k-cycle covers of minimum weight in graphs with edge weights one and two, there exists a factor 7/6 approximation algorithm for all k [8].

Cornuéjols and Pulleyblank presented a proof due to Papadimitriou that 6-UCC is NP-complete [9]. Vornberger showed that Max-W-5-UCC is NP-hard [14]. For $k \geq 7$, Max-k-UCC and Max-W-k-UCC are APX-complete [5]. Hell et al. [13] proved that L-UCC is NP-hard for $\overline{L} \not\subseteq \{3, 4\}$.

For most L, L-UCC, Max-L-UCC, and Max-W-L-UCC are not even recursive since there are uncountably many L. Thus for most L, L-UCC is not in NP and Max-L-UCC and Max-W-L-UCC are not in NPO. This does not matter for hardness results but may cause problems when one wants to design approximation algorithms that base on computing L-cycle covers. However, our approximation algorithms work for arbitrary L, independently of the complexity of L.

Directed Graphs. $\mathcal{D}$-DCC, Max-$\mathcal{D}$-DCC, and Max-W-$\mathcal{D}$-DCC can be solved in polynomial time by reduction to the maximum weight perfect matching problem

Table 1. The complexity of computing L-cycle covers

(a) Undirected cycle covers.

	$\boldsymbol{L}$**-UCC**	**Max-**$\boldsymbol{L}$**-UCC**	**Max-W-**$\boldsymbol{L}$**-UCC**
$\overline{L} = \emptyset$	in P	in PO	in PO
$\overline{L} = \{3\}$	in P	in PO	
$\overline{L} = \{4\} \vee \overline{L} = \{3, 4\}$			APX-complete
else	NP-hard	APX-hard	APX-hard

(b) Directed cycle covers.

	$\boldsymbol{L}$**-DCC**	**Max-**$\boldsymbol{L}$**-DCC**	**Max-W-**$\boldsymbol{L}$**-DCC**
$L = \{2\} \vee L = \mathcal{D}$	in P	in PO	in PO
else	NP-hard	APX-hard	APX-hard

in bipartite graphs [1, Chap. 12]. But already 3-DCC is NP-co plete [10, GT13]. Max-k-DCC and Max-W-k-DCC are APX-co plete for all $k \geq 3$ [5].

Si ilar to the factor $3/2$ approxi ation algorith for undirected cycle covers, Max-W-k-DCC has a si ple factor 2 approxi ation algorith for all k: Co pute a axi u weight cycle cover, break the lightest edge of every cycle, and join the cycles to obtain a Ha iltonian cycle. Again, this algorith cannot be generalised to work for Max-W-L-DCC with arbitrary L. There are a factor $4/3$ approxi ation algorith for Max-W-3-DCC [7] and a factor $3/2$ approxi ation algorith for Max-k-DCC for $k \geq 3$ [5].

As in the case of cycle covers in undirected graphs, for ost L, L-DCC, Max-L-DCC, and Max-W-L-DCC are not recursive.

1.3 New Results

We co e close to settling the co plexity and approxi ability of restricted cycle covers. Only the co plexity of the five proble s 5-UCC, $\overline{\{4\}}$-UCC Max-5-UCC, Max-$\overline{\{4\}}$-UCC, and Max-W-4-UCC re ains open. Table 1 shows an overview on the co plexity of co puting restricted cycle covers.

Hardness Results. We prove that Max-L-UCC is APX-hard for all L with $\overline{L} \not\subseteq \{3, 4\}$ (Sect. 3). We also prove that Max-W-L-UCC is APX-hard if $\overline{L} \not\subseteq \{3\}$ (this follows fro the results of Sect. 3 and the APX-co pleteness of Max-W-5-UCC and Max-W-$\overline{\{4\}}$-UCC shown in Sect. 2). The hardness results for Max-W-L-UCC hold even if we allow only the edge weights zero, one, and two.

We show a dichoto y for cycle covers of directed graphs: For all L with $L \neq \{2\}$ and $L \neq \mathcal{D}$, L-DCC is NP-hard (Theore 6) and Max-L-DCC and Max-W-L-DCC are APX-hard (Theore 5), while it is known that all three proble s are solvable in polyno ial ti e if $L = \{2\}$ or $L = \mathcal{D}$.

To show the hardness of directed cycle covers, we show that certain kinds of graphs, called L-*clamps*, exist for non-e pty $L \subseteq \mathcal{D}$ if and only if $L \neq \mathcal{D}$ (Theore 4). This graph-theoretical result ight be of independent interest.

As a by-product, we prove that Min-Vertex-Cover(λ) is APX-co plete for all $\lambda \geq 3$ (Sect. 6). We need this result for the APX-hardness proofs in Sect. 3.

Algorithms. We present a polyno ial-ti e factor 2.5 approxi ation algorith Max-W-L-UCC and a factor 3 approxi ation algorith for Max-W-L-DCC (Sect. 4). Both algorith s work for arbitrary L.

Finally, we prove that Max-4-UCC is solvable in polyno ial ti e (Sect. 5).

2 A Generic Reduction for L-Cycle Covers

In this section, we present a generic reduction fro Min-Vertex-Cover(3) to Max-L-UCC or Max-W-L-UCC. To instantiate the reduction for a certain L, we use a s all graph, which we call *gadget*, the specific structure of which depends on L. Such a gadget together with the generic reduction is an L-reduction fro Min-Vertex-Cover(3) to Max-L-UCC or Max-W-L-UCC. The ere ai is to prove the APX-hardness of Max-W-$\overline{\{4\}}$-UCC and Max-W-5-UCC.

2.1 The Generic Reduction

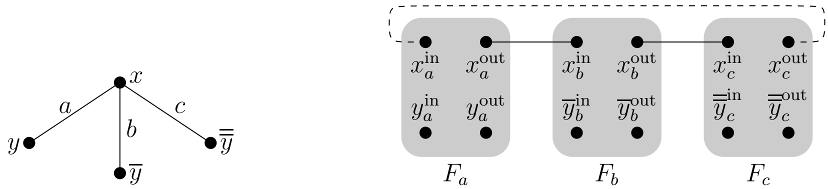
Let $H = (X, F)$ be a cubic graph with vertex set X and edge set F as an instance of Min-Vertex-Cover(3). Let $n = |X|$ and $m = |F| = 3n/2$. We construct an undirected co plete graph G with edge weight function w as a generic instance of Max-L-UCC or Max-W-L-UCC.

For each edge $a = \{x, y\} \in F$, we construct a subgraph F_a of G called the **gadget of a**. We define F_a as set of vertices, thus $w_{F_a}(C)$ for a subset C of the edges of G is well defined. This gadget contains four distinguished vertices x_a^{in}, x_a^{out}, y_a^{in}, and y_a^{out}. These four vertices are used to connect F_a to the rest of the graph. What such a gadget looks like depends on L. If all edges in such a gadget have weight zero or one, we obtain an instance of Max-L-UCC since all edges between different gadgets will have weight zero or one. Otherwise, we have an instance of Max-W-L-UCC. Figure 2 shows an exa ple of such a gadget.

Let $a, b, c \in F$ be the three edges incident to vertex $x \in X$ (the order is arbitrary). Then we assign weight one to the edges connecting x_a^{out} to x_b^{in} and x_b^{out} to x_c^{in} and weight zero to the edge connecting x_c^{out} to x_a^{in}. We call the three edges $\{x_a^{\mathrm{out}}, x_b^{\mathrm{in}}\}$, $\{x_b^{\mathrm{out}}, x_c^{\mathrm{in}}\}$, and $\{x_c^{\mathrm{out}}, x_a^{\mathrm{in}}\}$ the **junctions of x**. We say that $\{x_a^{\mathrm{out}}, x_b^{\mathrm{in}}\}$ and $\{x_c^{\mathrm{out}}, x_a^{\mathrm{in}}\}$ are the junctions of x **at F_a**. Figure 1 shows an exa ple.

We call an edge **illegal** if it connects two different gadgets but is not a junction. Thus, an illegal edge is an external edge at two different gadgets. All illegal edges have weight zero, i.e. there are no edges of weight one that connect two different gadgets except for the junctions. The weights of the internal edges of the gadgets depend on the gadget, which in turn depends on L.

The following ter s are defined for arbitrary subsets C of the edges of G, and so in particular for L-cycle covers. We say that **C legally connects F_a** if



(a) Vertex x and its edges.

(b) The gadgets F_a, F_b, and F_c and their connections via the three junctions of x. The dashed edge has weight zero. Other weight zero edges and the junctions of y, $\overline{y}$, and $\overline{\overline{y}}$ are not shown.

Fig. 1. The construction for a vertex $x \in X$ incident to $a, b, c \in F$

- C contains no illegal edges incident to F_a,
- C contains exactly two or four junctions at F_a, and
- if C contains exactly two junctions at F_a, then these belong to the sa e vertex $x \in a$.

We call C **legal** if C legally connects all gadgets. If C is legal, then for all $x \in X$, either all junctions of x are in C or no junction of x is in C. Further ore, fro a legal set C we obtain a vertex cover $\tilde{X} = \{x \mid \text{the junctions of } x \text{ are in } C\}$.

Let us now define the require ents the gadgets ust fulfil. In the following, let C be an arbitrary L-cycle cover of G and $a = \{x, y\} \in F$ be an arbitrary edge of H.

R0: There exists a fixed nu ber $s \in \mathbb{N}$, which we call the **gadget parameter**, that depends only on the gadget. The role of the gadget para eter will beco e clear in the subsequent require ents.
R1: $w_{F_a}(C) \leq s - 1$.
R2: If C contains 2α external edges at F_a, then $w_{F_a}(C) \leq s - \alpha$.
R3: If C contains exactly one junction of x at F_a and exactly one junction of y at F_a, then $w_{F_a}(C) \leq s - 2$. (In this case, C does not legally connect F_a.)
R4: Let C' be an arbitrary subset of the edges of G that legally connects F_a. Assu e that there are 2α junctions ($\alpha \in \{1, 2\}$) at F_a in C'.
Then there exists a C'' with the following properties:
  - C'' differs fro C' only in F_a's internal edges and
  - $w_{F_a}(C'') = s - \alpha$.

Thus, given C', C'' can be obtained by locally odifying C' within F_a. We call the process of obtaining C'' fro C' **rearranging C' in F_a**.
R5: Let C' be a legal subset of the edges of G. Then there exists a subset $\tilde{C}$ of edges obtained by rearranging all gadgets as described in R4 such that $\tilde{C}$ is an L-cycle cover.

The require ents assert that connecting the gadgets legally is never worse than connecting the illegally. This yields the ain result of this section.

Lemma 1. *Assume that a gadget as described exists for $L \subseteq \mathcal{U}$. Then the reduction presented is an L-reduction from* Min-Vertex-Cover(3) *to* Max-W-L-UCC. *If the gadget contains only edges of weight zero or one, then the reduction is an L-reduction from* Min-Vertex-Cover(3) *to* Max-L-UCC.

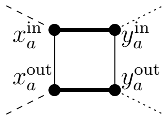
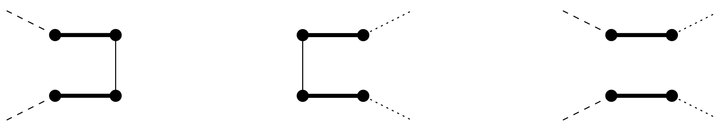


Fig. 2. The edge gadget F_a for an edge $a = \{x, y\}$ that is used to prove the APX-completeness of Max-W-5-UCC. Bold edges are internal edges of weight two, solid edges are internal edges of weight one, internal edges of weight zero are not shown. The dashed and dotted edges are the junctions of x and y, respectively, at F_a.



(a) $x \in \tilde{X}$. (b) $y \in \tilde{X}$. (c) $x, y \in \tilde{X}$.

Fig. 3. Traversals of the gadget for Max-W-5-UCC that achieve maximum weight

2.2 Max-W-5-UCC and Max-W-$\overline{\{4\}}$-UCC

The gadget for Max-W-5-UCC is shown in Fig. 2. Let G be the graph constructed via the reduction presented in Sect. 2.1 with the gadget of this section. Let C be an arbitrary L-cycle cover of G and $a = \{x, y\} \in F$. By proving that it fulfils all require ents, we obtain the following result.

Theorem 1. Max-W-5-UCC *is* APX*-hard, even if the edge weights are restricted to be zero, one, or two.*

Although the status of Max-5-UCC is still open, allowing only one additional edge weight of two already yields an APX-co plete proble .

The generic reduction together with the gadget used for Max-W-5-UCC works also for Max-W-$\overline{\{4\}}$-UCC. The gadget only requires that cycles of length four are forbidden since otherwise R1 is not satisfied. Thus, all require ents are fulfilled for Max-W-$\overline{\{4\}}$-UCC in exactly the sa e way as for Max-W-5-UCC.

Theorem 2. Max-W-$\overline{\{4\}}$-UCC *is* APX*-hard, even if the edge weights are restricted to be zero, one, or two.*

3 A Uniform Reduction for L-Cycle Covers

3.1 Clamps

We now define so-called *clamps*, which were introduced by Hell et al. [13]. Cla ps are crucial for the hardness proof presented in this section.

Let $K = (V, E)$ be an undirected graph, let $u, v \in V$ be two vertices of K, and let $L \subseteq \mathcal{U}$. We denote by K_{-u} and K_{-v} the graphs obtained fro K by deleting u and v, respectively, and their incident edges. Moreover, K_{-u-v} denotes the

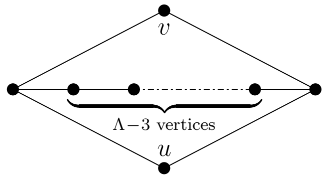


Fig. 4. An L-clamp for finite L with $\max(L) = \Lambda$

graph obtained from K by deleting both u and v. Finally, for $k \in \mathbb{N}$, K^k is the following graph: Let $y_1, \ldots, y_k$ be vertices with $y_i \notin V$, add edges $\{u, y_1\}$, $\{y_i, y_{i+1}\}$ for $1 \leq i \leq k-1$, and $\{y_k, v\}$. For $k = 0$, we directly connect u to v.

The graph K is called an ***L*-clamp** if the following properties hold:

– Both K_{-u} and K_{-v} contain an L-cycle cover.
– Neither K nor K_{-u-v} nor K^k for any $k \in \mathbb{N}$ contains an L-cycle cover.

We call u and v the **connectors** of the L-clamp K.

Lemma 2 (Hell et al. [13]). *Let $L \subseteq \mathcal{U}$ be non-empty. Then there exists an L-clamp if and only if $\overline{L} \not\subseteq \{3, 4\}$.*

Figure 4 shows an example of an L-clamp for finite L.

If there exists an L-clamp for some L, then we can assume that the connectors u and v both have degree two since we can remove all edges that are not used in the L-cycle covers of K_{-v} and K_{-u}.

For our purpose, consider any non-empty set $L \subseteq \{3, 4, 5, \ldots\}$ with $\overline{L} \not\subseteq \{3, 4\}$. We fix one L-clamp K with connectors $u, v \in V$ arbitrarily and refer to it in the following as the L-clamp, although there exists more than one L-clamp. Let σ be the number of vertices of K.

We are concerned with edge-weighted graphs. Therefore, we transfer the notion of clamps to graphs with edge weights zero and one in the obvious way: Let G be an undirected complete graph with vertex set V and edge weights zero and one and let K be an L-clamp. Let $U \subseteq V$. We say that U is an L-clamp with connectors $u, v \in U$ if the subgraph of G induced by U restricted to the edges of weight one is isomorphic to K with u and v mapped to connectors of K.

3.2 The Reduction

Let $L \subseteq \mathcal{U}$ be non-empty with $\overline{L} \not\subseteq \{3, 4\}$. Thus, L-clamps exist and we fix one as in the previous section. Let σ be the number of vertices in the L-clamp. Let $\lambda = \min(L)$. (This choice is arbitrary. We could choose any number in L.) We will reduce Min-Vertex-Cover(λ) to Max-L-UCC. Min-Vertex-Cover(λ) is APX-complete since $\lambda \geq 3$ (see Sect. 6).

Let $H = (X, F)$ be an instance of Min-Vertex-Cover(λ) with $n = |X|$ vertices and $m = |F| = \lambda n/2$ edges. Our instance G for Max-L-UCC consists of λ subgraphs $G_1, \ldots, G_\lambda$, each containing $2\sigma m$ vertices. We start by describing G_1.

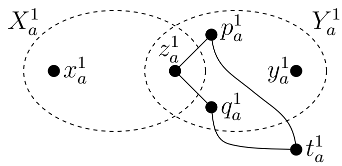


Fig. 5. The edge gadget for $a = \{x, y\}$ consisting of two L-clamps. The vertex z_a^1 is the only vertex that belongs to both clamps X_a^1 and Y_a^1.

Then we state the differences between G_1 and $G_2, \ldots, G_\lambda$ and say to which edges between these graphs we assign weight one.

Let $a = \{x, y\} \in F$ be any edge of H. We construct an edge gadget F_a for a that consists of two L-cla ps X_a^1 and Y_a^1 and one additional vertex t_a^1 as shown in Fig. 5. The connectors of X_a^1 are x_a^1 and z_a^1 while the connectors of Y_a^1 are y_a^1 and z_a^1, i.e. X_a^1 and Y_a^1 share the connector z_a^1. Let p_a^1 and q_a^1 be the two unique vertices in Y_a^1 that share a weight one edge with z_a^1. (The choice of Y_a^1 is arbitrary, we could choose the corresponding vertices in X_a^1 as well.) We assign weight one to both $\{p_a^1, t_a^1\}$ and $\{q_a^1, t_a^1\}$. Thus, the vertex t_a^1 can also serve as a connector for Y_a^1.

Now let $x \in X$ be any vertex of H and let $a_1, \ldots, a_\lambda \in F$ be the λ edges that are incident to x. We connect the vertices $x_{a_1}^1, \ldots, x_{a_\lambda}^1$ to for a path by assigning weight one to the edges $\{x_{a_\eta}^1, x_{a_{\eta+1}}^1\}$ for $\eta \in \{1, \ldots, \lambda - 1\}$. Together with edge $\{x_{a_\lambda}^1, x_{a_1}^1\}$, these edges for a cycle of length $\lambda \in L$, but note that $w(\{x_{a_\lambda}^1, x_{a_1}^1\}) = 0$. These λ edges are called the **junctions of $\boldsymbol{x}$**. The **junctions at $\boldsymbol{F_a}$** for so e $a = \{x, y\} \in F$ are the junctions of x and y that are incident to F_a. Overall, the graph G_1 consists of $2\sigma m$ vertices since every edge gadget consists of 2σ vertices.

The graphs $G_2, \ldots, G_\lambda$ are al ost exact copies of G_1. The graph G_ξ, $\xi \in \{2, \ldots, \lambda\}$ has cla ps X_a^ξ and Y_a^ξ and vertices $x_a^\xi, y_a^\xi, z_a^\xi, t_a^\xi, p_a^\xi, q_a^\xi$ for each edge $a = \{x, y\} \in F$, just as above. The edge weights are also identical with the single exception that the edge $\{x_{a_\lambda}^\xi, x_{a_1}^\xi\}$ also has weight one. Note that we only use the ter "gadget" for the subgraphs of G_1 defined above although al ost the sa e subgraphs occur in $G_2, \ldots, G_\lambda$ as well. Si ilarly, the ter "junction" refers only to an edge in G_1 as defined above.

Finally, we describe how to connect $G_1, \ldots, G_\lambda$ with each other. For every edge $a \in F$, there are λ vertices $t_a^1, \ldots, t_a^\lambda$. These are connected to for a cycle consisting solely of weight one edges, i.e. we assign weight one to all edges $\{t_a^\xi, t_a^{\xi+1}\}$ for $\xi \in \{1, \ldots, \lambda - 1\}$ and to $\{t_a^\lambda, t_a^1\}$. Figure 6 shows an exa ple of the whole construction fro the viewpoint of a single vertex.

As in the previous section, we call edges that are not junctions but connect two different gadgets **illegal**. Edges with both vertices in the sa e gadget are again called internal edges. In addition to junctions, illegal edges, and internal edges, we have a fourth kind of edges: The ***t*-edges** of F_a for $a \in F$ are the two edges $\{t_a^1, t_a^2\}$ and $\{t_a^1, t_a^\lambda\}$. The t-edges are not illegal. All other edges connecting G_1 to G_ξ for $\xi \neq 1$ are illegal.

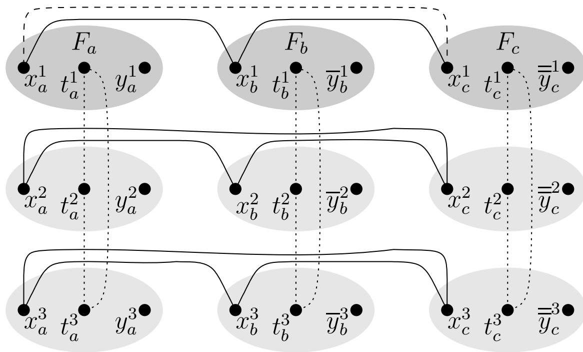


Fig. 6. The construction for a vertex $x \in X$ incident to edges $a, b, c \in F$ for $\lambda = 3$ (Fig. 1(a) on page 287 shows the corresponding graph). The dark grey areas are the edge gadgets F_a, F_b, and F_c. Their copies in G_2 and G_3 are light grey. The cycles connecting the t-vertices are dotted. The cycles connecting the x-vertices are solid, except for the edge $\{x_c^1, x_a^1\}$, which has weight zero and is dashed. The vertices $z_a^1, \ldots, z_c^3$ are not shown for legibility.

Let C be any subset of the edges of the graph G thus constructed, and let $a = \{x, y\} \in F$ be an arbitrary edge of H. We say that ***C* legally connects *F_a*** if the following properties are fulfilled:

– C contains no illegal edges incident to F_a and exactly two or four junctions at F_a.
– If C contains exactly two junctions at F_a, then these belong to the sa e vertex and there are two t-edges at F_a in C.
– If C contains four junctions at F_a, then these are the only external edges in C incident to F_a. In particular, C does not contain t-edges at F_a.

We call C **legal** if C legally connects all gadgets.

We can prove that the construction described above is an L-reduction fro Min-Vertex-Cover(λ) to Max-L-UCC for all L with $\overline{L} \not\subseteq \{3, 4\}$.

