Paul G. Spirakis
Maria Serna (Eds.)

LNCS 7878

Algorithms and Complexity

8th International Conference, CIAC 2013
Barcelona, Spain, May 2013
Proceedings

Springer

Paul G. Spirakis   Maria Serna (Eds.)

Algorithms and Complexity

8th International Conference, CIAC 2013
Barcelona, Spain, May 22-24, 2013
Proceedings

Springer

Volume Editors

Paul G. Spirakis
University of Patras
School of Engineering
Department of Computer Engineering and Informatics
265 00 Patras, Greece
E-mail: spirakis@cti.gr

Maria Serna
Universitat Politècnica de Catalunya
LSI Department
Edif Omega
Campus Nord
Jordi Girona 1-3
08034 Barcelona, Spain
E-mail: mjserna@lsi.upc.edu

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-38232-1 e-ISBN 978-3-642-38233-8
DOI 10.1007/978-3-642-38233-8
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013937318

CR Subject Classification (1998): F.2.2, F.2, E.1, G.2, I.3.5, C.2, F.1

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

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This volume contains the papers presented at the 8th International Conference on Algorithms and Complexity (CIAC 2013), which took place at the School of Mathematics and Statistics, Universitat Politècnica de Catalunya, Barcelona, during May 22–24, 2013. This series of conferences present research contributions in the theory and applications of algorithms and computational complexity.

The volume contains, in alphabetical order by first author, 31 accepted papers, selected by the Program Committee from 75 submissions received.

We thank all the authors who submitted papers, the members of the Program Committee, and the external reviewers. We gratefully acknowledge support from Universitat Politècnica de Catalunya, its Department of Software, the ALBCOM Research group, and the EATCS.

We would also like to thank Carme Àlvarez, Amalia Duch, and Maria Blesa for their help in the organization tasks.

March 2013

Paul Spirakis
Maria Serna

Organization

Program Committee

Paul Spirakis (Chair)	Patras University and Computer Technology Institute and Press Diophantus, Greece
Helmut Alt	Free University of Berlin, Germany
Tiziana Calamoneri	University of Rome La Sapienza, Italy
Ioannis Chatzigiannakis	Computer Technology Institute and Press Diophantus, Greece
Shlomi Dolev	Ben Gurion University, Israel
Sandor Fekete	Braunschweig University of Technology, Germany
Dimitris Fotakis	National Technical University of Athens, Greece
Kazuo Iwama	Kyoto University, Japan
Christos Kaklamanis	Patras University and Computer Technology Institute and Press Diophantus, Greece
Rajgoplan Kannan	Louisiana State University, USA
Alexis Kaporis	University of the Aegean, Greece
Stefano Leonardi	University of Rome La Sapienza, Italy
Alberto Marchetti Spaccamela	University of Rome La Sapienza, Italy
George Mertzios	Durham University, UK
Friedhelm Meyer Auf Der Heide	Heinz Nixdorf Institute and University of Paderborn, Germany
Othon Michail	Patras University and Computer Technology Institute and Press Diophantus, Greece
Sotiris Nikoletseas	Patras University and Computer Technology Institute and Press Diophantus, Greece
Vangelis Paschos	LAMSADE, University of Paris-Dauphine, France
Giuseppe Persiano	Università di Salerno, Italy
Jose Rolim	University of Geneva, Switzerland
Branislav Rovan	Comenius University, Slovakia
Maria Serna (Co-chair)	Universitat Politècnica de Catalunya, Spain
Emo Welzl	ETH Zurich, Switzerland

Steering Committee

Giorgio Ausiello	University of Rome La Sapienza, Italy
Josep Daz	Universitat Politècnica de Catalunya, Spain
Rossella Petreschi	University of Rome La Sapienza, Italy

Additional Reviewers

Adamczyk, Marek
Alvarez, Carme
Artigas, Danilo
Auletta, Vincenzo
Beyersdorff, Olaf
Bilo, Vittorio
Blesa, Maria J.
Bodlaender, Hans L.
Bonifaci, Vincenzo
Bonnet, Edouard
Buchin, Kevin
Busch, Costas
Butenko, Sergiy
Byrka, Jaroslaw
Cabello, Sergio
Chaplick, Steven
Cheilaris, Panagiotis
Clementi, Andrea
Cord-Landwehr, Andreas
Dantchev, Stefan
De Bonis, Annalisa
De Marco, Gianluca
Della Croce, Federico
Demange, Marc
Dimitrios, Michail
Dulman, Stefan
Eikel, Benjamin
Epstein, Leah
Escoffier, Bruno
Ferraioli, Diodato
Fineman, Jeremy
Finocchi, Irene
Fischer, Matthias
Forisek, Michal
Frigioni, Daniele
Galesi, Nicola
Geffert, Viliam
Giannopoulos, Panos
Gkatzelis, Vasilis
Gonçalves, Daniel
Grandoni, Fabrizio
Gravin, Nikolai
Gärtner, Bernd
Hanemann, Ariel
Heggernes, Pinar
Hoffmann, Michael
Jeż, Łukasz
Karanikolas, Nikos
Kentros, Sotirios
Kim, Eunjung
Kling, Peter
Kovacs, Annamaria
Kralovic, Rastislav
Kratochvil, Jan
Kroeller, Alexander
Kusters, Vincent
Kyropoulou, Maria
Lachish, Oded
Lampis, Michael
Letsios, Dimitrios
Levit, Vadim
Liao, Kewen
Liotta, Beppe
Lubiw, Anna
Lucarelli, Giorgio
Makris, Christos
Manlove, David
Markarian, Christine
Mihalak, Matus
Milis, Ioannis
Mittas, Nikolaos
Monti, Angelo
Morin, Pat
Moscardelli, Luca

Moser, Robin
Mulzer, Wolfgang
Mustata, Irina
Nika, Marily
Nisse, Nicolas
Nutov, Zeev
Panagopoulou, Panagiota
Papakonstantinopoulou, Katia
Pardubska, Dana
Penna, Paolo
Perez, Anthony
Pietrzyk, Peter
Proietti, Guido
Rapti, Maria
Raptopoulos, Christoforos
Rosen, Adi
Rossmanith, Peter
Sau, Ignasi
Scharf, Ludmila
Scheder, Dominik
Schlipf, Lena
Schmidt, Christiane
Schoengens, Marcel
Schöbel, Anita
Shalom, Mordechai
Sikora, Florian
Sinaimeri, Blerina
Sioutas, Spyros
Skoviera, Martin
Sommer, Christian
Speckmann, Bettina
Stanek, Martin
Syrgkanis, Vasilis
Tamaki, Suguru
Telelis, Orestis
Theodoridis, Evangelos
Thilikos, Dimitrios
Tsichlas, Kostas
Ueno, Kenya
Ventre, Carmine
Vrto, Imrich
Wahlström, Magnus
Weimann, Oren
Wiese, Andreas
Woeginger, Gerhard J.
Wu, Zhilin

Table of Contents

Approximation Algorithms for Disjoint *st*-Paths with Minimum Activation Cost

Hasna Mohsen Alqahtani and Thomas Erlebach

Department of Computer Science, University of Leicester, Leicester, UK
{hmha1,t.erlebach}@leicester.ac.uk

Abstract. In network activation problems we are given a directed or undirected graph $G = (V, E)$ with a family $\{f_{uv}(x_u, x_v) : (u, v) \in E\}$ of monotone non-decreasing activation functions from D^2 to $\{0, 1\}$, where D is a constant-size domain. The goal is to find activation values x_v for all $v \in V$ of minimum total cost $\sum_{v \in V} x_v$ such that the activated set of edges satisfies some connectivity requirements. Network activation problems generalize several problems studied in the network literature such as power optimization problems. We devise an approximation algorithm for the fundamental problem of finding the *Minimum Activation Cost Pair of Node-Disjoint st-Paths* (MA2NDP). The algorithm achieves approximation ratio 1.5 for both directed and undirected graphs. We show that a ρ-approximation algorithm for MA2NDP with fixed activation values for s and t yields a ρ-approximation algorithm for the *Minimum Activation Cost Pair of Edge-Disjoint st-Paths* (MA2EDP) problem. We also study the MA2NDP and MA2EDP problems for the special case $|D| = 2$.

1 Introduction

In this paper we consider network activation problems. In these problems we are given an *activation network*, which is a directed or undirected graph $G = (V, E)$ together with a family $\{f_{uv}(x_u, x_v) : (u, v) \in E\}$ of monotone non-decreasing activation functions from D^2 to $\{0, 1\}$, where D is a constant-size domain. The activation of an edge depends on the chosen values from the domain D at its endpoints. We say that an edge $(u, v) \in E$ is *activated* for chosen values x_u and x_v if $f_{uv}(x_u, x_v) = 1$. An activation function is called *monotone non-decreasing* if for every $(u, v) \in E$ we have that $f_{uv}(x_u, x_v) = 1$ implies $f_{uv}(y_u, y_v) = 1$ for any $y_u \geq x_u$, $y_v \geq x_v$. The goal is to determine activation values $x_v \in D$ for all $v \in V$ so that the total activation cost $\sum_{v \in V} x_v$ is minimized and the activated set of edges satisfies some connectivity requirements. Network activation problems were introduced by Panigrahi [11]. They generalize several known problems in wireless network design, e.g., minimum broadcast tree, installation cost optimization, and power optimization. For further applications and motivation for network activation problems we refer to [11,9].

We assume in the remainder of the paper that G is a directed graph. For the problems under consideration, the case of undirected graphs can be modelled by replacing each undirected edge $\{u, v\}$ by two directed edges (u, v) and (v, u) with the same activation function, i.e., $f_{uv}(x_u, x_v) = f_{vu}(x_v, x_u)$.

As D is a constant-size domain, we assume that the activation functions are specified by lookup tables. For each edge $(u, v) \in E$ we can then compute in polynomial time the minimum cost $c_{uv} = x_u^{uv} + x_v^{uv}$, where x_u^{uv} is the activation value on u and x_v^{uv} the activation value on v such that $f_{uv}(x_u^{uv}, x_v^{uv}) = 1$ and $x_u^{uv} + x_v^{uv}$ is minimized.

Related Work. The core objective of most network activation problems is to activate a network of minimum activation cost satisfying certain given connectivity requirements. The simplest connectivity requirement is to find an st-path for a specified pair of nodes s and t. Other examples of fundamental connectivity requirements are: spanning tree, k edge-disjoint st-paths, k node-disjoint st-paths, etc. Traditionally, these problems have been studied in a network model where each edge (or node) has a fixed cost.

In recent years, considerable work has been done on various network activation problems such as *Minimum Steiner Activation Network* (MSAN), *Minimum Spanning Activation Tree* (MSpAT), and *Minimum Activation Flow* (MAF). The problem of activating a network with k edge/node-disjoint paths between every pair of nodes is called *Minimum Edge/Node-connected Activation Network* (MEAN/MNAN). Panigrahi [11] gives an exact polynomial-time algorithm to solve the *Minimum Activation st-Path* (MAP) problem. However, he observes that the MAF problem (activating k edge-disjoint st-paths with minimum activation cost) is at least as hard as the ℓ-densest subgraph problem. As shown in [11], it is NP-hard to approximate MSpAT within a factor of $o(\log n)$. The MSpAT problem is a special case of the MSAN, MEAN and MNAN problems. Therefore, it is also NP-hard to approximate these problems within $o(\log n)$. Panigrahi presents $O(\log n)$-approximation algorithms for MSpAT, and also for MEAN and MNAN in the case of $k = 2$. Nutov [9] establishes a connection between network activation problems and edge-cost network design problems and shows that there exists a 2-approximation algorithm for the *Minimum Activation Cost k Node-Disjoint st-Paths* (MAkNDP) problem and a $2k$-approximation algorithm for the *Minimum Activation Cost k Edge-Disjoint st-Paths* (MAkEDP) problem.

Other relevant work has addressed power optimization [2,7,8]. In power optimization problems, each edge $(u, v) \in E$ has a threshold power requirement θ_{uv}. In the undirected case, edge (u, v) is activated for chosen values x_u and x_v if each of these values is at least θ_{uv}. In the directed case, edge (u, v) is activated if $x_u \geq \theta_{uv}$.

Power optimization is a special case of network activation problems. As mentioned in [11], in the power optimization setting the MEAN and MNAN problems have 4-approximation and 11/3-approximation algorithms, respectively, and it is known that the MSpAT problem is APX-hard. By a simple reduction to the shortest st-path problem, the *Minimum Power st-Path* problem is solvable in polynomial time for both directed and undirected networks [7]. Another problem that has been studied in the literature is finding the *Minimum Power k*

Edge-Disjoint st-Paths (MPkEDP). [6] shows that for both the directed and undirected variants, the MPkEDP problem is unlikely to admit even a polylogarithmic approximation algorithm. In contrast, the problem of finding Minimum Power k Node-Disjoint st-Paths in directed graphs can be solved in polynomial time [6,12].

The problem of finding node/edge disjoint st-paths with minimum cost in a network with edge costs is a well studied problem in graph theory. Polynomial-time algorithms have been known for decades [1,13,14]. These algorithms do not address the problem in the network activation setting, however. In this paper, we study the minimum activation cost pair of node/edge-disjoint st-paths problem. To the best of our knowledge, it is not yet known whether these problems are NP-hard. A *ρ-approximation algorithm* for a network activation problem is an algorithm that runs in polynomial time and always outputs a solution whose activation cost is at most ρ times the optimal activation cost for the given instance.

Our Results. We give a 1.5-approximation algorithm for the MA2NDP problem. We also show that a ρ-approximation algorithm for the MA2NDP problem with fixed activation values of s and t implies a ρ-approximation algorithm for the MA2EDP problem. For the case where the domain D has size 2 and all edges of the network have the same activation function, we prove that the MAkNDP problem is polynomial-time solvable for four of five cases of the activation function, and that the MAkEDP problem is NP-hard.

We employ ideas and techniques from the theory of network flows in order to establish approximation algorithms for our problems. The idea of the MA2NDP algorithm is to first guess the optimal activation values for the nodes s and t by enumeration. For each choice of activation values for s and t, we construct an edge-cost network from G. We then use ideas similar to Suurballe's algorithm [13], with modifications in the construction of the residual graph, to find the two node-disjoint st-paths. For the connection between the MA2NDP and MA2EDP problems, we design an approximation algorithm for the MA2EDP problem by using a ρ-approximation algorithm for the MA2NDP problem for every pair of nodes in the graph and then iteratively combining disjoint paths to/from an intermediate node into edge-disjoint paths with common nodes. We prove that this algorithm has approximation ratio ρ. For the special case where the domain D has size 2 and all edges have the same activation function, we show the NP-hardness of the MAkEDP problem by giving a reduction from the decision version of the maximum balanced complete bipartite subgraph (MaxBCBS) problem [4].

The remainder of the paper is organized as follows. We start by presenting our algorithm for the MA2NDP problem in Section 2. In Section 3, we establish the connection between the MA2NDP and MA2EDP problems and obtain a 1.5-approximation algorithm for MA2EDP. We then discuss the problem of finding k node/edge-disjoint st-paths with minimum activation cost in the case where $|D| = 2$ in Section 4. Finally, we conclude with a short section on future work and open questions. Some proofs are omitted due to space constraints.

2 Minimum Activation Cost Node-Disjoint *st*-Paths

The minimum activation k node-disjoint st-paths (MAkNDP) problem can be stated as follows: Given an activation network $G = (V, E)$ and source-destination pair $s, t \in V$, find activation values x_v for all $v \in V$ such that k node-disjoint st-paths $P^{st} = \{P_1, P_2, \ldots, P_k\}$ are activated and the total activation cost $\sum_{v \in V} x_v$ is minimized. [9] gave a 2-approximation algorithm for the MAkNDP problem. In this section we consider the special case $k = 2$ and give a 1.5-approximation algorithm for the MA2NDP problem.

Let FMA2NDP denote the variant of the MA2NDP problem where values $d \in D$ and $d' \in D$ are specified as activation values of s and t, respectively, and do not count towards the objective value. In other words, only solutions with $x_s = d$ and $x_t = d'$ are considered feasible for FMA2NDP, and the activation cost of a solution is $\sum_{v \in V \setminus \{s,t\}} x_v$.

The MA2NDP algorithm takes as input an activation network $G = (V, E)$ and a source-destination pair, $s, t \in V$. Its output is a set of activation values x_v for all $v \in V$ that activate a pair of node-disjoint st-paths, $P^{st} = \{P_1, P_2\}$. The algorithm enumerates all pairs $d, d' \in D$. For each choice of d and d', it solves the FMA2NDP problem with the values d and d' chosen as the activation values of s and t, respectively. Let $C(s, d, t, d')$ represent the activation cost of the solution for d, d'. In the end, the algorithm outputs the solution of minimum activation cost among the feasible solutions obtained for all pairs of values $d, d' \in D$, i.e., the solution of activation cost $\min_{x_s, x_t \in D}\{C(s, x_s, t, x_t)\}$.

The algorithm for the FMA2NDP problem with $x_s = d$ and $x_t = d'$ is as follows. For ease of presentation, we assume here that $D = \{0, 1, 2, \ldots, |D| - 1\}$. (The extension to arbitrary domains of constant size is straightforward.) We let $C(s, d, t, d')$ represent the total activation cost of the pair of node-disjoint activation paths P^{st} that the algorithm finds, or ∞ if such paths do not exist.

Step 1: Construct from G an edge-weighted graph $\bar{G}$ with two nodes s_d, $t_{d'}$ and $2|D|$ nodes $\{v_0^{in}, v_0^{out}, v_1^{in}, v_1^{out}, \ldots., v_{|D|-1}^{in}, v_{|D|-1}^{out}\}$ for every $v \in V \backslash \{s, t\}$. The edges of $\bar{G}$ are:
- For $a \in D$ and $v \in V \setminus \{s, t\}$, add a directed edge (v_a^{in}, v_a^{out}) with cost 0.
- For each $(u, v) \in E$ and $a, b \in D$ where $u, v \notin \{s, t\}$ and $f_{uv}(a, b) = 1$, add a directed edge (u_a^{out}, v_b^{in}) with cost b.
- For each $(s, v) \in E$ and $b \in D$ where $v \neq t$ and $f_{sv}(d, b) = 1$, add a directed edge (s_d, v_b^{in}) with cost b.
- For each $(v, t) \in E$ and $a \in D$ where $v \neq s$ and $f_{vt}(a, d') = 1$, add a directed edge $(v_a^{out}, t_{d'})$ with cost 0.
- If $(s, t) \in E$ and $f_{st}(d, d') = 1$, add a directed edge $(s_d, t_{d'})$ with cost 0.

Step 2: Run Dijkstra's algorithm on $\bar{G}$ to compute a shortest path P from s_d to $t_{d'}$. Let $C(P)$ be the edge-cost of P. If $\bar{G}$ has no such path, set $C(s, d, t, d') = \infty$ and skip Steps 3–5.

Step 3: Construct the residual network $\bar{G}_P$ induced by P:
- For each $v \in V \setminus \{s, t\}$ with $(v_a^{in}, v_a^{out}) \in P$ for some a, add a directed edge $(v_{\bar{a}}^{out}, v_{\bar{a}}^{in})$ with cost 0 for all $\bar{a} \in D$ with $\bar{a} \geq a$.

- For each $v \in V \setminus \{s,t\}$ such that $(v_a^{in}, v_a^{out}) \notin P$ for all $a \in D$, add (v_a^{in}, v_a^{out}) for all $a \in D$ with cost 0.
- For each $(u,v) \in E$ with $(u_a^{out}, v_b^{in}) \in P$ for some $a, b \in D$, add a directed edge $(v_{\bar{b}}^{in}, u_{\bar{a}}^{out})$ with cost $\bar{a} - a$ for all $\bar{a}, \bar{b} \in D$ where $\bar{a} \geq a$ and $\bar{b} \geq b$.
- For each $(u,v) \in E$ with $(u_a^{out}, v_b^{in}) \notin P$ for all $a, b \in D$ where v is used in P with activation value $\bar{b}$ (i.e., $v_{\bar{b}}^{in} \in P$), add edges $(u_{a'}^{out}, v_{b'}^{in})$ with cost $b' - \bar{b}$ for all $a' \in D$, $b' \geq \bar{b}$ such that $f_{uv}(a', b') = 1$.
- For each $(u,v) \in E$ with $(u_a^{out}, v_b^{in}) \notin P$ for all $a, b \in D$ where v is not used in P, add edges $(u_{a'}^{out}, v_{b'}^{in})$ with cost b' for all $a', b' \in D$ such that $f_{uv}(a', b') = 1$.
- For each $v \in V \setminus \{t\}$ with $(s,v) \in E$ and $(s_d, v_b^{in}) \notin P$ for all $b \in D$, add edges $(s, v_{b'}^{in})$ with cost b' for all $b' \in D$ such that $f_{sv}(d, b') = 1$.
- For each $v \in V \setminus \{s\}$ with $(v,t) \in E$ and $(v_b^{out}, t_{d'}) \notin P$ for all $b \in D$, add edges $(v_{b'}^{out}, t_{d'})$ with cost 0 for all $b' \in D$ such that $f_{vt}(b', d') = 1$.
- If $(s,t) \in E$ and $f_{st}(d, d') = 1$ and $(s_d, t_{d'}) \notin P$, add a directed edge $(s_d, t_{d'})$ with cost 0.

Step 4: Run Dijkstra's algorithm on the residual network $\bar{G}_P$ to identify a shortest path P' from s_d to $t_{d'}$. Let $C'(P')$ represent the edge-cost of P'. If no such path P' exists, set $C(s,d,t,d') = \infty$ and skip Step 5.

Step 5: Decompose P and P' into two node-disjoint paths, by removing from $P \cup P'$ the edge set which consists of the edges of P whose reverse edge is in P', and vice versa. Let P_1 and P_2 be the corresponding node-disjoint paths in G, and let $C(s,d,t,d')$ be the activation cost of $P^{st} = \{P_1, P_2\}$. Return $C(s,d,t,d')$ and P^{st}.

Note that the auxiliary graph $\bar{G}$ constructed in Step 1 has the property that any path Q from s_d to $t_{d'}$ in $\bar{G}$ with edge cost $C(Q)$ corresponds to an activated path Q' in G from s to t with activation cost $d + C(Q) + d'$, and vice versa. If the path Q uses an edge with head v_a^{in}, this corresponds to activating node v with activation value $x_v = a$ (and the cost of the edge 'pays' for this activation value). The shortest path constructed in Step 2 thus corresponds to a minimum activation cost st-path, under the constraint that $x_s = d$ and $x_t = d'$.

Let an instance of the FMA2NDP problem be given by a graph $G = (V, E)$ with designated nodes $s, t \in V$, a family F of activation functions from D^2 to $\{0,1\}$, and values $d, d' \in D$. Let $P^{st} = \{P_1, P_2\}$ be the paths found by the FMA2NDP algorithm and let $P^{OPT} = \{P_1^{OPT}, P_2^{OPT}\}$ be an optimum solution for this instance. We define $C_{ALG}(Q)$ as the activation cost of a path Q in G in the solution generated by the algorithm and $C_{OPT}(Q)$ as the activation cost of a path Q in the optimum solution. The edge cost of a path Q in $\bar{G}$ is denoted by $C(Q)$, and the edge cost of a path Q in $\bar{G}_P$ is denoted by $C'(Q)$.