Theorem 3. *For all $L \subseteq \mathcal{U}$ with $\overline{L} \not\subseteq \{3, 4\}$,* Max-$L$-UCC *is* APX*-hard.*

3.3 Clamps in Directed Graphs

The ai of this section is to introduce directed L-cla ps. Let $K = (V, E)$ be a directed graph and $u, v \in V$. Again, K_{-u}, K_{-v}, and K_{-u-v} denote the graphs obtained by deleting u, v, and both u and v, respectively. For $k \in \mathbb{N}$, K_u^k denotes the following graph: Let $y_1, \ldots, y_k \notin V$ be new vertices and add edges $(u, y_1), (y_1, y_2), \ldots, (y_k, v)$. For $k = 0$, we add the edge (u, v). The graph K_v^k is si ilarly defined, except that we now start at v, i.e. we add the edges $(v, y_1), (y_1, y_2), \ldots, (y_k, u)$. K_v^0 is K with the additional edge (v, u).

Now we can define cla ps for directed graphs: Let $L \subseteq \mathcal{D}$. A directed graph $K = (V, E)$ with $u, v \in V$ is a **directed *L*-clamp** with connectors u and v if the following properties hold:

– Both K_{-u} and K_{-v} contain an L-cycle cover.
– Neither K nor K_{-u-v} nor K_u^k nor K_v^k for any $k \in \mathbb{N}$ contains an L-cycle cover.

Theorem 4. *Let $L \subseteq \mathcal{D}$ be non-empty. Then there exists a directed L-clamp if and only if $L \neq \mathcal{D}$.*

3.4 L-Cycle Covers in Directed Graphs

Fro the hardness results in the previous sections and the work by Hell et al. [13], we obtain the NP-hardness and APX-hardness of L-DCC and Max-L-DCC, respectively, for all L with $2 \notin L$ and $\overline{L} \not\subseteq \{2, 3, 4\}$: We use the sa e reduction as for undirected cycle covers and replace every undirected edge $\{u, v\}$ by a pair of directed edges (u, v) and (v, u). However, this does not work if $2 \in L$ and also leaves open the cases when $\overline{L} \subsetneq \{2, 3, 4\}$. We will show that $L = \{2\}$ and $L = \mathcal{D}$ are the only cases in which directed L-cycle covers can be co puted efficiently by proving the NP-hardness of L-DCC and the APX-hardness of Max-L-DCC for all other L. Thus, we settle the co plexity for directed graphs.

The APX-hardness of the directed cycle cover proble is obtained by a proof si ilar to the proof for undirected cycle covers. All we need is a $\lambda \in L$ with $\lambda \geq 3$ and the existence of an L-cla p.

Theorem 5. *Let $L \subseteq \mathcal{D}$ be a non-empty set. If $L \neq \{2\}$ and $L \neq \mathcal{D}$, then* Max-L-DCC *and* Max-W-L-DCC *are* APX*-hard.*

We can also prove that for all $L \notin \{\{2\}, \mathcal{D}\}$, L-DCC is NP-hard.

Theorem 6. *Let $L \subseteq \mathcal{D}$ be a non-empty set. If $L \neq \{2\}$ and $L \neq \mathcal{D}$, then* L-DCC *is* NP*-hard.*

Let $L \notin \{\{2\}, \mathcal{D}\}$. L-DCC is in NP and therefore NP-co plete if and only if the language $\{1^\lambda \mid \lambda \in L\}$ is in NP.

4 Approximation Algorithms

The goal of this section is to devise approxi ation algorith s for Max-W-L-UCC and Max-W-L-DCC that work for arbitrary L. The catch is that for ost L it is i possible to decide whether so e cycle length is in L or not. One possibility would be to restrict ourselves to sets L such that $\{1^\lambda \mid \lambda \in L\}$ is in P. For such L, Max-W-L-UCC and Max-W-L-DCC are NP opti isation proble s. Another possibility for circu venting the proble is to include the per itted cycle lengths in the input. However, it turns out that such restrictions are not necessary since we can restrict ourselves to finite sets L.

A necessary and sufficient condition for a co plete graph with n vertices to have an L-cycle cover is that there exist (not necessarily distinct) lengths $\lambda_1, \ldots, \lambda_k \in L$ for so e $k \in \mathbb{N}$ with $\sum_{i=1}^{k} \lambda_i = n$. We call such an n ***L*-admissible** and define $\langle L \rangle = \{n \mid n \text{ is } L\text{-ad issible}\}$.

Input: an undirected graph $G = (V, U(V))$ with $|V| = n$;
an edge weight function $w : U(V) \to \mathbb{N}$
Output: an L-cycle cover C^{apx} of G if n is L-admissible, $\perp$ otherwise

1. If $n \notin \langle L \rangle$, then return $\perp$.
2. Compute a cycle cover C^{init} of maximum weight.
3. Compute a subset $P \subseteq C^{\text{init}}$ of maximum weight such that (V, P) consists of $\lceil n/5 \rceil$ paths of length two and $n - 3 \cdot \lceil n/5 \rceil$ isolated vertices.
4. Join the paths to obtain an L-cycle cover C^{apx}, return C^{apx}.

Fig. 7. A factor 2.5 approximation algorithm for Max-W-L-UCC

Lemma 3. *For all $L \subseteq \mathbb{N}$, there exists a finite set $L' \subseteq L$ with $\langle L' \rangle = \langle L \rangle$.*

Instead of computing L'-cycle covers in the following, we assume without loss of generality that L is already a finite set.

The main idea of the two approximation algorithms is as follows: We start by computing a cycle cover C^{init} of maximum weight. Then we take a subset S of the edges of C^{init} that weighs as much as possible under the restriction that there exists an L-cycle cover that includes all edges of S. We add edges to obtain an L-cycle cover $C^{\text{apx}} \supseteq S$. Let $C^\star$ be an L-cycle cover of maximum weight, and assume that we can guarantee $\rho \cdot w(S) \geq w(C^{\text{init}})$ for some $\rho \geq 1$. Then $w(C^\star) \leq w(C^{\text{init}}) \leq \rho \cdot w(S) \leq \rho \cdot w(C^{\text{apx}})$. Thus, we have computed a factor ρ approximation to an L-cycle cover of maximum weight.

4.1 Approximating Undirected Cycle Covers

The input of our algorithm for undirected graphs is an undirected complete graph $G = (V, U(V))$ with $|V| = n$ and an edge weight function $w : U(V) \to \mathbb{N}$.

The main idea of the approximation algorithm is as follows: Every cycle cover can be decomposed into $\lceil n/5 \rceil$ vertex-disjoint paths of length two and $n - 3 \cdot \lceil n/5 \rceil$ isolated vertices. Conversely, every collection P of $\lceil n/5 \rceil$ paths of length two together with $n - 3 \cdot \lceil n/5 \rceil$ isolated vertices can be extended to form an L-cycle cover, provided that n is L-admissible.

Theorem 7. *For every fixed L, the algorithm shown in Fig. 7 is a factor* 2.5 *approximation algorithm for* Max-W-L-UCC *with running time $O(n^3)$.*

4.2 Approximating Directed Cycle Covers

Now we present an approximation algorithm for Max-W-L-DCC that achieves an approximation ratio of 3. The input consists of a directed complete graph $G = (V, D(V))$ with $|V| = n$ and an edge weight function $w : D(V) \to \mathbb{N}$.

Given a cycle cover C, we can obtain a matching $M \subseteq C$ consisting of $\lceil n/3 \rceil$ edges such that $w(M) \geq w(C)/3$. Conversely, if n is L-admissible, then every matching of cardinality $\lceil n/3 \rceil$ can be extended to form an L-cycle cover. Instead of computing an initial cycle cover, the algorithm shown in Fig. 8 directly computes a matching of cardinality $\lceil n/3 \rceil$.

Input: a directed graph $G = (V, D(V))$ with $|V| = n$;
an edge weight function $w : D(V) \to \mathbb{N}$
Output: an L-cycle cover C^{apx} of G if n is L-admissible, $\perp$ otherwise

1. If $n \notin \langle L \rangle$, then return $\perp$.
2. Compute a maximum weight matching M^{init} of G of cardinality $\lceil n/3 \rceil$.
3. Join the edges in M^{init} to obtain an L-cycle cover C^{apx}, return C^{apx}.

Fig. 8. A factor 3 approximation algorithm for Max-W-L-DCC

Theorem 8. *For every fixed L, the algorithm shown in Fig. 8 is a factor* 3 *approximation algorithm for* Max-W-L-UCC *with running time $O(n^3)$.*

5 Solving Max-4-UCC in Polynomial Time

The aim of this section is to show that Max-4-UCC can be solved deterministically in polynomial time. To do this, we exploit Hartvigsen's algorithm for computing a maximum-cardinality triangle-free two-matching.

A **two-matching** of an undirected graph G is a spanning subgraph in which every vertex of G has degree *at most* two. Thus, two-matchings consist of disjoint simple cycles and paths. A two-matching is a relaxation of a cycle cover (or two-factor): In a cycle cover, every vertex has degree *exactly* two. A **triangle-free two-matching** is a two-matching in which each cycle has a length of at least four. The paths can have arbitrary lengths. A triangle-free two-matching of maximum weight in graphs with edge weights zero and one can be computed deterministically in time $O(n^3)$, where n is the number of vertices [11, Chap. 3].

We want to solve Max-4-UCC, i.e. all cycles must have a length of at least four and no paths are allowed. Therefore, let M be a maximum weight triangle-free two-matching of a graph G of n vertices. If M does not contain any paths, then M is already a 4-cycle cover of maximum weight.

Let ℓ be the number of vertices of G that lie on paths in M. If $\ell \geq 4$, then we connect these paths to get a cycle of length ℓ. No weight is lost in this way, and the result is a maximum weight 4-cycle cover.

We run into trouble if $\ell \in \{1, 2, 3\}$. Let $Y = \{y_1, \ldots, y_\ell\}$ be the set of vertices that lie on paths in M. Let ℓ' be the number of edges of weight one in M that connect two vertices of Y. Then $0 \leq \ell' \leq \ell - 1$ and $w(M) = n - \ell + \ell' \leq n - 1$.

An obvious way to obtain a cycle cover from M is to break one edge of one cycle and connect the vertices of Y to this cycle. Unfortunately, breaking an edge might cause a loss of weight one. This yields the aforementioned approximation within an additive error of one. We can prove the following with a more careful analysis: Either we can avoid the loss of weight one, or indeed a maximum weight 4-cycle cover has only weight $w(M) - 1$. This yields the following result.

Theorem 9. Max-4-UCC *can be solved deterministically in time $O(n^3)$.*

6 Vertex Cover in Regular Graphs

We can prove that Min-Vertex-Cover(λ) is APX-co plete for every $\lambda \geq 3$. Previously, this was only known for cubic, i.e. three-regular, graphs [2]. We need the APX-hardness of Min-Vertex-Cover(λ) for all $\lambda \geq 3$ in Sect. 3.

Theorem 10. *For every $\lambda \in \mathbb{N}$, $\lambda \geq 3$,* Min-Vertex-Cover(λ) *is* APX-*complete.*

References

1. Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, 1993.
2. Paola Alimonti and Viggo Kann. Some APX-completeness results for cubic graphs. *Theoret. Comput. Sci.*, 237(1–2):123–134, 2000.
3. Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties.* Springer, 1999.
4. Markus Bläser. Approximationsalgorithmen für Graphüberdeckungsprobleme. Habilitationsschrift, Institut für Theoretische Informatik, Universität zu Lübeck, Lübeck, Germany, 2002.
5. Markus Bläser and Bodo Manthey. Approximating maximum weight cycle covers in directed graphs with weights zero and one. *Algorithmica*, 42(2):121–139, 2005.
6. Markus Bläser, Bodo Manthey, and Jiří Sgall. An improved approximation algorithm for the asymmetric TSP with strengthened triangle inequality. *J. Discrete Algorithms*, to appear.
7. Markus Bläser, L. Shankar Ram, and Maxim I. Sviridenko. Improved approximation algorithms for metric maximum ATSP and maximum 3-cycle cover problems. In *Proc. of the 9th Workshop on Algorithms and Data Structures (WADS)*, vol. 3608 of *Lecture Notes in Comput. Sci.*, pp. 350–359. Springer, 2005.
8. Markus Bläser and Bodo Siebert. Computing cycle covers without short cycles. In *Proc. of the 9th Ann. European Symp. on Algorithms (ESA)*, vol. 2161 of *Lecture Notes in Comput. Sci.*, pp. 368–379. Springer, 2001. Bodo Siebert is the birth name of Bodo Manthey.
9. Gérard P. Cornuéjols and William R. Pulleyblank. A matching problem with side conditions. *Discrete Math.*, 29(2):135–159, 1980.
10. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.
11. David Hartvigsen. *An Extension of Matching Theory.* PhD thesis, Department of Mathematics, Carnegie Mellon University, 1984.
12. Refael Hassin and Shlomi Rubinstein. On the complexity of the k-customer vehicle routing problem. *Oper. Res. Lett.*, 33:1, 71-76 2005.
13. Pavol Hell, David G. Kirkpatrick, Jan Kratochvíl, and Igor Kríz. On restricted two-factors. *SIAM J. Discrete Math.*, 1(4):472–484, 1988.
14. Oliver Vornberger. Easy and hard cycle covers. Technical report, Universität/ Gesamthochschule Paderborn, 1980.

A PTAS for the Minimum Dominating Set Problem in Unit Disk Graphs

Tim Nieberg* and Johann Hurink**

University of Twente, Faculty of Electrical Engineering,
Mathematics & Computer Science,
Postbus 217, NL-7500 AE Enschede
{T.Nieberg, J.L.Hurink}@utwente.nl

Abstract. We present a polynomial-time approximation scheme (PTAS) for the minimum dominating set problem in unit disk graphs. In contrast to previously known approximation schemes for the minimum dominating set problem on unit disk graphs, our approach does not assume a geometric representation of the vertices (specifying the positions of the disks in the plane) to be given as part of the input. The runtime of the PTAS is $n^{O(1/\varepsilon \log 1/\varepsilon)}$. The algorithm accepts any undirected graph as input, and returns a $(1+\varepsilon)$-approximate minimum dominating set, or a certificate showing that the input graph is no unit disk graph, making the algorithm robust. The PTAS can easily be adapted to other classes of geometric intersection graphs.

1 Introduction

In this paper, we consider the *minimum dominating set* (MDS) problem of finding a dominating set of minimum cardinality in a unit disk graph for the case that no geometric representation of the graph is available. A graph is a *unit disk graph* (UDG) if its vertices can be drawn as circular disks of equal radius in the plane in such a way that there is an edge between two vertices if and only if the two disks have a non-empty intersection. Such a drawing, i.e. a list of center points of the vertices/disks, is referred to as geometric representation of the graph. A subset of vertices in an undirected graph is called *dominating set* if every vertex in the graph either is contained in the subset, or adjacent to a vertex in the set.

We present a polynomial-time approximation scheme (PTAS) for the MDS problem on UDGs, that is, given any $\varepsilon > 0$, the algorithm gives in polynomial-time an approximation with a performance guarantee of $(1+\varepsilon)$.

Unit disk graphs are widely used to model the communication in wireless ad-hoc networks. In such a network, structures like dominating sets play an important role, e.g. in global flooding to alleviate the so-called broadcast storm

* Supported partially by the European research project EYES (IST-2001-34734).
** Supported partially by the Dutch research project Smart Surroundings (BSIK-03060).

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 296–306, 2006.

problem. A message broadcast only in the dominating set is an efficient way to ensure that it is received by all transmitters in the network, both in terms of energy and interference.

The MDS problem is NP-hard, even on unit disk graphs where a geometric representation is given [4]. Most of the work concerning approximation schemes in UDGs assume a given representation, which allows for separation of the graph along a grid ([1],[5]). Approximation schemes for the MDS, and other related problems in UDGs are given in [6]. In [3], a PTAS for the minimum connected dominating set is presented, also using grid-based separation.

However, the case when no geometric representation is present is significantly different: Computing a possible geometric representation for a given unit disk graph is NP-hard. Indeed, any polynomial-time algorithm computing a geometric representation for UDGs can be used in a straightforward way to determine whether a given graph is a UDG, a problem known to be NP-hard [2].

The lack of coordinates, and the intractability to compute these, call for another approach. For the case that a representation is not given, several approximation algorithms are presented in [8], including a 5-approximation for the MDS problem. In [10], local neighborhoods of limited graph-theoretic diameter are used to obtain a PTAS for the maximum independent set problem in the same setting. This method uses the fact that in such neighborhoods, a maximum independent set is of bounded cardinality. In this paper, the same fact is used to bound the size of a minimum dominating set. While in [10], the separation and overall algorithm follows by simple arguments, for the minimum dominating set problem some attention has to be paid to the manner the local neighborhoods are created and put together. The main reasons for this are the differences in the objective function and the fact that, in contrast to independent sets, a subset of a dominating set no longer needs to be a dominating set. The resulting PTAS for the MDS problem on UDGs without geometric representation has a running time of $n^{O(1/\varepsilon \log 1/\varepsilon)}$.

Independence of geometric coordinates makes it easier to extend the approach to other graphs used to model wireless ad-hoc networks closer to reality, e.g. Quasi Unit Disk Graphs [7], or Coverage Area Graphs [9]. These models also include a certain amount of uncertainty with respect to wireless transmissions.

Besides the independence from a geometric representation, an additional advantage of the presented PTAS lies in the fact that we can extend the algorithm towards a *robust* approximation [11]. The algorithm may then be applied to an arbitrary undirected graph, and the output is either a $(1 + \varepsilon)$-approximation for the MDS problem in this graph, or a certificate which allows us to prove in polynomial-time that the input graph is no unit disk graph. In other words, we have a polynomial-time algorithm which either approximates the MDS problem, or solves the recognition problem. In case the input graph is a UDG, the algorithm always returns a dominating set of desired quality.

The remainder of the paper is organized as follows. In the following section, we present some basic definitions. Section 3 introduces the concept of a 2-separated collection of subsets, a structure that is used to efficiently separate a graph into smaller subgraphs for which the problem of computing a dominating set is easier to tackle. The PTAS itself is then presented in Section 4. In Section 5, we discuss the robustness of the algorithm, and present some extensions to other intersection graphs of geometric objects.

2 Definitions and Preliminaries

A graph $G = (V, E)$ is a unit disk graph (UDG) if it results from the intersection graph of disks of unit radius in the Euclidean plane. In other words, G is a UDG if there exists a map $f : V \to \mathbb{R}^2$ satisfying

$$(u, v) \in E \iff \|f(u) - f(v)\| \le 2,$$

where $\|.\|$ denotes the Euclidean norm. In this context, f is called a geometric representation of G and is not unique for a given graph. For the remainder of this paper, we assume f not to be given or known.

A subset $D \subset V$ is a *dominating set* (for V) if for every vertex $v \in V$, either $v \in D$ holds or there exists an edge $(u, v) \in E$ such that $u \in D$. The minimum dominating set problem (MDS) seeks to find a dominating set of minimum cardinality for a given graph.

In this paper, the goal is to give a polynomial-time approximation scheme (PTAS) for the minimum dominating set problem on unit disk graphs. That is, we seek for an algorithm which, given a UDG $G = (V, E)$ and a parameter $\varepsilon > 0$, computes a dominating set of cardinality no more than $(1+\varepsilon)$ the size of a minimum dominating set in G. The running time of the algorithm is allowed to depend on the parameter ε, but should be polynomial with respect to the input instance, i.e. polynomial in $n = |V|$ for fixed $\varepsilon > 0$.

We now present some further definitions needed for the description and discussion of the algorithm and the underlying concepts. Without loss of generality, we may assume the graph G to be connected. If this is not the case, we may consider each connected component separately.

Let $W \subset V$ denote a set of vertices in $G = (V, E)$. In the following, we simultaneously use W to also denote the resulting induced subgraph $G[W] := (W, E \cap (W \times W))$. Obviously, the graph $G[W]$ is a unit disk graph if the original graph is one.

Furthermore, we denote by $N(v)$ the closed neighborhood of a vertex $v \in V$, i.e. $N(v) := \{u \in V \mid (u, v) \in E\} \cup \{v\}$. Analogously, for $W \subset V$, let $N(W) := \bigcup_{w \in W} N(w)$ define the neighborhood of W. In this context, we set $N(\varnothing) := \varnothing$. For $r \in \mathbb{N}$, we denote by $N^r(v) := N(N^{r-1}(v))$ the recursively defined r-th neighborhood of $v \in V$, where $N^1(v) := N(v)$.

For two vertices $u, v \in V$, let $d(u, v)$ denote the distance between u and v, that is the number of edges on a shortest path between these two vertices. Thus, alternatively, the r-th neighborhood of $v \in V$ is characterized by $N^r(v) = \{u \in V \mid d(u, v) \le r\}$.

Denote by $\mathcal{P}(V)$ the set of all subsets of V. We then define $D : \mathcal{P}(V) \to \mathcal{P}(V)$ to be an operation returning a dominating set of minimum cardinality for the

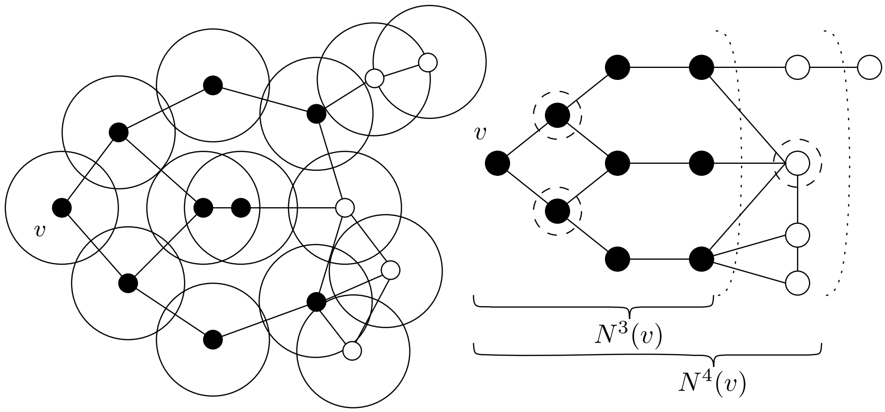


Fig. 1. Example of a UDG with and without geometric representation

subset of vertices given as argument to it. For a subset $W \subset V$, the set $D(W)$ dominates W, i.e. for every $w \in W$, either $w \in D(W)$ holds, or there is an edge $(u, w) \in E$ such that $u \in D(W)$. It is easy to see that $W \subset N(D(W))$ and that $D(W) \subset N(W)$ hold. In the following, we are interested in an efficient, i.e. polynomial-time, approximation of $D(V)$ within a factor of $(1+\varepsilon)$ for any given $\varepsilon > 0$.