Lemma 1. *For any $x_s, x_t \in D$ for which there are two node-disjoint st-paths, let $\bar{G}_P$ be the residual network of $\bar{G}$ imposed by P (Step 3). Then there exists a path $P' \in \bar{G}_P$ from s_d to $t_{d'}$ with edge-cost $C'(P')$ such that:*

$$C'(P') \leq C_{OPT}(P_1^{OPT} \setminus \{s,t\}) + C_{OPT}(P_2^{OPT} \setminus \{s,t\}) \tag{1}$$

Proof. Let $\tilde{G} \subset G$ be the network generated by edges that belong to P^{OPT} and the path in G that corresponds to P. Let $\tilde{G}_P$ be the (standard) residual network of $\tilde{G}$ imposed by P. There exists a path P^* from s to t in $\tilde{G}_P$. We have $C_{OPT}(P^*) = \sum_{v \in P^*} x_v$, where x_v is the activation value of v in the optimal solution. Clearly, $C_{OPT}(P^*)$ is at most the total cost of the optimal solution P^{OPT}. Consequently, $C_{OPT}(P^*) - (d+d') \leq C_{OPT}(P_1^{OPT} \backslash \{s,t\}) + C_{OPT}(P_2^{OPT} \backslash \{s,t\})$.

We want to prove that there is a path corresponding to P^* in $\bar{G}_P$ with edge cost at most $C_{OPT}(P^*) - d - d'$. To prove this, one can show that for each edge (u,v) of P^* there exists a corresponding edge in $\bar{G}_P$ whose cost is bounded by the activation value of v in P^{OPT}. As the algorithm computes a path P' with minimum edge cost in $\bar{G}_P$, we get that $C'(P') \leq \sum_{v \in P^* \backslash \{s,t\}} x_v = C_{OPT}(P^*) - (d+d')$. □

Theorem 1. *The algorithm computes a* 1.5*-approximation for the FMA2NDP problem.*

Proof. Since P is an st-path of minimum activation cost, we get that the activation cost of its intermediate nodes, which is equal to its edge cost $C(P)$, is bounded by

$$\begin{aligned} C(P) &\leq \min\{C_{OPT}\left(P_1^{OPT} \setminus \{s,t\}\right), C_{OPT}\left(P_2^{OPT} \setminus \{s,t\}\right)\} \\ &\leq \frac{C_{OPT}\left(P_1^{OPT} \setminus \{s,t\}\right) + C_{OPT}\left(P_2^{OPT} \setminus \{s,t\}\right)}{2} \end{aligned} \tag{2}$$

From Step 5 in the algorithm we notice that:

$$C_{ALG}(P_1 \setminus \{s,t\}) + C_{ALG}(P_2 \setminus \{s,t\}) \leq C(P) + C'(P') \tag{3}$$

From Lemma 1, (2) and (3) we get that the solution computed by the algorithm has objective value at most 1.5 times the optimal objective value. □

As our MA2NDP algorithm enumerates all possibilities for the activation values of s and t and outputs the solution of minimum activation cost among all computed solutions, Theorem 1 implies the following corollary.

Corollary 1. *There is a* 1.5*-approximation algorithm for MA2NDP.*

3 Minimum Activation Cost Edge-Disjoint *st*-Paths

The minimum activation cost k edge-disjoint st-paths problem (MAkEDP) can be stated as follows: Given an activation network $G = (V, E)$ and a source-destination pair $s, t \in V$, find activation values x_v for all $v \in V$ that activate a set of k edge-disjoint st-paths $P^{st} = \{P_1, P_2, \ldots, P_k\}$ such that the total cost $\sum_{v \in V} x_v$ is minimized. We consider the problem for $k = 2$, i.e., MA2EDP.

We observe that a pair of edge-disjoint st-paths can be viewed as the concatenation of pairs of node-disjoint paths between consecutive common nodes of the pair of edge-disjoint paths (see Fig. 1). This connection was used by Srinivas and Modiano [12] to derive a polynomial-time optimal algorithm for the special case of MA2EDP that arises in the power optimization setting. We generalize this method to the network activation setting and obtain the following theorem that connects the FMA2NDP and MA2EDP problems.

Theorem 2. *If there exist a ρ-approximation algorithm for FMA2NDP, then there exists a ρ-approximation algorithm for MA2EDP.*

The proof of this theorem is based on showing that the following MA2EDP algorithm computes a pair of edge-disjoint paths of activation cost at most ρ times the optimal activation cost. The MA2EDP algorithm takes as input an activation network $G = (V, E)$ and a source-destination pair, $s, t \in V$. Its output is a pair of edge-disjoint activated st-paths, $P^{st} = \{P_1, P_2\}$. The algorithm executes a ρ-approximation algorithm for FMA2NDP for each pair of nodes in G with specified activation values for that pair of nodes, and then iteratively combines disjoint paths to/from an intermediate node to obtain edge-disjoint paths. The MA2EDP algorithm can be specified via the following two steps:

Step 1: For every pair of nodes $u, u' \in V$ and every pair of activation values $d, d' \in D$, the algorithm runs the ρ-approximation algorithm for the FMA2NDP problem with source u, activated with $x_u = d$, and destination u', activated with $x_{u'} = d'$. This produces a pair of node-disjoint activation paths for each pair of nodes $u, u' \in V$ and specified activation values $x_u, x_{u'}$. Let $P^{(u,x_u,u',x_{u'})}$ denote this pair of node-disjoint paths (and the corresponding activation values of all nodes) and $C(u, x_u, u', x_{u'})$ its activation cost (or ∞, if such a pair of node-disjoint paths does not exist).

Step 2:

```
**for** each node $w \in V$:
  **for** each $x_w \in D$, each pair of nodes $u, u' \in V$ and each pair $x_u, x_{u'} \in D$:
    Combine the pairs of edge-disjoint paths $P^{(u,x_u,w,x_w)}$ and $P^{(w,x_w,u',x_{u'})}$
    into a pair of edge-disjoint paths $Q^{(u,x_u,u',x_{u'})}$ from $u$ to $u'$ and update
    the cost $C(u, x_u, u', x_{u'})$ via:

      $C(u, x_u, u', x_{u'}) =$
        $\min\{C(u, x_u, u', x_{u'}), C(u, x_u, w, x_w) + C(w, x_w, u', x_{u'}) - x_w\}$

    If the cost $C(u, x_u, u', x_{u'})$ changes by this update, set $P^{(u,x_u,u',x_{u'})}$ to
    $Q^{(u,x_u,u',x_{u'})}$.
```

The final output is the activation cost $\min_{x_s,x_t \in D}\{C(s, x_s, t, x_t)\}$ and the corresponding pair of edge-disjoint st-paths $P^{(s,x_s,t,x_t)}$.

To show that the MA2EDP algorithm actually finds a pair of edge-disjoint paths of activation cost at most $C(s, x_s, t, x_t)$, we have the following lemma.

Lemma 2. *Consider any time in the execution of the algorithm. Assume that at that time we have $C(u, x_u, u', x_{u'}) = T < \infty$. Then $P^{(u,x_u,u',x_{u'})}$ contains two edge-disjoint uu'-paths with activation cost at most T such that u has activation value at least x_u and u' has activation value at least $x_{u'}$.*

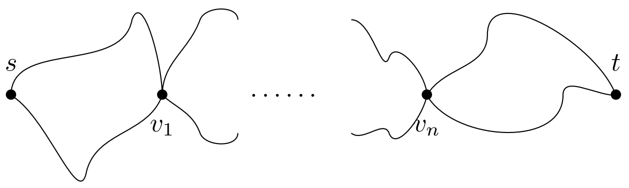


Fig. 1. A pair of edge-disjoint paths viewed as the concatenation of pairs of node-disjoint paths

Proof (of Theorem 2). Let an instance of the MA2EDP problem be given by $G = (V, E)$, $s, t \in V$ and a family F of activation functions. Let $V = \{v_1, v_2, \dots, v_n\}$, where the nodes are numbered in the order in which they are processed by the outer for-loop in Step 2. Define $C_k(u, x_u, v, x_v)$ as the value of $C(u, x_u, v, x_v)$ after $k \in \{0, ..., n\}$ iterations of the outer for-loop in Step 2 and $C_k^{OPT}(u, x_u, v, x_v)$ to be the optimal activation cost for two edge-disjoint paths from u, activated with x_u, to v, activated with x_v, for which the only intermediate nodes that are common between the paths are among $\{v_1, \dots, v_k\}$. Let $\bar{C}_k(u, x_u, v, x_v) = C_k(u, x_u, v, x_v) - x_u - x_v$ and $\bar{C}_k^{OPT}(u, x_u, v, x_v) = C_k^{OPT}(u, x_u, v, x_v) - x_u - x_v$. By induction, we will prove that for all $u, v \in V$ and $x_u, x_v \in D$ after k iterations the following holds:

$$\bar{C}_k(u, x_u, v, x_v) \;\le\; \rho\, \bar{C}_k^{OPT}(u, x_u, v, x_v) \tag{4}$$

Induction Base: If $k = 0$ (there is no common intermediate node), (4) holds as before the first iteration the algorithm uses a ρ-approximation for FMA2NDP.
Induction Step: Assume that the statement (4) holds for the case where all common nodes between the two paths are among $\{v_1, v_2, ..., v_{k-1}\}$. This means that for all $u, v \in V$, $x_u, x_v \in D$ after $k - 1$ iterations of the algorithm, we have $\bar{C}_{k-1}(u, x_u, v, x_v) \;\le\; \rho\, \bar{C}_{k-1}^{OPT}(u, x_u, v, x_v)$.

Now consider the k-th iteration, where v_k is considered as additional intermediate node for two edge-disjoint paths from u, activated with at least x_u, to v, activated with at least x_v:

If the optimum solution for the two edge-disjoint uv-paths with common nodes among $\{v_1, \dots, v_k\}$ uses only nodes among $\{v_1, v_2, .., v_{k-1}\}$ as common nodes, by induction hypothesis, (4) holds as $\bar{C}_k(u, x_u, v, x_v) \le \bar{C}_{k-1}(u, x_u, v, x_v)$.

If the optimum solution for the two edge-disjoint uv-paths with common nodes among $\{v_1, \dots, v_k\}$ uses the node v_k with activation value x_{v_k} as common node, then we have:

$$\begin{aligned}
\bar{C}_k(u, x_u, v, x_v) &\le \bar{C}_{k-1}(u, x_u, v_k, x_{v_k}) + \bar{C}_{k-1}(v_k, x_{v_k}, v, x_v) + x_{v_k} \\
&\le \rho\, \bar{C}_{k-1}^{OPT}(u, x_u, v_k, x_{v_k}) + \rho\, \bar{C}_{k-1}^{OPT}(v_k, x_{v_k}, v, x_v) + x_{v_k} \\
&\le \rho \left(\bar{C}_{k-1}^{OPT}(u, x_u, v_k, x_{v_k}) + \bar{C}_{k-1}^{OPT}(v_k, x_{v_k}, v, x_v) + x_{v_k} \right) \\
&= \rho\, \bar{C}_k^{OPT}(u, x_u, v, x_v)\,.
\end{aligned}$$

This completes the proof of the theorem. □

From Theorem 1 and Theorem 2, we obtain the following corollary.

Corollary 2. *The MA2EDP algorithm computes a* 1.5*-approximate solution for the MA2EDP problem.*

We remark that for the special case of power optimization, [6,12] gave an exact polynomial-time algorithm for the MAkNDP problem in directed graphs. Theorem 2 thus implies that there is an exact polynomial-time algorithm for the directed MA2EDP problem for power optimization, as was already shown in [12]. Nutov [8] shows that for arbitrary k there exists a k-approximation algorithm for the directed case of the MAkEDP problem.

4 Activation Networks with $|D| = 2$

In this section, we restrict the domain D to have size 2. This case is interesting from a theoretical point of view because it is the smallest non-trivial case for the size of the domain. From a practical point of view, this case corresponds to a simple setting where nodes have just two different activation states, e.g., low power and high power. Let $D = \{a, b\}$ with $a < b$. Note that the cost of a solution that activates B nodes with activation value b and $|V| - B$ nodes with activation value a is $a|V| + B(b - a)$. This means that minimizing the activation cost is equivalent to minimizing the number of nodes that have activation value b. In the rest of this section, we assume that all edges of the activation network have the same activation function $f : D^2 \to \{0, 1\}$.

4.1 Polynomial Cases of MAkNDP and MA2EDP

The following are all the different possibilities for a monotone non-decreasing activation function f with domain $D = \{a, b\}$:

1. $f(a, a) = 1$
2. $f(a, a) = f(a, b) = f(b, a) = 0, f(b, b) = 1$
3. $f(a, a) = f(a, b) = 0, f(b, a) = f(b, b) = 1$
4. $f(a, a) = f(b, a) = 0, f(a, b) = f(b, b) = 1$
5. $f(a, a) = 0, f(a, b) = f(b, a) = f(b, b) = 1$

The problem MAkNDP for activation function 1 is trivial as either the solution that activates all nodes with activation value a is optimal, or there is no feasible solution. For activation functions 2-4, we observe that the problem of minimizing the activation cost of node-disjoint paths from s to t is equivalent to the problem of minimizing the number of nodes used by the paths: For activation function 2, all nodes on all paths must be activated with value b. For activation functions 3, all nodes on all paths except node t must be activated with value b. For activation function 4, all nodes on all paths except node s must be activated with value b. To calculate the optimal solution in these cases, we first give unit cost to all edges of the graph, compute k node-disjoint paths of minimum edge-cost using a known polynomial-time algorithm for the minimum cost k-flow problem (with unit edge and node capacities), and finally activate the resulting network. Therefore, the MAkNDP problem with activation functions 1-4 can be solved in polynomial time for any k.

The problem with activation function 5 is polynomial-time solvable for $k = 2$. Assume that the minimum number of internal nodes (nodes excluding s, t) used by two node-disjoint paths from s to t is $M \geq 1$. We activate s with value b, and every other node on each of the two paths. If M is odd, one of the two paths must have an odd number o of internal nodes, the other an even number e of internal nodes, $M = o + e$. In total we activate $1 + (o + 1)/2 + e/2 = M/2 + 1.5$ nodes with value b, and this is optimal and independent of o, e. If M is even, there are two cases: If the two paths both have odd numbers of internal nodes, say o_1 and o_2, we activate $2 + (o_1 - 1)/2 + (o_2 - 1)/2 = M/2 + 1$ nodes with value b. If the two paths both have even numbers of internal nodes, say e_1 and e_2, we activate $1 + e_1/2 + e_2/2 = M/2 + 1$ nodes with value b. In both cases the optimal number of nodes activated with value b depends only on M. Thus, the MA2NDP problem for activation function 5 can also be solved by minimizing the number of nodes used by the two paths.

Note that minimizing the number of nodes used by the paths is not sufficient for activation function 5 and $k = 3$. If the three node-disjoint paths have 3, 1 and 1 internal nodes, respectively, the number of nodes that must be activated with value b is 3. If the three node-disjoint paths have 2, 2 and 1 internal nodes, respectively, the number of nodes that must be activated with value b is 4. In both cases the total number of nodes used by the three paths is the same, but only one of the two cases yields a solution with optimal activation cost.

For all five activation functions, it is easy to see that FMA2NDP is also polynomial-time solvable, and hence MA2EDP can be solved optimally in polynomial time by application of Theorem 2.

4.2 Hardness of MAkEDP

Panigrahi [11] showed that the MAkEDP problem is NP-hard since it generalizes the Node-Weighted k-Flow (NWkF) problem which is known to be NP-hard [10]. Nutov [10] proved the inapproximability of the NWkF problem by a reduction from the bipartite densest ℓ-subgraph problem to unit weight NWkF, and Panigrahi [11] observed that this inapproximability result can be adapted to MAkEDP as well. The reduction described in [10] uses parallel edges. Here, we use a similar approach that avoids parallel edges and establishes that the MAkEDP problem is NP-hard even in the case where $|D| = 2$, all edges have the same activation function, and there are no parallel edges. This is in contrast to the polynomial-time solvability of MAkNDP when $|D| = 2$ for activation functions 1–4 and arbitrary k. We show the hardness of the MAkEDP problem by giving a reduction from the decision version of the maximum balanced complete bipartite subgraph problem, which is NP-hard (Problem GT24 in [5]).

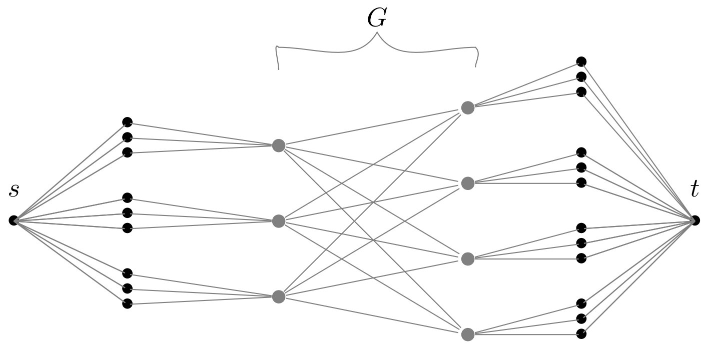


Fig. 2. A reduction of MaxBCBS (g=3)

Maximum Balanced Complete Bipartite Subgraph (MaxBCBS). Given a bipartite graph $G = (V_1 \cup V_2, E)$, find a maximum balanced complete bipartite subgraph (i.e. with the maximum number of nodes). Here, a balanced complete bipartite subgraph H is a complete bipartite subgraph such that $|H \cap V_1| = |H \cap V_2|$.

In the decision version of MaxBCBS, we are additionally given a parameter g and the question is to decide whether G contains a balanced complete bipartite subgraph with $2g$ nodes (and g^2 edges).

The Reduction. Let a bipartite instance of the decision version of MaxBCBS be given by $G = (V_1 \cup V_2, E)$ and parameter g. We construct an instance $\bar{G} = (\bar{V}, \bar{E})$ of MAkEDP as follows: We add to G as new nodes a source s and a target t. For each $v \in V_1 \cup V_2$ we add g new nodes $\{v_i : 1 \leq i \leq g\}$. For each $v \in V_1$, we add the edges $\{sv_i : 1 \leq i \leq g\} \cup \{v_i v : 1 \leq i \leq g\}$. For each $v \in V_2$, we add the edges $\{vv_i : 1 \leq i \leq g\} \cup \{v_i t : 1 \leq i \leq g\}$. The domain is $D = \{0, 1\}$, and for all $uv \in \bar{E}$, let $f_{uv}(1, 1) = 1$ and $f_{uv}(0, 0) = f_{uv}(1, 0) = f_{uv}(0, 1) = 0$. See Fig. 2. The reduction can be used for both directed and undirected graphs. We can show the following lemma and theorem.

Lemma 3. *There exists a balanced complete bipartite subgraph $K_{g,g}$ with $2g$ nodes in G if and only if there exist $k = g^2$ edge-disjoint paths in $\bar{G}$ of activation cost $2g^2 + 2g + 2$.*

Theorem 3. *The MAkEDP problem is NP-hard even for activation networks where the domain D is $\{0, 1\}$ and all edges $uv \in E$ have the same activation function $f : D^2 \rightarrow \{0, 1\}$.*

5 Conclusion

We have investigated the problem of finding disjoint st-paths of minimum activation cost in a given activation network. We gave a 1.5-approximation algorithm for the MA2NDP problem and showed that a ρ-approximation algorithm for the

FMA2NDP problem (MA2NDP with fixed activation values of s and t) can be used to obtain a ρ-approximation algorithm for the MA2EDP problem. For the restricted version of activation networks with $|D| = 2$ and a single activation function for all edges, we showed that MAkNDP can be solved in polynomial-time for arbitrary k (for $k = 2$ in one of the cases for the activation function). In addition, we showed that this restricted version of the MAkEDP problem is NP-hard.

The main open problem is to determine whether the MA2NDP problem is NP-hard. Our results show that a polynomial-time optimal algorithm for MA2NDP would imply a polynomial-time optimal algorithm for MA2EDP. It would also be interesting to study the case of domain size 2 in a setting where different edges can have different activation functions.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall, New Jersey (1993)
2. Calinescu, G., Kapoor, S., Olshevsky, A., Zelikovsky, A.: Network lifetime and power assignment in ad hoc wireless networks. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 114–126. Springer, Heidelberg (2003)
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. The MIT Press and McGraw-Hill Book Company (2001)
4. Feige, U., Kogan, S.: Hardness of approximation of the balanced complete bipartite subgraph problem. Technical report MCS04-04, Department of Computer Science and Applied Math., The Weizmann Institute of Science (2004)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability. A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, New York, San Francisco (1979)
6. Hajiaghayi, M.T., Kortsarz, G., Mirrokni, V.S., Nutov, Z.: Power optimization for connectivity problems. In: Jünger, M., Kaibel, V. (eds.) IPCO 2005. LNCS, vol. 3509, pp. 349–361. Springer, Heidelberg (2005)
7. Lando, Y., Nutov, Z.: On minimum power connectivity problems. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 87–98. Springer, Heidelberg (2007)
8. Nutov, Z.: Approximating minimum power covers of intersecting families and directed connectivity problems. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX 2006 and RANDOM 2006. LNCS, vol. 4110, pp. 236–247. Springer, Heidelberg (2006)
9. Nutov, Z.: Survivable network activation problems. In: Fernández-Baca, D. (ed.) LATIN 2012. LNCS, vol. 7256, pp. 594–605. Springer, Heidelberg (2012)
10. Nutov, Z.: Approximating Steiner networks with node-weights. SIAM J. Comput. 39(7), 3001–3022 (2010)
11. Panigrahi, D.: Survivable network design problems in wireless networks. In: 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1014–1027. SIAM (2011)
12. Srinivas, A., Modiano, E.: Finding Minimum Energy Disjoint Paths in Wireless Ad-Hoc Networks. Wireless Networks 11(4), 401–417 (2005)
13. Suurballe, J.W.: Disjoint paths in a network. Networks 4, 125–145 (1974)
14. Suurballe, J.W., Tarjan, R.E.: A quick method for finding shortest pairs of disjoint paths. Networks 14, 325–336 (1984)

Parameterized Complexity and Kernel Bounds for Hard Planning Problems

Christer Bäckström[1], Peter Jonsson[1], Sebastian Ordyniak[2], and Stefan Szeider[3,⋆]

[1] Department of Computer Science, Linköping University, Linköping, Sweden
[2] Faculty of Informatics, Masaryk University, Brno, Czech Republic
[3] Institute of Information Systems, Vienna University of Technology, Vienna, Austria

Abstract. The *propositional planning* problem is a notoriously difficult computational problem. Downey et al. (1999) initiated the parameterized analysis of planning (with plan length as the parameter) and Bäckström et al. (2012) picked up this line of research and provided an extensive parameterized analysis under various restrictions, leaving open only one stubborn case. We continue this work and provide a full classification. In particular, we show that the case when actions have no preconditions and at most e postconditions is fixed-parameter tractable if $e \leq 2$ and W[1]-complete otherwise. We show fixed-parameter tractability by a reduction to a variant of the Steiner Tree problem; this problem has been shown fixed-parameter tractable by Guo et al. (2007). If a problem is fixed-parameter tractable, then it admits a polynomial-time self-reduction to instances whose input size is bounded by a function of the parameter, called the *kernel*. For some problems, this function is even polynomial which has desirable computational implications. Recent research in parameterized complexity has focused on classifying fixed-parameter tractable problems on whether they admit polynomial kernels or not. We revisit all the previously obtained restrictions of planning that are fixed-parameter tractable and show that none of them admits a polynomial kernel unless the polynomial hierarchy collapses to its third level.