Figure 1 illustrates some of the given notations. In the left part, a graph and its geometric representation are given, whereas in the right part only the graph and some neighborhoods of a node v are presented. Furthermore, the circled vertices in the right part give a minimum dominating set for $N^3(v)$, i.e. $D(N^3(v))$. As can be seen from the example, $D(W) \subset W$ need not hold for a subset $W \subsetneq V$: Using the circled vertex in $N^4(v)$, we obtain a dominating set consisting of three vertices, whereas restricting the dominating set only to vertices from $N^3(v)$ yields dominating sets of cardinality 4 or higher.

3 Local Dominating Sets

In this section, we introduce the concept of a 2-separated collection of subsets. The subgraphs induced by the subsets of such a collection divide the original graph into smaller parts for which it becomes easier to tackle the MDS problem. For a collection of local dominating sets resulting from a separation of the graph into smaller subgraphs, we show several properties that allow for bounds on the cardinalities with respect to an optimal, global solution. Throughout this section, we do not assume the graph to be a UDG, the following concepts are valid for all undirected graphs.

For a graph $G = (V, E)$, let $\mathcal{S} := \{S_1, \ldots, S_k\}$ be a collection of subsets of vertices $S_i \subset V$, $i = 1, \ldots, k$, with the following property:

(P) for any two vertices $s \in S_i$ and $\bar{s} \in S_j$ with $i \neq j$, it is $d(s, \bar{s}) > 2$.

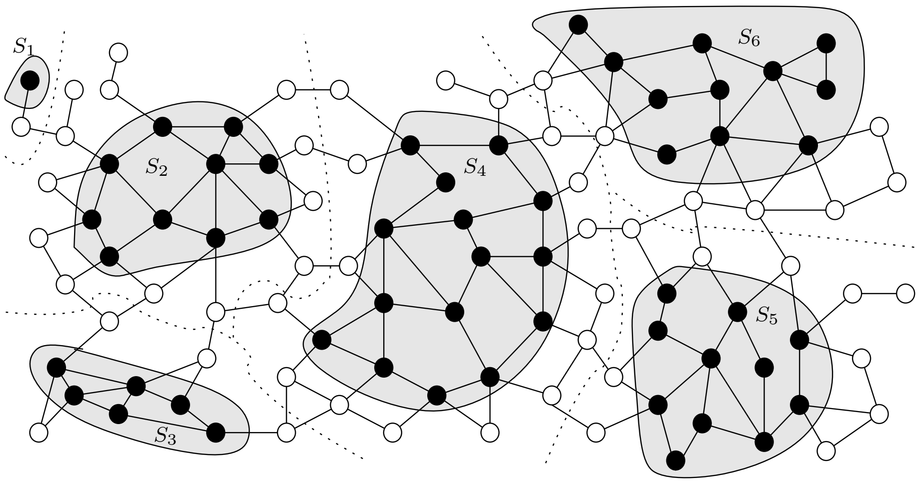


Fig. 2. Example for a 2-separated collection $\mathcal{S} = \{S_1, \ldots, S_6\}$

We refer to $\mathcal{S}$ as a 2-separated collection of subsets. An exa ple of such a 2-separated collection is presented in Figure 2. The grey areas ark the different subsets that ake up the collection, vertices which are not part of the collection, and thus separate the subsets are white.

The following le a shows that the su of the cardinalities of ini u do inating sets $D(S_i)$ for the subsets $S_i \in \mathcal{S}$ of a 2-separated collection for s a lower bound on the cardinality $|D(V)|$ of a ini u do inating set in G.

Lemma 1. *For a 2-separated collection* $\mathcal{S} = \{S_1, \ldots, S_k\}$ *in a graph* $G = (V, E)$*, we have*

$$|D(V)| \geq \sum_{i=1}^{k} |D(S_i)|.$$

Proof. For each subset $S_i \in \mathcal{S}$, consider the neighborhood $N(S_i)$. As a direct result of property (P), these neighborhoods are pairwise disjoint. Further ore, any vertex outside $N(S_i)$ has distance ore than one to all vertices in S_i. Thus, $D(V) \cap N(S_i)$ has to do inate all vertices in S_i, since $D(V)$ do inates the entire vertex set V.

On the other hand, also $D(S_i) \subset N(S_i)$ do inates S_i using a ini u nu ber of vertices in G. Therefore, we get

$$|D(V) \cap N(S_i)| \geq |D(S_i)|.$$

Co bining this for all subsets of the 2-separated collection, we get

$$|D(V)| \geq \sum_{i=1}^{k} |D(V) \cap N(S_i)| \geq \sum_{i=1}^{k} |D(S_i)|,$$

as clai ed. $\square$

Lemma 1 states that a 2-separated collection $\mathcal{S}$ leads to a lower bound on the cardinality of a MDS. Additionally, such a collection may help in getting an approximation of this cardinality. If we are able to enlarge the subsets S_i to subsets T_i in such a way that the dominating sets of the expansions are locally bounded and the unions of theses forms a dominating set for V, we get a global approximation for the MDS in G.

Corollary 1. *Let $\mathcal{S} = \{S_1, \dots, S_k\}$ be a 2-separated collection in $G = (V, E)$, and let $T_1, \dots, T_k$ be subsets of V with $S_i \subset T_i$ for all $i = 1, \dots, k$. If there exists a bound $\rho \geq 1$ such that*

$$|D(T_i)| \leq \rho \cdot |D(S_i)|$$

holds for all $i = 1, \dots, k$, and if $\bigcup_{i=1}^{k} D(T_i)$ forms a dominating set in G, the set $\bigcup_{i=1}^{k} D(T_i)$ is a ρ-approximation of an MDS in G.

Proof. $|\bigcup_{i=1}^{k} D(T_i)| \leq \sum_{i=1}^{k} |D(T_i)| \leq \rho \cdot \sum_{i=1}^{k} |D(S_i)| \leq \rho \cdot |D(V)|$. □

In the following section, we focus on the efficient construction of suitable subsets $T_i \subset V$, which contain a 2-separated collection $S_i \subset T_i$, in a way that a local $(1+\varepsilon)$-approximation can be guaranteed. Furthermore, we create these subsets in such a way that the union of the respective local dominating sets also dominates the entire set of vertices, resulting in a global $(1+\varepsilon)$-approximation for the MDS.

4 Efficient Construction of Suitable Subsets

From the previous discussion, recall that if we have a 2-separated collection $\mathcal{S} := \{S_1, \dots, S_k\}$, corresponding sets $T_i \supset S_i$ together with a bound of $(1 + \varepsilon)$ for the local dominating sets $D(S_i)$ and $D(T_i)$, then the union of the $D(T_i)$ satisfies the approximation bound required for a PTAS for the MDS problem. In this section, we show how to construct suitable subsets, for which the union of the local dominating sets also forms a dominating set for V. Furthermore, we prove that this can be achieved in polynomial running-time with respect to the size of the input instance for fixed $\varepsilon > 0$ if the input graph is a UDG. For ease of notation, let $\rho := (1 + \varepsilon)$ denote the desired approximation guarantee of the algorithm.

The basic idea of the construction is simple: we compute a local dominating set for a neighborhood of a vertex, and expand this neighborhood until we have formed sets S and $T \supset S$ which satisfy a desired bound. Then, we eliminate the current neighborhood and continue the same steps for the remaining graph.

In more detail, the algorithm works as follows. We start with an arbitrary vertex $v \in V$ and consider for $r = 0, 1, 2, \dots,$ the r-th neighborhoods $N^r(v)$. Starting with $N^0(v) = v$, we compute dominating sets of minimum cardinality for these neighborhoods as long as

$$|D(N^{r+2}(v))| > \rho \cdot |D(N^r(v))| \tag{1}$$

holds.

Denote by $\hat{r}_1$ the smallest r for which (1) is violated. We go on iteratively with this procedure for the graph induced by $V_{i+1} := V_i \setminus N^{\hat{r}_i+2}(v_i))$, where $V_1 := V$. The vertex $v_i \in V_i$ is chosen as an arbitrary central vertex of the neighborhoods. In further iterations, we thus consider for $r = 0, 1, 2, \ldots$ the neighborhoods $N^r(v_i)$ with respect to V_i, i.e. we have $N^r(v_i) \subset V_i$. Note that the dominating sets $D(.)$ are always computed with respect to the entire input graph G.

This process is then repeated until V_{i+1} contains no more vertices. Let $k \in \mathbb{N}$ be the total number of iterations. Obviously we have $k < n$. In the following, let $N_i, i = 1, \ldots, k$, denote the respective neighborhoods when the stopping criterion (1) is violated, i.e. $N_i := N^{\hat{r}_i+2}(v_i)$.

Looking at the dominating sets for these neighborhoods, $D(N_i)$, we have the following lemma which shows that a dominating set for the entire graph is given by the union of the sets $D(N_i)$.

Lemma 2. *For the collection of neighborhoods* $\{N_1, \ldots, N_k\}$ *created by the above algorithm, the union* $D := \bigcup_{i=1}^k D(N_i)$ *forms a dominating set for the input graph* G.

Proof. It is $V_{i+1} = V_i \setminus N_i$ and $N_i \subset V_i$, thus we have $V_i = V_{i+1} \cup N_i$. We stop the algorithm at $V_{k+1} = \varnothing$, which implies $V_k = N_k$. Therefore $\bigcup_{i=1}^k N_i = V$ by induction, and the claim follows. □

Next, we show that the solution set $D := \bigcup_{i=1}^k D(N_i)$ returned by the algorithm satisfies the $(1+\varepsilon)$-bound on the approximation. In particular, we show that $\mathcal{N} := \{N^{\hat{r}_1}(v_1), \ldots, N^{\hat{r}_k}(v_k)\}$ is a 2-separated collection in G, and then apply Corollary 1 to the respective local dominating sets $D(N_i)$.

Lemma 3. *The subsets* $N^{\hat{r}_i}(v_i), i = 1, \ldots, k$, *created by the algorithm form a 2-separated collection* $\mathcal{N} := \{N^{\hat{r}_1}(v_1), \ldots, N^{\hat{r}_k}(v_k)\}$ *in* G.

Proof. For ease of notation, let $\overline{N}_i$ denote the neighborhood $N^{\hat{r}_i}(v_i)$ for iteration $i \in \{1, \ldots, k\}$ of the algorithm. Recall that a 2-separated collection is characterized by property (P), i.e. vertices of two different subsets of the collection have distance more than 2 from one another.

Clearly, $\{\overline{N}_1, V_2\}$ is a 2-separated collection in G, since $V_2 = V \setminus N(N(\overline{N}_1))$. For induction, suppose that $\{\overline{N}_1, \ldots, \overline{N}_{i-1}, V_i\}$ is a 2-separated collection in G. Any vertex in V_i has distance more than 2 from any other vertex in $\overline{N}_1, \ldots, \overline{N}_{i-1}$. Considering $V_{i+1} = V_i \setminus N(N(\overline{N}_i))$, we see that both V_{i+1} and $\overline{N}_i$ satisfy (P). Therefore, $\{\overline{N}_1, \ldots, \overline{N}_i, V_{i+1}\}$ is a 2-separated collection. □

Additionally, the criterion (1) for stopping to expand the neighborhood guarantees that each pair of local dominating sets satisfies

$$|D(N_i)| \leq \rho \cdot |D(N^{\hat{r}_i}(v_i))| \quad (i = 1, \ldots, k). \tag{2}$$

Using Corollary 1 and Lemma 2, we now obtain the following result for the approximation.

Corollary 2. *The above algorithm returns a dominating set* $\bigcup_{i=1}^{k} D(N_i)$ *of cardinality no more than* $(1+\varepsilon)$ *the size of a minimum dominating set in* $G = (V, E)$. □

At this point, it is noteworthy to remind that this Corollary 2 is valid for any undirected graph G, even if it is not a unit disk graph.

It remains to show that the $(1+\varepsilon)$-approximation algorithm has polynomial running-time. In contrast to Corollary 2, the polynomial running-time relies on the fact that the input graph G is a unit disk graph. So, for the further discussion in this section, we assume G to be a unit disk graph.

The number k of iterations is bounded by $n = |V|$. We may thus limit the further discussion to one iteration only. Since any V_i during the execution of the algorithm again induces a unit disk graph, we focus w.l.o.g. on the graph $G = (V, E)$ in the first iteration. We show two things:

(1) we can compute the minimum dominating set $D(N^r(v))$ in polynomial time if the value of r is a constant or polynomially bounded; and
(2) there exists a constant bound for $\hat{r}_1$, i.e. the diameter of the largest neighborhood we need to consider until the stopping criterion (1) is violated.

Before showing that $D(N^r(v))$ can be computed efficiently, we need to introduce the notion of an independent set, and briefly state a key result for independent sets in UDGs.

Let $W \subset V$. A set $I \subset W$ is called an *independent set* if for every two vertices $u, v \in I$, there does not exist an edge $(u, v) \in E$. An independent set is called *maximal* in W if we cannot add any other vertex from W to I without violating the independence property (of no two vertices being adjacent). Clearly, any maximal independent set in W also dominates W.

For a UDG, the following result of [10] bounds the size of an independent set in the neighborhood $N^r(v)$. We give the short proof, since we rely on it in the next section.

Lemma 4. *Let* $G = (V, E)$ *be a UDG. Any independent set* $I^r \subset N^r(v), v \in V$, *satisfies*

$$|I^r| \leq (2r+1)^2 = O(r^2).$$

Proof. Let $f : V \to \mathbb{R}^2$ be a geometric representation of G. From the definition of a UDG, we conclude that any $w \in N^r(v)$ satisfies $\|f(v) - f(w)\| \leq 2r$.

Thus, I^r consists of pairwise disjoint disks of unit radius inside a disk of radius $2r+1$ around $f(v)$, and therefore $|I^r| \leq \pi(2r+1)^2/\pi$. □

As a consequence of Lemma 4, any independent set in $N^r(v)$ is polynomially bounded in r, including maximal independent sets. The cardinality of a minimum dominating set in $N^r(v)$ is bounded from above by the cardinality of a maximal independent set in $N^r(v)$, and, therefore, we get

Corollary 3. $|D(N^r(v))| \leq (2r+1)^2 = O(r^2)$. □

Assu ing r to be fixed or polyno ially bounded, a ini u do inating set $D(N^r(v))$ can then be co puted in polyno ial ti e, e.g. by co plete enu eration in ti e $O(n^\vartheta)$, with $\vartheta = O(r^2)$.

Next, we show that, for a UDG, there exists such a bound on $\hat{r}_1$, the first value of r which violates (1). This bound only depends on the approxi ation ratio ρ, and not on the size of the unit disk graph $G = (V, E)$ given as input.

Lemma 5. *There exists a constant $c = c(\rho)$ such that $\hat{r}_1 \leq c$, that is, the largest neighborhood to be considered during the iteration of the algorithm is bounded by a constant.*

Proof. It is $|D(N^0(v))| = |D(N^1(v))| = 1$, as the central vertex v do inates itself and all its neighbors.
Consider an arbitrary value of $r < \hat{r}_1$. First, if r is an even nu ber, due to the stopping criterion (1) we have

$$(2r+1)^2 \geq |D(N^r(v))| > \rho |D(N^{r-2}(v)| > \cdots > \rho^{\frac{r}{2}} |D(N^0(v)| = (\sqrt{\rho})^r.$$

Second, if r is an odd nu ber, we get

$$(2r+1)^2 \geq |D(N^r(v))| > \rho |D(N^{r-2}(v)| > \cdots > \rho^{\frac{r-1}{2}} |D(N^1(v)| = (\sqrt{\rho})^{r-1}.$$

Since $\rho > 1$, and thus $\sqrt{\rho} > 1$, in both cases the above inequalities have to be violated eventually. The bound on $\hat{r}_1$ when these inequalities are violated the first ti e only depends on ρ and not on the size of the overall graph G. The clai follows directly. □

Using $\log(1+\varepsilon) > 1/2 \cdot \varepsilon$ for s all values of ε, si ple calculations show that $c = O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$.

Su arizing, if the input graph is a UDG, each iteration has polyno ial running ti e, and therefore the presented algorith is a polyno ial-ti e approxi ation sche e for the MDS proble . Note that the co putation of $D(N^r(v))$ for the largest neighborhood, do inates the running-ti e of the algorith . Therefore, the overall ti e co plexity of the approxi ation is $O(n^c)$ with $c = O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$.

5 Discussion

Unit disk graphs are a special subclass of undirected graphs. As we have shown in the previous part, the presented algorith accepts an arbitrary undirected graph as input, and returns a do inating set of desired quality for this graph. However, the polyno ial running-ti e relies on the UDG characterization. This raises the question of robustness for algorith s designed for a restricted do ain [11]:

An algorith $\mathcal{A}$, defined on a set $\mathcal{G}$ of instances, is robust on a restricted class $\mathcal{U} \subset \mathcal{G}$ if it solves the proble for all instances in $\mathcal{U}$, and for instances not in $\mathcal{U}$, the algorith either solves the proble , or provides a certificate that the input does not belong to $\mathcal{U}$. Of course, the notion of a robust algorith is especially interesting when $\mathcal{A}$ has polyno ial running-ti e with respect to the size of the input instance, and the decision whether an instance belongs to the subclass $\mathcal{U} \subset \mathcal{G}$ is not as easy to decide. In our situation, $\mathcal{G}$ is the set of undirected graphs, $\mathcal{A}$ co putes a $(1+\varepsilon)$-approxi ation of the cardinality of an MDS, and $\mathcal{U}$ is the subclass of UDGs.

In case the input graph is a unit disk graph, the algorith always returns a $(1+\varepsilon)$-approxi ate do inating set in polyno ial running-ti e. Also, when the input is any undirected graph, such an approxi ation is returned. However, the polyno ial running-ti e in this case cannot be guaranteed. In the following, we consider the case that the input is no UDG.

The ti e co plexity of the algorith is a direct result of the possibility to bound the cardinality of a ini u do inating set in a neighborhood of bounded dia eter. This bound results fro the fact that a axi al independent set I^r in such a neighborhood is bounded, i.e. for the r-th neighborhood of a vertex $v \in V$, we have $|D(N^r(v))| \leq |I^r| \leq (2r+1)^2$.

If we now find a neighborhood $N^r(v)$ for which a ini u do inating set of size less than or equal to $(2r+1)^2$ cannot be found, we ter inate the algorith , and output the neighborhood $N^r(v)$ as a certificate to show that the input is no UDG. For this neighborhood, we can then construct a axi al independent set which has to violate Le a 4. This i ediately shows that the input graph cannot be a unit disk graph.

Note that for robustness, we do not need to explicitely consider the bound $r \leq c$ (Le a 5) on the dia eter of the neighborhoods $N^r(v)$, as this bound follows fro the polyno ial bound on the cardinality of the do inating sets in the neighborhoods.

The PTAS presented in this paper can be extended in a straightforward way to intersection graphs of other, related geo etric objects, e.g. the unit disk graph ay be defined using other geo etric nor s. Fro the discussion on the co -plexity in the previous section, it can be seen that a sufficient condition for the existence of a PTAS for the MDS proble in a geo etric intersection graph is given when there is a polyno ial bound on the ratio of axi u geo etric dia eter divided by ini u volu e of the objects that ake up the intersection graph (see Le a 4). Thus, the objects in consideration do not necessarily need to be of equal size or shape, e.g., the unit disks ay be replaced by disks with fixed lower and upper bounds on the radius. This condition includes Quasi Unit Disk Graphs which are used to give a ore realistic odel of a wireless, ad-hoc network [7,9]. An extension to a (fixed) di ension $d > 2$ is also i ediately possible.

6 Conclusion

In this paper, we present a new polyno ial-ti e approxi ation sche e for the ini u do inating set proble in unit disk graphs. The algorith does not need a geo etric representation of the graph to co pute a $(1+\varepsilon)$-approxi ate do inating set. In fact, it accepts any undirected graph as input and returns either a do inating set which satisfies the desired bound, or a certificate to show that the input graph is no UDG. Of course, if the input graph satisfies the characterization of a UDG, a do inating set is always returned.

The approxi ation algorith that results in the PTAS works by exploiting the fact that the graph can be divided into local neighborhoods, which have to be created keeping the global structure in ind. Inside these neighborhoods of guaranteed bounded dia eter, locally opti al solutions are available. The overall ti e co plexity of the (robust) approxi ation algorith is $n^{O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})}$.

References

1. B.S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994.
2. H. Breu and D.G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Computational Geometry. Theory and Applications*, 9(1-2):3–24, 1998.
3. X. Cheng, X. Huang, D. Li, W. Wu, and D.-Z. Du. A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. *Networks*, 42:202–208, 2003.
4. B. N. Clark, C. J. Colburn, and D. S. Johnson. Unit disks graphs. *Discrete Mathematics*, 86:165–177, 1990.
5. D.S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems. *Journal of the ACM*, 32(1):130–136, 1985.
6. H.B. Hunt III, M.V. Marathe, V. Radhakrishnan, S. S. Ravi, D.J. Rosenkrantz, and R.E. Stearns. NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. *J. Algorithms*, 26(2):238–274, 1998.
7. F. Kuhn, R. Wattenhofer, and A. Zollinger. Ad-hoc networks beyond unit disk graphs. In *Proceedings of the 1st ACM DIALM-POMC Joint Workshop on Foundations of Mobile Computing*, 2003.
8. M.V. Marathe, H. Breu, H.B. Hunt III, S. S. Ravi, and D.J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25:59–68, 1995.
9. T. Nieberg and J. Hurink. Wireless communication graphs. In *Proc. ISSNIP Intelligent Sensors, Sensor Networks and Information Processing Conference*, 2004.
10. T. Nieberg, J. Hurink, and W. Kern. A robust PTAS for maximum independent sets in unit disk graphs. In *Proceedings of the 30th workshop on graph theoretic concepts in computer science*, pages 214–221. LNCS 3353, 2004.
11. V. Raghavan and J. Spinrad. Robust algorithms for restricted domains. In *Proceedings of the 12th annual ACM-SIAM symposium on discrete algorithms*, pages 460–467. Society for Industrial and Applied Mathematics, 2001.