1 Introduction

The propositional planning problem has been the subject of intensive study in knowledge representation, artificial intelligence and control theory and is relevant for a large number of industrial applications [13]. The problem involves deciding whether an *initial state*—an n-vector over some set D–can be transformed into a *goal state* via the application of *operators* each consisting of *preconditions* and *post-conditions* (or *effects*) stating the conditions that need to hold before the operator can be applied and which conditions will hold after the application of the operator, respectively. It is known that deciding whether an instance has a solution is PSPACE-complete, and it remains at least NP-hard under various

⋆ Research supported by the ERC, grant reference 239962.

restrictions [6,3]. In view of this intrinsic difficulty of the problem, it is natural to study it within the framework of Parameterized Complexity which offers the more relaxed notion of *fixed-parameter tractability* (FPT). A problem is fixed-parameter tractable if it can be solved in time $f(k)n^{O(1)}$ where f is an arbitrary function of the parameter and n is the input size. Indeed, already in a 1999 paper, Downey, Fellows and Stege [8] initiated the parameterized analysis of propositional planning, taking the minimum number of steps from the initial state to the goal state (i.e., the length of the solution plan) as the parameter; this is also the parameter used throughout this paper. More recently, Bäckström et al. [1] picked up this line of research and provided an extensive analysis of planning under various syntactical restrictions, in particular the syntactical restrictions considered by Bylander [6] and by Bäckström and Nebel [3], leaving open only one stubborn class of problems where operators have no preconditions but may involve up to e postconditions (effects).

New Contributions

We provide a full parameterized complexity analysis of propositional planning without preconditions. In particular, we show the following dichotomy:

(1) Propositional planning where operators have no preconditions but may have up to e postconditions is fixed-parameter tractable for $e \leq 2$ and W[1]-complete for $e > 2$.

W[1] is a parameterized complexity class of problems that are believed to be not fixed-parameter tractable. Indeed, the fixed-parameter tractability of a W[1]-complete problem implies that the Exponential Time Hypothesis fails [7,11]. We establish the hardness part of the dichotomy (1) by a reduction from a variant of the k-Clique problem. The case $e = 2$ is known to be NP-hard [6]. Its difficulty comes from the fact that possibly one of the two postconditions might set a variable to its desired value, but the other postcondition might change a variable from a desired value to an undesired one. This can cause a chain of operators so that finally all variables have their desired value. We show that this behaviour can be modelled by means of a certain problem on Steiner trees in directed graphs, which was recently shown to be fixed-parameter tractable by Guo, Niedermeier and Suchy [15]. We would like to point out that this case (0 preconditions, 2 postconditions) is the only fixed-parameter tractable case among the NP-hard cases in Bylander's system of restrictions (see Table 1).

Our second set of results is concerned with bounds on problem kernels for planning problems. It is known that a decidable problem is fixed-parameter tractable if and only if it admits a polynomial-time self-reduction where the size of the resulting instance is bounded by a function f of the parameter [10,14,12]. The function f is called the *kernel size.* By providing upper and lower bounds on the kernel size, one can rigorously establish the potential of polynomial-time preprocessing for the problem at hand. Some NP-hard combinatorial problems such as

Table 1. Complexity of Bounded Planning, restricting the number of preconditions (p) and effects (e). The problems in FPT do not admit polynomial kernels. Results marked with * are obtained in this paper. All other parameterized results are from [1] and all classical results are from [6].

	$e = 1$	$e = 2$	fixed $e > 2$	arbitrary e
$p = 0$	in P in P	in FPT* NP-C	W[1]-C* NP-C	W[2]-C NP-C
$p = 1$	W[1]-C NP-H	W[1]-C NP-H	W[1]-C NP-H	W[2]-C Pspace-C
fixed $p > 1$	W[1]-C NP-H	W[1]-C Pspace-C	W[1]-C Pspace-C	W[2]-C Pspace-C
arbitrary p	W[1]-C Pspace-C	W[1]-C Pspace-C	W[1]-C Pspace-C	W[2]-C Pspace-C

k-Vertex Cover admit polynomially sized kernels, for others such as k-Path an exponential kernel is the best one can hope for [4]. We examine all planning problems that we have previously been shown to be fixed-parameter tractable on whether they admit polynomial kernels. Our results are negative throughout. In particular, it is unlikely that the FPT part in the above dichotomy (1) can be improved to a polynomial kernel:

(2) Propositional planning where operators have no preconditions but may have up to 2 postconditions does not admit a polynomial kernel unless co-NP $\subseteq$ NP/poly.

Recall that by Yap's Theorem [17] co-NP $\subseteq$ NP/poly implies the (unlikely) collapse of the Polynomial Hierarchy to its third level. We establish the kernel lower bound by means of the technique of *OR-compositions* [4]. We also consider the "PUBS" fragments of planning as introduced by Bäckström and Klein [2]. These fragments arise under combinations of syntactical properties (postunique (P), unary (U), Boolean (B), and single-valued (S); definitions are provided in Section 3).

(3) None of the fixed-parameter tractable but NP-hard PUBS restrictions of propositional planning admits a polynomial kernel, unless co-NP $\subseteq$ NP/poly.

According to the PUBS lattice (see Figure 1), only the two maximal restrictions PUB and PBS need to be considered. Moreover, we observe from previous results that a polynomial kernel for restriction PBS implies one for restriction PUB. Hence this leaves restriction PUB as the only one for which we need to show a super-polynomial kernel bound. We establish the latter, as above, by using OR-compositions.

The full proofs of statements marked with $\star$ are omitted due to space restrictions and can be found at http://arxiv.org/abs/1211.0479.

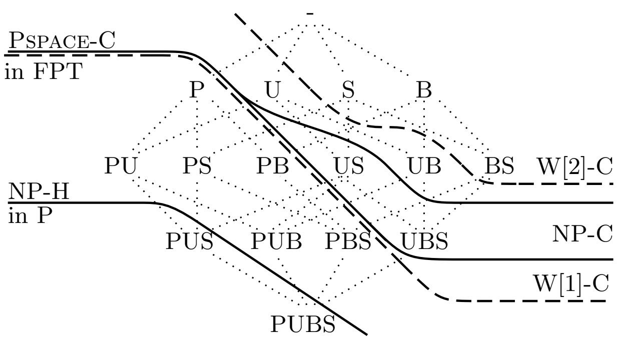


Fig. 1. Complexity of BOUNDED PLANNING for the restrictions P, U, B and S illustrated as a lattice defined by all possible combinations of these restrictions [1]. As shown in this paper, PUS and PUBS are the only restrictions that admit a polynomial kernel, unless the Polynomial Hierarchy collapses.

2 Parameterized Complexity

We define the basic notions of Parameterized Complexity and refer to other sources [9,11] for an in-depth treatment. A *parameterized problem* is a set of pairs $\langle \mathbb{I}, k \rangle$, the *instances*, where $\mathbb{I}$ is the main part and k the *parameter*. The parameter is usually a non-negative integer. A parameterized problem is *fixed-parameter tractable (FPT)* if there exists an algorithm that solves any instance $\langle \mathbb{I}, k \rangle$ of size n in time $f(k)n^c$ where f is an arbitrary computable function and c is a constant independent of both n and k. FPT is the class of all fixed-parameter tractable decision problems.

Parameterized complexity offers a completeness theory, similar to the theory of NP-completeness, that allows the accumulation of strong theoretical evidence that some parameterized problems are not fixed-parameter tractable. This theory is based on a hierarchy of complexity classes FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \cdots$ where all inclusions are believed to be strict. An *fpt-reduction* from a parameterized problem P to a parameterized problem Q if is a mapping R from instances of P to instances of Q such that (i) $\langle \mathbb{I}, k \rangle$ is a YES-instance of P if and only if $\langle \mathbb{I}', k' \rangle = R(\mathbb{I}, k)$ is a YES-instance of Q, (ii) there is a computable function g such that $k' \leq g(k)$, and (iii) there is a computable function f and a constant c such that R can be computed in time $O(f(k) \cdot n^c)$, where n denotes the size of $\langle \mathbb{I}, k \rangle$.

A *kernelization* [11] for a parameterized problem P is an algorithm that takes an instance $\langle \mathbb{I}, k \rangle$ of P and maps it in time polynomial in $|\mathbb{I}| + k$ to an instance $\langle \mathbb{I}', k' \rangle$ of P such that $\langle \mathbb{I}, k \rangle$ is a YES-instance if and only if $\langle \mathbb{I}', k' \rangle$ is a YES-instance and $|\mathbb{I}'|$ is bounded by some function f of k. The output $\mathbb{I}'$ is called a *kernel*. We say P has a *polynomial kernel* if f is a polynomial. Every fixed-parameter tractable problem admits a kernel, but not necessarily a polynomial kernel.

An *OR-composition algorithm* for a parameterized problem P maps t instances $\langle \mathbb{I}_1, k\rangle, \ldots, \langle \mathbb{I}_t, k\rangle$ of P to one instance $\langle \mathbb{I}', k'\rangle$ of P such that the algorithm runs in time polynomial in $\sum_{1 \leq i \leq t} |\mathbb{I}_i| + k$, the parameter k' is bounded by a polynomial in the parameter k, and $\langle \mathbb{I}', k'\rangle$ is a YES-instance if and only if there is an $1 \leq i \leq t$ such that $\langle \mathbb{I}_i, k\rangle$ is a YES-instance.

Proposition 1 (Bodlaender, et al. [4]). *If a parameterized problem P has an OR-composition algorithm, then it has no polynomial kernel unless* co-NP $\subseteq$ NP/poly.

A *polynomial parameter reduction* from a parameterized problem P to a parameterized problem Q is an fpt-reduction R from P to Q such that (i) R can be computed in polynomial time (polynomial in $|\mathbb{I}| + k$), and (ii) there is a polynomial p such that $k' \leq p(k)$ for every instance $\langle \mathbb{I}, k\rangle$ of P with $\langle \mathbb{I}', k'\rangle = R(\langle \mathbb{I}, k\rangle)$. The *unparameterized version* $\tilde{P}$ of a parameterized problem P has the same YES and NO-instances as P, except that the parameter k is given in unary 1^k.

Proposition 2 (Bodlaender, Thomasse, and Yeo [5]). *Let P and Q be two parameterized problems such that there is a polynomial parameter reduction from P to Q, and assume that $\tilde{P}$ is* NP*-complete and $\tilde{Q}$ is in* NP*. Then, if Q has a polynomial kernel also P has a polynomial kernel.*

3 Planning Framework

We will now introduce the SAS$^+$ formalism for specifying propositional planning problems [3]. We note that the propositional STRIPS language can be treated as the special case of SAS$^+$ satisfying restriction B (which will be defined below). More precisely, this corresponds to the variant of STRIPS that allows negative preconditions; this formalism is often referred to as PSN.

Let $V = \{v_1, \ldots, v_n\}$ be a finite set of *variables* over a finite *domain* D. Implicitly define $D^+ = D \cup \{\mathbf{u}\}$, where $\mathbf{u}$ is a special value (the *undefined value*) not present in D. Then D^n is the set of *total states* and $(D^+)^n$ is the set of *partial states* over V and D, where $D^n \subseteq (D^+)^n$. The value of a variable v in a state $s \in (D^+)^n$ is denoted $s[v]$. A *SAS$^+$ instance* is a tuple $\mathbb{P} = \langle V, D, A, I, G\rangle$ where V is a set of variables, D is a domain, A is a set of *actions*, $I \in D^n$ is the *initial state* and $G \in (D^+)^n$ is the *goal*. Each action $a \in A$ has a *precondition* $\mathrm{pre}(a) \in (D^+)^n$ and an *effect* $\mathrm{eff}(a) \in (D^+)^n$. We will frequently use the convention that a variable has value $\mathbf{u}$ in a precondition/effect unless a value is explicitly specified. Let $a \in A$ and let $s \in D^n$. Then a is *valid in s* if for all $v \in V$, either $\mathrm{pre}(a)[v] = s[v]$ or $\mathrm{pre}(a)[v] = \mathbf{u}$. Furthermore, the *result of a in s* is a state $t \in D^n$ defined such that for all $v \in V$, $t[v] = \mathrm{eff}(a)[v]$ if $\mathrm{eff}(a)[v] \neq \mathbf{u}$ and $t[v] = s[v]$ otherwise.

Let $s_0, s_\ell \in D^n$ and let $\omega = \langle a_1, \ldots, a_\ell\rangle$ be a sequence of actions. Then ω is a *plan from s_0 to s_ℓ* if either (i) $\omega = \langle\rangle$ and $\ell = 0$ or (ii) there are states $s_1, \ldots, s_{\ell-1} \in D^n$ such that for all i, where $1 \leq i \leq \ell$, a_i is valid in s_{i-1} and s_i is the result of a_i in s_{i-1}. A state $s \in D^n$ is a *goal state* if for all $v \in V$, either

$G[v] = s[v]$ or $G[v] = \mathbf{u}$. An action sequence ω is a *plan for* $\mathbb{P}$ if it is a plan from I to some goal state $s \in D^n$. We will study the following problem:

> BOUNDED PLANNING
> *Instance:* A tuple $\langle \mathbb{P}, k \rangle$ where $\mathbb{P}$ is a SAS$^+$ instance and k is a positive integer.
> *Parameter:* The integer k.
> *Question:* Does $\mathbb{P}$ have a plan of length at most k?

We will consider the following four syntactical restrictions, originally defined by Bäckström and Klein [2].

- **P** (postunique): For each $v \in V$ and each $x \in D$ there is at most one $a \in A$ such that $\text{eff}(a)[v] = x$.
- **U** (unary): For each $a \in A$, $\text{eff}(a)[v] \neq \mathbf{u}$ for exactly one $v \in V$.
- **B** (Boolean): $|D| = 2$.
- **S** (single-valued): For all $a, b \in A$ and all $v \in V$, if $\text{pre}(a)[v] \neq \mathbf{u}$, $\text{pre}(b)[v] \neq \mathbf{u}$ and $\text{eff}(a)[v] = \text{eff}(b)[v] = \mathbf{u}$, then $\text{pre}(a)[v] = \text{pre}(b)[v]$.

For any set R of such restrictions we write R-BOUNDED PLANNING to denote the restriction of BOUNDED PLANNING to only instances satisfying the restrictions in R. Additionally we will consider restrictions on the number of preconditions and effects as previously considered in [6]. For two non-negative integers p and e we write (p, e)-BOUNDED PLANNING to denote the restriction of BOUNDED PLANNING to only instances where every action has at most p preconditions and at most e effects. Table 1 and Figure 1 summarize results from [6,3,1] combined with the results presented in this paper.

4 Parameterized Complexity of $(0, e)$-Bounded Planning

In this section we completely characterize the parameterized complexity of BOUNDED PLANNING for planning instances without preconditions. It is known [1] that BOUNDED PLANNING without preconditions is contained in the parameterized complexity class W[1]. Here we show that $(0, e)$-BOUNDED PLANNING is also W[1]-hard for every $e > 2$ but it becomes fixed-parameter tractable if $e \leq 2$. Because $(0, 1)$-BOUNDED PLANNING is trivially solvable in polynomial time this completely characterized the parameterized complexity of BOUNDED PLANNING without preconditions.

4.1 Hardness Results

Theorem 1. $(0, 3)$-BOUNDED PLANNING *is* W[1]*-hard.*

Proof. We devise a parameterized reduction from the following problem, which is W[1]-complete [16].

Multicolored Clique
Instance: A k-partite graph $G = (V, E)$ with partition $V_1, \dots, V_k$ such that $|V_i| = |V_j| = n$ for $1 \leq i < j \leq k$.
Parameter: The integer k.
Question: Are there vertices $v_1, \dots, v_k$ such that $v_i \in V_i$ for $1 \leq i \leq k$ and $\{v_i, v_j\} \in E$ for $1 \leq i < j \leq k$? (The graph $K = (\{v_1, \dots, v_k\}, \{\, \{v_i, v_j\} : 1 \leq i < j \leq k \,\})$ is a *k-clique* of G.)

Let $\mathbb{I} = (G, k)$ be an instance of this problem with partition $V_1, \dots, V_k$, $|V_1| = \dots = |V_k| = n$ and parameter k. We construct a $(0, 3)$-Bounded Planning instance $\mathbb{I}' = (\mathbb{P}', k')$ with $\mathbb{P}' = \langle V', D', A', I', G' \rangle$ such that $\mathbb{I}$ is a Yes-instance if and only if so is $\mathbb{I}'$.

We set $V' = V(G) \cup \{\, p_{i,j} : 1 \leq i < j \leq k \,\}$, $D' = \{0, 1\}$, $I' = \langle 0, \dots, 0 \rangle$, $G'[p_{i,j}] = 1$ for every $1 \leq i < j \leq k$ and $G'[v] = 0$ for every $v \in V(G)$. Furthermore, the set A' contains the following actions:

- For every $v \in V(G)$ one action a_v with $\text{eff}(a_v)[v] = 0$;
- For every $e = \{v_i, v_j\} \in E(G)$ with $v_i \in V_i$ and $v_j \in V_j$ one action a_e with $\text{eff}(a_e)[v_i] = 1$, $\text{eff}(a_e)[v_j] = 1$, and $\text{eff}(a_e)[p_{i,j}] = 1$.

Clearly, every action in A' has no precondition and at most 3 effects.

The theorem will follow after we have shown the that G contains a k-clique if and only if $\mathbb{P}$ has a plan of length at most $k' = \binom{k}{2} + k$. Suppose that G contains a k-clique with vertices $v_1, \dots, v_k$ and edges $e_1, \dots, e_{k''}$, $k'' = \binom{k}{2}$. Then $\omega' = \langle a_{e_1}, \dots, a_{e_{k''}}, a_{v_1}, \dots, a_{v_k} \rangle$ is a plan of length k' for $\mathbb{P}'$. For the reverse direction suppose that ω' is a plan of length at most k' for $\mathbb{P}'$. Because $I'[p_{i,j}] = 0 \neq G'[p_{i,j}] = 1$ the plan ω' has to contain at least one action a_e where e is an edge between a vertex in V_i and a vertex in V_j for every $1 \leq i < j \leq k$. Because $\text{eff}(a_{e=\{v_i,v_j\}})[v_i] = 1 \neq G[v_i] = 0$ and $\text{eff}(a_{e=\{v_i,v_j\}})[v_j] = 1 \neq G[v_j] = 0$ for every such edge e it follows that ω' has to contain at least one action a_v with $v \in V_i$ for every $1 \leq i \leq k$. Because $k' = \binom{k}{2} + k$ it follows that ω' contains exactly $\binom{k}{2}$ actions of the form a_e for some edge $e \in E(G)$ and exactly k actions of the form a_v for some vertex $v \in V(G)$. It follows that the graph $K = (\{\, v : a_v \in \omega \,\}, \{\, e : a_e \in \omega \,\})$ is a k-clique of G. □

4.2 Fixed-Parameter Tractability

Before we show that $(0, 2)$-Bounded Planning is fixed-parameter tractable we need to introduce some notions and prove some simple properties of $(0, 2)$-Bounded Planning. Let $\mathbb{P} = \langle V, D, A, I, G \rangle$ be an instance of Bounded Planning. We say an action $a \in A$ has an effect on some variable $v \in V$ if $\text{eff}(a)[v] \neq \mathbf{u}$, we call this effect *good* if furthermore $\text{eff}(a)[v] = G[v]$ or $G[v] = \mathbf{u}$ and we call the effect *bad* otherwise. We say an action $a \in A$ is *good* if it has only good effects, *bad* if it has only bad effects, and *mixed* if it has at least one good and at least one bad effect. Note that if a valid plan contains a bad action then this action can always be removed without changing the validity of the plan.

Consequently, we only need to consider good and mixed actions. Furthermore, we denote by $B(V)$ the set of variables $v \in V$ with $G[v] \neq \mathbf{u}$ and $I[v] \neq G[v]$.

The next lemma shows that we do not need to consider good actions with more than 1 effect for $(0,2)$-Bounded Planning.

Lemma 1 ($\star$). *Let $\mathbb{I} = \langle \mathbb{P}, k \rangle$ be an instance of* $(0,2)$-Bounded Planning. *Then $\mathbb{I}$ can be fpt-reduced to an instance $\mathbb{I}' = \langle \mathbb{P}', k' \rangle$ of* $(0,2)$-Bounded Planning *where $k' = k(k+3)+1$ and no good action of $\mathbb{I}'$ effects more than one variable.*

Theorem 2. $(0,2)$-Bounded Planning *is fixed-parameter tractable.*

Proof. We show fixed-parameter tractability of $(0,2)$-Bounded Planning by reducing it to the following fixed-parameter tractable problem [15].

> Directed Steiner Tree
> *Instance:* A set of nodes N, a weight function $w : N \times N \rightarrow (\mathbb{N} \cup \{\infty\})$, a root node $s \in N$, a set $T \subseteq N$ of terminals , and a weight bound p.
> *Parameter:* $p_M = \frac{p}{\min\{\, w(u,v) \,:\, u,v \in N\,\}}$.
> *Question:* Is there a set of arcs $E \subseteq N \times N$ of weight $w(E) \leq p$ (where $w(E) = \sum_{e \in E} w(e)$) such that in the digraph $D = (N, E)$ for every $t \in T$ there is a directed path from s to t? We will call the digraph D a *directed Steiner Tree (DST)* of weight $w(E)$.

Let $\mathbb{I} = \langle \mathbb{P}, k \rangle$ where $\mathbb{P} = \langle V, D, A, I, G \rangle$ be an instance of $(0,2)$-Bounded Planning. Because of Lemma 1 we can assume that A contains no good actions with two effects. We construct an instance $\mathbb{I}' = \langle N, w, s, T, p \rangle$ of Directed Steiner Tree where $p_M = k$ such that $\mathbb{I}$ is a Yes-instance if and only if $\mathbb{I}'$ is a Yes-instance. Because $p_M = k$ this shows that $(0,2)$-Bounded Planning is fixed-parameter tractable.

We are now ready to define the instance $\mathbb{I}'$. The node set N consists of the root vertex s and one node for every variable in V. The weight function w is ∞ for all but the following arcs: (i) For every good action $a \in A$ the arc from s to the unique variable $v \in V$ that is effected by a gets weight 1. (ii) For every mixed action $a \in A$ with some good effect on some variable $v_g \in V$ and some bad effect on some variable $v_b \in V$, the arc from v_b to v_g gets weight 1.

We identify the root s from the instance $\mathbb{I}$ with the node s, we let T be the set $B(V)$, and $p_M = p = k$.

Claim 1 ($\star$). *$\mathbb{P}$ has a plan of length at most k if and only if $\mathbb{I}'$ has a DST of weight at most $p_M = p = k$.*

The theorem follows. □

5 Kernel Lower Bounds

Since $(0,2)$-Bounded Planning is fixed-parameter tractable by Theorem 2 it admits a kernel. Next we provide strong theoretical evidence that the problem does not admit a polynomial kernel. The proof of Theorem 3 is based on an OR-composition algorithm and Proposition 1.

Theorem 3 ($\star$). $(0,2)$-Bounded Planning *has no polynomial kernel unless* co-NP $\subseteq$ NP/poly.

In previous work [1] we have classified the parameterized complexity of the "PUBS" fragments of Bounded Planning. It turned out that the problems fall into four categories (see Figure 1): (i) polynomial-time solvable, (ii) NP-hard but fixed-parameter tractable, (iii) W[1]-complete, and (iv) W[2]-complete. The aim of this section is to further refine this classification with respect to kernelization. The problems in category (i) trivially admit a kernel of constant size, whereas the problems in categories (iii) and (iv) do not admit a kernel at all (polynomial or not), unless W[1] = FPT or W[2] = FPT, respectively. Hence it remains to consider the six problems in category (ii), each of them could either admit a polynomial kernel or not. We show that none of them does.

According to our classification [1], the problems in category (ii) are exactly the problems R-Bounded Planning, for $R \subseteq \{P, U, B, S\}$, such that $P \in R$ and $\{P, U, S\} \not\subseteq R$.

Theorem 4. *None of the problems R-*Bounded Planning *for $R \subseteq \{P, U, B, S\}$ such that $P \in R$ and $\{P, U, S\} \not\subseteq R$ (i.e., the problems in category (ii)) admits a polynomial kernel unless* co-NP $\subseteq$ NP/poly.

The remainder of this section is devoted to establish Theorem 4. The relationship between the problems as indicated in Figure 1 greatly simplifies the proof. Instead of considering all six problems separately, we can focus on the two most restricted problems $\{P, U, B\}$-Bounded Planning and $\{P, B, S\}$-Bounded Planning. If any other problem in category (ii) would have a polynomial kernel, then at least one of these two problems would have one. This follows by Proposition 2 and the following facts:

1. The unparameterized versions of all the problems in category (ii) are NP-complete. This holds since the corresponding classical problems are strongly NP-hard, hence the problems remain NP-hard when k is encoded in unary (as shown by Bäckström and Nebel [3]);
2. If $R_1 \subseteq R_2$ then the identity function gives a polynomial parameter reduction from R_2-Bounded Planning to R_1-Bounded Planning.

Furthermore, the following result of Bäckström and Nebel [3, Theorem 4.16] even provides a polynomial parameter reduction from $\{P, U, B\}$-Bounded Planning to $\{P, B, S\}$-Bounded Planning. Consequently, $\{P, U, B\}$-Bounded Planning remains the only problem for which we need to establish a superpolynomial kernel lower bound.

Proposition 3 (Bäckström and Nebel [3]). *Let $\mathbb{I} = \langle \mathbb{P}, k \rangle$ be an instance of $\{P, U, B\}$-*Bounded Planning*. Then $\mathbb{I}$ can be transformed in polynomial time into an equivalent instance $\mathbb{I}' = \langle \mathbb{P}', k' \rangle$ of $\{P, B, S\}$-*Bounded Planning *such that $k = k'$.*

Hence, in order to complete the proof of Theorem 4 it only remains to establish the next lemma.

Lemma 2. $\{P, U, B\}$-Bounded Planning *has no polynomial kernel unless* co-NP $\subseteq$ NP/poly.

Proof. Because of Proposition 1, it suffices to devise an OR-composition algorithm for $\{P, U, B\}$-Bounded Planning. Suppose we are given t instances $\mathbb{I}_1 = \langle \mathbb{P}_1, k \rangle, \dots, \mathbb{I}_t = \langle \mathbb{P}_t, k \rangle$ of $\{P, U, B\}$-Bounded Planning where $\mathbb{P}_i = \langle V_i, D_i, A_i, I_i, G_i \rangle$ for every $1 \leq i \leq t$. It has been shown in [1, Theorem 5] that $\{P, U, B\}$-Bounded Planning can be solved in time $O^*(S(k))$ (where $S(k) = 2 \cdot 2^{(k+2)^2} \cdot (k+2)^{(k+1)^2}$ and the O^* notation suppresses polynomial factors). It follows that $\{P, U, B\}$-Bounded Planning can be solved in polynomial time with respect to $\sum_{1 \leq i \leq t} |\mathbb{I}_i| + k$ if $t > S(k)$. Hence, if $t > S(k)$ this gives us an OR-composition algorithm as follows. We first run the algorithm for $\{P, U, B\}$-Bounded Planning on each of the t instances. If one of these t instances is a Yes-instance then we output this instance. If not then we output any of the t instances. This shows that $\{P, U, B\}$-Bounded Planning has an OR-composition algorithm for the case that $t > S(k)$. Hence, in the following we can assume that $t \leq S(k)$.

Given $\mathbb{I}_1, \dots, \mathbb{I}_t$ we will construct an instance $\mathbb{I} = \langle \mathbb{P}, k' \rangle$ of $\{P, U, B\}$-Bounded Planning as follows. For the construction of $\mathbb{I}$ we need the following auxiliary gadget, which will be used to calculate the logical "OR" of two binary variables. The construction of the gadget uses ideas from [3, Theorem 4.15]. Assume that v_1 and v_2 are two binary variables. The gadget $\mathrm{OR}_2(v_1, v_2, o)$ consists of the five binary variables o_1, o_2, o, i_1, and i_2. Furthermore, $\mathrm{OR}_2(v_1, v_2, o)$ contains the following actions:

- the action a_o with $\mathrm{pre}(a_o)[o_1] = \mathrm{pre}(a_o)[o_2] = 1$ and $\mathrm{eff}(a_o)[o] = 1$;
- the action a_{o_1} with $\mathrm{pre}(a_{o_1})[i_1] = 1$, $\mathrm{pre}(a_{o_1})[i_2] = 0$ and $\mathrm{eff}(a_{o_1})[o_1] = 1$;
- the action a_{o_2} with $\mathrm{pre}(a_{o_2})[i_1] = 0$, $\mathrm{pre}(a_{o_2})[i_2] = 1$ and $\mathrm{eff}(a_{o_2})[o_2] = 1$;
- the action a_{i_1} with $\mathrm{eff}(a_{i_1})[i_1] = 1$;
- the action a_{i_2} with $\mathrm{eff}(a_{i_2})[i_2] = 1$;
- the action a_{v_1} with $\mathrm{pre}(a_{v_1})[v_1] = 1$ and $\mathrm{eff}(a_{v_1})[i_1] = 0$;
- the action a_{v_2} with $\mathrm{pre}(a_{v_2})[v_2] = 1$ and $\mathrm{eff}(a_{v_2})[i_2] = 0$;

We now show that $\mathrm{OR}_2(v_1, v_2, o)$ can indeed be used to compute the logical "OR" of the variables v_1 and v_2. We need the following claim.

Claim 2 ($\star$). *Let* $\mathbb{P}(\mathrm{OR}_2(v_1, v_2, o))$ *be a* $\{P, U, B\}$-Bounded Planning *instance that consists of the two binary variables v_1 and v_2, and the variables and actions of the gadget* $\mathrm{OR}_2(v_1, v_2, o)$. *Furthermore, let the initial state of* $\mathbb{P}(\mathrm{OR}_2(v_1, v_2, o))$ *be any initial state that sets all variables of the gadget* $\mathrm{OR}_2(v_1, v_2, o)$ *to* 0 *but assigns the variables v_1 and v_2 arbitrarily, and let the goal state of* $\mathbb{P}(\mathrm{OR}_2(v_1, v_2, o))$ *be defined by $G[o] = 1$. Then* $\mathbb{P}(\mathrm{OR}_2(v_1, v_2, o))$ *has a plan if and only if its initial state sets at least one of the variables v_1 or v_2 to* 1. *Furthermore, if there is such a plan then its length is* 6.

We continue by showing how we can use the gadget $\mathrm{OR}_2(v_1, v_2, o)$ to construct a gadget $\mathrm{OR}(v_1, \dots, v_r, o)$ such that there is a sequence of actions of $\mathrm{OR}(v_1, \dots, v_r, o)$ that sets the variable o to 1 if and only if at least one of the external variables $v_1, \dots, v_r$ are initially set to 1. Furthermore, if there is such a sequence of actions then its length is at most $6\lceil \log r \rceil$. Let T be a rooted binary tree with root s that has r leaves $l_1, \dots, l_r$ and is of smallest possible height. For every node $t \in V(T)$ we make a copy of our binary OR-gadget such that the copy of a leave node l_i is the gadget $\mathrm{OR}_2(v_{2i-1}, v_{2i}, o_{l_i})$ and the copy of an inner node $t \in V(T)$ with children t_1 and t_2 is the gadget $\mathrm{OR}_2(o_{t_1}, o_{t_2}, o_t)$ (clearly this needs to be adapted if r is odd or an inner node has only one child). For the root node with children t_1 and t_2 the gadget becomes $\mathrm{OR}_2(o_{t_1}, o_{t_2}, o)$. This completes the construction of the gadget $\mathrm{OR}(v_1, \dots, v_r, o)$. Using Claim 2 it is easy to verify that the gadget $\mathrm{OR}(v_1, \dots, v_r, o)$ can indeed be used to compute the logical "OR" or the variables $v_1, \dots, v_r$.

We are now ready to construct the instance $\mathbb{I}$. $\mathbb{I}$ contains all the variables and actions from every instance $\mathbb{I}_1, \dots, \mathbb{I}_t$ and of the gadget $\mathrm{OR}(v_1, \dots, v_t, o)$. Additionally, $\mathbb{I}$ contains the binary variables $v_1, \dots, v_t$ and the actions $a_1, \dots, a_t$ with $\mathrm{pre}(a_i) = G_i$ and $\mathrm{eff}(a_i)[v_i] = 1$. Furthermore, the initial state I of $\mathbb{I}$ is defined as $I[v] = I_i[v]$ if v is a variable of $\mathbb{I}_i$ and $I[v] = 0$, otherwise. The goal state of $\mathbb{I}$ is defined by $G[o] = 1$ and we set $k' = k + 6\lceil \log t \rceil$. Clearly, $\mathbb{I}$ can be constructed from $\mathbb{I}_1, \dots, \mathbb{I}_t$ in polynomial time and $\mathbb{I}$ is a YES-instance if and only if at least one of the instances $\mathbb{I}_1, \dots, \mathbb{I}_t$ is a YES-instance. Furthermore, because $k' = k + 6\lceil \log t \rceil \leq k + 6\lceil \log S(k) \rceil = k + 6\lceil 1 + (k+2)^2 + (k+1)^2 \cdot \log(k+2) \rceil$, the parameter k' is polynomial bounded by the parameter k. This concludes the proof of the lemma. $\square$

6 Conclusion

We have studied the parameterized complexity of BOUNDED PLANNING with respect to the parameter plan length. In particular, we have shown that $(0, e)$-BOUNDED PLANNING is fixed-parameter tractable for $e \leq 2$ and W[1]-complete for $e > 2$. Together with our previous results [1] this completes the full classification of planning in Bylander's system of restrictions (see Table 1). Interestingly, $(0, 2)$-BOUNDED PLANNING turns out to be the only nontrivial fixed-parameter tractable case (where the unparameterized version is NP-hard).

We have also provided a full classification of kernel sizes for $(0, 2)$-BOUNDED PLANNING and all the fixed-parameter tractable fragments of BOUNDED PLANNING in the "PUBS" framework. It turns out that none of the nontrivial problems (where the unparameterized version is NP-hard) admits a polynomial kernel unless the Polynomial Hierarchy collapses. This implies an interesting *dichotomy* concerning the kernel size: we only have constant-size and superpolynomial kernels—polynomially bounded kernels that are not of constant size are absent.

References

1. Bäckström, C., Chen, Y., Jonsson, P., Ordyniak, S., Szeider, S.: The complexity of planning revisited - a parameterized analysis. In: Hoffmann, J., Selman, B. (eds.) Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, Toronto, Ontario, Canada, July 22-26. AAAI Press (2012)

2. Bäckström, C., Klein, I.: Planning in polynomial time: the SAS-PUBS class. Comput. Intelligence 7, 181–197 (1991)
3. Bäckström, C., Nebel, B.: Complexity results for SAS+ planning. Comput. Intelligence 11, 625–656 (1995)
4. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. J. of Computer and System Sciences 75(8), 423–434 (2009)
5. Bodlaender, H.L., Thomassé, S., Yeo, A.: Kernel bounds for disjoint cycles and disjoint paths. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 635–646. Springer, Heidelberg (2009)
6. Bylander, T.: The computational complexity of propositional STRIPS planning. Artificial Intelligence 69(1-2), 165–204 (1994)
7. Chen, J., Huang, X., Kanj, I.A., Xia, G.: Strong computational lower bounds via parameterized complexity. J. of Computer and System Sciences 72(8), 1346–1367 (2006)
8. Downey, R., Fellows, M.R., Stege, U.: Parameterized complexity: A framework for systematically confronting computational intractability. In: Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future. AMS-DIMACS, vol. 49, pp. 49–99. American Mathematical Society (1999)
9. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science. Springer, New York (1999)
10. Fellows, M.R.: The lost continent of polynomial time: Preprocessing and kernelization. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 276–277. Springer, Heidelberg (2006)
11. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science. An EATCS Series, vol. XIV. Springer (2006)
12. Fomin, F.V.: Kernelization. In: Ablayev, F., Mayr, E.W. (eds.) CSR 2010. LNCS, vol. 6072, pp. 107–108. Springer, Heidelberg (2010)
13. Ghallab, M., Nau, D.S., Traverso, P.: Automated planning - theory and practice. Elsevier (2004)
14. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. ACM SIGACT News 38(2), 31–45 (2007)
15. Guo, J., Niedermeier, R., Suchý, O.: Parameterized complexity of arc-weighted directed steiner problems. SIAM J. Discrete Math. 25(2), 583–599 (2011)
16. Pietrzak, K.: On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. J. of Computer and System Sciences 67(4), 757–771 (2003)
17. Yap, C.-K.: Some consequences of nonuniform conditions on uniform classes. Theoretical Computer Science 26(3), 287–300 (1983)

Selfish Resource Allocation in Optical Networks[⋆]

Evangelos Bampas[1], Aris Pagourtzis[1], George Pierrakos[2], and Vasilis Syrgkanis[3]

[1] School of Elec. & Comp. Engineering, National Technical University of Athens
{ebamp,pagour}@cs.ntua.gr

[2] School of Electrical Engineering & Computer Sciences, UC Berkeley
georgios@cs.berkeley.edu

[3] Computer Science Dept., Cornell University
vasilis@cs.cornell.edu

Abstract. We introduce Colored Resource Allocation Games as a new model for selfish routing and wavelength assignment in multifiber all-optical networks. Colored Resource Allocation Games are a generalization of congestion and bottleneck games where players have their strategies in multiple copies (colors). We focus on two main subclasses of these games depending on the player cost: in Colored Congestion Games the player cost is the sum of latencies of the resources allocated to the player, while in Colored Bottleneck Games the player cost is the maximum of these latencies. We investigate the pure price of anarchy for three different social cost functions and prove tight bounds for each separate case. We first consider a social cost function which is particularly meaningful in the setting of multifiber all-optical networks, where it captures the objective of fiber cost minimization. Additionally, we consider the two usual social cost functions (maximum and average player cost) and obtain improved bounds that could not have been derived using earlier results for the standard models for congestion and bottleneck games.

1 Introduction

Potential games are a widely used tool for modeling network optimization problems under a non-cooperative perspective. Initially studied in [1] with the introduction of congestion games and further extended in [2] in a more general framework, they have been successfully applied to describe selfish routing in communication networks (e.g. [3]). The advent of optical networks as the technology of choice for surface communication has introduced new aspects of networks that are not sufficiently captured by the models proposed so far. In this work, we propose a class of potential games which are more suitable for modeling selfish routing and wavelength assignment in multifiber optical networks.

⋆ This research has been co-financed by the European Union (European Social Fund–ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) — Research Funding Program: THALIS Investing in knowledge society through the European Social Fund.

In optical networks, it is highly desirable that all communication should be carried out *transparently*, that is, each signal should remain on the same wavelength from source to destination. The need for efficient access to the optical bandwidth has given rise to the study of several optimization problems in the past years. The most well-studied among them is the problem of assigning a path and a color (wavelength) to each communication request in such a way that paths of the same color are edge-disjoint and the number of colors used is minimized. Nonetheless, it has become clear that the number of wavelengths in commercially available fibers is rather limited—and will probably remain such in the foreseeable future. Therefore, the use of multiple fibers has become inevitable in large scale networks. In the context of multifiber optical networks several optimization problems have been defined and studied, the objective usually being to minimize either the maximum fiber multiplicity per edge or the sum of these maximum multiplicities over all edges of the graph.

1.1 Contribution

We introduce Colored Resource Allocation Games, a class of games that can model non-cooperative versions of routing and wavelength assignment problems in multifiber all-optical networks. They can be viewed as an extension of congestion games where each player has his strategies in multiple copies (colors). When restricted to (optical) network games, facilities correspond to edges of the network and colors to wavelengths. The number of players using an edge in the same color represents a lower bound on the number of fibers needed to implement the corresponding physical link. Having this motivation in mind, we consider the case in which each player's cost is equal to the *maximum* edge congestion encountered on her path (*max* player cost), as well as the case in which each player's cost is equal to the *sum* of edge congestions encountered on her path (*sum* player cost). For our purposes of using Colored Resource Allocation games to model resource allocation in optical networks, it makes sense to restrict our study to the class of identity latency functions.

We use the price of anarchy (PoA) introduced in [4] as a measure of the deterioration of the quality of solutions caused by the lack of coordination. We estimate the price of anarchy of our games under three different social cost functions. The first one (SC_{fib}) is specially designed for the setting of multifiber all-optical networks: it is equal to the sum over all facilities of the maximum color congestion on each facility. Note that in the optical network setting this function represents the total fiber cost needed to accommodate all players; hence, it captures the objective of a well-studied optimization problem ([5–8]). The other two social cost functions are standard in the literature (see e.g. [9]): the first ($SC_{\max}$) is equal to the maximum player cost and the second (SC_{sum}) is equal to the sum of player costs (equivalently, the average player cost).

Let us also note that the $SC_{\max}$ function under the *max* player cost captures the objective of another well known problem, namely minimizing the maximum fiber multiplicity over all edges of the network [7, 10, 11]. In addition, note that our model admits a number of different interpretations as discussed in [12].

Table 1. The pure price of anarchy of Colored Bottleneck Games (*max* player cost) under different social costs. Results for classical bottleneck games are shown in the right column.

	Colored Bottleneck Games	Bottleneck Games
$SC_{\text{fib}}(A) = \sum_{f \in F} \max_{a \in [W]} n_{f,a}(A)$	$\frac{\lvert E_A \rvert}{\lvert E_{\text{OPT}} \rvert} \left\lceil \frac{N}{W} \right\rceil$	—
$SC_{\max}(A) = \max_{i \in [N]} C_i(A)$	$\Theta\left(\frac{N}{W}\right)$	$\Theta(N)$ [13]
$SC_{\text{sum}}(A) = \sum_{i \in [N]} C_i(A)$	$\Theta\left(\frac{N}{W}\right)$	$\Theta(N)$ [13]

Table 2. The pure price of anarchy of Colored Congestion Games (*sum* player cost) under different social costs. Results for classical congestion games are shown in the right column.

	Colored Congestion Games	Congestion Games
$SC_{\text{fib}}(A) = \sum_{f \in F} \max_{a \in [W]} n_{f,a}(A)$	$\Theta\left(\sqrt{W \lvert F \rvert}\right)$	—
$SC_{\max}(A) = \max_{i \in [N]} C_i(A)$	$\Theta\left(\sqrt{\frac{N}{W}}\right)$	$\Theta\left(\sqrt{N}\right)$ [9]
$SC_{\text{sum}}(A) = \sum_{i \in [N]} C_i(A)$	$\frac{5}{2}$	$\frac{5}{2}$ [9]

Our main contribution is the derivation of tight bounds on the price of anarchy for Colored Resource Allocation Games. These bounds are summarized in Tables 1 and 2. It can be shown that the bounds for Colored Congestion Games remain tight even for network games.

Observe that known bounds for classical congestion and bottleneck games can be obtained from our results by simply setting $W = 1$. On the other hand, one might notice that our games can be casted as classical congestion or bottleneck games with $W \lvert F \rvert$ facilities. However we are able to derive better upper bounds for most cases by exploiting the special structure of the players' strategies.

1.2 Related Work

One of the most important solution concepts in the theory of non-cooperative games is the *Nash equilibrium* [14], a stable state of the game in which no player has incentive to change strategy unilaterally. A fundamental question in this theory concerns the existence of *pure* Nash equilibria. For congestion and bottleneck games [1, 2, 15] it has been shown with the use of potential functions that they converge to a pure Nash equilibrium.

In [16] Roughgarden introduces a canonical framework for studying the price of anarchy; in particular he identifies the following canonical sufficient condition, which he calls the "smoothness condition":

$$\sum_{i=1}^{n} C_i(A_i^*, A_{-i}) \leq \lambda SC(A^*) + \mu SC(A) \ .$$

The key idea is that, by showing that a game is (λ, μ)-smooth, i.e. that it satisfies the condition above for some choice of λ and μ, we immediately get an upper bound of $\frac{\lambda}{1-\mu}$ on the price of anarchy of the game. Hence, bounding the price of anarchy reduces to the problem of identifying λ and μ which minimize the aforementioned quantity, and for which the game is (λ, μ)-smooth. From the games and welfare functions that we analyze only colored congestion games from the perpsective of SC_{sum} are smooth, a property implied by the existing analysis of Christodoulou et al [9] and which we show remains tight even in our setting. On the contrary, our other two social cost functions and our bottleneck game analysis do not seem to admit a similar smoothness argument, and therefore a different approach is required in order to upper bound the price of anarchy for these settings.

Bottleneck games have been studied in [13, 15, 17, 18]. In [13] the authors study atomic routing games on networks, where each player chooses a path to route her traffic from an origin to a destination node, with the objective of minimizing the maximum congestion on any edge of her path. They show that these games always possess at least one optimal pure Nash equilibrium (hence the price of stability is equal to 1) and that the price of anarchy of the game is determined by topological properties of the network. A further generalization is the model of Banner and Orda [15], where they introduce the notion of bottleneck games. In this model they allow arbitrary latency functions on the edges and consider both splittable and unsplittable flows. They show existence, convergence and non-uniqueness of equilibria and they prove that the price of anarchy for these games is exponential in the users' demand.

Since bottleneck games traditionally have price of anarchy that is rather high (proportional to the size of the network in many cases), in [19] the authors study bottleneck games when the utility functions of the players are exponential functions of their congestion, and they show that for this class of *exponential bottleneck games* the price of anarchy is in fact logarithmic. Finally [20] investigate the computational problem of finding a pure Nash equilibrium in bottleneck games, as well as the performance of some natural (de-centralized) improvement dynamics for finding pure Nash equilibria.

Selfish path coloring in single fiber all-optical networks has been studied in [21–24]. Bilò and Moscardelli [21] consider the convergence to Nash equilibria of selfish routing and path coloring games. Bilò et al. [22] consider several information levels of local knowledge that players may have and give bounds for the price of anarchy in chains, rings and trees. The existence of Nash equilibria and the complexity of recognizing and computing a Nash equilibrium for selfish routing and path coloring games under several payment functions are considered by Georgakopoulos et al. [23]. In [24] upper and lower bounds for the price of anarchy of selfish path coloring with and without routing are presented under functions that charge a player only according to her own strategy.