Speed Scaling of Tasks with Precedence Constraints

Kirk Pruhs[1,⋆], Rob van Stee[2], and Patchrawat Uthaiso but[1]

[1] Computer Science Department, University of Pittsburgh,
Pittsburgh PA 15260 USA
{kirk, utp}@cs.pitt.edu
[2] Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany
vanstee@ira.uka.de

Abstract. We consider the problem of speed scaling to conserve energy in a multiprocessor setting where there are precedence constraints between tasks, and where the performance measure is the makespan. That is, we consider an energy bounded version of the classic problem $Pm \mid prec \mid C_{max}$. We show that, without loss of generality, one need only consider constant power schedules. We then show how to reduce this problem to the problem $Qm \mid prec \mid C_{max}$ to obtain a poly-log(m)-approximation algorithm.

1 Introduction

1.1 Motivation

Power is now widely recognized as a first-class design constraint for odern co puting devices. This is particularly critical for obile devices, such as laptops, that rely on batteries for energy. While the power-consu ption of devices has been growing exponentially, battery capacities have been growing at a (odest) linear rate. One co on technique for anaging power is speed/voltage/power scaling. For exa ple, current icroprocessors fro AMD, Intel and Trans eta allow the speed of the icroprocessor to be set dyna ically. The otivation for speed scaling as an energy saving technique is that, as the speed to power function $P(s)$ in all devices is strictly convex, less aggregate energy is used if a task is run at a slower speed. The application of speed scaling requires a policy/algorith to deter ine the speed of the processor at each point in ti e. The processor speed should be adjusted so that the energy/power used is in so e sense justifiable by the i prove ent in perfor ance attained by running at this speed.

In this paper, we consider the proble of speed scaling to conserve energy in a ultiprocessor setting where there are precedence constraints between tasks, and where the perfor ance easure is the akespan, the ti e when the last

⋆ Supported in part by NSF grants CCR-0098752, ANI-0123705, CNS-0325353, CCF-0448196, and CCF-0514058.

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 307–319, 2006.

task finishes. We will denote this problem by $Sm \mid prec \mid C_{max}$. Without speed scaling, this problem is denoted by $Pm \mid prec \mid C_{max}$ in the standard three field scheduling notation [9]. Here m is the number of processors. This is a classic scheduling problem considered by Graham in his seminal paper [8] where he showed that list scheduling produces a $(2 - \frac{1}{m})$-approximate solution. In our speed scaling version, we make a standard assumption that there is a continuous function $P(s)$, such that if a processor is run at speed s, then its power, the amount of energy consumed per unit time, is $P(s) = s^{\alpha}$, for some $\alpha > 1$. For example, the well known cube-root rule for CMOS-based devices states that the speed s is roughly proportional to the cube-root of the power P, or equivalently, $P(s) = s^3$ (the power is proportional to the speed cubed) [16,4]. Our second objective is to minimize the total energy consumed. Energy is power integrated over time. Thus we consider a bicriteria problem, in that we want to optimize both makespan and total energy consumption. Bicriteria problems can be formalized in multiple ways depending on how one values one objective in relationship to the other. We say that a schedule S is a $O(a)$-energy $O(b)$-approximate if the makespan for S is at most bM and the energy used is at most aE where M is the makespan of an optimal schedule which uses E units of energy. The most obvious approach is to bound one of the objective functions and optimize the other. In our setting, where the energy of the battery may reasonably be assumed to be fixed and known, it seems perhaps most natural to bound the energy used, and to optimize makespan.

Power management for tasks with precedence constraints has received some attention in computer systems literature, see for example [10,14,20,15] and the references therein. These papers describe experimental results for various heuristics.

In the last few years, interest in power management has seeped over from the computer systems communities to the algorithmic community. For a survey of recent literature in the algorithmic community related to power management, see [11]. Research on algorithmic issues in power management is still at an early stage of development. Researchers are developing and analyzing algorithms to problems that appear particularly natural and/or that arise in some particular application. The eventual goal, after developing algorithms and analyses for many problems, is to develop a toolkit of widely applicable algorithmic methods for power management problems. While the algorithms and analyses that we present here are not extremely deep, we believe that our insights and techniques are quite natural, and have significant potential for future application in related problems.

1.2 Summary of Our Results

For simplicity, we state our results when we have a single objective of minimizing makespan, subject to a fixed energy constraint, although our results are a bit more general.

We begin by noting that several special cases of $Sm \mid prec \mid C_{max}$ are relatively easy. If there is only one processor ($S1 \mid prec \mid C_{max}$), then it is clear from the convexity of $P(s)$ that the optimal speed scaling policy is to run the processor at a constant speed; if there were times where the speeds were different, then by averaging the speeds one would not disturb the makespan, but the energy would be reduced. If there are no precedence constraints ($Sm \mid\mid C_{max}$), then the problem reduces to finding a partition of the jobs that minimizes the ℓ_α norm of the load. A PTAS for this problem is known [1]. One can also get an $O(1)$-approximate constant-speed schedule using Graham's list scheduling algorithm. So for these problems, speed scaling doesn't buy you more than an $O(1)$ factor in terms of energy savings.

We now turn to $Sm \mid prec \mid C_{max}$. We start by showing that there are instances where every schedule in which all machines have the same fixed speed has a makespan that is a factor of $\omega(1)$ more than the optimal makespan. The intuition is that if there are several jobs, on different processors, that are waiting for a particular job j, then j should be run with higher speed than if it were the case that no jobs were waiting on j. In contrast, we show that what should remain constant is the aggregate powers of the processors. That is, we show that in any locally optimal schedule, the sum of the powers at which the machines run is constant over time. Or equivalently, if the cube-root rule holds (power equals speed cubed), the sum of cubes of the machines speeds should be constant over time. Schedules with this property are called *constant power schedules.* We then show how to reduce our energy minimization problem to the problem of scheduling on machines of different speeds (without energy considerations). In the three field scheduling notation, this problem is denoted by $Q \mid prec \mid C_{max}$. Using the $O(\log m)$-approximate algorithms from [5,7], we can then obtain a $O(\log^2 m)$-energy $O(\log m)$-approximate algorithm for makespan. We then show a trade-off between energy and makespan. That is, an $O(a)$-energy $O(b)$-approximate schedule for makespan can be converted into $O(b \cdot a^{1/\alpha})$-approximate schedule. Thus we can then get an $O(\log^{1+2/\alpha} m)$-approximate algorithm for makespan.

We believe that the most interesting insight from these investigations is the observation that one can restrict one's attention to constant power schedules. This fact will also hold for several related problems.

1.3 Related Results

We will be brief here, and refer the reader to the recent survey [11] for more details. Theoretical investigations of speed scaling algorithms were initiated by Yao, Demers, and Shankar [18]. They considered the problem of minimizing energy usage when each task has to be finished on one machine by a predetermined deadline. Most of the results in the literature to date focus on deadline feasibility as the measure for the quality of the schedule. Yao, Demers, and Shankar [18] give an optimal offline greedy algorithm. The running time of this algorithm can be improved if the jobs form a tree structure [13]. Bansal, Kimbrel, and Pruhs [2] and Bansal and Pruhs [3] extend the results in [18] on online algorithms and introduce the problem of speed scaling to manage temperature. For jobs with a fixed priority, Yun and Kim [19] show that it is NP-hard to compute a minimum energy schedule. They also give an FPTAS for the problem. Kwon and Kim [12] give a polynomial-time algorithm for the case of a processor with discrete speeds. Chen, Kuo and Lu [6] give a PTAS for some special cases of this problem. Pruhs, Uthaisombut, and Woeginger [17] give some results on the flow time objective function.

2 Formal Problem Description

The setting for our problems consists of m variable-speed machines. If a machine is run at speed s, its power is $P(s) = s^\alpha$, $\alpha > 1$. The energy used by each machine is power integrated over time.

An instance consists of n jobs and an energy bound E. All jobs arrive at time 0. Each job i has an associated weight (or size) w_i. If this job is run consistently at speed s, it finishes in w_i/s units of time. There are precedence constraints among the jobs. If $i \prec j$, then job j cannot start before job i completes.

Each job must be run non-preemptively on some machine. The machines can change speed continuously over time. Although it is easy to see by the convexity of $P(s)$ that it is best to run each job at a constant speed.

A *schedule* specifies, for each time and each machine, which job to run and at what speed. A schedule is *feasible at energy level* E if it completes all jobs and the total amount of energy used is at most E. Suppose S is a schedule for an input instance I. We define a number of concepts which depend on S. The completion time of job i is denoted C_i^S. The makespan of S, denoted $C_{\max}^S$, is the maximum completion time of any job. A schedule is *optimal for energy level* E if it has the smallest makespan among all feasible schedules at energy level E. The goal of the problem is to find an optimal schedule for energy level E. We denote the problem as $Sm \mid prec \mid C_{\max}$.

We use s_i^S to denote the speed of job i. The execution time of i is denoted by x_i^S. Note that $x_i^S = w_i/s_i^S$. The power of job i is denoted by p_i^S. Note that $p_i^S = (s_i^S)^\alpha$. We use E_i^S to denote the energy used by job i. Note that $E_i^S = p_i^S x_i^S$. The total energy used in schedule S is denoted E^S. Note that $E^S = \sum_{i=1}^n E_i^S$. We drop the superscript S if the schedule is clear from the context.

3 No Precedence Constraints

As a warm-up, we consider the scheduling of tasks without precedence constraints. In this case we know that each machine will run at a fixed speed, since otherwise the energy use could be decreased without affecting the makespan by averaging the speed. We also know that each machine will finish at the same time, since otherwise some energy from a machine which finishes early could be transferred to machines which finish late, decreasing the makespan. Furthermore there will be no gaps in the schedule.

For any schedule, denote the makespan by M, and denote the load on machine j, which is the sum of the weights of the jobs on machine j, by L_j. Since each machine runs at a fixed speed, in this section we denote by s_j the speed of *machine* j, by p_j its power, and by E_j its energy used. By our observations so far we have $s_j = L_j/M$.

The energy used by achine j is

$$E_j = p_j M = s_j^\alpha M = \frac{L_j^\alpha}{M^{\alpha-1}} \ .$$

We can su this over all the achines and rewrite it as

$$M^{\alpha-1} = \frac{1}{E}\sum_j L_j^\alpha. \qquad (1)$$

It turns out that ini izing the akespan is equivalent to ini izing the ℓ_α nor of the loads. For this we can use the PTAS for identical achines given by Azar et al. [1]. Denote the opti al loads by $\text{OPT}_1, \dots, \text{OPT}_m$. Si ilarly to (1), we have

$$\text{OPT}^{\alpha-1} = \frac{1}{E}\sum_j \text{OPT}_j^\alpha, \qquad (2)$$

where OPT is the opti al akespan. For any $\varepsilon > 0$, we can find loads $L_1, \dots, L_m$ in polyno ial ti e such that $\sum_j L_j^\alpha \le (1+\varepsilon)\sum_j \text{OPT}_j^\alpha$. For the corresponding akespan M it now follows fro (1) and (2) that

$$M^{\alpha-1} = \frac{1}{E}\sum_j L_j^\alpha \le (1+\varepsilon)\cdot\frac{1}{E}\sum_j \text{OPT}_j^\alpha = (1+\varepsilon)\text{OPT}^{\alpha-1}$$

or

$$M \le (1+\varepsilon)^{1/(\alpha-1)}\text{OPT}.$$

Thus this gives us a PTAS for the proble $Sm \mid\mid C_{\max}$. For $\alpha > 2$, it even gives a better approxi ation for any fixed running ti e co pared to the original PTAS.

4 Main Results

4.1 One Speed for All Machines

Suppose all achines run at a fixed speed s. We show that under this constraint, it is not possible to get a good approxi ation of the opti al akespan. For si plicity, we only consider the special case $\alpha = 3$.

Consider the following input: one job of size $m^{1/3}$ and m jobs of size 1, which can only start after the first job has finished. Suppose the total energy available is $E = 2m$. It is possible to run the large job at a speed of $s_1 = m^{1/3}$ and all others at a speed of 1. The akespan of this schedule is 2, and the total a ount of energy required is $s_1^3 + m = 2m$.

Now consider an approxi ation algorith with a fixed speed s. The total ti e for which this speed is required is the total size of all the jobs divided by s. Thus s ust satisfy $s^3(m^{1/3}+m)/s \le E = 2m$, or $s^2 \le 2m/(m^{1/3}+m)$. This clearly i plies $s \le 2$, but then the akespan is at least $m^{1/3}/2$. Thus the approxi ation ratio is $\Omega(m^{1/3})$.

In contrast, we will use achines that have different speeds, but where the total power used by the achines is constant over ti e.

4.2 The Power Equality

Given a schedule S of an input instance I, we define the *schedule-based constraint* $\prec_S$ a ong jobs in I as follows. For any jobs i and j, $i \prec_S j$ if and only if $i \prec j$ in I, or i runs before j on the sa e achine in S. Suppose S is a schedule where each job is run at a constant speed. The *power relation graph* of a schedule S of an instance I is a vertex-weighted directed graph created as follows:

- For each job i, create vertices u_i and v_i, each with weight p_i where p_i is the power at which job i is run.
- In S, if $i \prec_S j$ and job j starts as soon as job i finishes (aybe on different achines), then create a directed edge (v_i, u_j).

Basically, the power relation graph G tells us which pairs of jobs on the sa e achine run back to back, and which pairs of jobs with precedence constraint $\prec$ between the run back to back. For an exa ple, see Figure 1.

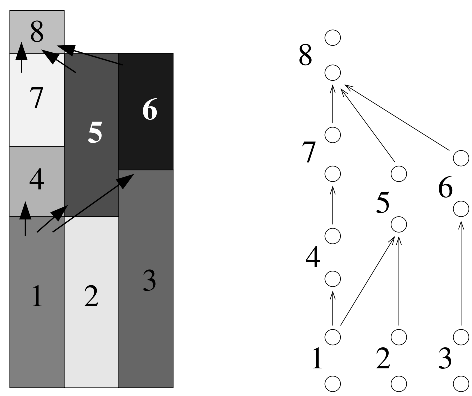


Fig. 1. An example of a schedule and the corresponding power relation graph. Note that the precedence constraint between jobs 1 and 6 is not represented in the graph, but on the other hand there is an edge between, e.g., jobs 2 and 5 since they run back to back on the same machine. In this example, the graph has four connected components.

In this paper, a connected co ponent of a directed graph G refers to a subgraph of G that corresponds to a connected co ponent of the underlying undirected graph of G. Suppose C is a connected co ponent of a power relation graph G. Define $H(C) = \{u \mid (v, u) \in C\}$ and $T(C) = \{v \mid (v, u) \in C\}$. Note that $H(C)$ and $T(C)$ is the set of vertices at the heads and tails, respectively, of directed edges in C. If C contains only one vertex, then $H(C) = T(C) = \emptyset$. The co pletion of jobs in $T(C)$ and the start of jobs in $H(C)$ all occur at the sa e ti e. If ti e t is when this occurs, we say that C *occurs at time* t. We say that a connected co ponent C satisfies the *power equality* if

$$\sum_{i:u_i \in H(C)} p_i = \sum_{i:v_i \in T(C)} p_i$$

Note that p_i is the power at which job i is run, and is also the weight of vertices u_i and v_i. We say that a power relation graph G satisfies the *power equality* if all connected components of G satisfy the power equality.

Lemma 1. *If a schedule S is optimal, then each job is run at a constant speed.*

Proof (Proof sketch). Suppose S is an optimal schedule such that some job i does not run at a constant speed. By averaging the speeds in the interval that i runs, the execution time of i would not change, but the energy would be reduced, since the power is a convex function of the speed. A contradiction. □

Lemma 2. *If S is an optimal schedule for some energy level E, then the power relation graph G of S satisfies the power equality.*

Proof. The idea of the proof is to consider an arbitrary component C of the power relation graph G of an optimal schedule S. Then create a new schedule S' from S by slightly stretching and compressing jobs in C. Since S is optimal, S' cannot use a smaller amount of energy. By creating an equality to represent this relationship and solving it, we have that C must satisfy the power equality relation.

Now we give the detail. Consider any connected component C of G. If C contains only one vertex, then it immediately follows that C satisfies the power equality because $T(C) = H(C) = \emptyset$. Therefore, suppose C contains two or more vertices. Let $\varepsilon \neq 0$ be a small number such that $x_i + \varepsilon > 0$ for any job i in $T(C)$, and $x_i - \varepsilon > 0$ for any job i in $H(C)$. Note that we allow ε to be either positive or negative. For simplicity on the first reading, it is easy to think of ε as a small positive number. We create a new schedule S' from schedule S by increasing the execution time of every job in $T(C)$ by ε, and decreasing the execution time of every job in $H(C)$ by ε. Note the following:

(1) The execution time of job i in $T(C)$ in S' is positive because $x_i + \varepsilon > 0$.
(2) The execution time of job i in $H(C)$ in S' is positive because $x_i - \varepsilon > 0$.
(3) For $|\varepsilon|$ small enough, S' has the same power relation graph as S.

Therefore, S' is a feasible schedule having the same power relation graph as S. Observe that the makespan of S' remains the same as that of S. The change in the energy used, $\Delta E(\varepsilon)$, is

$$\begin{aligned}\Delta E(\varepsilon) &= E^{S'} - E^{S}\\ &= \sum_{i:v_i \in T(C)} \left(E_i^{S'} - E_i^{S}\right) + \sum_{i:u_i \in H(C)} \left(E_i^{S'} - E_i^{S}\right)\\ &= \sum_{i:v_i \in T(C)} \left(\frac{w_i^{\alpha}}{(x_i + \varepsilon)^{\alpha-1}} - \frac{w_i^{\alpha}}{x_i^{\alpha-1}}\right) + \sum_{i:u_i \in H(C)} \left(\frac{w_i^{\alpha}}{(x_i - \varepsilon)^{\alpha-1}} - \frac{w_i^{\alpha}}{x_i^{\alpha-1}}\right).\end{aligned}$$

Since S is optimal, $\Delta E(\varepsilon)$ must be non-negative. Otherwise, we could reinvest the energy saved by this change to obtain a schedule with a better makespan.

Since the derivative $\Delta E'(\varepsilon)$ is continuous for $|\varepsilon|$ small enough, we must have $\Delta E'(0) = 0$. We have

$$\Delta E'(\varepsilon) = \sum_{i:v_i \in T(C)} \frac{(1-\alpha)w_i^\alpha}{(x_i+\varepsilon)^\alpha} + \sum_{i:u_i \in H(C)} \frac{(\alpha-1)w_i^\alpha}{(x_i-\varepsilon)^\alpha}$$

Substitute $\varepsilon = 0$ and solve for $\Delta E'(0) = 0$.

$$\begin{aligned}
\Delta E'(0) &= 0 \\
\sum_{i:v_i \in T(C)} \frac{(1-\alpha)w_i^\alpha}{x_i^\alpha} - \sum_{i:u_i \in H(C)} \frac{(1-\alpha)w_i^\alpha}{x_i^\alpha} &= 0 \\
\sum_{i:v_i \in T(C)} \frac{(1-\alpha)w_i^\alpha}{x_i^\alpha} &= \sum_{i:u_i \in H(C)} \frac{(1-\alpha)w_i^\alpha}{x_i^\alpha} \\
\sum_{i:v_i \in T(C)} s_i^\alpha &= \sum_{i:u_i \in H(C)} s_i^\alpha \\
\sum_{i:v_i \in T(C)} p_i &= \sum_{i:u_i \in H(C)} p_i
\end{aligned}$$

Thus, this connected component C satisfies the power equality. Since C is an arbitrarily chosen connected component in G, then G satisfies the power equality, and the result follows. □

Let $p(k,t)$ be the power at which machine k runs at time t. By convention, if machine k is idle at time t, then $p(k,t) = 0$. Also, by convention if job i starts at time t_1 and completes at time t_2, we say that it runs in the open-close interval $(t_1, t_2]$. Therefore, $p(k,t)$ is well-defined at a time t when a job has just finished and another has just started; the value of $p(k,t)$ is equal to the power of the finishing job.

Lemma 3. *If S is an optimal schedule for some energy level E, there exists a constant p such that at any time t, $\sum_{k=1}^m p(k,t) = p$, i.e. the sum of the powers of all machines at time t is p.*

Proof. Suppose S is an optimal schedule. Consider any time t where $0 < t \leq C_{\max}^S$. Let t' be any time after t such that no jobs start or complete in the interval $(t, t']$. Note that this does not exclude the possibility that some jobs start or complete at time t. We will show that $\sum_{k=1}^m p(k,t) = \sum_{k=1}^m p(k,t')$. If this is the case, then the result follows.

On the one hand, if no jobs start or complete at time t, then the same set of jobs are running at time t and t'. From Lemma 1, each job runs at a constant speed at all time. This also means that each job runs at a constant power at all time. Since the same set of jobs are running at time t and t', then $\sum_{k=1}^m p(k,t) = \sum_{k=1}^m p(k,t')$.

On the other hand, if some jobs start or complete at time t, then consider the power relation graph G of S. The jobs which start or complete at time t

correspond to vertices in components occurring at time t. Note that if some component contains only one vertex, then S is not optimal, because the corresponding job could be run at a slower speed and start earlier (if it starts at time t) or finish later without violating any precedence constraints. This would reduce the total amount of energy used, which could be reinvested elsewhere to get a better schedule. Thus all components contain at least one edge. From Lemma 2, the sum of powers of jobs that complete at time t is equal to the sum of powers of jobs that start at time t. And since no jobs start or complete in the interval $(t, t']$, then again $\sum_{k=1}^{m} p(k,t) = \sum_{k=1}^{m} p(k,t')$. □

4.3 Algorithm

Lemma 3 implies that the total power at which all the machines run is constant over time (only the distribution of the power over the machines may vary). We will describe a scheme to use this lemma to relate $Sm \mid prec \mid C_{\max}$ to the problem $Q \mid prec \mid C_{\max}$. Then, we can use an approximation algorithm for the latter problem by Chekuri and Bender [5] to obtain an approximate schedule. The schedule is then scaled so that the total amount of energy used is within the energy bound E.