Selfish path multicoloring games are introduced in [12] where it is proved that the pure price of anarchy is bounded by the number of available colors and by the length of the longest path; constant bounds for the price of anarchy in specific topologies are also provided. In those games, in contrast to the ones studied here, routing is given in advance and players choose only colors.

2 Model Definition

We use the notation $[X]$ for the set $\{1, \ldots, X\}$, where X is a positive natural number.

Definition 1 (Colored Resource Allocation Games). *A Colored Resource Allocation Game is defined as a tuple $\langle F, N, W, \{\mathcal{E}_i\}_{i\in[N]}\rangle$ such that:*

1. *F is a set of facilities f_i.*
2. *$[W]$ is a set of colors.*
3. *$[N]$ is a set of players.*
4. *$\mathcal{E}_i$ is a set of possible facility combinations for player i such that:*
   a. *$\forall\ i \in [N] : \mathcal{E}_i \subseteq 2^F$,*
   b. *$S_i = \mathcal{E}_i \times [W]$ is the set of possible strategies of player i, and*
   c. *$A_i = (E_i, a_i) \in S_i$ is the notation of a strategy for player i, where $E_i \in \mathcal{E}_i$ denotes the set of facilities and $a_i \in [W]$ denotes the color chosen by the player.*
5. *$A = (A_1, \ldots, A_N)$ is a strategy profile for the game.*
6. *For a strategy profile A, $\forall f \in F$, $\forall c \in [W]$, $n_{f,c}(A)$ is the number of players that use facility f in color c in strategy profile A.*

Depending on the player cost function we define two subclasses of Colored Resource Allocation Games:

– *Colored Bottleneck Games (CBG), where the player cost is*

$$C_i(A) = \max_{e\in E_i} n_{e,a_i}(A) \ .$$

– *Colored Congestion Games (CCG), where the player cost is*

$$C_i(A) = \sum_{e\in E_i} n_{e,a_i}(A) \ .$$

For each of the above variations we will consider three different social cost functions:

– $SC_{\text{fib}}(A) = \sum_{f\in F} \max_{c\in[W]} n_{f,c}(A)$.
– $SC_{\max}(A) = \max_{i\in[N]} C_i(A)$.
– $SC_{\text{sum}}(A) = \sum_{i\in[N]} C_i(A)$. Note that, in the case of CCG games, the sum social cost can also be expressed as $SC_{\text{sum}}(A) = \sum_{f\in F} \sum_{c\in[W]} n^2_{f,c}(A)$.

From the definition of pure Nash equilibrium we can derive the following two facts that hold in Colored Congestion and Bottleneck Games respectively:

Fact 1. *For a pure Nash equilibrium A of a CCG game it holds:*

$$\forall E'_i \in \mathcal{E}_i, \forall c' \in [W] : C_i(A) \leq \sum_{e \in E'_i} (n_{e,c'}(A) + 1) \ . \tag{1}$$

Fact 2. *For a pure Nash equilibrium A of a CBG game it holds:*

$$\forall E'_i \in \mathcal{E}_i, \forall c' \in [W] : C_i(A) \leq \max_{e \in E'_i}(n_{e,c'}(A) + 1) \ . \tag{2}$$

Equivalently:

$$\forall E_i \in \mathcal{E}_i, \forall c \in [W], \exists e \in E_i : C_i(A) \leq n_{e,c}(A) + 1 \ . \tag{3}$$

In the rest of the paper, we will only deal with pure Nash equilibria and we will refer to them simply as Nash equilibria.

3 Colored Bottleneck Games

By a standard lexicographic argument, one can show that every CBG game has at least one pure Nash equilibrium and that the price of stability [25] is 1.

3.1 Price of Anarchy for Social Cost SC_{fib}

Definition 2. *We define E_S to be the set of facilities used by at least one player in the strategy profile $S = (A_1, \dots, A_N)$, i.e., $E_S = E_1 \cup \dots \cup E_N$.*

Theorem 1. *The price of anarchy of any CBG game with social cost SC_{fib} is at most $\frac{|E_A|}{|E_{\text{OPT}}|} \lceil \frac{N}{W} \rceil$, where A is a worst-case Nash equilibrium and* OPT *is an optimal strategy profile.*

Proof. We exclude from the sum over the facilities, those facilities that are not used by any player since they do not contribute to the social cost. Thus we focus on facilities with $\max_c n_{e,c} > 0$. Let A be a worst-case Nash equilibrium and let $c_{\max}(e)$ denote the color with the maximum multiplicity at facility e. Let P_i be a player that uses the facility copy $(e, c_{\max}(e))$. Since $C_i(A) = \max_{e \in E_i} n_{e,a_i}(A)$ it must hold that $n_{e,c_{\max}(e)}(A) \leq C_i(A)$. In fact, we can state the following general property:

$$\forall e \in F, \ \exists i \in [N] : n_{e,c_{\max}(e)} \leq C_i(A) \ . \tag{4}$$

Suppose that there exists a player with cost $\lceil \frac{N}{W} \rceil + 1$ or more. From Fact 2, at least $\lceil \frac{N}{W} \rceil$ players must play each of the other colors. By a simple calculation, this implies that there are at least $N + 1$ players in the game, a contradiction.

We conclude that each player's cost is at most $\lceil \frac{N}{W} \rceil$, thus $C_i(A) \leq \lceil \frac{N}{W} \rceil$. Moreover, it is easy to see that $SC_{\text{fib}}(\text{OPT}) \geq |E_{\text{OPT}}|$. From the above we can conclude:

$$\frac{SC_{\text{fib}}(A)}{SC_{\text{fib}}(\text{OPT})} \leq \frac{|E_A|}{|E_{\text{OPT}}|} \left\lceil \frac{N}{W} \right\rceil \ . \tag{5}$$

□

Theorem 2. *There exists a class of CBG games with social cost SC_{fib} with* $\text{PoA} = \frac{|E_A|}{|E_{\text{OPT}}|} \lceil \frac{N}{W} \rceil$.

Proof. Consider a game in which each player i has the following strategy set: $\mathcal{E}_i = \{\{f_i\}, \{f_1, \ldots, f_M\}\}$, where $M \geq N \geq W$. In the worst-case Nash equilibrium A, all players will play the second strategy leading to $SC_{\text{fib}}(A) = M \lceil \frac{N}{W} \rceil = |E_A| \lceil \frac{N}{W} \rceil$. On the other hand in the optimal outcome all players will play the first strategy leading to $SC_{\text{fib}}(\text{OPT}) = N = |E_{\text{OPT}}|$. Thus the price of anarchy for this instance is $\text{PoA} = \frac{|E_A|}{|E_{\text{OPT}}|} \lceil \frac{N}{W} \rceil$. □

3.2 Price of Anarchy for Social Cost $SC_{\max}$

Theorem 3. *The price of anarchy of any CBG game with social cost $SC_{\max}$ is at most $\lceil \frac{N}{W} \rceil$.*

Proof. It is easy to see that $SC_{\max}(\text{OPT}) \geq 1$. We established in the proof of Theorem 1 that the maximum player cost in a Nash equilibrium is $\lceil \frac{N}{W} \rceil$. Therefore, for any worst-case Nash equilibrium A, $SC_{\max}(A) \leq \lceil \frac{N}{W} \rceil$. □

Theorem 4. *There exists a class of CBG games with social cost $SC_{\max}$ with* $\text{PoA} = \lceil \frac{N}{W} \rceil$.

Proof. Consider the following class of CBG games. We have N players and N facilities. Each player P_i has two possible strategies: $\mathcal{E}_i = \{\{f_i\}, \{f_1, \ldots, f_N\}\}$. In a worst-case Nash equilibrium, all players choose the second strategy and they are equally divided in the W colors. This leads to player cost $\lceil \frac{N}{W} \rceil$ for each player and thus to a social cost $\lceil \frac{N}{W} \rceil$. In the optimal strategy profile, all players would choose their first strategy leading to player and social cost equal to 1. Thus the price of anarchy for this instance is $\lceil \frac{N}{W} \rceil$. □

3.3 Price of Anarchy for Social Cost SC_{sum}

Theorem 5. *The price of anarchy of any CBG game with social cost SC_{sum} is at most $\lceil \frac{N}{W} \rceil$.*

Proof. As before, we know that the maximum player cost in a Nash equilibrium is $\lceil \frac{N}{W} \rceil$, therefore the social cost is at most $N \cdot \lceil \frac{N}{W} \rceil$. Moreover, $SC_{\text{sum}}(\text{OPT}) \geq N$. Thus the price of anarchy is bounded by $\lceil \frac{N}{W} \rceil$. □

The instance used in the previous section can also be used here to prove that the above inequality is tight for a class of CBG games.

4 Colored Congestion Games

4.1 Price of Anarchy for Social Cost SC_{fib}

Theorem 6. *The price of anarchy of any CCG game with social cost SC_{fib} is at most $\mathcal{O}\left(\sqrt{W\,|F|}\right)$.*

Proof. We denote by $\overline{n_e(S)}$ the vector $[n_{e,c_1}(S), \ldots, n_{e,c_W}(S)]$. We can rewrite the social cost as $SC_{\text{fib}}(S) = \sum_{e\in F} \max_{c\in[W]} n_{e,c}(S) = \sum_{e\in F} \|\overline{n_e(S)}\|_\infty$. From norm inequalities, we have:

$$\frac{\|\overline{n_e(S)}\|_2}{\sqrt{W}} \leq \|\overline{n_e(S)}\|_\infty \leq \|\overline{n_e(S)}\|_2 \ , \tag{6}$$

hence:

$$SC_{\text{fib}}(S) = \sum_{e\in F} \|\overline{n_e(S)}\|_\infty \leq \sum_{e\in F} \sqrt{\sum_c n_{e,c}^2(S)} \leq \sqrt{|F|}\sqrt{\sum_{e\in F}\sum_c n_{e,c}^2(S)} \ , \tag{7}$$

where the last inequality is a manifestation of the norm inequality $\|\boldsymbol{x}\|_1 \leq \sqrt{n}\|\boldsymbol{x}\|_2$, where $\boldsymbol{x}$ is a vector of dimension n. Now, from the first inequality of (6) we have:

$$SC_{\text{fib}}(S) \geq \frac{1}{\sqrt{W}} \sum_{e\in F} \sqrt{\sum_c n_{e,c}^2(S)} \geq \frac{1}{\sqrt{W}} \sqrt{\sum_{e\in F}\sum_c n_{e,c}^2(S)} \ . \tag{8}$$

Combining (8) and (7), we get:

$$\frac{1}{\sqrt{W}}\sqrt{SC_{\text{sum}}(S)} \leq SC_{\text{fib}}(S) \leq \sqrt{|F|}\sqrt{SC_{\text{sum}}(S)} \ . \tag{9}$$

From [9] we know that the price of anarchy with social cost $SC_{\text{sum}}(S)$ is at most $5/2$. Let A be a worst-case Nash equilibrium under social cost SC_{fib} and let OPT be an optimal strategy profile. From (9) we know that $SC_{\text{fib}}(A) \leq \sqrt{|F|}\sqrt{SC_{\text{sum}}(A)}$ and $SC_{\text{fib}}(\text{OPT}) \geq \frac{1}{\sqrt{W}}\sqrt{SC_{\text{sum}}(\text{OPT})}$. Thus:

$$\text{PoA} = \frac{SC_{\text{fib}}(A)}{SC_{\text{fib}}(\text{OPT})} \leq \sqrt{W\,|F|}\sqrt{\frac{SC_{\text{sum}}(A)}{SC_{\text{sum}}(\text{OPT})}} \leq \sqrt{W\,|F|}\sqrt{\frac{5}{2}} \ . \tag{10}$$

□

Theorem 7. *There exists a class of CCG games with social cost SC_{fib} with* $\text{PoA} = \sqrt{W\,|F|}$.

Proof. Consider a colored congestion game with N players, $|F| = N$ facilities and $W = N$ colors. Each player has as strategies the singleton sets consisting of one facility: $\mathcal{E}_i = \{\{f_1\}, \{f_2\}, \ldots, \{f_N\}\}$.

The above instance has a worst-case equilibrium with social cost N when all players choose a different facility in an arbitrary color. On the other hand in the optimum strategy profile players fill all colors of the necessary facilities. This needs $\frac{N}{W}$ facilities with maximum capacity over their colors 1. Thus the optimum social cost is $\frac{N}{W}$ leading to a $\text{PoA} = \sqrt{W\,|F|}$. □

4.2 Price of Anarchy for Social Cost $SC_{\max}$

Theorem 8. *The price of anarchy of any CCG game with social cost $SC_{\max}$ is at most $\mathcal{O}\left(\sqrt{\frac{N}{W}}\right)$.*

Proof. Let A be a Nash equilibrium and let OPT be an optimal strategy profile. Without loss of generality, we assume that player 1 is a maximum cost player: $SC_{\max}(A) = C_1(A)$. Thus, we need to bound $C_1(A)$ with respect to the optimum social cost $SC_{\max}(\mathrm{OPT}) = \max\limits_{j \in [N]} C_j(\mathrm{OPT})$.

Since A is a Nash equilibrium, no player benefits from changing either her color or her choice of facilities. We denote by $\mathrm{OPT}_1 = (E_1^\star, a_1^\star)$ the strategy of player P_1 in OPT. Since A is a Nash equilibrium it must hold:

$$\forall c \in [W]: \ C_1(A) \leq \sum_{e \in E_1^\star} (n_{e,c}(A) + 1) \leq \sum_{e \in E_1^\star} n_{e,c}(A) + C_1(\mathrm{OPT}) \ . \tag{11}$$

The second inequality holds since any strategy profile cannot lead to a cost for a player that is less than the size of her facility combination.

Let $I \subset [N]$ be the set of players that, in A, use some facility $e \in E_1^\star$. The sum of their costs is:

$$\begin{aligned} \sum_{i \in I} C_i(A) \geq \sum_{e \in E_1^\star} \sum_{c \in [W]} n_{e,c}^2(A) \geq \frac{(\sum_{e \in E_1^\star} \sum_{c \in [W]} n_{e,c}(A))^2}{|E_1^\star| W} \geq \\ \frac{(W \min_{c \in [W]} \sum_{e \in E_1^\star} n_{e,c}(A))^2}{|E_1^\star| W} \geq \frac{W (\min_{c \in [W]} \sum_{e \in E_1^\star} n_{e,c}(A))^2}{|E_1^\star|} \ . \end{aligned} \tag{12}$$

The first inequality holds since a player in I might use facilities (e, c) not in $E_1^\star$ and the second inequality holds from the Cauchy-Schwarz inequality. Denoting by $c_{\min}$ the color $\arg\min_{c \in [W]} \sum_{e \in E_1^\star} n_{e,c}(A)$, we have:

$$\left(\sum_{e \in E_1^\star} n_{e,c_{\min}}(A) \right)^2 \leq \frac{|E_1^\star|}{W} \sum_{i \in I} C_i(A) \ . \tag{13}$$

We know from [9] that:

$$\sum_{i \in [N]} C_i(A) \leq \frac{5}{2} \sum_{i \in [N]} C_i(\mathrm{OPT}) \ . \tag{14}$$

Combining the above two inequalities we have:

$$\left(\sum_{e \in E_1^\star} n_{e,c_{\min}}(A) \right)^2 \leq \frac{|E_1^\star|}{W} \sum_{i \in I} C_i(A) \leq \frac{|E_1^\star|}{W} \sum_{i \in [N]} C_i(A) \leq \frac{5}{2} \frac{|E_1^\star|}{W} \sum_{i \in [N]} C_i(\mathrm{OPT}) \tag{15}$$

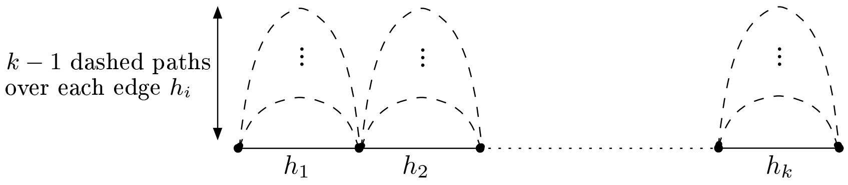


Fig. 1. A worst-case instance that proves the asymptotic tightness of the upper bound on the price of anarchy of CCG games with social cost $SC_{\max}$, depicted as a network game. A dashed line represents a path of length k connecting its two endpoints.

Combining with (11) for $c_{\min}$, we get

$$C_1(A) \le C_1(\mathrm{OPT}) + \sqrt{\frac{5}{2}\frac{|E_1^\star|}{W}\sum_{i\in[N]} C_i(\mathrm{OPT})} \ . \tag{16}$$

Since $|E_1^\star| \le C_1(\mathrm{OPT})$ and $C_i(\mathrm{OPT}) \le SC_{\max}(\mathrm{OPT})$ for any $i \in [N]$, we get

$$C_1(A) \le \left(1 + \sqrt{\frac{5}{2}\frac{N}{W}}\right) SC_{\max}(\mathrm{OPT}) \ . \tag{17}$$

□

Theorem 9. *There exists a class of CCG games with social cost* $SC_{\max}$ *with* $\mathrm{PoA} = \Theta\left(\sqrt{\frac{N}{W}}\right)$.

Proof. Given integers $k > 1$ and $W > 0$, we will describe the lower bound instance as a network game. The set of colors is $[W]$. The network consists of a path of $k+1$ nodes $n_0, \ldots, n_k$. In addition, each pair of neighboring nodes n_i, n_{i+1} is connected by $k-1$ edge-disjoint paths of length k. Figure 1 provides an illustration.

In this network, W major players want to send traffic from n_0 to n_k. For every i, $0 \le i \le k-1$, there are $(k-1)W$ minor players that want to send traffic from node n_i to node n_{i+1}. In the worst-case equilibrium A all players choose the short central edge, leading to social cost $SC_{\max}(A) = k^2$. In the optimum the minor players are equally divided on the dashed-line paths and the major players choose the central edge. This leads to $SC_{\max}(\mathrm{OPT}) = k$, and the price of anarchy is therefore:

$$\mathrm{PoA} = k = \Theta\left(\sqrt{\frac{N}{W}}\right) \ . \tag{18}$$

□

4.3 Price of Anarchy for Social Cost SC_{sum}

The price of anarchy of CCG games with social cost SC_{sum} is upper-bounded by 5/2, as proved in [9]. For the lower bound, we use a slight modification of the instance described in [9]. We have NW players and $2N$ facilities. The facilities are separated into two groups: $\{h_1, \ldots, h_N\}$ and $\{g_1, \ldots, g_N\}$. Players are divided into N groups of W players. Each group i has strategies $\{h_i, g_i\}$ and $\{g_{i+1}, h_{i-1}, h_{i+1}\}$. The optimal allocation is for all players in the i-th group to select their first strategy and be equally divided in the W colors, leading to $SC_{\text{sum}}(\text{OPT}) = 2NW$. In the worst-case Nash equilibrium, players choose their second strategy and are equally divided in the W colors, leading to $SC_{\text{sum}}(A) = 5NW$. Thus, the price of anarchy of this instance is 5/2 and the upper bound remains tight in our model as well.

5 Discussion

In this paper we introduced Colored Resource Allocation Games, a class of games which generalize both congestion and bottleneck games. The main feature of these games is that players have their strategies in multiple copies (colors). Therefore, these games can serve as a framework to describe routing and wavelength assignment games in multifiber all-optical networks. Although we could cast such games as classical congestion games, it turns out that the proliferation of resources together with the structure imposed on the players' strategies allows us to prove better upper bounds.

Regarding open questions, it would be interesting to consider more general latency functions. This would make sense both in the case where fiber pricing is not linear in the number of fibers, and also in the case where the network operator seeks to determine an appropriate pricing policy so as to reduce the price of anarchy. Another interesting direction is to examine which network topologies result in better system behavior.

References

1. Rosenthal, R.W.: A class of games possessing pure-strategy Nash equilibria. Int. J. Game Theory 2, 65–67 (1973)
2. Monderer, D., Shapley, L.S.: Potential games. Games and Economic Behavior 14, 124–143 (1996)
3. Roughgarden, T.: Selfish routing and the price of anarchy. The MIT Press (2005)
4. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
5. Nomikos, C., Pagourtzis, A., Zachos, S.: Routing and path multicoloring. Inf. Process. Lett. 80(5), 249–256 (2001)
6. Nomikos, C., Pagourtzis, A., Potika, K., Zachos, S.: Routing and wavelength assignment in multifiber WDM networks with non-uniform fiber cost. Computer Networks 50(1), 1–14 (2006)
7. Andrews, M., Zhang, L.: Complexity of wavelength assignment in optical network optimization. In: INFOCOM. IEEE (2006)

8. Andrews, M., Zhang, L.: Minimizing maximum fiber requirement in optical networks. J. Comput. Syst. Sci. 72(1), 118–131 (2006)
9. Christodoulou, G., Koutsoupias, E.: The price of anarchy of finite congestion games. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, STOC, pp. 67–73 (2005)
10. Li, G., Simha, R.: On the wavelength assignment problem in multifiber WDM star and ring networks. IEEE/ACM Trans. Netw. 9(1), 60–68 (2001)
11. Margara, L., Simon, J.: Wavelength assignment problem on all-optical networks with k fibres per link. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 768–779. Springer, Heidelberg (2000)
12. Bampas, E., Pagourtzis, A., Pierrakos, G., Potika, K.: On a noncooperative model for wavelength assignment in multifiber optical networks. IEEE/ACM Trans. Netw. 20(4), 1125–1137 (2012)
13. Busch, C., Magdon-Ismail, M.: Atomic routing games on maximum congestion. In: Cheng, S.-W., Poon, C.K. (eds.) AAIM 2006. LNCS, vol. 4041, pp. 79–91. Springer, Heidelberg (2006)
14. Nash, J.: Non-cooperative games. The Annals of Mathematics 54(2), 286–295 (1951)
15. Banner, R., Orda, A.: Bottleneck routing games in communication networks. In: INFOCOM. IEEE (2006)
16. Roughgarden, T.: Intrinsic robustness of the price of anarchy. In: Mitzenmacher, M. (ed.) STOC, pp. 513–522. ACM (2009)
17. Fotakis, D., Kontogiannis, S., Koutsoupias, E., Mavronicolas, M., Spirakis, P.: The structure and complexity of Nash equilibria for a selfish routing game. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 123–134. Springer, Heidelberg (2002)
18. Libman, L., Orda, A.: Atomic resource sharing in noncooperative networks. Telecommunication Systems 17(4), 385–409 (2001)
19. Kannan, R., Busch, C.: Bottleneck congestion games with logarithmic price of anarchy. In: Kontogiannis, S., Koutsoupias, E., Spirakis, P.G. (eds.) SAGT 2010. LNCS, vol. 6386, pp. 222–233. Springer, Heidelberg (2010)
20. Harks, T., Hoefer, M., Klimm, M., Skopalik, A.: Computing pure Nash and strong equilibria in bottleneck congestion games. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part II. LNCS, vol. 6347, pp. 29–38. Springer, Heidelberg (2010)
21. Bilò, V., Moscardelli, L.: The price of anarchy in all-optical networks. In: Kralovic, R., Sýkora, O. (eds.) SIROCCO 2004. LNCS, vol. 3104, pp. 13–22. Springer, Heidelberg (2004)
22. Bilò, V., Flammini, M., Moscardelli, L.: On Nash equilibria in non-cooperative all-optical networks. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 448–459. Springer, Heidelberg (2005)
23. Georgakopoulos, G.F., Kavvadias, D.J., Sioutis, L.G.: Nash equilibria in all-optical networks. In: Deng, X., Ye, Y. (eds.) WINE 2005. LNCS, vol. 3828, pp. 1033–1045. Springer, Heidelberg (2005)
24. Milis, I., Pagourtzis, A., Potika, K.: Selfish routing and path coloring in all-optical networks. In: Janssen, J., Prałat, P. (eds.) CAAN 2007. LNCS, vol. 4852, pp. 71–84. Springer, Heidelberg (2007)
25. Anshelevich, E., Dasgupta, A., Kleinberg, J., Tardos, E., Wexler, T., Roughgarden, T.: The price of stability for network design with fair cost allocation. In: Proceedings. 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 295–304 (October 2004)

Average Optimal String Matching in Packed Strings*

Djamal Belazzougui[1] and Mathieu Raffinot[2]

[1] Department of Computer Science, University of Helsinki, Finland
djamal.belazzougui@cs.helsinki.fi
[2] LIAFA, Univ. Paris Diderot - Paris 7, 75205 Paris Cedex 13, France
raffinot@liafa.univ-paris-diderot.fr

Abstract. In this paper we are concerned with the basic problem of string pattern matching: preprocess one or multiple fixed strings over alphabet σ so as to be able to efficiently search for all occurrences of the string(s) in a given text T of length n. In our model, we assume that text and patterns are tightly packed so that any single character occupies $\log \sigma$ bits and thus any sequence of k consecutive characters in the text or the pattern occupies exactly $k \log \sigma$ bits. We first show a data structure that requires $O(m)$ words of space (more precisely $O(m \log m)$ bits of space) where m is the total size of the patterns and answers to search queries in average-optimal $O(n/y)$ time where y is the length of the shortest pattern ($y = m$ in case of a single pattern). This first data structure, while optimal in time, still requires $O(m \log m)$ bits of space, which might be too much considering that the patterns occupy only $m \log \sigma$ bits of space. We then show that our data structure can be compressed to only use $O(m \log \sigma)$ bits of space while achieving query time $O(n(\log_\sigma m)^\varepsilon/y)$, with ε any constant such that $0 < \varepsilon < 1$. We finally show two other direct applications: average optimal pattern matching with worst-case guarantees and average optimal pattern matching with k differences. In the meantime we also show a slightly improved worst-case efficient multiple pattern matching algorithm.

1 Introduction

The string matching problem consists of finding all occurrences of a given pattern $p = p_1p_2 \ldots p_m$ in a large text $T = t_1t_2 \ldots t_n$, both sequences of characters from a finite character set Σ of size $\sigma = |\Sigma|$. This problem is fundamental in computer science and has a wide range of applications in text retrieval, symbol manipulation, computational biology, and network security.

This problem has been deeply studied and since the 70's there exist algorithms like the Morris-Pratt algorithm (MP) [27] or the well-known Knuth-Morris-Pratt (KMP -a variation of MP) [24] that are average and worst case time $O(n + m)$, that is, linear in the size of the text and that of the pattern. This problem has been further investigated and a time average lower bound in $\Omega(n \log_\sigma(m)/m)$

* Work partially supported by the french ANR-2010-COSI-004 MAPPI Project.

(assuming equiprobability and independence of letters) has been proved by A.C. Yao in [33]. Since this seminal work, many average optimal algorithms have been proposed, from both a theoretical point of view, like the Backward Dawg Matching (BDM) [15], than from a practical point of view, like the Backward Nondeterministic Dawg Matching (BNDM) [29]. Worst case linear and average optimal algorithms also appeared, mainly by combining a forward algorithm like KMP with a backward search algorithm similar to BDM or BNDM. The Double-Forward [2] is however an exception to this general approach since it combines two forward algorithms sharing a unique data structure. It is the simplest algorithm reaching these optimal complexities published so far.

A natural extension to the string matching problem is to search for all occurrences of multiple strings instead of a single one. We extend the notation m to be the total size of the patterns. In a similar way to the single pattern case, there exist $O(n+m)$ linear time algorithms, the most famous being the Aho-Corasick algorithm [1], and also average optimal algorithms like Multi-BDM [29] and Dawg-Match [14].

The optimal results and algorithms we mentioned above are valid in classical models. In this paper we are concerned with the single and multiple string pattern matching problem in the RAM model with word length $\omega = \Omega(\log(n+m))$, assuming that text and patterns are tightly packed so that any single character occupies $\log\sigma$ bits and thus any sequence of k consecutive characters in the text or the pattern occupies exactly $k\log\sigma$ bits.

The single and multiple string matching problems have already been studied in this model both from a worst case and average point of view. For the worst case bound, the main studies are from Fredriksson [18], Bille [9] and Belazzougui [3,4]. Those studies made some progress on both single and multiple string matching problems. Very recently Benkiki et al. [8] obtained the optimal $O(n\frac{\log\sigma}{\omega} + occ)$ query time for the single string case, however their result requires the availability of non-standard instructions. Eventually Breslauer et al. [10] obtained an algorithm with the unconditional optimal $O(n\frac{\log\sigma}{\omega} + occ)$ query time for the single string matching problem. The optimal worst-case bound for the multiple string variant is still an open problem despite the progress made in the recent studies.

This paper however mainly focuses on average optimal string matching. This question has already been considered by Fredriksson also in [18] where he presented a general speed-up framework based on the notion of super-alphabet and on the use of tabulation (four russian technique). The single string matching algorithm obtained using these techniques is optimal on average, but at the cost of a huge memory requirement. Precisely, the algorithm is $O(n/y)$ where y is the length of the shortest pattern ($y = m$ in case of a single pattern) while requiring $O(\sigma m)$ space.

In this paper we first explain a data structure that requires only $O(m)$ words of space and answers to queries in average optimal $O(n/y)$. However, our first data structure, while leading to optimal query times, still requires $O(m\log m)$ bits of space which might be still too much considering that the patterns occupy only $m\log\sigma$ bits of space. We then show that our data structure can be compressed to only require $O(m \log \sigma)$ bits of space while achieving query time $O(n(\log_\sigma m)^\varepsilon / y)$. We eventually show two other direct applications: average optimal pattern matching with worst case guarantees and average optimal string pattern matching with k differences. In the meantime we also show improved solutions for worst-case efficient multiple pattern matching algorithm (which we use for the average-optimal pattern matching with worst-case guarantees).

Model and Assumptions. In this paper, we assume a standard word RAM model with word length $\omega = \Omega(\log n)$ (where n is the size of the input) with all standard operations (including multiplication and addition) supported in constant time. We assume that the text and the patterns are from the same alphabet σ. In our bounds we often refer to three different kinds of time measures:

1. Worst-case time of an algorithm refers to a deterministic time that holds regardless of the input and of the random choices made by the algorithm.
2. Randomized or randomized expected time of an algorithm measures the average time of the algorithm over all possible random choices of the algorithm regardless of the input (it holds for any chosen input).
3. Expected or average time of an algorithm measures the average time of the algorithm considering a probability model on the input. In our case the input is either a text or a pattern and the probability model simply assumes that the positions of the input are all independent and of the same probability $1/\sigma$.

We quantify the space either in bits or in words. To translate between words and bits, the space is simply multiplied by a factor $\log n$. This is justified, since the model only assumes that $\omega = \Omega(\log n)$ and thus allows the case $\omega = \Theta(\log n)$. Note also that even if $\omega \gg \log n$, we can still simulate any algorithm designed to use words of size $\Theta(\log n)$ instead of ω with only a constant-factor slowdown.

2 The Basic Algorithm

We first consider the single string matching problem and we state the following result.

Theorem 1. *Given a pattern (string) p of length m over an alphabet of size σ, we can build a data structure of size $O(m \log m)$ bits so that we can report all the occ occurrences of the pattern p in any packed text T of length n in expected time $O(n/m)$. The construction of the data structure takes $O(m)$ randomized expected time (in which case the data structure occupies $m \log m + O(m)$ bits only) and $O(m \log \log m)$ time in the worst-case.*

Proof. We first present the randomized construction. We use a window of size m characters. We build a minimal perfect hash function f [21] on the set F_t of factors of the pattern of length $t = 3 \log_\sigma m$ characters. This hash function occupies $O(|F_t|)$ bits of space and can be evaluated in $O(1)$ time. Before building this minimal perfect hash function we first extract the factors in the following way: we traverse the pattern p and then at each iteration i extract in $O(1)$ time the characters $s_i = p[i..i+t-1]$ of the pattern (note that s_i is actually just a bitvector of $3\log n$ bits) and add the pair (s_i, i) at the end of a temporary array M initially empty. Then, we just sort this array (using radix-sort) in time $O(m)$ where the pairs are ordered according to their first component (the s_i component). Then we store a table $T[1..m]$. For every $s \in F_t$ occurring at position j in p (that is we have $s = p[j..t-1]$), we set $T[f(s)] = j$. Thus the hash function f can be used in conjunction with the table T in order to get at least the position of one occurrence of every substring of p of length t. Note that the space occupancy of f is $O(m-t) = O(m)$ bits while the space occupancy of T is $(m-t)\log m \leq m \log m$ bits. Note also the hash function f can be evaluated in $O(1)$ time as it operates on strings of length $t = 3\log_\sigma m$ characters, which is $O(\log m) = O(w)$ bits. Next, the matching of p in a text T follows the usual strategy used in the other average optimal string matching algorithms. That is, we use a window (a window is defined by just a pointer to the text plus an integer defining the size of the window in number of characters) of size $2m-t$ (except at the last step where the window could be of smaller size). Before starting the first iteration, the window is placed at the beginning of the text (that is, take $w = T[1..2m-t]$). Then at each iteration i starting from $i = 1$ do the following:

1. Consider the substring $q = w[m-t+1..m]$.
2. Compute $j = T[f(q)]$ and compare q with the substring $p[j..j+t-1]$.
3. In case the two substrings match, do an intensive search for p in the window using any string matching method that runs in reasonable time.
4. Check whether $i(m-t+1)+m \geq n$ in which case the search is finished, otherwise continue with the next step.
5. Finally shift the window by $m-t+1$ positions by setting $w = T[i(m-t+1)+1..(i+1)(m-t+1)+m]$ (or $w = T[i(m-t+1)+1..n]$ whenever $(i+1)(m-t+1)+m > n$), increment i and go to step 1 of the next iteration.

It is easy to check that the algorithm above runs in expected time $O(n/m)$ time assuming that the characters are generated independently uniformly at random. The query time follows from the fact that we are doing $O(n/(m-t+1)) = O(n/m)$ iterations and at each iteration the only non-constant step is the intensive search, which costs $O(m^2)$ (assuming the most naive search algorithm) but takes place only with probability at most $O(m/\sigma^{3\log_\sigma m}) = (1/m^2)$ and thus only contributes $O(1)$ to the cost of search.

We now describe the deterministic construction. Instead of storing a perfect hash function on all factors of lengths $t = 3\log_\sigma m$ characters, mapping them to interval $[1..m]$ and then storing their corresponding pointers in the table T, we instead store all those factors using the deterministic dictionary of [30]. This dictionary occupies linear space and can be constructed in time $O(n \log\log n)$ when built on a set of n keys of lengths $O(\log n)$ bits each. In our case, we have $\Theta(m)$ keys of length $t = 3\log_\sigma m$ characters, which is $3\log m$ bits and thus the construction of [30] takes $O(m \log\log m)$ worst-case time. Note that the space occupancy of this construction is also linear $O(m \log m)$ bits of space and the query time is constant.

What remains is to show how to combine the two data structures. The combination consists only in first trying to build the randomized data structure fixing some maximal amount of time cm for a suitably chosen c. Then if the construction fails to terminate in that time, we build the deterministic data structure. Note, however that the deterministic construction will be very far from practical both from the space and time point of view. $\square$

This result improves upon Fredriksson's result since the memory our algorithm requires is only $O(m)$ words of space or $O(m \log m)$ bits. Using the same approach we extend our result to optimally solve in average the multiple string matching problem.

Theorem 2. *Given a set of patterns (strings) S of total lengths m over an integer alphabet of size σ where the shortest pattern is of length $y \geq 4 \log_\sigma m$, we can build a data structure of size $O(m \log m)$ bits so that we can report the occurrences of any of the patterns in S in any packed text T of length n in expected time $O(n/y)$. The construction of the data structure takes $O(m)$ randomized expected time (in which case the data structure occupies $m \log m + O(m)$ bits only) and $O(m \log\log m)$ time in the worst-case (in which case the data structures occupies $O(m \log m)$ bits of space).*

Proof. The algorithm is almost the same as that of theorem 1 except for the following points:

1. The window size is now $2y - t$ where y is the length of the shortest pattern in the set of patterns and $t = 3 \log_\sigma m$.
2. We index all the substrings of length $t = 3 \log_\sigma m$ of all of the patterns. That is, the function f will store all factors of the strings.
3. The intensive search looks for all of the patterns in the window. The time for this intensive search is $O(ym) = O(m^2)$.
4. The window is advanced by $y - t + 1$ characters at each iteration.

The query time can be easily bounded by the same analysis used in theorem 1. That is, at each iteration the only non-constant time step is the intensive search that takes $O(ym) = O(m^2)$ time, but is triggered only with probability $O(1/m^2)$ and thus contributes a $O(1)$ cost. The probability $O(1/m^2)$ is deduced from the fact that any string of length $3 \log_\sigma m$ generated at random has a probability $O(1/\sigma^{3 \log_\sigma m}) = O(1/m^3)$ of colliding with any substring of one of the patterns and thus, the probability of colliding with any substring of any of the patterns is $O(1/m^2)$. Note that the window is advanced by $t' = y - t + 1 = y - 3 \log_\sigma m + 1$ at each iteration. Since $y \geq 4 \log_\sigma m$, we deduce that $t' \geq y/4$. Thus we have about $4n/y$ iteration where at iteration an expected $O(1)$ time is spent which gives a total of $O(n/y)$ time. $\square$

3 Succinct Representation

The space required by the representation used in the two previous theorems is $O(m \log m)$ (only $m \log m + O(m)$ bits for a randomized construction).

This space usage is not succinct in the sense that it is larger than the space used by the pattern(s), which is $m \log \sigma$ bits only. In the context of single string matching, this is probably not a problem as m is small enough to fit entirely in memory. However in the case of multiple string matching, the cumulative size of the patterns may exceed the available memory (or at least do not fit into fastest levels of memory). In the last decade many efforts have been devoted to reduce the space required by the full text indexing data structures that allow us to answer efficiently (with a small sacrifice in query time) to queries asking for all the occurrences of a pattern in a text, but in which the text is fixed and the patterns are given as queries. Note that this is the converse of our case, in which the text is given as a query and the pattern(s) is/are fixed. In this section we show that the first two theorems can also be modified to obtain a different space/time trade-off between the space used by the data structure and the time required to match the text. Our solution makes use of results from succinct full-text indexing. our solution makes use of results from succinct full-text indexing literature, namely the compressed text index of [19] built using the space-efficient methods described in [22,23]. In [19] the following lemma was shown:

Lemma 1. *There exists a succinct full text index which can be built on a text T of length m (or a collection of texts of total length m) over an alphabet of size σ such that:*

1. *The full text index occupies space $O(m \log \sigma)$ bits and;*
2. *It can return for any string p the range of suffixes (a range $[i..j]$ in the suffix array) of T prefixed by p in time $O(|p|/\log_\sigma m + (\log_\sigma m)^\varepsilon)$.*

The construction of the data structure takes either $O(m \log \sigma)$ randomized time or $O(m \log m)$ deterministic time and uses $O(m \log m)$ bits of space during the construction.

The construction algorithm in [19] uses $O(m \log m)$ bits of temporary space during the construction though the constructed index occupies only $O(m \log \sigma)$ bits of space. In our case we strive to reduce the peak consumption during both construction and matching phases and thus need to use a construction algorithm that uses space comparable to the constructed index. This is achieved by the following result:

Lemma 2 ([22,23]). *Given a text T of length n (or a collection of texts of total length m) over an alphabet of size σ, the indexes of [19] can be constructed in worst-case time $O(m \log m)$ using temporary space $O(m \log \sigma)$ bits of space (in addition to the final space of the index).*

The result about our succinct representation for average-optimal string matching is summarized by the following theorem:

Theorem 3. *Given a set of patterns (strings) S of total lengths m over an integer alphabet of size σ where the shortest pattern is of length $y \geq 4 \log_\sigma m$, we can build a data structure of size $O(m \log \sigma)$ bits so that we can report all of the occurrences of any of the patterns in time $O(n(\log_\sigma m)^\varepsilon / y)$ for any $\varepsilon \in (0, 1)$.*

The index can be built in worst-case time $O(m \log m)$ using $O(m \log \sigma)$ bits of temporary space.

Proof. For achieving the improved space bounds we will incorporate lemma 1 into the first step of the algorithm, which was to find whether the string $q = w[m - t + 1..m]$ matches some substring of some text. The hash data structure used for that purpose in theorem 2 occupies space $O(m \log m)$. The reason for needing that much space was because of the table T which stores m pointers of $\log m$ bits each. We now show an alternative way to match the strings using a compressed index. For that, we simply index the pattern(s) using the index of lemma 1. Then, checking if q exists takes $O(t/\log_\sigma m + (\log_\sigma m)^\varepsilon) = O((\log_\sigma m)^\varepsilon)$ time. The space used by the index is $O(m \log \sigma)$ bits. Building the index (using lemma 2) takes $O(m \log m)$ time and uses at most $O(m \log \sigma)$ bits of temporary space. □

The main advantage of theorem 3 over the first two theorems is that the peak memory use during the preprocessing or the matching phases never exceeds $O(m \log \sigma)$ bits of space (against $\Theta(m \log m)$ bits of peak memory use in the two first theorems). The drawback is that the construction and matching phases are slightly slower than those of theorems 2 and 1.

Corollary 1. *Given a pattern p of length m over an integer alphabet of size σ, we can build in time $O(m \log m)$ a data structure of size $O(m \log \sigma)$ bits so that we can report all the occ occurrences of p in any packed text T of length n in expected time $O(\frac{n}{m}(\log_\sigma m)^\varepsilon)$ for any $\varepsilon \in (0, 1)$. The peak memory usage during the construction or matching phases is $O(m \log \sigma)$ bits.*

Note that this corollary still improves when compared with the standard algorithm which examines the pattern and the text character by character and has query time $O(\frac{n}{m} \log_\sigma m)$, slower than our corollary by a factor $(\log_\sigma m)^{1-\varepsilon}$.

4 Worst-Case Guarantees

The optimal average time algorithm we presented above runs in worst case time $O(\frac{n}{m} + occ)$. However, in many real life uses of string matching, one does not know any property of the data source. Thus, even if on average the algorithm is optimal, it is cautious to bound its worst case time to avoid a *bad* instance to block the software. We thus extend our approach to guarantee a worst case time bound at least as worst as the fastest forward algorithm designed for (multiple) string(s) matching when text and patterns are tightly packed. For this sake we adapt the approach used in the Double-Forward algorithm [2] to packed strings.

The idea is simple. We consider the same approach used for proving Theorem 1. We hash the last $3 \log_\sigma m$ characters of the current search window through the minimal perfect hashing function we built on all $3 \log_\sigma m$ characters long factor of the pattern. If the hash test fails, we shift the search window to the right just after the first character of the $3 \log_\sigma m$ last characters we tested. This situation is shown in figure 1. Note that this step is the same as in the proof of Theorem 1.

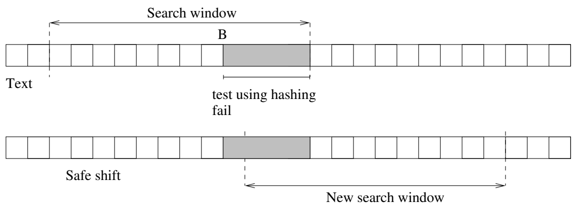


Fig. 1. Hash test fail case. The search window is shifted after the first of the $3\log_\sigma m$ characters tested.

The algorithm changes if the test passes. Assume that at some point we began a forward scan of the text using the algorithm of [4] that stopped in a position A in the text after having read a prefix u of the current search window. Figure 2 illustrates this case. We simply continue this forward search from A', passing the end of the current search window and reading characters until the longest suffix of the text (read by blocks of characters) that matches a prefix u' of the pattern is small enough. We then repeat the global search scheme from this new window.

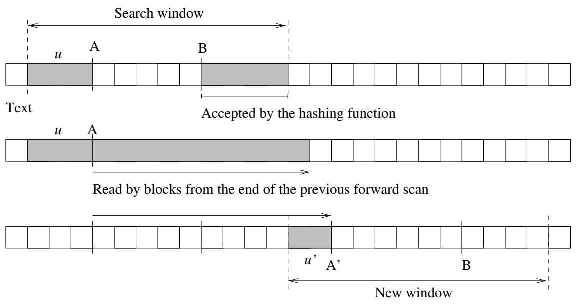

Search window
u
A
B
Text
Accepted by the hashing function
u
A
Read by blocks from the end of the previous forward scan
u'
A'
B
New window

Fig. 2. Hash test success case. The forward search stopped in A' is continued until passing the end of the initial window until point A'.

It is obvious that this new algorithm remains optimal on average. As a forward algorithm, we use the very recent algorithm of Breslauer et al. [10]. The algorithm achieves preprocessing time $O(m/\log_\sigma m+\omega)$ and optimal worst-case query time $O(n\frac{\log\sigma}{\omega})$. We thus get the following result:

Theorem 4. *Given a pattern p of length m characters over an integer alphabet of size σ, we can build in randomized $O(m+\omega)$ time and worst-case $O(m\log\log m+\omega)$ time a data structure of size $O(m\log m)$ bits so that we can report all of the occ occurrences of p in any packed text T of length n in expected time $O(n/m)$ and worst-case time $O(\frac{n\log\sigma}{\omega}+occ)$.*

4.1 Multiple String Matching

We extend the previous algorithm to match a set of strings. We use as a forward algorithm an improved version of the multiple string algorithm searching described in [3,4], whose original version runs in $O(n(\frac{\log d + \log y + \log\log m}{y} + \frac{\log\sigma}{\omega}) + occ)$ time using a linear-space data structure that can be built in $O(m\log m)$ time. We first prove the following (slightly) improved result based on [3,4]:

Theorem 5. *Given a set of d patterns (strings) S of total length m characters over an integer alphabet of size σ where the shortest pattern is of length y, we can build in worst case time $O(m\log m)$ a data structure of size $O(m\log m + \omega)$ bits so that we can report all of the occ occurrences of any of the patterns in S in any packed text T of length n in worst-case time $O(n(\frac{\frac{\log d}{\log\log d} + \log y + \log\log m}{y} + \frac{\log\sigma}{\omega}) + occ)$. The data structure uses temporary space $O(m\log m + \omega^2)$.*

Proof. To get the improved result we first notice that the text matching phase in the multiple string matching algorithm of [4] proceeds in $\Theta(n/y)$ iterations, at each iteration reading y consecutive characters and spending $O(\log d + \log y + \log\log m + y\frac{\log\sigma}{\omega})$ time, where the two terms $\log d$ and $\log y$ are due to :

1. The use of a string B-tree, which is a data structure used for longest suffix matching. The string B-tree stores a set S' of $O(dy)$ strings (actually $O(dy)$ pointers to text substrings of length y) and answers to a longest suffix matching query for a query string of length y in time $O(y\frac{\log\sigma}{\omega} + \log(dy))$.
2. The use of a two dimensional rectangle stabbing data structure of Chazelle [13]. This data structures stores up to $m' = O(dy)$ rectangles using $O(m') = O(dy)$ space and answers to rectangle stabbing queries that ask to report all the rectangles that contain a given query point. The original query time was $O(\log m' + occ)$ where occ is the number of reported rectangles. However this query time was later improved to $O(\frac{\log m'}{\log\log n'} + occ)$ [34].