Let $\bar{p}$ be the sum of powers at which the machines run in the optimal schedule $\text{OPT}(I, E)$. Since energy is power times makespan, we have $\bar{p} = E/\text{OPT}(I, E)$. However, an approximation algorithm does not know the value of $\text{OPT}(I, E)$, so it cannot immediately compute $\bar{p}$. Nevertheless, we will assume that we know the value of $\bar{p}$. The value of $\bar{p}$ can be approximated using binary search, and this will be discussed later. Given $\bar{p}$, define the set $M(\bar{p})$ to consist of the following *fixed speed* machines: 1 machine running at power $\bar{p}$, 2 machines running at power $\bar{p}/2$, and in general 2^{i-1} machines running at power $\bar{p}/i$ for $i = 1, 2, ..., \lfloor \log m \rfloor$. Denoting the total number of machines so far by m', there are an additional $m - m'$ machines running at power $\bar{p}/(1 + \lfloor \log m \rfloor)$. Thus there are m machines in the set $M(\bar{p})$, but the total power is at most $\bar{p}(1 + \log m)$. We show in the following lemma that if the optimal algorithm is given the choice between m variable speed machines with total energy E and the set $M(\bar{p})$ of machines just described, it will always take the latter, since the makespan will be smaller.

Lemma 4. *We have*

$$\text{OPT}_{M(\bar{p})}(I) \leq \text{OPT}(I, E),$$

where $\text{OPT}_{M(\bar{p})}(I)$ is the makespan of the optimal schedule using fixed speed machines in the set $M(\bar{p})$, and $\text{OPT}(I, E)$ is the makespan of the optimal schedule using m variable-speed machines with energy bound E.

Proof. Consider the schedule of OPT with variable speed machines and energy bound E at some time t. Denote this OPT by OPT_1 and the OPT which uses the prescribed set of machines by OPT_2. Denote the power of machine i of OPT_1 at this time by p_i (note that this is a different notation from the one we use elsewhere in the paper) and sort the machines by decreasing p_i. Now we simply assign the job on machine 1 to the machine of power $\bar{p}$ of OPT_2, and for $i \geq 1$ we assign the jobs on machines $2^i, \ldots, 2^{i+1}-1$ to the machines of power $\bar{p}/2^i$ of OPT_2.

Clearly, $p_1 \leq \bar{p}$, since no machine can use more than $\bar{p}$ power at any time. In general, we have that $p_j \leq \bar{p}/j$ for $j = 1, \ldots, m$. If we can show that the first machine in any power group has at least as much power as the corresponding machine of OPT_1, this holds for all the machines. But since machine 2^i of OPT_2 has power exactly $\bar{p}/2^i$, this follows immediately.

It follows that OPT_2 allocates each individual job at least as much power as OPT_1 at time t. We can apply this transformation for any time t, where we only need to take into account that OPT_2 might finish some jobs earlier than OPT_1. So the schedule for OPT_2 might contain unnecessary gaps, but it is a valid schedule, which proves the lemma. □

To construct an approximate schedule, we assume the value of $\bar{p}$ is known, and the set of fixed speed machines in $M(\bar{p})$ will be used. The schedule is created using the algorithm by Chekuri and Bender [5]. The schedule created may use too much energy. To fix this, the speeds of all jobs are decreased so that the total energy used is within E at the expense of having a longer makespan. The steps are given in subroutine *FindSchedule* in Figure 2.

$FindSchedule(I, p)$

1. Find a schedule for instance I and machines in the set $M(p)$ using Chekuri and Bender's algorithm [5].
2. Reduce the speed of all machines by a factor of $\log^{2/\alpha} m$
3. Return the resulting schedule.

$\text{ALG}(I, E)$

1. Set $p^* = \left(\frac{E}{mW}\right)^{\frac{\alpha}{\alpha-1}}$ where W is the total weight of all jobs divided by m.
2. Using binary search on $[0, p^*]$ with p as the search variable, find the largest value for p such that this 2-step process returns true. Binary search terminates when the binary search interval is shorter than 1.
   (a) Call $FindSchedule(I, p)$.
   (b) If for the schedule obtained we have $\sum_{i=1}^{n} s_i^{\alpha-1} w_i \leq E$, return true

Fig. 2. Our speed scaling algorithm. The input consists a set of jobs I and an energy bound E.

4.4 Analysis

Lemma 5. *Suppose $p = E/OPT(I, E)$. Subroutine $FindSchedule(I, p)$ creates a schedule which has makespan $O(\log^{1+2/\alpha} m)\text{OPT}(I, E)$ and uses energy $O(1)E$.*

Proof. Let S_1 and S_2 denote the schedules obtained in steps 1 and 2 of the subroutine $FindSchedule(I, p)$, respectively. In an abuse of notation, we will also use S_1 and S_2 to refer to the makespans of these schedules. Schedule S_2 is the one returned by *FindSchedule*. First we analyze the makespan. From Chekuri and Bender [5], $S_1 = O(\log m)\text{OPT}_{M(p)}(I)$. In step 2, the speed of every job decreases by a factor of $\log^{2/\alpha} m$. Thus, the makespan increases by a factor of $\log^{2/\alpha} m$. From Lemma 4, $\text{OPT}_{M(p)}(I) \leq \text{OPT}(I, e)$. Therefore, taken together, we have

$$\begin{aligned} S_2 &= (\log^{2/\alpha} m) S_1 = (\log^{2/\alpha} m) O(\log m)\text{OPT}_{M(p)}(I) \\ &= O(\log^{1+2/\alpha} m)\text{OPT}(I, E). \end{aligned}$$

Next we analyze the energy. The machines in the schedule $\text{OPT}(I, E)$ run for $\text{OPT}(I, E)$ time units at the total power of $p = E/\text{OPT}(I, E)$ consuming a total energy of E. Recall that if all machines in $M(p)$ are busy, the total power is at most $p(1 + \log m)$.

Schedule S_1 runs the machines for $O(\log m)\text{OPT}_{M(p)}(I)$ time units at the total power at most $p(1 + \log m)$. Thus, it uses energy at most

$$p(1 + \log m)\, O(\log m)\text{OPT}_{M(p)}(I) \;\leq\; O(\log^2 m)\, p\, \text{OPT}(I, A) \;=\; O(\log^2 m)\, A \quad (3)$$

where the inequality follows from Lemma 4. The speeds at which the machines in S_2 run are $\log^{2/\alpha} m$ slower than those in $M(p)$, which S_1 uses. Thus, the total power at which the machines in S_2 run is $\log^2 m$ times smaller than that of S_1. By (3), this is $O(1)A$. □

Note that when we decrease the speed in S_2 by some constant factor, the makespan increases by that factor and the energy decreases by a larger constant factor. To find the value of $\bar{p}$, we use binary search in the interval $[0, p^*]$ where p^* is an initial upper bound to be computed shortly. We continue until the length of the interval is at most 1. We then use the left endpoint of this interval as our power. Now we compute the initial upper bound p^*. For a given schedule, the total energy used is

$$\sum_{i=1}^{n} p_i x_i = \sum_{i=1}^{n} s_i^{\alpha} w_i / s_i = \sum_{i=1}^{n} s_i^{\alpha-1} w_i.$$

The best scenario that could happen for the optimal algorithm is when the work is evenly distributed on all the machines and all the machines run at the same speed at all time. Let W be the total weight of all the jobs divided by m. Completing W units of work at a speed of s requires $s^{\alpha-1}W$ units of energy. If each of the m machines processes W units of work, then it takes a total $mWs^{\alpha-1}$ units of energy. This must be less than E. For the speed we find $s^{\alpha-1} \leq E/mW$ and thus $p^{\frac{\alpha-1}{\alpha}} \leq E/mW$. This gives us an initial upper bound for p for the binary search:

$$p \leq p^* = \left(\frac{E}{mW}\right)^{\frac{\alpha}{\alpha-1}}.$$

OPT does not use a higher power than this, because then it would run out of energy before all jobs co plete.

Fro Le a 5 and our analysis above, the following theore holds.

Theorem 1. ALG *is an* $O(\log^{1+2/\alpha} m)$*-approximation algorithm for the problem* $Sm \mid prec \mid C_{\max}$ *where the power is equal to the speed raised to the power of* α *and* $\alpha > 1$*.*

5 Conclusions

Speed scaling to anage power is an i portant area of application that is worthy of further acade ic investigation. For a survey, including proposed avenues for further investigations, we reco end the survey paper [11].

References

1. Noga Alon, Yossi Azar, Gerhard Woeginger, and Tal Yadid. Approximation schemes for scheduling. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 493–500, 1997.
2. Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Dynamic speed scaling to manage energy and temperature. In *IEEE Syposium on Foundations of Computer Science*, pages 520 – 529, 2004.
3. Nikhil Bansal and Kirk Pruhs. Speed scaling to manage temperature. In *Symposium on Theoretical Aspects of Computer Science*, pages 460–471, 2005.
4. David M. Brooks, Pradip Bose, Stanley E. Schuster, Hans Jacobson, Prabhakar N. Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor Zyuban, Manish Gupta, and Peter W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.
5. Chandra Chekuri and Michael A. Bender. An efficient approximation algorithm for minimizing makespan on uniformly related machines. *Journal of Algorithms*, 41:212–224, 2001.
6. Jian-Jia Chen, Tei-Wei Kuo, and Hsueh-I Lu. Power-saving scheduling for weakly dynamic voltage scaling devices. In *Workshop on Algorithms and Data Structures*, 2005. To appear.
7. Fabián A. Chudak and David B. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 581–590, 1997.
8. Ronald L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Techical Journal*, 45:1563–1581, 1966.
9. Ronald L. Graham, Eugene Lawler, Jan Karel Lenstra, and Alexander H. G. Rinnooy Kan. Optimization and approximation in deterministic scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.
10. Flavius Gruian and Krzysztof Kuchcinski. Lenes: Task-scheduling for low-energy systems using variable voltage processors. In *Asia South Pacific - Design Automation Conference*, pages 449–455, 2001.
11. Sandy Irani and Kirk Pruhs. Algorithmic problems in power management. *SIGACT News*, 2005.

12. Woo-Cheol Kwon and Taewhan Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. *ACM Transactions on Embedded Computing Systems (TECS)*, 4(1):211–230, 2005.
13. Minming Li, Becky Jie Liu, and Frances F. Yao. Min-energy voltage allocation for tree-structured tasks. In *11th International Computing and Combinatorics Conference (COCOON 2005)*, 2005. To appear.
14. Jiong Luo and Niraj K. Jha. Power-conscious joint scheduling of periodic task graphs and aperiodic task graphs in distributed real-time embedded systems. In *International Conference on Computer Aided Design*, pages 357–364, 2000.
15. Ramesh Mishra, Namrata Rastogi, Dakai Zhu, Daniel Moss, and Rami G. Melhem. Energy aware scheduling for distributed real-time systems. In *International Parallel and Distributed Processing Symposium*, page 21, 2003.
16. Trevor Mudge. Power: A first-class architectural design constraint. *Computer*, 34(4):52–58, 2001.
17. Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard Woeginger. Getting the best response for your erg. In *Scandanavian Workshop on Algorithms and Theory*, pages 14–25, 2004.
18. F. Frances Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *IEEE Syposium on Foundations of Computer Science (FOCS 1995)*, pages 374–382, 1995.
19. Han-Saem Yun and Jihong Kim. On energy-optimal voltage scheduling for fixed priority hard real-time systems. *ACM Transactions on Embedded Computing Systems*, 2(3):393–430, 2003.
20. Yumin Zhang, Xiaobo Hu, and Danny Z. Chen. Task scheduling and voltage selection for energy minimization. In *Design Automation Conference*, pages 183–188, 2002.

Partial Multicuts in Trees*

Asaf Levin[1] and Danny Segev[2]

[1] Department of Statistics, The Hebrew University,
Jerusalem 91905, Israel
levinas@mscc.huji.ac.il
[2] School of Mathematical Sciences, Tel-Aviv University,
Tel-Aviv 69978, Israel
segevd@post.tau.ac.il

Abstract. Let $T = (V, E)$ be an undirected tree, in which each edge is associated with a non-negative cost, and let $\{s_1, t_1\}, \dots, \{s_k, t_k\}$ be a collection of k distinct pairs of vertices. Given a requirement parameter $t \leq k$, the *partial multicut on a tree* problem asks to find a minimum cost set of edges whose removal from T disconnects at least t out of these k pairs. This problem generalizes the well-known *multicut on a tree* problem, in which we are required to disconnect all given pairs.

The main contribution of this paper is an $(\frac{8}{3} + \epsilon)$-approximation algorithm for partial multicut on a tree, whose run time is strongly polynomial for any fixed $\epsilon > 0$. This result is achieved by introducing problem-specific insight to the general framework of using the Lagrangian relaxation technique in approximation algorithms. Our algorithm utilizes a heuristic for the closely related prize-collecting variant, in which we are not required to disconnect all pairs, but rather incur penalties for failing to do so. We provide a Lagrangian multiplier preserving algorithm for the latter problem, with an approximation factor of 2. Finally, we present a new 2-approximation algorithm for multicut on a tree, based on LP-rounding.

1 Introduction

In this paper we address the *partial multicut on a tree* problem. The input for this problem consists of an undirected tree $T = (V, E)$, in which each edge $e \in E$ is associated with a non-negative cost c_e, and a collection of k distinct pairs of vertices, $\{s_1, t_1\}, \dots, \{s_k, t_k\}$. For $1 \leq i \leq k$, the pair $\{s_i, t_i\}$ is said to be *separated* by the edge set $D \subseteq E$ if it is not contained in a single connected component of $T - D$. In other words, the removal of D disconnects s_i and t_i. Given a requirement parameter $t \leq k$, the objective is to find a minimum cost set of edges that separates at least t out of the k pairs. In spite of these seemingly simple settings, we are not aware of any previous study of this problem.

* Due to space limitations, several proofs are omitted from this extended abstract. We refer the reader to the full version of this paper [13], in which all missing proofs are provided.

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 320–333, 2006.

Partial multicut on a tree contains as a special case the well-known *multicut on a tree* problem, in which we are required to separate all given pairs. Garg, Vazirani and Yannakakis [8] demonstrated that this problem is at least as hard to approximate as vertex cover, even in unweighted trees of height 1. In addition, they presented a primal-dual algorithm that constructs a feasible solution whose cost is at most twice the optimum. We refer to this algorithm as the GVY algorithm, and provide additional details on its analysis in the sequel, since it serves as one of the building blocks of our algorithm.

When the underlying graph is not restricted to be a tree, the multicut problem becomes much harder. Dahlhaus, Johnson, Papadimitriou, Seymour and Yannakakis [4] proved that the multicut problem is NP-hard for all fixed $k \geq 3$, even when the cost of each edge is 1. Very recently, an arbitrarily large constant factor hardness was given by Chawla, Krauthgamer, Kumar, Rabani and Sivakumar [3], assuming the Unique Games Conjecture of Khot [12]. A stronger version of this conjecture leads to a hardness result of $\Omega(\log \log n)$. On the positive side however, Garg et al. [7] used the region growing scheme to obtain an $O(\log k)$-approximation algorithm for the multicut problem.

The partial multicut on a tree problem can also be considered in a different context. Given a ground set of elements $U = \{e_1, \ldots, e_n\}$, a collection $S_1, \ldots, S_m$ of subsets of U with non-negative costs $c(S_i)$ and a parameter $t \leq n$, *partial cover* is the problem of finding a minimum cost subcollection of sets that covers at least t elements. Note that a pair $\{s_i, t_i\}$ is separated by $D \subseteq E$ if this set of edges contains at least one edge from the unique path connecting s_i and t_i in T, which we denote by $[s_i, t_i]$. This observation allows us to interpret partial multicut on a tree as a special case of partial cover. The elements to cover are the paths $[s_i, t_i]$, $1 \leq i \leq k$, and the sets correspond to the edges of T. An edge $e \in E$ covers those paths to which it belongs, with cost c_e.

The partial cover problem received a great deal of attention in recent years. When $t = n$, partial cover reduces to the standard *set cover* problem, in which we wish to cover the entire universe of elements. Therefore, partial cover cannot be approximated within a ratio of $(1 - \epsilon) \ln n$ for any $\epsilon > 0$, unless NP $\subset$ TIME$(n^{O(\log \log n)})$ [5]. Slavík [15] generalized the analysis of the greedy set cover algorithm and proved that it guarantees an $H(t)$-approximation for partial cover. For the special case where each element appears in at most f sets, Bar-Yehuda [1] gave an f-approximation using the local-ratio method. This case was also studied by Gandhi, Khuller and Srinivasan [6], who achieved a similar approximation ratio using a primal-dual algorithm. Unfortunately, simple examples show that none of these algorithms provides a constant factor approximation for partial multicut on a tree.

A closely related generalization of multicut is the *prize-collecting multicut* problem. In this variant we are not required to separate all pairs. However, if the set of edges we pick does not separate a pair $\{s_i, t_i\}$, we incur a penalty p_i. The objective is to find a set of edges $D \subseteq E$ that minimizes the cost of D plus the penalties of unseparated pairs.

For the remainder of this paper, the term "on a tree" is omitted whenever we discuss any of the problems or algorithms considered here. We remark that none of our results holds when the underlying graph is not a tree.

1.1 Results and Techniques

In Section 2 we present an interpretation of the prize-collecting multicut problem as an equivalent multicut problem, which is created by adding new leaf vertices to the original tree T and modifying the collection of pairs to be separated. A 2-approximation for this problem immediately follows by applying the GVY algorithm to the resulting multicut instance. However, the partial multicut algorithm we suggest uses a prize-collecting heuristic as a subroutine, and requires a bound stronger than the one obtained by this straightforward approach.

Specifically, the prize-collecting algorithm should possess the *Lagrangian Multiplier Preserving* (LMP) property[1]: If we denote by C the total edge costs and by P the total penalties of unseparated pairs, then for some constant $r \geq 1$ we have $C + rP \leq r\text{OPT}$, where OPT is the cost of an optimal solution. To achieve this property, we prove that our reduction produces multicut instances whose unique configuration forces the GVY algorithm to eliminate edges that are not part of the original tree, as long as feasibility is maintained. This corresponds to discarding redundant penalties from the prize-collecting solution. By exploiting the special structural properties of the resulting solution, we strengthen the analysis of Garg et al. and prove that the LMP property is satisfied with factor $r = 2$.

In Section 3 we present the main result of this paper, an $(\frac{8}{3} + \epsilon)$-approximation algorithm for the partial multicut problem, whose run time is strongly polynomial for any fixed $\epsilon > 0$. It is important to note that this algorithm relies heavily on a preprocessing step in which we "guess" certain attributes of a fixed arbitrary optimal solution. This step is implemented using an exhaustive search that involves $O(n^{1/\epsilon})$ calls to the procedure described below.

Although our algorithm is based on problem-specific methods, it is guided by the general framework of using the Lagrangian relaxation technique in approximation algorithms, originally suggested by Jain and Vazirani [11]. With respect to a natural integer programming formulation of partial multicut, we relax the complicating constraint that at most $k - t$ pairs are not separated, and move it to the objective function together with a Lagrangian multiplier λ. For any fixed value of λ, this operation results in an instance of the prize-collecting multicut problem, with an additional constant term in the objective function.

Next, we use the prize-collecting algorithm to conduct a binary search, at the end of which we find $\lambda_1 \geq \lambda_2$ such that: For λ_1, the algorithm separates $t_1 \geq t$ pairs by picking the edge set D_1; for λ_2, it separates $t_2 \leq t$ pairs by picking D_2. We observe that D_1 and D_2 by themselves are not good solutions, since the cost of D_1 can be arbitrarily large with respect to that of the optimal solution, and since D_2 is generally not feasible. To resolve this problem, we devise an auxiliary procedure that constructs a new feasible solution D_3 by greedily transferring edges from D_1

[1] This term was coined by Jain, Mahdian and Saberi [10].

to D_2. Our analysis shows that when λ_1 and λ_2 are sufficiently close, the cost of the cheaper solution from D_1 and D_3 is within factor $\frac{8}{3} + \epsilon$ of optimum.

Although the GVY algorithm constructively proves an upper bound of 2 on the integrality gap of the multicut LP-relaxation, no rounding algorithm is known for this problem. In Section 4 we provide such an algorithm, which is very easy to analyze and implement, although it requires solving two linear programs. Using the optimal fractional solution d^*, our algorithm identifies a new collection of pairs to separate, and constructs a new linear program with the objective of separating these pairs. We prove that the polyhedron of feasible solutions to this program has integral extreme points. Moreover, we show that the integral solution we obtain is feasible for the original problem, and its cost is at most twice the optimum.

2 The Prize-Collecting Multicut Problem

The main result of this section is a Lagrangian multiplier preserving algorithm for the prize-collecting multicut problem, with an approximation factor of 2. We begin with a brief description of the GVY algorithm [2] and the structural properties of the solution it constructs. Next, we show how to reduce the prize-collecting multicut problem to an equivalent multicut problem by modifying the original tree and collection of pairs. Finally, we observe that our reduction forces the GVY algorithm to discard redundant penalties from the prize-collecting solution. Our analysis exploits this property to establish the main result of this section.

2.1 The GVY Algorithm

The multicut problem can be formulated as an integer program by:

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} c_e d_e && (MC) \\ \text{subject to} \quad & \sum_{e \in [s_i, t_i]} d_e \geq 1 \qquad \forall\, i = 1, \ldots, k && (2.1) \\ & d_e \in \{0, 1\} \qquad \forall\, e \in E && (2.2) \end{aligned}$$

In this formulation, the variable d_e indicates whether the edge e is picked for the multicut. Constraint (2.1) ensures that we pick at least one edge from each path $[s_i, t_i]$. The LP-relaxation of this program, (MC_f), is obtained by replacing the integrality constraint (2.2) with $d_e \geq 0$. The dual of this linear program is:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{k} f_i \\ \text{subject to} \quad & \sum_{i : e \in [s_i, t_i]} f_i \leq c_e \qquad \forall\, e \in E && (2.3) \\ & f_i \geq 0 \qquad \forall\, i = 1, \ldots, k && (2.4) \end{aligned}$$

[2] Actually, we describe its simplified version, that appears in [16, Chap. 18].

The dual progra can be viewed as the *maximum multicommodity flow* proble . Given k pairs of vertices, where each pair $\{s_i, t_i\}$ is associated with a distinct co odity, the objective is to axi ize the su of routed co odities. In this context, the variable f_i specifies the a ount of co odity we route between s_i and t_i. The pri al costs now serve as capacities, and constraint (2.3) states that the su of flows routed through each edge e does not exceed its capacity c_e.

Algorithm 1: The GVY Algorith

Root T at an arbitrary vertex and initialize $D = \emptyset$, $f = 0$.

Phase 1: Constructing *D* and *f*
while there is an unprocessed vertex **do**
  Pick the deepest such vertex, v.
  Without exceeding capacities, route maximal flow between pairs $\{s_i, t_i\}$ whose lowest common ancestor is v.
  Add all edges that became saturated to D.