To get our improved query time, we will use an alternative data structure for longest suffix matching. We use a compacted trie built on the set S'. The compacted trie will use $O(dy)$ pointers of $O(\log(dy)) = O(\log m)$ bits. The compacted trie can be built in time $O(m\log m)$ and uses space $O(m\log m)$ bits. In order to accelerate the traversal of the trie we will use super-characters built from every $\frac{\omega}{\log\sigma}$ consecutive characters. The compacted trie will be built on the original set of strings considered as strings over the super-alphabet. This is done by grouping every sequence of $\frac{\omega}{\log\sigma}$ characters into a single one (strings whose length is not multiple of $\frac{\omega}{\log\sigma}$ are padded with special characters before grouping). Then a node of the trie will have children labeled with characters from the new alphabet. Note that each character of the new alphabet occupies $O(\omega)$ bits. In order to be able to determine which child to follow, we need to build a dictionary on the set of labels of each node. However this would occupy in total $O(m\omega)$ bits of space. In order to reduce the space, we will group all the distinct labels of all the nodes in the trie, and build a perfect hash function that maps all the distinct labels to a range of size $m^{O(1)}$ (we call the resulting labels reduced labels). The resulting function occupies $O(\omega)$ bits of space . A deterministic perfect hash function occupying constant space can be built in deterministic $O(m\log m)$ time [20] or randomized $O(m)$ time using universal hashing [11]. Then for each node we build a local deterministic dictionary that associates a pointer to the child corresponding to each reduced label (we call it child dictionary). As each reduced label is the range $m^{O(1)}$ means that it can be described using $O(\log m)$ bits. Thus the total space occupied by all child dictionaries will be $O(m \log m)$ bits. The compacted trie above is able to to return the length of the longest suffix matching up to lengths multiple of $\frac{\omega}{\log \sigma}$ (old alphabet) characters.

In order to terminate the longest suffix matching query up to additional $\frac{\omega}{\log \sigma} - 1$ characters, we will for each node build a z-fast trie [5,7] on all child labels. The z-fast trie will be built on the set of labels of the children of each node. A z-fast trie supports longest suffix matching queries in time $O(\log |p|)$ on a pattern p. It uses a linear number of pointers to the original strings (see [7]). The probabilistic z-fast trie presented in [5,7], can be made deterministic by building a deterministic perfect hash function on the path labelling each node in the z-fast trie using [20]. The paths are of length at most ω bits, we can reuse the same strategy above to map the labelling paths to $O(\log m)$ bits.

In order to further reduce the used space we partition our initial set of strings into groups of ω strings and store only the first string of each group in the structure above (we call it sampled trie), but in addition for each node of the sampled trie store a pointer to the corresponding node in the original non-sampled trie. Then we build the same above trie structure on each group of ω (call them group tries). This ensures that building the above structure on each group requires just $O(\omega^2)$ bits of space. That way the total space is bounded by $O(\omega^2 + m \log m)$ bits of space. Finally a query will first start in the sampled trie, then follow a pointer to the non-sampled trie to match at most one additional character. This isolates a range that spans at most two groups which can then be searched using the local group tries. □

We can now replace the $\log d + \log y$ term with the addition of the query times of the alternative longest suffix matching data structure and the improved query time for the rectangle stabbing problem to obtain a total query time $O(\frac{\frac{\log(dy)}{\log\log(dy)} + \log y + \log\log m}{y} + occ) = O(\frac{\frac{\log d}{\log\log d} + \log y + \log\log m}{y} + occ)$.

Now back to the average-optimal multiple string matching algorithm. As the size of the search window in the algorithm is y, the size of the smallest string searched, the complexities we obtain on average depend on y instead of m. And we get:

Theorem 6. *Given a set of d patterns (strings) S of total length m characters over an integer alphabet of size σ where the shortest pattern is of length $y \geq 4 \log_\sigma m$, we can build in worst case time $O(m \log m)$ a data structure of size $O(m \log m)$ bits so that we can report all of the occ occurrences of any of the patterns in S in any packed text T of length n in expected time $O(n/y)$ and in worst case time $O(n(\frac{\frac{\log d}{\log\log d} + \log y + \log\log m}{y} + \frac{\log \sigma}{\omega}) + occ)$.*

5 Approximate String Matching

Approximate string matching considering edit distance is to find all positions in a text where at least one string of the set of patterns matches the text up to k errors. The approximate string matching problem can be efficiently solved for relatively small k (in regard to the length of the strings) using exact string matching as a filter. This approach is also used in [18]. The idea is the following. Assume a string to match the text up to k errors. If this string is first divided into $k+1$ pieces, one of these pieces must exactly match the text. Thus, the approach is to split the string patterns in $k+1$ pieces and search for all those pieces simultaneously in the text using the multiple string matching algorithm of Theorem 2. If the pieces are of the same length, $m/(k+1)$, the average time required to search all pieces is $O(n/(m/k+1)) = O((k+1)n/m) = O(kn/m)$.

If one of those pieces matches, then a complete match of the corresponding strings is checked using a classical $O(km)$ algorithm.

Considering a probability model in which all positions are independent and of the same probability $1/\sigma$, a rough upper bound of the number of verifications is $O(nk(1/\sigma)^{(m/k)})$. For $k < m/\log_\sigma m$, the time of the multiple string matching algorithm dominates and the average time remains $O(kn/m)$.

References

1. Aho, A.V., Corasick, M.J.: Efficient string matching: an aid to bibliographic search. Commun. ACM 18(6), 333–340 (1975)
2. Allauzen, C., Raffinot, M.: Simple optimal string matching. Journal of Algorithms 36, 102–116 (2000)
3. Belazzougui, D.: Worst case efficient single and multiple string matching in the ram model. In: Iliopoulos, C.S., Smyth, W.F. (eds.) IWOCA 2010. LNCS, vol. 6460, pp. 90–102. Springer, Heidelberg (2011)
4. Belazzougui, D.: Worst-case efficient single and multiple string matching on packed texts in the word-ram model. J. Discrete Algorithms 14, 91–106 (2012)
5. Belazzougui, D., Boldi, P., Pagh, R., Vigna, S.: Monotone minimal perfect hashing: searching a sorted table with $o(1)$ accesses. In: SODA, pp. 785–794 (2009)
6. Belazzougui, D., Boldi, P., Pagh, R., Vigna, S.: Fast prefix search in little space, with applications. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part I. LNCS, vol. 6346, pp. 427–438. Springer, Heidelberg (2010)
7. Belazzougui, D., Boldi, P., Vigna, S.: Dynamic Z-fast tries. In: Chavez, E., Lonardi, S. (eds.) SPIRE 2010. LNCS, vol. 6393, pp. 159–172. Springer, Heidelberg (2010)
8. Ben-Kiki, O., Bille, P., Breslauer, D., Gasieniec, L., Grossi, R., Weimann, O.: Optimal Packed String Matching. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011). Leibniz International Proceedings in Informatics (LIPIcs), vol. 13, pp. 423–432 (2011)
9. Bille, P.: Fast searching in packed strings. In: Kucherov, G., Ukkonen, E. (eds.) CPM 2009 Lille. LNCS, vol. 5577, pp. 116–126. Springer, Heidelberg (2009)
10. Breslauer, D., Gąsieniec, L., Grossi, R.: Constant-time word-size string matching. In: Kärkkäinen, J., Stoye, J. (eds.) CPM 2012. LNCS, vol. 7354, pp. 83–96. Springer, Heidelberg (2012)
11. Carter, L., Wegman, M.N.: Universal classes of hash functions (extended abstract). In: STOC, pp. 106–112 (1977)

12. Chan, T.M., Larsen, K.G., Patrascu, M.: Orthogonal range searching on the ram, revisited. In: Symposium on Computational Geometry, pp. 1–10 (2011)
13. Chazelle, B.: Filtering search: A new approach to query-answering. SIAM J. Comput. 15(3), 703–724 (1986)
14. Crochemore, M., Czumaj, A., Sieniec, L.G., Jarominek, S., Lecroq, T., Plandowski, W., Rytter, W.: Fast multi-pattern matching. Rapport I.G.M. 93-3, Université de Marne-la-Vallée (1993)
15. Crochemore, M., Rytter, W.: Text algorithms. Oxford University Press (1994)
16. Czumaj, A., Crochemore, M., Gasieniec, L., Jarominek, S., Lecroq, T., Plandowski, W., Rytter, W.: Speeding up two string-matching algorithms. Algorithmica 12, 247–267 (1994)
17. Farach, M.: Optimal suffix tree construction with large alphabets. In: FOCS, pp. 137–143 (1997)
18. Fredriksson, K.: Faster string matching with super-alphabets. In: Laender, A.H.F., Oliveira, A.L. (eds.) SPIRE 2002. LNCS, vol. 2476, pp. 44–57. Springer, Heidelberg (2002)
19. Grossi, R., Vitter, J.S.: Compressed suffix arrays and suffix trees with applications to text indexing and string matching. SIAM J. Comput. 35(2), 378–407 (2005)
20. Hagerup, T., Miltersen, P.B., Pagh, R.: Deterministic dictionaries. J. Algorithms 41(1), 69–85 (2001)
21. Hagerup, T., Tholey, T.: Efficient minimal perfect hashing in nearly minimal space. In: Ferreira, A., Reichel, H. (eds.) STACS 2001. LNCS, vol. 2010, pp. 317–326. Springer, Heidelberg (2001)
22. Hon, W.-K., Lam, T.W., Sadakane, K., Sung, W.-K., Yiu, S.-M.: A space and time efficient algorithm for constructing compressed suffix arrays. Algorithmica 48(1), 23–36 (2007)
23. Hon, W.-K., Sadakane, K., Sung, W.-K.: Breaking a time-and-space barrier in constructing full-text indices. SIAM J. Comput. 38(6), 2162–2178 (2009)
24. Knuth, D.E., Morris Jr, J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM J. Comput. 6(1), 323–350 (1977)
25. Lecroq, T.: Recherches de mot. Thèse de doctorat, Université d'Orléans, France (1992)
26. Manber, U., Myers, E.W.: Suffix arrays: A new method for on-line string searches. SIAM J. Comput. 22(5), 935–948 (1993)
27. Morris Jr., J.H., Pratt, V.R.: A linear pattern-matching algorithm. Report 40, University of California, Berkeley (1970)
28. Morrison, D.R.: Patricia-practical algorithm to retrieve information coded in alphanumeric. J. ACM 15(4), 514–534 (1968)
29. Navarro, G., Raffinot, M.: Fast and flexible string matching by combining bit-parallelism and suffix automata. ACM Journal of Experimental Algorithmics 5, 4 (2000)
30. Ružić, M.: Constructing efficient dictionaries in close to sorting time. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 84–95. Springer, Heidelberg (2008)
31. Ruzic, M.: Making deterministic signatures quickly. ACM Transactions on Algorithms 5(3) (2009)
32. Sadakane, K.: Compressed suffix trees with full functionality. Theory Comput. Syst. 41(4), 589–607 (2007)
33. Yao, A.C.: The complexity of pattern matching for a random string. SIAM J. Comput. 8(3), 368–387 (1979)
34. Shi, Q., JáJá, J.: Novel Transformation Techniques Using Q-Heaps with Applications to Computational Geometry. SIAM J. Comput. 34(6), 1474–1492 (2005)

Parameterized Complexity of DAG Partitioning

René van Bevern[1,⋆], Robert Bredereck[1,⋆⋆], Morgan Chopin[2,⋆⋆⋆], Sepp Hartung[1], Falk Hüffner[1,†], André Nichterlein[1], and Ondřej Suchý[3,‡]

[1] Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany
{rene.vanbevern,robert.bredereck,sepp.hartung, falk.hueffner,andre.nichterlein}@tu-berlin.de
[2] LAMSADE, Université Paris-Dauphine, France
chopin@lamsade.dauphine.fr
[3] Faculty of Information Technology, Czech Technical University in Prague, Prague, Czech Republic
ondrej.suchy@fit.cvut.cz

Abstract. The goal of tracking the origin of short, distinctive phrases (*memes*) that propagate through the web in reaction to current events has been formalized as DAG Partitioning: given a directed acyclic graph, delete edges of minimum weight such that each resulting connected component of the underlying undirected graph contains only one sink. Motivated by NP-hardness and hardness of approximation results, we consider the parameterized complexity of this problem. We show that it can be solved in $O(2^k \cdot n^2)$ time, where k is the number of edge deletions, proving fixed-parameter tractability for parameter k. We then show that unless the Exponential Time Hypothesis (ETH) fails, this cannot be improved to $2^{o(k)} \cdot n^{O(1)}$; further, DAG Partitioning does not have a polynomial kernel unless NP $\subseteq$ coNP/poly. Finally, given a tree decomposition of width w, we show how to solve DAG Partitioning in $2^{O(w^2)} \cdot n$ time, improving a known algorithm for the parameter pathwidth.

1 Introduction

The motivation of our problem comes from a data mining application. Leskovec et al. [6] want to track how short phrases (typically, parts of quotations) show up on different news sites, sometimes in mutated form. For this, they collected from 90 million articles phrases of at least four words that occur at least ten times. They then created a directed graph with the phrases as vertices and draw an arc from phrase p to phrase q if p is shorter than q and either p has small edit distance from q (with words as tokens) or there is an overlap of at least 10 consecutive words. Thus, an arc (p, q) indicates that p might originate from q.

⋆ Supported by DFG project DAPA (NI 369/12-1).
⋆⋆ Supported by DFG project PAWS (NI 369/10-1).
⋆⋆⋆ Supported by DAAD.
† Supported by DFG project PABI (NI 369/7-2).
‡ A major part of this work was done while with the TU Berlin, supported by the DFG project AREG (NI 369/9).

Since all arcs are from shorter to longer phrases, the graph is a directed acyclic graph (DAG). The arcs are weighted according to the edit distance and the frequency of q. A vertex with no outgoing arc is called a *sink*. If a phrase is connected to more than one sink, its ultimate origin is ambiguous. To resolve this, Leskovec et al. [6] introduce the following problem.

DAG Partitioning [6]
Input: A directed acyclic graph $D = (V, A)$ with positive integer edge weights $\omega : A \to \mathbb{N}$ and a positive integer $k \in \mathbb{N}$.
Output: Is there a set $A' \subseteq A$, $\sum_{a \in A'} \omega(a) \leq k$, such that each connected component in $D' = (V, A \setminus A')$ has exactly one sink?

While the work of Leskovec et al. [6] had a large impact (for example, it was featured in the New York Times), there are few studies on the computational complexity of DAG Partitioning so far. Leskovec et al. [6] show that DAG Partitioning is NP-hard. Alamdari and Mehrabian [1] show that moreover it is hard to approximate in the sense that if P $\neq$ NP, then for any fixed $\varepsilon > 0$, there is no $(n^{1-\varepsilon})$-approximation, even if the input graph is restricted to have unit weight arcs, maximum outdegree three, and two sinks.

In this paper, we consider the parameterized complexity of DAG Partitioning. (We assume familiarity with parameterized analysis and concepts such as problem kernels (see e. g. [4, 7])). Probably the most natural parameter is the maximum weight k of the deleted edges; edges get deleted to correct errors and ambiguity, and we can expect that for sensible inputs only few edges need to be deleted.

Unweighted DAG Partitioning is similar to the well-known Multiway Cut problem: given an undirected graph and a subset of the vertices called the *terminals*, delete a minimum number k of edges such that each terminal is separated from all others. DAG Partitioning in a connected graph can be considered as a Multiway Cut problem with the sinks as terminals and the additional constraint that not all edges going out from a vertex may be deleted, since this creates a new sink. Xiao [8] gives a fixed-parameter algorithm for solving Multiway Cut in $O(2^k \cdot n^{O(1)})$ time. We show that a simple branching algorithm solves DAG Partitioning in the same running time (Theorem 3). We also give a matching lower bound: unless the Exponential Time Hypothesis (ETH) fails, DAG Partitioning cannot be solved in $O(2^{o(k)} \cdot n^{O(1)})$ time (Corollary 1). We then give another lower bound for this parameter by showing that DAG Partitioning does not have a polynomial kernel unless NP $\subseteq$ coNP/poly (Theorem 5).

An alternative parameterization considers the structure of the underlying undirected graph of the input. Alamdari and Mehrabian [1] show that if this graph has pathwidth ϕ, DAG Partitioning can be solved in $2^{O(\phi^2)} \cdot n$ time, and thus DAG Partitioning is fixed-parameter tractable with respect to pathwidth. They ask if DAG Partitioning is also fixed-parameter tractable with respect to the parameter treewidth. We answer this question positively by giving an algorithm based on dynamic programming that given a tree decomposition of width w solves DAG Partitioning in $O(2^{O(w^2)} \cdot n)$ time (Theorem 7).

Due to space constraints, we defer some proofs to a journal version.

2 Notation and Basic Observation

All graphs in this paper are finite and simple. We consider directed graphs $D = (V, A)$ with *vertex set* V and *arc set* $A \subseteq V \times V$, as well as undirected graphs $G = (V, E)$ with vertex set V and *edge set* $E \subseteq \{\{u, v\} \mid u, v \in V\}$. For a (directed or undirected) graph G, we denote by $G \setminus E'$ the subgraph obtained by removing from it the arcs or edges in E'. We denote by $G[V']$ the subgraph of G induced by the vertex set $V' \subseteq V$. The set of out-neighbors and in-neighbors of a vertex v in a directed graph is $N^+(v) = \{u : (v, u) \in A\}$ and $N^-(v) = \{u : (u, v) \in A\}$, respectively. Moreover, for a set of arcs B and a vertex v we let $N_B(v) := \{u \mid (u, v) \in B \text{ or } (v, u) \in B\}$ and $N_B[v] := N_B(v) \cup \{v\}$. The out-degree, the in-degree, and the degree of a vertex $v \in V$ are $d^+(v) = |N^+(v)|$, $d^-(v) = |N^-(v)|$, and $d(v) = d^+(v) + d^-(v)$, respectively. A vertex is *a sink* if $d^+(v) = 0$ and *isolated* if $d(v) = 0$. We say that *u can reach v* (*v is reachable from u*) in D if there is an oriented path from u to v in D. In particular, u is always reachable from u. Furthermore, we use *connected component* as an abbreviation for *weakly connected component*, that is, a connected component in the underlying undirected graph. The *diameter* of D is the maximum length of a shortest path between two different vertices in the underlying undirected graph of D.

The following easy to prove structural result about minimal DAG Partitioning solutions is fundamental to our work.

Lemma 1. *Any minimal solution for* DAG Partitioning *has exactly the same sinks as the input.*

Proof. Clearly, no sink can be destroyed. It remains to show that no new sinks are created. Let $D = (V, A)$ be a DAG and $A' \subseteq A$ a minimal set such that $D' = (V, A \setminus A')$ has exactly one sink in each connected component. Suppose for a contradiction that there is a vertex t that is a sink in D' but not in D. Then there exists an arc $(t, v) \in A$ for some $v \in V$. Let C_v and C_t be the connected components in D' containing v and t respectively and let t_v be the sink in C_v. Then, for $A'' := A' \setminus \{(t, v)\}$, $C_v \cup C_t$ is one connected component in $(V, A \setminus A'')$ having one sink t_v. Thus, A'' is also a solution with $A'' \subsetneq A'$, a contradiction. □

3 Classical Complexity

Since DAG Partitioning is shown to be NP-hard in general, we determine whether relevant special cases are efficiently solvable. Alamdari and Mehrabian [1] already showed that DAG Partitioning is NP-hard even if the input graph has two sinks. We complement these negative results by showing that the problem remains NP-hard even if the diameter or the maximum degree is a constant.

Theorem 1. DAG Partitioning *is solvable in polynomial time on graphs of diameter one, but NP-complete on graphs of diameter two.*

Theorem 1 can be proven by reducing from general DAG Partitioning and by adding a gadget that ensures diameter two.

Theorem 2. DAG Partitioning *is solvable in linear time if D has maximum degree two, but NP-complete on graphs of maximum degree three.*

Proof. Any graph of maximum degree two consists of cycles or paths. Thus, the underlying graph has treewidth at most two and we can therefore solve the problem in linear time using Theorem 7.

We prove the NP-hardness on graphs of maximum degree three. To this end, we use the reduction from Multiway Cut to DAG Partitioning by Leskovec et al. [6]. In the instances produced by this reduction, we then replace vertices of degree greater than three by equivalent structures of maximum degree three.

Multiway Cut
Input: An undirected graph $G = (V, E)$, a weight function $w : E \to \mathbb{N}$, a set of terminals $T \subseteq V$, and an integer k.
Output: Is there a subset $E' \subseteq E$ with $\sum_{e \in E'} w(e) \leq k$ such that the removal of E' from G disconnects each terminals from all the others?

We first recall the reduction from Multiway Cut to DAG Partitioning. Let $I = (G = (V, E), w, T, k)$ be an instance of Multiway Cut. Since Multiway Cut remains NP-hard for three terminals and unit weights [3], we may assume that $w(e) = 1$ for all $e \in E$ and $|T| = 3$. We now construct the instance $I' = (D = (V', E'), k')$ of DAG Partitioning from I as follows. Add three vertices r_1, r_2, r_3 forming the set V_1, a vertex v' for each vertex $v \in V$ forming the set V_2, and a vertex $e_{\{u,v\}}$ for every edge $\{u, v\} \in E$ forming the set V_3. Now, for each terminal $t_i \in T$ insert the arc (t'_i, r_i) in E'. For each vertex $v \in V \setminus T$, add the arcs (v', r_i) for $i = 1, 2, 3$. Finally, for every edge $\{u, v\} \in E$ insert the arcs $(e_{\{u,v\}}, u')$ and $(e_{\{u,v\}}, v')$. Set $k' = k + 2(n - 3)$. We claim that I is a yes-instance if and only if I' is a yes-instance.

Suppose that there is a solution $S \subseteq E$ of size at most k for I. Then the following yields a solution of size at most k' for I': If a vertex $v \in V$ belongs to the same component as terminal t_i, then remove every arc (v', r_j) with $j \neq i$. Furthermore, for each edge $\{u, v\} \in S$ remove one of the two arcs $(e_{\{u,v\}}, u')$ and $(e_{\{u,v\}}, v')$. One can easily check that we end up with a valid solution for I'. Conversely, suppose that we are given a minimal solution of size at most k' for I'. Notice that one has to remove at least two of the three outgoing arcs of each vertex $v' \in V_2$ and that we cannot discard all three because, contrary to Lemma 1, this would create a new sink. Thus, we can define the following valid solution for I: remove an edge $\{u, v\} \in E$ if and only if one of the arcs $(e_{\{u,v\}}, u')$ and $(e_{\{u,v\}}, v')$ is deleted. Again the correctness can easily be verified.