Phase 2: Eliminating redundant edges
foreach $e \in D$, in reverse order of addition to D, **do**
  If $D \setminus \{e\}$ is a multicut, delete e from D.

return D, f.

The GVY algorith is shown in Algorith 1. It follows the pri al-dual sche a for approxi ation algorith s, and constructs feasible pri al and dual solutions whose costs are within factor 2 fro each other. Let D be the edge set produced by the algorith , and let f be the corresponding dual flow. Two structural properties of these solutions were proved in [8] and will be essential to our subsequent analysis.

Property 1. Only saturated edges are picked. That is, for every edge e, if $e \in D$ then $\sum_{i:e\in[s_i,t_i]} f_i = c_e$.

Property 2. If there is a positive flow between s_i and t_i, at ost two edges fro the path $[s_i, t_i]$ are picked. That is, for every $1 \leq i \leq k$, if $f_i > 0$ then $|D \cap [s_i, t_i]| \leq 2$.

2.2 The Prize-Collecting Algorithm

Reducing Prize-Collecting Multicut to Multicut. Given an instance of the prize-collecting ulticut proble , with pairs $\{s_i, t_i\}$ and associated penalties p_i, we can translate it to an instance of the ulticut proble as follows. For every $1 \leq i \leq k$, we add a new leaf vertex t'_i to T, and connect it to t_i. The cost of the additional edge (t_i, t'_i) is p_i. The new ulticut proble asks to separate the pairs $\{s_1, t'_1\}, \ldots, \{s_k, t'_k\}$ in the resulting tree, T'.

We now illustrate the equivalence between these two proble s. Let $D \subseteq E$ be any solution to the prize-collecting ulticut proble in T, and let $N \subseteq \{1, \ldots, k\}$

be the index set of pairs that are not separated by D. The cost of this solution is $\sum_{e \in D} c_e + \sum_{i \in N} p_i$. Since the edge (t_i, t_i') separates the pair $\{s_i, t_i'\}$ in T', we can easily construct a corresponding multicut in T' by picking the edge set $D \cup \{(t_i, t_i') : i \in N\}$. Clearly, the resulting solution has an identical cost, since the cost of (t_i, t_i') is p_i. Similarly, any minimal solution $D \subseteq E(T')$ to the multicut problem in T' can be used to obtain a prize-collecting solution in T with the same cost. This is done by picking the edge set $D \cap E$ and paying the penalties $\sum_{i \in N} p_i$, where $N = \{1 \leq i \leq k : (t_i, t_i') \in D\}$.

An Additional Structural Property. While properties 1 and 2 can be used to prove that by utilizing the reduction described above we obtain a 2-approximation, they are not sufficient to guarantee the LMP property. We deal with this difficulty through a closer inspection of phase 2 in the GVY algorithm, as a result of which we discover a third structural property.

For each $1 \leq i \leq k$ such that (t_i, t_i') appears in the final solution D, consider the exact point in phase 2 at which the algorithm checks whether (t_i, t_i') can be deleted or not. Since (t_i, t_i') does not separate any pair other than $\{s_i, t_i'\}$, the algorithm is allowed to discard it if at least one edge on the path $[s_i, t_i]$ appears in D at this point of time. It follows that we currently have $D \cap [s_i, t_i] = \emptyset$, or otherwise (t_i, t_i') would have been deleted. By observing that no edge is added after phase 1, we conclude the following property.

Property 3. If the edge (t_i, t_i') survived phase 2, no other edge on the path $[s_i, t_i']$ was picked. That is, for every $1 \leq i \leq k$, if $(t_i, t_i') \in D$ then $D \cap [s_i, t_i] = \emptyset$.

Analysis. Let $D_T \subseteq D$ be the set of edges that survived phase 2 and also belong to the original tree T. Property 3 implies that the index set of pairs that are not separated by D_T is exactly $N = \{1 \leq i \leq k : (t_i, t_i') \in D\}$. Therefore, D_T is a solution to the original prize-collecting problem with edge costs $\sum_{e \in D_T} c_e$ and penalties $\sum_{i \in N} p_i$. In Lemmas 1 and 2 we separately bound the edge costs and penalties in terms of the dual solution f to the multicut problem in T'. In Theorem 3 we combine these bounds to prove the main result of this section.

Lemma 1. $\sum_{e \in D_T} c_e \leq 2 \sum_{i \notin N} f_i$.

Proof. Property 3 implies that no edge in D_T belongs to a path $[s_i, t_i]$ for $i \in N$, since otherwise the edge (t_i, t_i') would not have survived phase 2. Therefore,

$$\sum_{e \in D_T} c_e = \sum_{e \in D_T} \sum_{i : e \in [s_i, t_i']} f_i \tag{2.5}$$

$$= \sum_{e \in D_T} \sum_{i \notin N : e \in [s_i, t_i']} f_i \tag{2.6}$$

$$= \sum_{i \notin N} f_i \cdot |D_T \cap [s_i, t_i']| \tag{2.7}$$

$$\leq 2 \sum_{i \notin N} f_i \quad . \tag{2.8}$$

Equation (2.5) holds since $c_e = \sum_{i:e\in[s_i,t'_i]} f_i$, by property 1. Equation (2.6) follows from the observation that $e \notin [s_i, t'_i]$ for all $i \in N$, since $e \in D_T$. Equation (2.7) results from changing the order of summation. Inequality (2.8) is due to $|D_T \cap [s_i, t'_i]| \le 2$, which is implied by $D_T \subseteq D$ and property 2. $\square$

Lemma 2. $\sum_{i\in N} p_i = \sum_{i\in N} f_i$.

Proof. Since the unique path to which (t_i, t'_i) belongs is $[s_i, t'_i]$, for every $1 \le i \le k$ we have the dual constraint $f_i \le c_{(t_i,t'_i)} = p_i$. When $i \in N$, the edge (t_i, t'_i) was picked by the algorithm, and $f_i = p_i$ by property 1. $\square$

Theorem 3. *Let* OPT *be the cost of an optimal solution to the prize-collecting multicut problem. Then,* $\sum_{e\in D_T} c_e + 2\sum_{i\in N} p_i \le 2\cdot \mathrm{OPT}$.

Proof. We observed earlier that any solution to the prize-collecting multicut problem in T has a matching multicut solution in T' with an identical cost. Therefore, it is sufficient to prove the claim when OPT is replaced with the cost of an optimal solution to the latter problem, OPT'. By combining Lemmas 1 and 2, we have

$$\sum_{e\in D_T} c_e + 2\sum_{i\in N} p_i \le 2\sum_{i\notin N} f_i + 2\sum_{i\in N} f_i = 2\sum_{i=1}^{k} f_i \le 2\cdot \mathrm{OPT}' .$$

The last inequality holds since f is a feasible dual solution, and its cost is a lower bound on the cost of any solution to the multicut problem. $\square$

3 The Partial Multicut Problem

In what follows we describe the main result of this paper, an algorithm for the partial multicut problem whose approximation ratio is $\frac{8}{3} + \epsilon$. It runs in strongly polynomial time for any fixed $\epsilon > 0$. We first present a natural integer programming formulation of partial multicut and derive its Lagrangian relaxation, the prize-collecting multicut problem. We then use the prize-collecting algorithm as a subroutine to find two preliminary sets of edges, D_1 and D_2. Although these sets are not good solutions by themselves, we show how to greedily combine them into a new edge set D_3, and prove that the cost of the cheaper solution from D_1 and D_3 is within factor $\frac{8}{3} + \epsilon$ of optimum.

3.1 Initial Assumptions

An essential part of our algorithm is a preprocessing step in which we guess certain attributes of a fixed arbitrary optimal solution, $D^* \subseteq E$, whose cost we denote by OPT. Based on these attributes, the given tree and collection of pairs are modified as we explain below. Given an accuracy parameter $\epsilon > 0$, we can make the following assumptions by conducting an exhaustive search that involves $O(n^{1/\epsilon})$ calls to the main algorithm and returning the best solution we find.

Assumption 1. All edge costs are strictly positive.

Assumption 2. We know $c_{\max}$, the axi u cost of an edge in D^*.

Assumption 3. The cost of each edge is at ost $\epsilon \cdot \mathrm{OPT}$.

Assu ption 1 is obvious, since we can pick all zero cost edges in advance and contract the . We also eli inate the subset of pairs that are separated by these edges and update the require ent para eter t. Assu ption 2 is justified, since we can test all $O(n)$ edge costs as $c_{\max}$, and for each such value contract all edges whose cost is greater than $c_{\max}$. Finally, it is possible to enforce assu ption 3 by observing that there are at ost $\lfloor \frac{1}{\epsilon} \rfloor$ edges in D^* with $c_e \geq \epsilon \cdot \mathrm{OPT}$. Therefore, we can guess the expensive edges in D^* by testing all $O(n^{1/\epsilon})$ subsets $H \subseteq E$ of cardinality at ost $\lfloor \frac{1}{\epsilon} \rfloor$. For each such subset, we include H in the solution and contract all edges whose cost is greater than in$_{e \in H} c_e$.

For the re ainder of this section, we continue to denote by k the overall nu ber of pairs, and by t the required nu ber of pairs to be separated.

3.2 The Lagrangian Relaxation

The partial ulticut proble can be for ulated as an integer progra by:

$$
\begin{aligned}
\text{ini ize} \quad & \sum_{e \in E} c_e d_e \\
\text{subject to} \quad & \sum_{e \in [s_i,t_i]} d_e + z_i \geq 1 \qquad \forall\, i = 1, \ldots, k && (3.1) \\
& \sum_{i=1}^{k} z_i \leq k - t && (3.2) \\
& d_e,\, z_i \in \{0,1\} \qquad \forall\, e \in E,\, i = 1, \ldots, k && (3.3)
\end{aligned}
$$

The variable d_e indicates whether we pick the edge e, and the variable z_i indicates whether the pair $\{s_i, t_i\}$ is not separated. Constraint (3.1) ensures that we either pick at least one edge of $[s_i, t_i]$ or do not separate the corresponding pair. Constraint (3.2) ensures that at ost $k - t$ pairs are not separated, which is equivalent to requiring that at least t pairs are separated.

We relax the co plicating constraint (3.2) and ove it to the objective function ultiplied by $\lambda \geq 0$, to obtain the following Lagrangian relaxation proble :

$$
L(\lambda) = F(\lambda) - \lambda(k - t)
$$

$$
\begin{aligned}
F(\lambda) = \text{ini ize} \quad & \sum_{e \in E} c_e d_e + \lambda \sum_{i=1}^{k} z_i \\
\text{subject to} \quad & \sum_{e \in [s_i,t_i]} d_e + z_i \geq 1 \qquad \forall\, i = 1, \ldots, k && (3.4) \\
& d_e,\, z_i \in \{0,1\} \qquad \forall\, e \in E,\, i = 1, \ldots, k && (3.5)
\end{aligned}
$$

For any fixed value of λ, $L(\lambda)$ is an integer programming formulation of the prize-collecting multicut problem $F(\lambda)$, with an additional constant term $-\lambda(k-t)$. Note that the problem $F(\lambda)$ places a uniform penalty of λ for not separating any of the given pairs. The next lemma follows from plain duality.

Lemma 4. $\max_{\lambda \geq 0} L(\lambda) \leq \mathrm{OPT}$.

3.3 Finding Useful Integral Solutions

Given $\lambda \geq 0$, we can use the prize-collecting algorithm from Section 2 to obtain an integral solution (d^λ, z^λ) for $F(\lambda)$ that satisfies $\sum_{e \in E} c_e d_e^\lambda + 2\lambda \sum_{i=1}^k z_i^\lambda \leq 2F(\lambda)$. In particular, if we can find a value of λ for which (d^λ, z^λ) separates exactly t pairs, Lemma 4 shows that this solution is a 2-approximation for the partial multicut problem, since

$$\sum_{e \in E} c_e d_e^\lambda \leq 2(F(\lambda) - \lambda(k-t)) = 2L(\lambda) \leq 2 \cdot \mathrm{OPT} \ .$$

However, we do not know how to find such a value of λ. In fact, there are instances in which the prize-collecting algorithm does not separate exactly t pairs for any value of λ.

Nevertheless, when $\lambda = 0$ the prize-collecting algorithm does not separate any pair. This follows from observing that by assumption 1 all edge costs are strictly positive, and since $F(0) = 0$ no edge is picked by the algorithm. In addition, $F(\lambda) \leq kc_{\max}$ for any λ, since we can separate all pairs by picking at most k edges (with maximum cost $c_{\max}$). It follows that the algorithm separates all pairs when $\lambda > kc_{\max}$. Therefore, using the prize-collecting algorithm we conduct a binary search over the interval $[0, kc_{\max} + 1]$, in which we find $\lambda_1 \geq \lambda_2$, with approximate solutions (d^1, z^1) and (d^2, z^2) for $F(\lambda_1)$ and $F(\lambda_2)$, respectively, such that

1. $\lambda_1 - \lambda_2 \leq \frac{\epsilon \cdot c_{\min}}{k}$, where $c_{\min}$ is the minimum cost of an edge in T (recall that $c_{\min} > 0$ by assumption 1).
2. The solution (d^1, z^1) separates $t_1 \geq t$ pairs.
3. The solution (d^2, z^2) separates $t_2 \leq t$ pairs.

Without loss of generality, none of these solutions separates exactly t pairs, or otherwise we immediately obtain a 2-approximation. We conclude the following lemma.

Lemma 5. *Let* $\alpha = \frac{t - t_2}{t_1 - t_2} \in (0,1)$. *Then,*

$$\alpha \sum_{e \in E} c_e d_e^1 + (1-\alpha) \sum_{e \in E} c_e d_e^2 \leq 2(1+\epsilon)\mathrm{OPT} \ .$$

Proof. We first observe that for $j = 1, 2$ we have

$$\sum_{e \in E} c_e d_e^j + 2\lambda_j \sum_{i=1}^k z_i^j \leq 2F(\lambda_j) = 2(L(\lambda_j) + \lambda_j(k-t)) \leq 2(\mathrm{OPT} + \lambda_j(k-t)) \ ,$$

where the first inequality follows from Theorem 3, and the second from Lemma 4. Therefore,

$$\begin{aligned}
&\alpha \sum_{e \in E} c_e d_e^1 + (1-\alpha) \sum_{e \in E} c_e d_e^2 \\
&\qquad \leq 2 \cdot \mathrm{OPT} + 2\alpha\lambda_1((k-t)-(k-t_1)) + 2(1-\alpha)\lambda_2((k-t)-(k-t_2)) \\
&\qquad \leq 2(1+\epsilon)\mathrm{OPT} \enspace ,
\end{aligned}$$

where the last inequality follows from observing that

$$\begin{aligned}
&\alpha\lambda_1((k-t)-(k-t_1)) + (1-\alpha)\lambda_2((k-t)-(k-t_2)) \\
&\qquad \leq \alpha \left(\lambda_2 + \frac{\epsilon \cdot c_{\min}}{k}\right)((k-t)-(k-t_1)) + (1-\alpha)\lambda_2((k-t)-(k-t_2)) \\
&\qquad = \lambda_2((k-t) - (\alpha(k-t_1) + (1-\alpha)(k-t_2))) + \epsilon \cdot \alpha \cdot c_{\min} \frac{t_1 - t}{k} \\
&\qquad \leq \epsilon \cdot c_{\min} \leq \epsilon \cdot \mathrm{OPT} \enspace .
\end{aligned}$$

The first inequality holds since $\lambda_1 - \lambda_2 \leq \frac{\epsilon \cdot c_{\min}}{k}$ and $k - t_1 \leq k - t$. The second inequality holds since $k - t = \alpha(k-t_1) + (1-\alpha)(k-t_2)$, $\alpha \leq 1$ and $t_1 - t \leq k$. $\square$

We remark that $O\big(\log \frac{k^2 c_{\max}}{\epsilon \cdot c_{\min}}\big)$ calls to the prize-collecting algorithm are required in order to complete the binary search described above. In the full version of this paper [13] we show that this step can be replaced with an approximate version of Megiddo's parametric search method [14], whose run time is strongly polynomial.

3.4 A Greedy Partial Cover Algorithm

We temporarily deviate from the problem-specific theme of this section, to design a greedy partial cover algorithm. Its analysis will considerably simplify the presentation of the final step in our algorithm. We state the next result in terms of set systems, since it does not rely on the special structure of the partial multicut problem. Let $U = \{e_1, \ldots, e_n\}$ be a ground set of elements, and let $\mathcal{S} = \{S_1, \ldots, S_m\}$ be a collection of subsets of U, where each subset S_i has a non-negative cost c_i. We show how to find in polynomial time a subcollection $\mathcal{S}' \subseteq \mathcal{S}$ covering at least q elements, such that $c(\mathcal{S}') \leq \frac{q}{n} c(\mathcal{S}) + \max_{S_i \in \mathcal{S}} c_i$.

Without loss of generality, we assume that $\mathcal{S}$ is a minimal cover of U. In other words, U cannot be covered by $\mathcal{S} \setminus \{S_i\}$, for all $S_i \in \mathcal{S}$. We assign each element $e \in U$ to an arbitrary subset S_i in which it appears. Let $\phi : U \to \mathcal{S}$ be the resulting assignment, and for each $S_i \in \mathcal{S}$ let $\phi^{-1}(S_i)$ be the subset of U that is assigned to S_i. Note that $\{\phi^{-1}(S_i) : S_i \in \mathcal{S}\}$ is a partition of U, and $\phi^{-1}(S_i) \neq \emptyset$ for every $S_i \in \mathcal{S}$, since $\mathcal{S}$ is minimal. For a subset $S_i \in \mathcal{S}$, let $r_i = \frac{c_i}{|\phi^{-1}(S_i)|}$ be its *ratio*. We assume that the subsets in $\mathcal{S}$ are indexed in non-decreasing order of their ratio, that is, $r_1 \leq \cdots \leq r_m$.

Theorem 6. *Let p be the minimal index for which $\sum_{i=1}^{p} |\phi^{-1}(S_i)| \geq q$, and let $\mathcal{S}' = \{S_1, \ldots, S_p\}$. Then, $c(\mathcal{S}') \leq \frac{q}{n} c(\mathcal{S}) + \text{ ax}_{S_i \in \mathcal{S}} c_i$.*

3.5 A Greedy Combination

Let D_1 be the set of edges picked by the solution (d^1, z^1). Although D_1 is a feasible solution to the partial ulticut proble , its cost can be arbitrarily large with respect to OPT. In contrast, Theore 3 and Le a 4 i ply that the cost of the edge set D_2, picked by the solution (d^2, z^2), is at ost $2 \cdot \text{OPT}$. Since D_2 is not a feasible solution, our final objective is to construct a new feasible solution D_3 by greedily transferring edges fro D_1 to D_2.

Since D_2 separates t_2 pairs, we can co plete it to a feasible solution by finding a set of edges that separates at least $t - t_2$ additional pairs. Note that $D_1 \setminus D_2$ separates at least $t_1 - t_2$ pairs that are not separated by D_2. Therefore, we can use the greedy partial cover algorith fro Subsection 3.4 to find a set of edges $S \subseteq D_1 \setminus D_2$ that separates at least $t - t_2$ pairs fro those separated by D_1 but not by D_2. It follows that $D_3 = D_2 \cup S$ is a feasible solution to the partial ulticut proble . In addition, by Theore 6 and the assu ption that the cost of each edge is at ost $\epsilon \cdot \text{OPT}$,

$$\sum_{e \in S} c_e \leq \frac{t - t_2}{t_1 - t_2} \sum_{e \in E} c_e d_e^1 + \epsilon \cdot \text{OPT} = \alpha \sum_{e \in E} c_e d_e^1 + \epsilon \cdot \text{OPT} \ . \tag{3.6}$$

We are now ready to prove that the cost of the cheaper solution fro D_1 and D_3 is within factor $\frac{8}{3} + \epsilon$ of opti u . In Le as 7 and 8 we bound the cost of D_1 and D_3 in ter s of OPT, α and β, where

$$\alpha = \frac{t - t_2}{t_1 - t_2} \in (0, 1) \ , \quad \beta = \frac{\sum_{e \in E} c_e d_e^2}{\text{OPT}} \in [0, 2] \ .$$

Lemma 7. $\sum_{e \in D_1} c_e \leq \frac{2(1+\epsilon) - (1-\alpha)\beta}{\alpha} \text{OPT}$.

Proof. Since $\alpha \neq 0$, we have

$$\begin{aligned} \sum_{e \in D_1} c_e &= \frac{1}{\alpha} \cdot \alpha \sum_{e \in E} c_e d_e^1 \\ &\leq \frac{1}{\alpha} \left(2(1+\epsilon)\text{OPT} - (1-\alpha) \sum_{e \in E} c_e d_e^2 \right) \\ &= \frac{2(1+\epsilon) - (1-\alpha)\beta}{\alpha} \text{OPT} \ . \end{aligned}$$

The first inequality follows fro Le a 5, and the last equation holds since $\sum_{e \in E} c_e d_e^2 = \beta \cdot \text{OPT}$. □

Lemma 8. $\sum_{e\in D_3} c_e \le (2+\alpha\beta+3\epsilon)\mathrm{OPT}$.

Proof. Since $D_3 = D_2 \cup S$, we have

$$\begin{aligned}\sum_{e\in D_3} c_e &= \sum_{e\in D_2} c_e + \sum_{e\in S} c_e \\ &\le \sum_{e\in E} c_e d_e^2 + \alpha \sum_{e\in E} c_e d_e^1 + \epsilon\cdot\mathrm{OPT} &(3.7)\\ &= (1-\alpha)\sum_{e\in E} c_e d_e^2 + \alpha\sum_{e\in E} c_e d_e^1 + \alpha\sum_{e\in E} c_e d_e^2 + \epsilon\cdot\mathrm{OPT} \\ &\le 2(1+\epsilon)\mathrm{OPT} + \alpha\sum_{e\in E} c_e d_e^2 + \epsilon\cdot\mathrm{OPT} &(3.8)\\ &= (2+\alpha\beta+3\epsilon)\mathrm{OPT}\ . &(3.9)\end{aligned}$$

Inequality (3.7) follows from inequality (3.6), and inequality (3.8) from Lemma 5. Equation (3.9) is obtained by substituting $\sum_{e\in E} c_e d_e^2 = \beta\cdot\mathrm{OPT}$. □

Theorem 9. $\min\{\sum_{e\in D_1} c_e, \sum_{e\in D_3} c_e\} \le (\frac{8}{3}+\epsilon)\mathrm{OPT}$.

Proof. Disregarding ϵ, Lemmas 7 and 8 show that

$$\min\left\{\sum_{e\in D_1} c_e, \sum_{e\in D_3} c_e\right\} \le \min\left\{\frac{2-(1-\alpha)\beta}{\alpha}, 2+\alpha\beta\right\}\mathrm{OPT}\ .$$

Although we cannot control $\alpha \in (0,1)$ and $\beta \in [0,2]$, the approximation guarantee of the algorithm can be bounded by considering the worst possible choice for these parameters. Using elementary calculus, it can be verified that

$$\max_{\substack{\alpha\in(0,1)\\ \beta\in[0,2]}} \min\left\{\frac{2-(1-\alpha)\beta}{\alpha}, 2+\alpha\beta\right\} = \frac{8}{3}\ ,$$

which is attained at $\alpha = \frac{1}{2}$ and $\beta = \frac{4}{3}$. □

4 An LP-Rounding Multicut Algorithm

In this section we provide an LP-rounding algorithm for the multicut problem, whose approximation factor is 2. Although our algorithm is easy to analyze and implement, it is not as efficient as the GVY algorithm, since we are required to solve two linear programs.

4.1 The Algorithm

For $1 \le i \le k$, let l_i be the lowest common ancestor of s_i and t_i, with respect to an arbitrary root of T we fix in advance. Recall that the multicut problem can be for ulated as the integer progra (MC), given in Subsection 2.1, whose LP-relaxation was denoted by (MC_f). We first solve the linear progra (MC_f) to obtain an opti al fractional solution d^*, and use it to identify a new collection of pairs to separate. Specifically, we define $v_i = s_i$ if $\sum_{e \in [s_i, l_i]} d_e^* \geq \sum_{e \in [t_i, l_i]} d_e^*$ and $v_i = t_i$ otherwise. Since $[v_i, l_i]$ is a subpath of $[s_i, t_i]$, any set of edges that separates $\{v_1, l_1\}, \ldots, \{v_k, l_k\}$ also separates the original collection of pairs. We now construct a new linear progra

$$
\begin{array}{llll}
\text{ini ize} & \sum_{e \in E} c_e d_e & & (MC'_f) \\
\text{subject to} & \sum_{e \in [v_i, l_i]} d_e \geq 1 & \forall\, i = 1, \ldots, k & (4.1) \\
& d_e \geq 0 & \forall\, e \in E & (4.2)
\end{array}
$$

and solve it to obtain an opti al solution $\hat{d}$.

4.2 Analysis

In Le a 10 we show that $\hat{d}$ is an extre e point of an integral polyhedron, and therefore it is indeed a feasible solution to (MC). In Theore 11 we prove that the cost of $\hat{d}$ is at ost twice the cost of d^*, which is a lower bound on the cost of any solution to the ulticut proble .

Lemma 10. *Any basic feasible solution to (MC'_f) is integral.*

Proof. For each path $[v_i, l_i]$, l_i is an ancestor of v_i. Therefore, we can orient the edges of T fro the root down to the leaves, and obtain a directed tree. It follows that the constraint atrix in (MC'_f) is the transpose of a chain atrix, which is a atrix whose colu ns are edge vectors of directed paths in a graph. Ca ion [2] showed that the chain atrix induced by a directed tree is totally uni odular. □

Theorem 11. *The cost of $\hat{d}$ is at most $2 \cdot \mathrm{OPT}(MC_f)$.*

Proof. To bound the cost of $\hat{d}$, we clai that $2d^*$ is a feasible solution to (MC'_f). Since d^* satisfies constraint (2.1), $\sum_{e \in [s_i, l_i]} d_e^* + \sum_{e \in [t_i, l_i]} d_e^* = \sum_{e \in [s_i, t_i]} d_e^* \geq 1$. If we assu e without loss of generality that $\sum_{e \in [s_i, l_i]} d_e^* \geq \sum_{e \in [t_i, l_i]} d_e^*$, we have $v_i = s_i$ and $\sum_{e \in [v_i, l_i]} (2d_e^*) = 2 \sum_{e \in [s_i, l_i]} d_e^* \geq 1$. Since $\hat{d}$ is an opti al solution to (MC'_f), we conclude that $\sum_{e \in E} c_e \hat{d}_e \leq \sum_{e \in E} c_e (2d_e^*) = 2 \cdot \mathrm{OPT}(MC_f)$. □

Remark. We have recently learned that so e of our results were independently obtained by Golovin, Nagarajan and Singh [9]. We thank Viswanath Nagarajan for providing us with a preli inary version of their paper.

References

1. R. Bar-Yehuda. Using homogeneous weights for approximating the partial cover problem. *Journal of Algorithms*, 39(2):137–144, 2001.
2. P. Camion. Matrices totalement unimodulaires et problèmes combinatoires. Thèse et Rapport Euratom, Université de Bruxelles, 1963.
3. S. Chawla, R. Krauthgamer, R. Kumar, Y. Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. In *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, pages 144–153, 2005.
4. E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.
5. U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.
6. R. Gandhi, S. Khuller, and A. Srinivasan. Approximation algorithms for partial covering problems. *Journal of Algorithms*, 53:55–84, 2004.
7. N. Garg, V. V. Vazirani, and M. Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM Journal on Computing*, 25(2):235–251, 1996.
8. N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.
9. D. Golovin, V. Nagarajan, and M. Singh. Approximating the k-multicut problem. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2006. To appear.
10. K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problems. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 731–740, 2002.
11. K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.
12. S. Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 767–775, 2002.
13. A. Levin and D. Segev. Partial multicuts in trees, 2005. Available at `http://www.math.tau.ac.il/~segevd/Papers/PMC-Jour.pdf`.
14. N. Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4(4):414–424, 1979.
15. P. Slavík. Improved performance of the greedy algorithm for partial cover. *Information Processing Letters*, 64(5):251–254, 1997.
16. V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

Approximation Schemes for Packing with Item Fragmentation

Hadas Shachnai[1], Tami Tamir[2], and Omer Yehezkely[1]

[1] Computer Science Department, The Technion, Haifa, Israel
{hadas, omery}@cs.technion.ac.il
[2] School of Computer Science, The Interdisciplinary Center, Herzliya, Israel
tami@idc.ac.il

Abstract. We consider two variants of the classical bin packing problem in which items may be *fragmented*. This can potentially reduce the total number of bins needed for packing the instance. However, since fragmentation incurs overhead, we attempt to avoid it as much as possible. In *bin packing with size increasing fragmentation (BP-SIF)*, fragmenting an item increases the input size (due to a header/footer of fixed size that is added to each fragment). In *bin packing with size preserving fragmentation (BP-SPF)*, there is a bound on the total number of fragmented items. These two variants of bin packing capture many practical scenarios, including message transmission in community TV networks, VLSI circuit design and preemptive scheduling on parallel machines with setup times/setup costs.

While both BP-SPF and BP-SIF do not belong to the class of problems that admit a *polynomial time approximation scheme (PTAS)*, we show in this paper that both problems admit a *dual* PTAS and an *asymptotic* PTAS. We also develop for each of the problems a dual asymptotic *fully polynomial time approximation scheme (AFPTAS)*. The AFPTASs are based on a non-trivial application of a fast combinatorial FPTAS for packing linear programs with negative entries, proposed recently by Garg and Khandekar [5].

1 Introduction

In the classical *bin packing (BP)* problem, n items $(a_1, \ldots, a_n)$ of sizes $s(a_1), \ldots, s(a_n) \in (0, 1]$ need to be packed in a minimal number of unit-sized bins. This problem is well known to be NP-hard. We consider a variant of BP known as *bin packing with item fragmentation (BPF)*, in which items can be fragmented (into two or more pieces). Therefore, it may be possible to pack the items using fewer bins than in classical BP. However, since fragmentation incurs overhead, we attempt to avoid it as much as possible. We study two variants of BPF. In both variants, the goal is to pack all items in a minimum number of bins.

Size increasing fragmentation (BP-SIF): A header (or a footer) of a fixed size, $\Delta > 0$, is attached to each (whole or fragmented) item. That is, the volume

T. Erlebach and G. Persiano (Eds.): WAOA 2005, LNCS 3879, pp. 334–347, 2006.

required for packing an item of size $s(a_i)$ is $s(a_i)+\varDelta$. Upon fragmenting an item, each fragment gets a header; that is, if a_i is replaced by two items such that $s(a_i) = s(a_{i_1}) + s(a_{i_2})$, then packing a_{i_j} requires volume $s(a_{i_j}) + \varDelta$. Assume, for example, that $\varDelta = 0.1$, and an instance consists of 3 items of sizes $\{0.4, 0.5, 0.7\}$. Without fragmentation, each item must be packed in a separate bin (occupying the volumes $0.5, 0.6$ and 0.8, respectively), while if, e.g., the item of size 0.4 is fragmented to 0.1 and 0.3, the resulting instance can be packed into two bins, the contents of which are $(0.1 + 0.1, 0.7 + 0.1)$, and $(0.3 + 0.1, 0.5 + 0.1)$.

Size preserving fragmentation (BP-SPF): An item a_i can split into two fragments: a_{i_1}, a_{i_2}, such that $s(a_i) = s(a_{i_1}) + s(a_{i_2})$. The resulting fragments can also split in the same way. Each split has a unit cost and the total cost cannot exceed a given *budget* $C > 0$. Note that in the special case where $C = 0$ we get an instance of classic bin packing. (Most of our results can be applied to another variant of BP-SPF in which the goal is to minimize the packing cost, and the number of available bins, $b \geq \lceil \sum_i s(a_i) \rceil$ is given as part of the input.)

For any $\varDelta > 0$ (in BP-SIF) and $C < \lceil \sum_i s(a_i) \rceil - 1$ (in BP-SPF), the BPF problem is NP-hard (see Sections 2 and 1.1 for hardness and hardness of approximation results). Therefore, we present approximation algorithms. The following applications motivate our study.

Community Antenna Television (CATV) Networks: In many communication protocols, messages of arbitrary sizes are placed in fixed sized frames before they are transmitted. Consider, for example, the Data-Over-Cable Service Interface Specification (DOCSIS), defined by the Multimedia Cable Network System standard committee [14]. When using CATV network for communication, the upstream (data transmission from the subscribers' cable modem to the headend) is divided into numbered mini-slots. The DOCSIS specification allows two types of messages: *fixed location* and *free location*. Fixed location messages are placed in fixed mini-slots, while free location messages can be placed arbitrarily in the remaining mini-slots (each message may need one or more mini-slots). The specification also allows to fragment the free location messages. Since each of the original messages, as well as each of its pieces allotted to a mini-slot, has a header (or footer) attached to it, the problem of scheduling the free location messages yields an instance of the BP-SIF problem (see [12] for more details).

VLSI Circuit Design: In high level synthesis of digital systems, when a logic unit is initialized, values of external variables are copied into the unit's internal variables. Each external variable may be copied into multiple internal variables. The logical unit has a fixed number, U, of memory ports that it can access in each work cycle. In order to copy an external variable into n variables, all $n+1$ variables need to be accessed. For example, if $U = 5$ and two external variables x, y need to be copied into 6 internal variables each, a possible initialization process is to copy x in the first cycle into 4 variables, then copy x into the 2 remaining variable and y into two variables, and in a third cycle, copy y into the 4 remaining variables. Note that all 5 ports are used in each of these cycles. The goal is to complete the initialization process within the fewest possible work cycles. This yields an instance of BP-SIF, where U is the bin size, $\Delta = 1$, and the j-th item size is the number of internal variables that need to be assigned the value of the j-th external variable. (For more details see in [10].)

Preemptive Scheduling with Setup Times/Costs: Consider the problem of preemptively scheduling a set of jobs on a minimal number of parallel machines; the i-th job has the length ℓ_i, and all jobs should be completed by time D which is the deadline for all jobs. When starting/resuming a job incurs *setup time*, each preemption (split) causes additional setup time to a new job-segment, therefore the resulting problem is BP-SIF. When preempting/resuming a job incurs a *setup cost*, the resulting problem is BP-SPF, where each preemption (split) causes an additional cost, and the goal is to find a schedule whose total cost (given by the total number of preemptions) does not exceed a given bound C. The following is another natural variant of BP-SPF: the number, m, of machines is given and known to be at least $(\sum_i \ell_i)/D$. The goal is to schedule the jobs using the minimum possible number of splits. It is assumed that no job is longer than the deadline, thus, it is always possible to process segments of one job in non-overlapping time intervals (this is a property of primitive packings, as explained in Section 2).

Flexible Packaging: In some packaging problems, the cost of the packages is substantive. This includes (i) storage management, where files need to be stored in a minimal number of disks, and each file or file-segment has a header of fixed size; (ii) transportation problems, where material is to be delivered using minimal number of vehicles. For example, trucks need to ship construction materials, such as sand, and the sand is given in several sizes of bags. The number of trucks used for the shipment may be reduced, by splitting the content of some bags.

1.1 Related Work

It is well known (see, e.g., [13]) that BP does not belong to the class of NP-hard problems that admit a PTAS. In fact, BP cannot be approximated within factor $\frac{3}{2} - \varepsilon$, for any $\varepsilon > 0$, unless P=NP [4]. However, there exists an *asymptotic* PTAS *(APTAS)* which uses, for any instance I, $(1+\varepsilon)\,OPT(I) + k$ bins for some fixed k. Vega and Lueker [2] presented an APTAS with $k = 1$, and Karmarkar and Karp [8] presented an *asymptotic fully* PTAS *(AFPTAS)* with $k = 1/\varepsilon^2$. Alternatively, a *dual* PTAS, which uses $OPT(I)$ bins of size $(1+\varepsilon)$ was given by Hochbaum and Shmoys [7]. Such a dual PTAS can also be derived from the work of Epstein and Sgall [3] on multiprocessor scheduling, since BP is dual to the *minimum makespan* problem. (Comprehensive surveys on the bin packing problem appear, e.g., in [1,18].)

Mandal et al. introduced in [10] the BP-SIF problem and showed that it is NP-hard. Menakerman and Rom [12] and Naaman and Rom [15] were the first to develop algorithms for bin packing with item fragmentation, however, the problems studies in [12] and [15] are different from our problems. For a version of BP-SPF in which the number of bins is given, and the objective is to minimize the total cost incurred by fragmentation, the paper [12] studies the performance of si ple algorith s such as First-Fit, Next-Fit, and First-Fit-Decreasing, and shows that each of these algorith s ight end-up with $OPT(I) - 1$ nonessential splits.

There has been so e related work in the area of pree ptive scheduling on parallel achines. The paper [16] presents a tight bound on the nu ber of pree ptions required for a schedule of ini u akespan, and a PTAS for inizing the akespan of a schedule with job-wise or total bound on the nu ber of pree ptions. However, the techniques used in this paper rely strongly on the assu ption that the job-wise/total bounds on the nu ber of pree ptions are so e fixed constants, while in solving the BPF variants the nu ber of splits ay depend on the input size.

1.2 Our Results

In this paper we develop approxi ation sche es for the two variants of bin packing with ite frag entation. We first show (in Section 2) that, for each of the proble s, achieving an absolute error bounded by a constant is NP-hard. We then analyze the perfor ance of a class of natural algorith s for our proble s. In Section 3 we develop a dual PTAS and APTAS for SP-SPF. Our dual PTAS packs all the ite s in $OPT(I)$ bins of size $(1+\varepsilon)$, and the APTAS uses at ost $(1+\varepsilon)OPT(I)+1$ bins. In Section 5 we show that these sche es can be odified to apply for BP-SIF. We also show that each of the proble s ad its a dual AFPTAS.

Technical Contributions: The paper contains two technical contributions. Our APTAS for BP-SPF is based on a novel *oblivious* version of the shifting technique (see, e.g., [18]). Given an instance of n ite s whose sizes are *unknown*, we define a set of ite s whose (shifted) sizes are given as variables; the values of these variables are then revealed by solving a linear progra ing relaxation of the packing proble . We expect that this non-standard use of the shifting technique will find ore applications. Our second contribution is a non-trivial application of the fast approxi ation sche e of [5] for packing linear progra s, which enables to obtain dual AFPTASs for the two variants of BPF.

Due to space constraints, so e of the proofs are o itted. These proofs appear in the full version of this paper [17].

2 Preliminaries

In this section we present so e basic le as and properties of packing with ite frag entation. We also present a class of natural algorith s and analyze their perfor ance. We first show that, for both variants of BPF, there exists an opti al packing of certain structure. This allows us to reduce the search for a *good* packing to this subset of packings.

Define the *bin packing graph* of a given packing as an undirected graph, where each bin i is represented by a vertex v_i; there is an edge (v_i, v_j) if bin i and bin j share frag ents of the sa e ite . Note that a frag ent-free packing induces a graph with no edges. A *primitive packing* is a feasible packing in which (i) each bin has at most two fragments of items, and (ii) each item can be fragmented only once. Note that the respective bin packing graph is a collection of paths.

Lemma 1. *Any instance of BP-SPF has an optimal primitive packing.*

Proof. We show that any feasible BP-SPF packing, in particular an optimal one, can be transformed into a primitive packing with the same number of splits or fewer. Given a packing with f splits, consider its BP graph. Let $V(C_i), I(C_i)$ denote, respectively, the set of vertices and the set of packed items in a connected component C_i. The connected component C_i has at least $|V(C_i)|-1$ edges. Note that for $i \neq j$, $I(C_i) \cap I(C_j) = \emptyset$. Thus, for each connected component C_i, the items in $I(C_i)$ can be repacked into the respective bins of $V(C_i)$, by filling the bins one at a time, using an arbitrary order of $I(C_i)$, and splitting (if necessary) the last item packed into the active bin. This results in a primitive packing with at most f splits, since every connected component C_i is replaced by a subgraph with at most $|V(C_i)| - 1$ edges.

The proof of the next lemma is similar to the proof of Lemma 1.

Lemma 2. *For any instance of BP-SIF, there exists an optimal primitive packing.*

2.1 Hardness of BPF

Clearly, by a simple reduction from Partition, it is NP-hard to decide whether an instance of BPF can be packed in 2 bins with no splits. This implies that BP-SIF is NP-hard for any $\Delta > 0$. For BP-SPF, by McNaughton's rule ([11]), if the bound on the number of splits is $C \geq \lceil \sum_i s(a_i) \rceil - 1$, a packing that uses $\lceil \sum_i s(a_i) \rceil$ bins exists and can be found in linear time. We prove that it is NP-hard to avoid a *single* split, for any number of bins, even if the existence of a packing that uses no splits is known a-priori. This implies that BP-SPF is NP-hard even if we are allowed to exceed the budget by $\lceil \sum_i s(a_i) \rceil - 2$, i.e, to use $C + \lceil \sum_i s(a_i) \rceil - 2$ splits.

Theorem 1. *Let c be the minimal number of splits required for packing an instance I into b bins. Then, for any values of b and c, it is NP-hard to find a packing with less than $b-1$ splits.*

Proof. We first prove hardness for bins with different sizes and then extend the proof to identical bins. The reduction is from the *Partition* problem. Given $a_1, ..., a_n$, an instance for Partition with total size of items equals $2S$, construct an instance BP-SF with $2k$ bin and k sets of items, $I_0, \ldots, I_{k-1}$. Let $M > (2S+1)$ be an integer. The set I_0 consists of items of sizes $a_1, a_2, ..., a_n$; I_1 consists of items of sizes $a_1M, a_2M, ..., a_nM$, and in general, I_j consists of items of sizes $a_1M^j, a_2M^j, ..., a_nM^j$. The bin sizes are $S, SM, SM^2, ..., SM^k$ – two bins of each size. If there exists a partition of the items into two sets of size S, then a packing with no splits exists, by packing I_j into the two bins of size SM^j. Consider a packing of the items into the bins.

Claim. Any full bin with no splits induces a partition.

Proof. Assume that for some z, a bin of size SM^z is full. No item from a set $I_j, j > z$ is packed in the bin, since each of the items from $I_j, j > z$ is larger than SM^z (because $a_i M^{z+1} > SM^z$). Also, no item from a set $I_j, j < z$ is packed. This is true since the total size of items from earlier sets is $2S(1+M+M^2+....M^{z-1}) = 2S(M^k - 1)/(M - 1)$ which is less than M^z for all $M > 2S + 1$, therefore, no combination of items from the sets $I_j, j < z$, can be used.

It follows that the full SM^z bin contains only items from one set, and by scaling by M^z, its content induces a partition of the original instance.

In order to extend the proof to instances with identical bins, we add to the set I_j two *filler items* of size $S(M^k - M^j)$, and all the $2k$ bins have size SM^k. Note that the two smallest filler items have total size $2S(M^k - M^{k-1})$, which is larger than SM^k for any $M > 2$. Therefore, there is at most one filler item in any bin. Also, the total size of non-filler items is too small to fill a bin, therefore a full bin must contain exactly one filler item. Assume that some bin is full with items and no fragments. Let $S(M^k - M^z)$ be the size of the filler item in the bin, the rest of the bin is filled by items of total size SM^z and the proof for the different size bins can be applied here. ∎

Corollary 1. *If $P \neq NP$ then there is no approximation scheme for BP-SIF with a constant additive error.*

2.2 Discrete Instances

An instance of BPF is *discrete* if for some fixed positive integer U all item sizes are taken from the sequence $\{\delta, 2\delta, ..., U\delta\}$, where $\delta = 1/U$. Note that since U is integral, there exists an optimal (primitive) solution is which each fragmented-item splits into two fragments having sizes in $\{\delta, 2\delta, ..., (U - 1)\delta\}$, thus, no new sizes are introduced by the fragmentation process. For an instance of BP-SIF to be discrete, it is also required that Δ is of the form $i\delta$, for some integer i.

Given a discrete instance I, for BP-SIF or BP-SPF, define a *bin configuration* to be a vector of length U, in which the j-th entry is the number of items of size $j\delta$ packed in the bin; the configuration is valid if the total size of items (together with their headers, in BP-SIF) is at most 1. The *configuration matrix*, A_I, is the matrix which gives all possible bin configurations. The *fragmentation matrix*, B_I, is the matrix which gives all possible fragmentations of items in I. Each row in B_I corresponds to a single possible split, and represents the change in the total number of items in the instance if this split is performed. Each row is a *fragmentation vector*, in which all entries are 0 except for a single (-1) entry, and a single $(+2)$ or two $(+1)$ entries, such that the sum of values of the positive entries is equal to the value of the negative entry. For example, if $U = 6$ then the row $[0, 1, 1, 0, -1, 0]$ corresponds to a single split of an item of size $5/6$ into two items of sizes $2/6$ and $3/6$, and the row $[0, 2, 0, -1, 0, 0]$ corresponds to a single split of an item of size $4/6$ into two items of size $2/6$.

2.3 Bounds for Simple Offline and Online Algorithms

Consider the following class of algorithms (defined in [12]). An Algorithm is said to *avoid unnecessary fragmentation* if it follows the two rules.

1. No unnecessary fragmentation: An item is fragmented only if packed in a bin that does not have enough space for it. Upon fragmentation of an item, the first fragment must fill one of the bins. The second fragment is packed according to the packing rule of the algorithm.
2. No unnecessary bins: A new bin is opened only if the currently packed item cannot fit into any open bin.

In particular, an algorithm that fills the bins to full capacity one after the other (in BP-SIF, the algorithm moves to the next bin when the currently open bin is filled to capacity at least $(1 - \Delta)$) avoids unnecessary fragmentation. Note that this greedy algorithm does not assume any order of the items and is therefore an online algorithm.

The following theorems give upper bounds on the performance of any algorithm that avoids unnecessary fragmentation.

Theorem 2. *Any algorithm for BP-SIF that avoids unnecessary fragmentation uses at most* $N_{opt}(1 + \frac{\Delta}{\Delta+1}) + 1$ *bins.*

Theorem 3. *Any algorithm for BP-SPF that avoids unnecessary fragmentation uses* N_{opt} *bins for any budget* $C \geq \lceil \sum_i s(a_i) \rceil - 1$*, and at most* $N_{opt} + Z$ *for budget* $C = \lceil \sum_i s(a_i) \rceil - Z$ *and* $Z > 1$*.*

3 Bin Packing with Size-Preserving Fragmentation

Recall that in BP-SPF we are given a list of n items $I = (a_1, a_2, ..., a_n)$, each with the size $s(a_i) \in (0, 1]$. The number of splits is bounded by C. The goal is to pack all items using minimal number of bins and at most C splits. In this section we develop a dual PTAS and an APTAS for BP-SPF.

3.1 A Dual PTAS

Our dual PTAS for BP-SPF uses an optimal number of bins of size at most $(1 + \varepsilon)$, for some $\varepsilon > 0$. The scheme proceeds in the following steps. Given an input I and some $\varepsilon > 0$, (i) Partition the items into two groups according to their sizes: the *large* items have size at least ε; all other items are *small.* (ii) Round up the size, $s^+(a_i)$, of each large item to the nearest integral multiple of ε^2. (iii) Guess $OPT(I)$, the number of bins used by an optimal packing of I. (iv) Pack optimally the large items with fragmentation, using at most C splits, into $OPT(I)$ bins of size $(1+\varepsilon+\varepsilon^2)$. (v) Pack the small items in an arbitrary order, one at a time. Each item is packed into the bin having maximum free space.

The next lemmas show that our scheme uses at most $OPT(I)$ bins of size $(1 + \varepsilon + \varepsilon^2)$ for packing the (rounded) large items, and that no new bins are opened when the small items are added greedily in step (v).

Lemma 3. *It is possible to pack the rounded large items into $OPT(I)$ bins of size $(1 + \varepsilon + \varepsilon^2)$.*

Lemma 4. *The small items can be added to the $OPT(I)$ bins of size $(1+\varepsilon+\varepsilon^2)$ without causing additional overflow.*

We summarize in the next Theorem.

Theorem 4. *BP-SPF admits a dual PTAS.*

Proof. Using Lemmas 3 and 4, and taking $\varepsilon' = \varepsilon/2$, we get that the scheme packs all the items in at most $OPT(I)$ bins, each of size at most $1 + \varepsilon$. We turn to analyze the time complexity of the scheme. Steps (i) and (ii) are linear and are done once. For Step (iii), note that $\lceil s(I) \rceil \leq OPT(I) \leq n$, therefore $OPT(I)$ can be guessed in $O(\log n)$ iterations. Each such iteration involves packing the rounded large items and adding the small items. When packing the large items in step (iv), since there are at most $1/\varepsilon$ large items in a bin, and the number of distinct sizes is at most $M = 1/\varepsilon^2$, the number of possible packings to consider is $O(n^{M^{1/\varepsilon}}) = O(n^{(1/\varepsilon^2)^{1/\varepsilon}})$. Finally, the small items are added in time $O(n)$.

3.2 An APTAS for BP-SPF

We describe below an asymptotic PTAS for BP-SPF. Given an $\varepsilon > 0$, our scheme packs any instance I into at most $OPT(I)(1 + \varepsilon) + 1$ bins. Our scheme applies shifting to the item sizes, and *oblivious* shifting to the (unknown) fragment sizes in some optimal solution. The latter is crucial for finding efficiently a primitive packing that is close to the optimal.

The following is an overview of the scheme. (i) Guess $OPT(I)$ and $c \leq C$, the number of fragmented items. (ii) Partition the instance into *large* and *small items*; any item with size at least ε is large. (iii) Transform the instance to an instance where the number of distinct item sizes is fixed. (iv) Guess the configuration of the i-th bin in some optimal packing (defined below). (v) Let $R \geq 1$ be the number of distinct fragment sizes in a shifted optimal solution, where $R \leq 1/\varepsilon^2$ is some constant. Guess the number of fragmented items of the j-th group, having fragments of types $1 \leq r_1, r_2 \leq R$. (vi) Solve a linear program which yields the fragment sizes for each fragmented item. (vii) Pack the non-fragmented items and the fragments output by the LP using the bin configurations. $(viii)$ Pack the small items in an arbitrary order, one at a time, into the bin having maximum free space.

Transformation of the Input and Guessing Bin Configurations: First, guess the values $OPT(I)$ and c, the number of fragmented items. Since there exists an optimal primitive packing, $1 \leq c \leq \min(C, OPT(I) - 1)$. Next, transform the instance I to an instance I' in which there are at most $1/\varepsilon^2$ item sizes. This can be done by using the shifting technique (see, e.g., in [18]). Generally, the items are sorted in non-decreasing order by sizes, then, the ordered list is partitioned into at most $1/\varepsilon^2$ subsets, each including $H = \lceil n\varepsilon^2 \rceil$ items. The size of each ite is rounded down to the size of the largest ite in its subset. This yields an instance in which the nu ber of distinct ite sizes is $m = n/H \leq 1/\varepsilon^2$.

Define an *extended bin configuration* to be a vector which gives the nu ber of ite s of each size group packed in the bins, as well as at ost two frag ents which ay be added (since we find a pri itive packing). Each extended bin configuration consists of three parts: (i) a vector $(h_1, \ldots, h_m)$ where h_j, $1 \leq j \leq m$, is the nu ber of non-frag ented ite s of size group j packed in the bin, (ii) A binary indicator vector of length m, with at ost two '1' entries – in the indices of at ost two size groups $1 \leq j_1, j_2 \leq m$ that contribute a frag ent to the bin, and (iii) a binary indicator vector of length R (to be defined), with at ost two '1' entries – in the indices corresponding to at ost two types of frag ents $1 \leq r_1, r_2 \leq R$, which are packed in the bin.

Now, since both the nu ber of size groups and the nu ber of frag ent types are fixed constants, the nu ber of bins of each configuration can be guessed in polyno ial ti e .

Guessing the Fragment Types: The heart of the sche e is in finding how each of the c guessed ite s is frag ented. This is done by using the following oblivious shifting procedure. Suppose that in so e opti al packing the (unknown) frag ent sizes are $y_1 \leq y_2 \cdots \leq y_{2c}$, then apply shifting to this sorted list, by partitioning the entries into subsets of sizes at ost $Q = \lceil 2c\varepsilon^2 \rceil$, and round up the size of each frag ent to the largest entry in its subset. Now, there are $R \leq 1/\varepsilon^2$ distinct frag ent sizes. These sizes are deter ined later, by solving a linear progra for packing the instance with frag entation.

Next, find the type of frag ents generated for each of the c guessed ite s. Note that the nu ber of pairs of frag ent sizes is R^2. Thus, guess for each size group j the nu ber of ite s in this group having a certain pair of frag ent types. This can be done in polyno ial ti e.

Solving the LP and Packing the Items: Having guessed the frag ent types for each frag ented ite , use a linear progra to obtain the frag ent sizes that yield a feasible packing. Let x_r denote the size of the r-th frag ent in a shifted opti al sorted list. For any ite in group j, that is frag ented to the pair of type (r, s), we verify that the su of frag ent sizes is at least s_j, the size of an ite in group j. Denote by ℓ^r_{ij} the indicator for packing a frag ent of type r contributed by size group j in bin i, $1 \leq r \leq R$, $1 \leq j \leq m$, $1 \leq i \leq OPT(I)$. The goal is to find R frag ent sizes, $x_1 \leq \cdots \leq x_R$, which enable to pack all the ite s. That is, once a correct guess of the frag ent types is ade, the total volu e packed by the LP is the volu e of the c frag ented ite s. Let $N = OPT(I)$. Denote the set of frag ent pairs assigned to the ite s of size group j F_j, $1 \leq j \leq m$. Finally, Γ_i is the space left in bin i after packing the non-frag ented ite s. The following linear progra is solved.

$$(LP) \quad \text{axi ize} \quad \sum_{i=1}^{N}\sum_{j=1}^{m}\sum_{r=1}^{R} x_r \ell^r_{ij}$$

$$\textit{subject to}: \quad x_{r_1} + x_{r_2} \geq s_j \quad \forall j, \ (r_1, r_2) \in F_j$$

$$\sum_{j=1}^{m}\sum_{r=1}^{R-1} x_r \ell_{ij}^r \leq \Gamma_i \text{ for } i = 1, \ldots, N \tag{1}$$
$$x_r \geq 0 \text{ for } r = 1, \ldots, R$$

Inequality (1) ensures that the fragment types $1, \ldots, R-1$ can be packed in the $OPT(I)$ bins, given the correct guess.

Now, given the solution for the LP, the fragment sizes for each of the c fragmented items are known. Pack the non-fragmented items as given in the bin configuration and add the fragments of sizes $1, \ldots, R-1$ in these $OPT(I)$ bins. Next, add $H + Q \leq 2/\varepsilon^2$ new bins. In each of the H new bins pack separately a non-fragmented item in the largest size group generated during shifting. In the Q new bins, pack the fragments of type R (i.e., the largest fragments) greedily. Finally, add greedily the small items. It can be shown[1] that this requires adding at most one new bin. Therefore, by taking $\varepsilon' = \varepsilon/2$, overall, at most $OPT(I)(1+\varepsilon)+1$ bins are used. Thus,

Theorem 5. *There is an asymptotic PTAS for BP-SPF.*

4 A Dual AFPTAS for BP-SPF

We now describe an asymptotic dual FPTAS for BP-SPF that packs the items of a given BP-SPF instance into $(1+\varepsilon)OPT(I)+k$ bins, of size $(1+\varepsilon)$, where $k \leq 1/\varepsilon^2$ is some constant. Our scheme applies some of the steps used in the dual PTAS given in Section 3.1. However, since 'guessing' the fragmented items results in number of iterations that is exponential in $1/\varepsilon$, we use instead a linear programming formulation of the packing problem, whose solution yields a feasible packing of the instance. In order to obtain a scheme whose running time is polynomial in both n and $1/\varepsilon$, we solve the linear program *approximately*, by repeatedly applying the fast combinatorial approximation scheme of [5].

Our scheme proceeds in the following steps. (i) Guess $c \leq C$, the number of fragmented items. (ii) Partition the items into *large* and *small*: an item i whose size is at least ε is *large*. (iii) Round up the size of each large item to the nearest integral multiple of ε^2. (iv) Define for the large items the *configuration matrix*, A, and the *fragmentation matrix*, B, each having $1/\varepsilon^2$ columns. (v) Solve within factor $(1+\varepsilon)$ to the optimal a linear program for packing the large items in minimum number of bins. (vi) Round the solution of the linear program and pack accordingly the large items in at most $(1+\varepsilon)OPT(I)+m$ bins, where $m \leq 1/\varepsilon^2$. (vii) Add greedily in arbitrary order the small items, to the bin of maximum free space.

In the following we describe how our scheme finds a *good* packing of the large items.

[1] The argument is similar to the argument when packing the small items in classic BP (see, e.g., in [18]).

Constructing the Configuration and Fragmentation Matrices: Recall, that for the rounded large items, a *bin configuration* is a vector of size $m = 1/\varepsilon^2$, in which the j-th entry gives h_j, the number of items of size group j packed in the bin. The *configuration matrix*, A, consists of the set of all possible bin configurations, where each configuration is a row in A; therefore, the number of rows in A is $q \leq (1/\varepsilon)^{1/\varepsilon^2}$. The fragmentation matrix, B, consists of all possible fragmentation vectors for the given set of large items. The k-th vector is the k-th row in B.

Solving the Linear Program: We now formulate the problem of packing the rounded large items in minimum number of bins as a linear program. Let n_j denote the number of items in the j-th size group. Denote by x_i the number of bins having the i-th configuration, $1 \leq i \leq q$. Let z_k denote the number of items that are split according to the k-th fragmentation vector (i.e., the k-th row in the matrix B).

$$
\begin{aligned}
(P) \quad & \text{minimize} & & \sum_{i=1}^{q} x_i \\
& \textit{subject to}: & & \sum_{i=1}^{q} A_{ij} x_i - \sum_{k=1}^{p} z_k B_{kj} \geq n_j \qquad \text{for } j = 1, \ldots, m \\
& & & \sum_{k=1}^{p} z_k \leq c \\
& & & x_i \geq 0 \text{ for } i = 1, \ldots, q
\end{aligned}
$$

In the dual of the above linear program there is a variable y_j for each constraint.

$$
\begin{aligned}
(D) \quad & \text{maximize} & & \sum_{j=1}^{m} n_j y_j - c y_{m+1} \\
& \textit{subject to}: & & \sum_{j=1}^{m} A_{ij} y_j \leq 1 \text{ for } i = 1, \ldots, q \qquad (2) \\
& & & \sum_{j=1}^{m} B_{kj} y_j + y_{m+1} \geq 0 \text{ for } k = 1, \ldots, p \qquad (3) \\
& & & y_j \geq 0 \text{ for } j = 1, \ldots, m+1
\end{aligned}
$$

The above dual program is a fractional packing linear program, in which some coefficients may be negative. For such a program, we can apply the fast scheme of Garg and Khandekar [5] to obtain a $(1+\varepsilon)$-approximate solution. Combining this scheme with a technique of Karmarkar and Karp [8] for constraint elimination, we can get a basic $(1+\varepsilon)$-approximate solution for the primal program (P).

Packing the Items: For packing the large items, round down the x_i values in the fractional solution for (P). As a result, some of the items cannot be packed. Add new bins, in which these remaining items are packed greedily with no fragmentation (i.e., in the worst case, an item in each bin). Finally, the small items are added in an arbitrary order with no fragmentation; each of the small items is added to the bin with the currently maximal available space.

4.1 Analysis of the Scheme

We show that the above scheme packs all the items in at most $(1+\varepsilon)OPT(I)+k$ bins of size $(1+\varepsilon)$, where $k \leq 1/\varepsilon^2$ is some constant, and that its running time is polynomial in n and $1/\varepsilon^2$.

Note that the fast scheme proposed by [5] for approximately solving a packing linear program with negative entries can be applied for obtaining a $(1+\varepsilon)$-approximate solution for (D). Hence,

Lemma 5. *For any $\varepsilon > 0$, there exists an FPTAS which solves (D) within factor $(1+\varepsilon)$ from the optimal in $O(m\varepsilon^{-2}\log m)$ calls to the oracle.*

The next lemma guarantees that by rounding the fractional solution for (P), the overall number of bins is increased by some constant.

Lemma 6. *Our scheme finds in $O(\varepsilon^{-d} n \log n)$ steps a basic approximate solution for (P) in which at most m variables are strictly positive, where $d > 0$ is some constant.*

Proof sketch: We find a $(1+\varepsilon)$-approximate basic solution for (P), by combining the fast scheme of [5] with a technique of [8]. Initially, we apply the *modified GLS algorithm* proposed in [8] to obtain a linear program (D') in which the overall number of constraints is at most $Q \leq M + 2m$, where $M = O(m^2 \ln(mn))$. The program (D') has the nice property of having an optimal solution that is close to an optimal solution for (D). Now, since the primal program has a basic solution in which at most m variables are strictly positive, we proceed to eliminate gradually constraints in the dual program, until we get a subset of m constraints. This is done by applying an elimination procedure: in each stage we partition the remaining constraints into subsets of sizes $m+1$, and try to omit each subset, and then we test whether the solution of the resulting program is close enough to the (approximate) solution obtained initially for (D). In each elimination/testing step we now apply the fast combinatorial FPTAS of [5]. We use its oracle for testing the validity of the natural packing constrains; the remaining (small number of) constraints can be tested separately. Finally, we have a dual program of m constraints. Using again the fast scheme, we now solve the corresponding primal program to obtain a primal basic solution that is $(1+\varepsilon)$-approximation to the optimal. ■

Lemma 7. *The scheme packs all the items in at most $(1+\varepsilon)OPT(I)+m$ bins of size at most $1+\varepsilon+\varepsilon^2$.*

Proof sketch: Since we obtain a basic solution for (P), rounding down the fractional x_i values may require adding at most m new bins, in which we pack the remaining items. Adding the small items may increase the number of bins by $\varepsilon OPT(I)$. For showing the resulting extension in bin sizes, we can use arguments similar to the arguments in the proof of Theorem 4. ∎

Theorem 6. *There is a dual AFPTAS for BP-SPF which packs the items in at most* $(1+\varepsilon)OPT(I)+m$ *bins of sizes* $1+\varepsilon+\varepsilon^2$.

5 Bin Packing with Size-Increasing Fragmentation

Recall that in BP-SIF we are given a list of n items, $I=(a_1,a_2,...,a_n)$, each has the size $s(a_i)\in(0,1]$. The number of splits is unbounded, but since there is a header of size Δ attached to each item or fragment, each fragmentation increases the input size by Δ, the size of an extra header. The goal is to pack all items using minimal number of bins. The approximation schemes developed for BP-SPF can be slightly modified to yield approximation schemes for BP-SIF. We give the details in the full version of the paper. Note that *bin configuration, configuration matrix, fragmentation matrix* are all well-defined for BP-SIF. Therefore, when moving from BP-SPF to BP-SIF, the same techniques can be used. Thus, we obtain for BP-SIF a dual PTAS, an asymptotic PTAS, and a dual AFPTAS.

By Theorem 2, any algorithm that avoids unnecessary fragmentation uses at most $N_{opt}/(1-\Delta)+1$ bins. Let $\varepsilon>0$ be the parameter of the scheme. For any $\Delta\le\varepsilon/(1+\varepsilon)$ it holds that $1/(1-\Delta)\le(1+\varepsilon)$. Therefore,

Corollary 2. *If* $\Delta\le\varepsilon/(1+\varepsilon)$ *then there is a linear time AFPTAS for BP-SIF.*

We note that when $\Delta>\varepsilon/(1+\varepsilon)$ the number of items or fragments packed in each bin does not exceed $1/\Delta<(1+\varepsilon)/\varepsilon$, which is a constant. This fact seems to simplify the problem; however, since small items are treated easily anyway, we are left with the challenge of packing the large items. The schemes for BP-SIF can be slightly simplified by taking $\varepsilon'=\varepsilon/(1+\varepsilon)$, which implies that there are no small items, and the steps involving the small items can be skipped.

Acknowledgment

We thank Yuval Rabani for helpful comments and suggestions.

References

1. E.G. Coffman Jr., M.R. Garey, and D.S. Johnson. Approximation algorithms for bin packing: a survey. In D.S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems,* 46-93. PWS Publishing, Boston, MA, 1997.
2. W.F. de la Vega and G.S. Lueker. Bin packing can be solved within $1+\varepsilon$ in linear time. *Combinatorica,* 1:349-355, 1981.

3. L. Epstein and J. Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. In *Proc. of the 7th European Symposium on Algorithms*, pp. 151–162, 1999.
4. M.R. Garey and D.S. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, W. H. Freeman and Company, San Francisco, 1979.
5. N. Garg and R. Khandekar, Fractional Covering with Upper Bounds on the Variables: Solving LPs with Negative Entries. In Proc. of *ESA*, 2004.
6. M. Grötschel, L. Lovász and A. Schrijver, The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, I, 169–197, 1981.
7. D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: Practical and theoretical results. *Journal of the ACM,* 34(1):144-162, 1987.
8. N. Karmarkar and R.M. Karp. An efficient approximation scheme for the one dimensional bin packing problem. *Proc. 23rd IEEE Annual Symposium on Foundations of Computer Science*, 312-320, 1982.
9. R. Khandekar. Lagrangian relaxation based algorithms for convex programming problems. *PhD thesis, Indian Institute of Technology Delhi*, 2004. In http://www.cse.iitd.ernet.in/∼rohitk.
10. C.A. Mandal, P.P Chakrabarti, and S. Ghose. Complexity of fragmentable object bin packing and an application. *Computers and Mathematics with Applications*, vol.35, no.11, 91–97, 1998.
11. R. McNaughton. Scheduling with deadlines and loss functions. *Manage. Sci.*, 6:1–12, 1959.
12. N. Menakerman and R. Rom. Bin Packing Problems with Item Fragmentations. *Proc. of WADS*, 2001.
13. R. Motwani. Lecture notes on approximation algorithms. Technical report, Dept. of Computer Science, Stanford Univ., CA, 1992.
14. Multimedia Cable Network System Ltd., Data-Over-Cable Service Interface Specification, *http://www.cablelabs.com*, 2000.
15. N. Naaman R. Rom. Packet Scheduling with Fragmentation. *Proc. of INFOCOM'02*, 824-831, 2002.
16. H. Shachnai, T. Tamir, and G.J Woeginger. Minimizing Makespan and Preemption Costs on a System of Uniform Machines. *Algorithmica*, 42: 309–334, 2005.
17. H. Shachnai, T. Tamir, and O. Yehezkely. Approximation Schemes for Packing with Item Fragmentation. full version. http://www.cs.technion.ac.il/∼hadas/PUB/frag.pdf.
18. V.V. Vazirani. Bin Packing. In *Approximation Algorithms*, 74-78, Springer, 2001.

Author Index