It remains now to modify the instance I' to get a new instance I'' of maximum degree three. For each vertex $v \in V'$ with $|N^-(v)| = |\{w_1, \ldots, w_{d^-(v)}\}| \geq 2$, do the following: For $j = 2, \ldots, d^-(v)$ remove the arc (w_j, v) and add the vertex w'_j together with the arc (w_j, w'_j). Moreover, add the arcs (w_1, w'_2), $(w_{d^-(v)}, v)$, and (w'_j, w'_{j+1}) for each $j = 2, \ldots, d^-(v) - 1$. Now, every vertex has maximum degree four. Notice that, by Lemma 1, among the arcs introduced so far, only the arcs (w_j, w'_j) can be deleted, as otherwise we would create new sinks. The correspondence between deleting the arc (w_j, w'_j) in the modified instance and

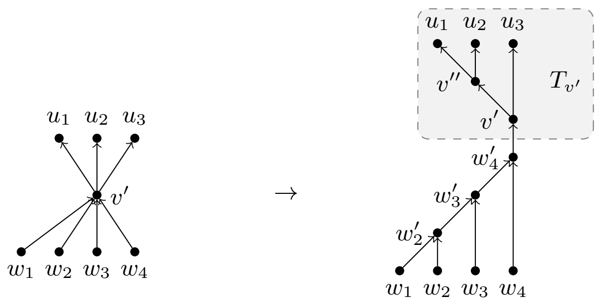


Fig. 1. Construction of the "tree structure" for a vertex $v' \in V_2$

deleting the arc (w_j, v) in I' is easy to see. Thus the modified instance is so far equivalent to I. Notice also that the vertices in I'' that have degree larger than three are exactly the degree-four vertices in V_2. In order to decrease the degree of these vertices, we carry out the following modifications. For each vertex $v' \in V_2$, with $N^+(v') = \{u_1, u_2, u_3\}$, remove (v', u_1) and (v', u_2), add a new vertex v'', and insert the three arcs (v', v''), (v'', u_1), and (v'', u_2) (see Figure 1). This concludes the construction of I'' and we now prove the correctness. Let $T_{v'}$ be the subgraph induced by $\{v', v'', u_1, u_2, u_3\}$ where $v' \in V_2$. It is enough to show that exactly two arcs from $T_{v'}$ have to be removed in such a way that there remains only one path from v' to exactly one of the u_i. Indeed, we have to remove at least two arcs: otherwise, two sinks will belong to the same connected component. Next, due to Lemma 1, it is not possible to remove more than two arcs from $T_{v'}$. Moreover, using again Lemma 1, the two discarded arcs leave a single path from v' to exactly one of the u_i. This completes the proof. □

4 Parameterized Complexity: Bounded Solution Size

In this section, we investigate the influence of the parameter solution size k on the complexity of DAG Partitioning. To this end, notice that in Lemma 1 we proved that any minimal solution does not create new sinks. Hence, the task is to separate at minimum cost the existing sinks by deleting arcs without creating new ones. Note that this is very similar to the Multiway Cut problem: In Multiway Cut the task is to separate at minimum cost the given terminals. Multiway Cut was shown by Xiao [8] to be solvable in $O(2^k \min(n^{2/3}, m^{1/2})nm)$ time. However, the algorithm relies on minimum cuts and is rather complicated. In contrast, by giving a simple search tree algorithm running in $O(2^k \cdot n^2)$ time for DAG Partitioning, we show that the additional constraint to not create new sinks makes the problem arguably simpler.

Our search tree algorithm exploits the fact that no new sinks are created in the following way: Assume that there is a vertex v with only one outgoing arc pointing to a vertex u. Then, for any minimal solution, u and v are in the same connected component. This leads to the following data reduction rule, which enables us to provide the search tree algorithm. It is illustrated in Figure 2.

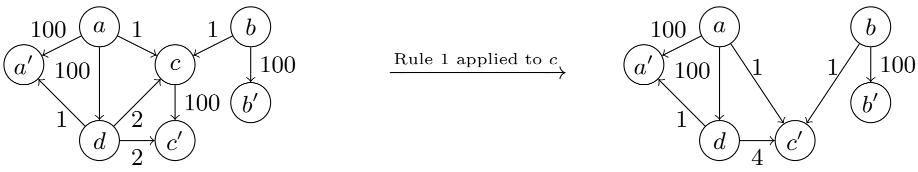


Fig. 2. Exemplary DAG Partitioning instance. Reduction Rule 1 transforms the instance into an equivalent instance: All paths from any vertex to c' can be disconnected with the same costs as before. In every solution with costs less than 100, a must be in the same component as a' and b must be in the same component as b'. Let $S = \{a', b', c'\}$ be the sinks. Furthermore, d is a sink in $D[V \setminus S]$ which must be disconnected from all but one sink. However, since a is adjacent to d removing the cheapest arc (d, a') does not lead to a solution, because a' and c' would remain in the same component. In contrast, by removing (a, c'), (b, c') and (d, c') one obtains the unique solution, which has cost 6.

Reduction Rule 1. *Let $v \in V$ be a vertex with outdegree one and u its unique out-neighbor. Then for each arc $(w, v) \in A$, add an arc (w, u) with the same weight. If there already is an arc (w, u) in A, then increase the weight of (w, u) by $\omega(w, v)$. Finally, delete v.*

The correctness of this rule follows from the discussion above. Clearly, to a vertex $v \in V$, it can be applied in $O(n)$ time. Thus, in $O(n^2)$ time, the input graph can be reduced until no further reduction by Reduction Rule 1 is possible. Thereafter, each vertex has at least two outgoing arcs and, thus, there is a vertex that has at least two sinks as out-neighbors. Exactly this fact we exploit for a search tree algorithm, yielding the following theorem:

Theorem 3. DAG Partitioning *can be solved in $O(2^k \cdot n^2)$ time.*

Proof. Since each connected component can be treated independently, we assume without loss of generality that the input graph D is connected. Moreover, we assume that Reduction Rule 1 is not applicable to D.

Let S be the set of sinks in D. If all vertices are sinks, then D has only one vertex and we are done. Otherwise, let r be a sink in $D[V \setminus S]$ (such a sink exists, since any subgraph of an acyclic graph is acyclic). Then the d arcs going out of r all end in sinks, that is, r is directly connected to d sinks. Since Reduction Rule 1 is not applicable, $d > 1$, and at most one of the d arcs may remain. We recursively branch into d cases, each corresponding to the deletion of $d - 1$ arcs. In each branch, k is decreased by $d - 1$ and the recursion stops as soon as it reaches 0. Thus, we have a branching vector (see e. g. [7]) of $(\underbrace{d-1, d-1, \ldots, d-1}_{d})$, which yields a branching number not worse than 2, since $2^{d-1} \geq d$ for every $d \geq 2$. Therefore, the size of the search tree is bounded by $O(2^k)$. Fully executing Reduction Rule 1 and finding r can all be done in $O(n^2)$ time. □

Limits of Kernelization and Parameterized Algorithms. In the remainder of this section we investigate the theoretical limits of kernelization and parameterized algorithms with respect to the parameter k. Specifically, we show that unless the exponential time hypothesis (ETH) fails, DAG Partitioning cannot be solved in subexponential time, that is, the running time stated in Theorem 3 cannot be improved to $2^{o(k)}\text{poly}(n)$. Moreover, by applying a framework developed by Bodlaender et al. [2] we prove that DAG Partitioning, unless NP $\subseteq$ coNP/poly, does not admit a polynomial kernel with respect to k.

Towards both results, we first recall the Karp reduction (a polynomial-time computable many-to-one reduction) from 3-SAT into DAG Partitioning given by Alamdari and Mehrabian [1]. The 3-SAT problem is, given a formula F in conjunctive normal form with at most three literals per clause, to decide whether F admits a satisfying assignment.

Lemma 2 ([1, Sect. 2]). *There is a Karp reduction from* 3-SAT *to* DAG Partitioning.

Proof. We briefly recall the construction given by Alamdari and Mehrabian [1].

Let φ be an instance of 3-SAT with the variables $x_1, \dots, x_n$ and the clauses $C_1, \dots C_m$. We construct an instance (D, ω, k) with $k := 4n + 2m$ for DAG Partitioning that is a yes-instance if and only if φ is satisfiable. Therein, the weight function ω will assign only two different weights to the arcs: A *normal arc* has weight one and a *heavy arc* has weight $k + 1$ and thus cannot be contained in any solution.

Construction: We start constructing the DAG D by adding the special vertices f, f', t and t' together with the heavy arcs (f, f') and (t, t'). The vertices f' and t' will be the only sinks in D. For each variable x_i, introduce the vertices x_i^t, x_i^f, x_i and $\overline{x_i}$ together with the heavy arcs (t, x_i^t) and (f, x_i^f) and the normal arcs (x_i^t, x_i), $(x_i^t, \overline{x_i})$, (x_i^f, x_i), $(x_i^f, \overline{x_i})$, (x_i, f'), $(\overline{x_i}, f')$, (x_i, t'), and $(\overline{x_i}, t')$. For each clause C, add a vertex C together with the arc (t', C). Finally, for each clause C and each variable x_i, if the positive (or negative) literal of x_i appears in C, then add the arc (C, x_i) ($(C, \overline{x_i})$, resp.). This completes the construction of D.

Correctness: One can prove that (D, ω, k) is a yes-instance for DAG Partitioning if and only if φ is satisfiable.

Limits of Parameterized Algorithms. The Exponential Time Hypothesis (ETH) was introduced by Impagliazzo et al. [5] and states that 3-SAT cannot be solved in $2^{o(n)}\text{poly}(n)$ time, where n denotes the number of variables.

Corollary 1. *Unless the ETH fails,* DAG Partitioning *cannot be solved in* $2^{o(k)}\textit{poly}(n)$ *time.*

Proof. The reduction provided in the proof of Lemma 2 reduces an instance of 3-SAT consisting of a formula with n variables to an equivalent instance (D, ω, k) of DAG Partitioning with $k = 4n + 2m$. In order to prove Corollary 1, it remains to show that we can upper-bound k by a linear function in n. Fortunately, this is done by the so-called *Sparsification Lemma* [5], which allows us to assume that the number of clauses in the 3-SAT instance that we reduce from is linearly bounded in the number of variables. □

Limits of Problem Kernelization. We first recall the basic concepts and the main theorem of the framework introduced by Bodlaender et al. [2].

Theorem 4 ([2, Corollary 10]). *If some set $L \subseteq \Sigma^*$ is NP-hard under Karp reductions and L cross-composes into the parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$, then there is no polynomial-size kernel for Q unless NP $\subseteq$ coNP/poly.*

Here, a problem L *cross-composes* into a parameterized problem Q if there is a polynomial-time algorithm that transform the instances $I_1, \ldots, I_s$ of Q into an instance (I, k) for L such that k is bounded by a polynomial in $\max_{i=1}^{s} |I_j| + \log s$ and $(I, k) \in L$ if and only if there is an instance $I_j \in Q$, where $1 \leq j \leq s$. Furthermore, it is allowed to assume that the input instances $I_1, \ldots, I_s$ belong to the same equivalence class of a *polynomial equivalence relation* $R \subseteq \Sigma^* \times \Sigma^*$, that is, an equivalence relation such that it can be decided in polynomial time whether two inputs are equivalent and each set $S \subseteq \Sigma^*$ is partitioned into at most $\max_{x \in S}(|x|)^{O(1)}$ equivalence classes.

In the following we show that 3-SAT cross-composes into DAG Partitioning parameterized by k. In particular, we show how the reduction introduced in Lemma 2 can be extended to a cross-composition.

Lemma 3. 3-SAT *cross-composes to* DAG Partitioning *parameterized by k.*

Proof. Let $\varphi_1, \ldots, \varphi_s$ be instances of 3-SAT. We assume that each of them has n variables and m clauses. This is obviously a polynomial equivalence relation. Moreover, we assume that s is a power of two, as otherwise we can take multiple copies of one of the instances. We construct a DAG $D = (V, A)$ that forms together with an arc-weight function ω and $k := 4n + 2m + 4\log s$ an instance of DAG Partitioning that is a yes-instance if and only if φ_i is satisfiable for at least one $1 \leq i \leq s$.

Construction: For each instance φ_i let D_i be the DAG constructed as in the proof of Lemma 2. Note that (D_i, k') with $k' := 4n + 2m$ is a yes-instance if and only if φ_i is satisfiable. To distinguish them between multiple instances, we denote the special vertices f, f', t, and t' in D_i by f_i, f'_i, t_i, and t'_i. For all $1 \leq i \leq s$, we add D_i to D and we identify the vertices $f_1, f_2, \ldots, f_t$ to a vertex f and, analogously, we identify the vertices $f'_1, f'_2, \ldots, f'_t$ to a vertex f'. Furthermore, we add the vertices t, t', and t'' together with the heavy arcs (t, t'') and (t, t') to D. As in the proof of Lemma 2, a heavy arc has weight $k + 1$ and thus cannot be contained in any solution. All other arcs, called normal, have weight one.

Add a balanced binary tree O with root in t'' and the leaves $t_1, \ldots, t_s$ formed by normal arcs which are directed from the root to the leaves. For each vertex in O, except for t'', add a normal arc from f. Moreover, add a balanced binary tree I with root t' and the leaves $t'_1, \ldots, t'_s$ formed by normal arcs that are directed from the leaves to the root. For each vertex in I, except for t', add a normal arc to f'. This completes the construction of D.

Correctness: One can prove that (D, ω, k) is a yes-instance if and only if φ_i is satisfiable for at least one $1 \leq i \leq s$. □

Lemma 3 and Theorem 4 therefore yield:

Theorem 5. DAG Partitioning *does not have a polynomial-size kernel with respect to k, unless NP $\subseteq$ coNP/poly.*

We note that Theorem 5 can be strengthened to graphs of constant diameter or unweighted graphs.

5 Partitioning DAGs of Bounded Treewidth

In the meme tracking application, edges always go from longer to shorter phrases by omitting or modifying words. It is thus plausible that the number of phrases of some length on a path between two phrases is bounded. Thus, the underlying graphs are tree-like and in particular have bounded treewidth. In this section, we investigate the computational complexity of DAG Partitioning measured in terms of distance to "tree-like" graphs. Specifically, we show that if the input graph is indeed a tree with uniform edge weights, then we can solve the instance in linear time by data reduction rules (see Theorem 6). Afterwards, we prove that this can be extended to weighted graphs of constant treewidth and, actually, we show that DAG Partitioning is fixed-parameter tractable with respect to treewidth. This improves the algorithm for pathwidth given by Alamdari and Mehrabian [1], as the treewidth of a graph is at most its pathwidth.

Warm-Up: Partitioning Trees

Theorem 6. DAG Partitioning *is solvable in linear time if the underlying undirected graph is a tree with uniform edge weights.*

To prove Theorem 6, we employ data reduction on the tree's leaves. Note that Reduction Rule 1 removes all leaves of a tree that have outdegree one and leaves that are the only out-neighbor of their parent. In this case, Reduction Rule 1 can be realized by merely deleting such leaves. In cases where Reduction Rule 1 is not applicable to any leaves, we apply the following data reduction rule:

Reduction Rule 2. *Let $v \in V$ be a leaf with in-degree one and in-neighbor w. If w has more than one out-neighbor, then delete v and decrement k by one.*

We can now prove that as long as the tree has leaves, one of Reduction Rule 1 and Reduction Rule 2 applies, thus proving Theorem 6.

Partitioning DAGs of Bounded Treewidth. We note without proof that DAG Partitioning can be characterized in terms of monadic second-order logic (MSO), hence it is FPT with respect to treewidth (see e.g. [7]). However, the running time bound that this approach yields is far from practical. Therefore, we give an explicit dynamic programming algorithm.

Theorem 7. *Given a tree decomposition of the underlying undirected graph of width w,* DAG Partitioning *can be solved in $O(2^{O(w^2)} \cdot n)$ time. Hence, it is fixed-parameter tractable with respect to treewidth.*

Suppose we are given $D = (V, A)$, ω, k and a tree decomposition (T, β) for the underlying undirected graph of D of width w. Namely, T is a tree and β is a mapping that assigns to every node x in $V(T)$ a set $V_x = \beta(x) \subseteq V$ called *a bag* (for more details on treewidth see e. g. [7]). We assume without loss of generality, that the given tree decomposition is nice and is rooted in a vertex r with an empty bag. For a node x of $V(T)$, we denote by U_x the union of V_y over all y descendant of x, A_x^V denotes $A(D[V_x])$, and A_x^U denotes $A(D[U_x])$.

Furthermore, for a DAG G we define the transitive closure with respect to the reachability as $\text{Reach}^*(G) := (V(G), A(G) \cup \{(u, v) \mid u, v \in V(G), u \neq v, \text{ and there is an oriented path from } u \text{ to } v \text{ in } G\})$.

Solution Patterns. Our algorithm is based on leaf to root dynamic programming. The behavior of a partial solution is described by a structure which we call a pattern. Let x be a node of T and V_x its bag. Let P be a DAG with $V(P)$ consisting of V_x and at most $|V_x|$ additional vertices such that each vertex in $V(P) \setminus V_x$ is a non-isolated sink. Let $\mathcal{Q}$ be a partition of $V(P)$ into at most $|V_x|$ sets $Q_1, \ldots, Q_q$, such that each connected component of P is within one set of $\mathcal{Q}$ and each Q_i contains at most one vertex of $V(P) \setminus V_x$. Let R be a subgraph of $P[V_x]$. We call $(P, \mathcal{Q}, R)$ a *pattern* for x. In the next paragraphs, we give an account of how the pattern $(P, \mathcal{Q}, R)$ describes a partial solution.

Intuitively, the DAG P stores the vertices of a bag and the sinks these vertices can reach in the graph modified by the partial solution. The partition $\mathcal{Q}$ refers to a possible partition of the vertices of the graph such that each part is a connected component and each connected component contains exactly one sink. Finally, the graph R is the intersection of the partial solution with V_x.

Formally, a pattern describes a partial solution as follows. Let $A' \subseteq A_x^U$ be a set of arcs such that no connected component of $D_x(A') = D[U_x] \setminus A'$ contains two different sinks in $U_x \setminus V_x$ and every sink in a connected component of $D_x(A')$ which contains a vertex of V_x can be reached from some vertex of V_x in $D_x(A')$. A sink in $U_x \setminus V_x$ is called *interesting* in $D_x(A')$ if it is reachable from at least one vertex of V_x. Let $P_x(A')$ be the DAG on $V_x \cup V'$, where V' is the set of interesting sinks in $D_x(A')$ such that there is an arc (u, v) in $P_x(A')$ if the vertex u can reach the vertex v in the DAG $D_x(A')$. Let $\mathcal{Q}_x(A')$ be the partition of $V_x \cup V'$ such that the vertices u and v are in the same set of $\mathcal{Q}_x(A')$ if and only if they are in the same connected component of $D_x(A')$. Finally, by $R_x(A')$ we denote the DAG $D[V_x] \setminus A'$.

Let $(P, \mathcal{Q}, R)$ be a pattern for x. We say that A' *satisfies* the pattern $(P, \mathcal{Q}, R)$ at x if no connected component of $D_x(A')$ contains two different sinks in $U_x \setminus V_x$, every sink in a connected component of $D_x(A')$ which contains a vertex of V_x can be reached from some vertex of V_x in $D_x(A')$, $P = P_x(A')$ (there is an isomorphism between P and $P_x(A')$ which is identical on V_x, to be precise), $\mathcal{Q}$ is a coarsening of $\mathcal{Q}_x(A')$, and $R = R_x(A')$. Formally, $\mathcal{Q}$ is a coarsening of $\mathcal{Q}_x(A')$ if for each set $Q \in \mathcal{Q}_x(A')$ there exists a set $Q' \in \mathcal{Q}$ such that $Q \subseteq Q'$. Note, that some coarsenings of $\mathcal{Q}_x(A')$ may not form a valid pattern together with $P_x(A')$ and $R_x(A')$.

For each node x of $V(T)$ we have a single table Tab_x indexed by all possible patterns for x. The entry $\mathit{Tab}_x(P, \mathcal{Q}, R)$ stores the minimum weight of a set A' that satisfies the pattern $(P, \mathcal{Q}, R)$ at x. If no set satisfies the pattern, we store ∞. As the root has an empty bag, Tab_r has exactly one entry containing the minimum weight of set A' such that in $D_r(A') = D \setminus A'$ no connected component contains two sinks. Obviously, such a set forms a solution for D. Hence, once the tables are correctly filled, to decide the instance (D, ω, k), it is enough to test whether the only entry of Tab_r is at most k.

The Algorithm. Now we show how to fill the tables. First we initialize all the tables by ∞. By updating the entry $\mathit{Tab}_x(P, \mathcal{Q}, R)$ with m we mean setting $\mathit{Tab}_x(P, \mathcal{Q}, R) := m$ if $m < \mathit{Tab}_x(P, \mathcal{Q}, R)$. For a leaf node x we try all possible subsets $A' \subseteq A_x^U = A_x^V$, and for each of them and for each coarsening $\mathcal{Q}$ of $\mathcal{Q}_x(A')$ we update $\mathit{Tab}_x(P_x(A'), \mathcal{Q}, R_x(A'))$ with $\omega(A')$. Note that in this case, as there are no vertices in $P_x(A') \setminus V_x$, every coarsening of $\mathcal{Q}_x(A')$ forms a valid pattern together with $P_x(A')$ and $R_x(A')$. In the following we assume that by the time we start the computation for a certain node of T, the computations for all its children are already finished.

Consider now the case, where x is a forget node with a child y, and assume that $v \in V_y \setminus V_x$. For each pattern $(P, \mathcal{Q}, R)$ for y we distinguish several cases. In each of them we set $R' = R \setminus \{v\}$. If v is isolated in P and there is a set $\{v\}$ in $\mathcal{Q}$ (case (i)), then we let $P' = P \setminus \{v\}$, $\mathcal{Q}'$ be a partition of $V(P')$ obtained from $\mathcal{Q}$ by removing the set $\{v\}$, and update $\mathit{Tab}_x(P', \mathcal{Q}', R')$ with $\mathit{Tab}_y(P, \mathcal{Q}, R)$. If v is a non-isolated sink and $v \in Q_i \in \mathcal{Q}$ such that $Q_i \subseteq V_y$ (case (ii)), then we update $\mathit{Tab}_x(P, \mathcal{Q}, R')$ with $\mathit{Tab}_y(P, \mathcal{Q}, R)$ ($P' = P$, $\mathcal{Q}' = \mathcal{Q}$ in this case). If v is not a sink in P and there is no sink in $V(P) \setminus V_y$ such that v is its only in-neighbor (case (iii)), then let $P' = P \setminus \{v\}$ and $\mathcal{Q}'$ be a partition of $V(P')$, obtained from $\mathcal{Q}$ by removing v from the set it is in, and update $\mathit{Tab}_x(P', \mathcal{Q}', R')$ with $\mathit{Tab}_y(P, \mathcal{Q}, R)$. If there is a sink $u \in V(P) \setminus V_y$ such that v is its only in-neighbor and $\{u, v\}$ is a set of $\mathcal{Q}$ (case (iv)), then let $P' = P \setminus \{u, v\}$ and $\mathcal{Q}'$ be a partition of $V(P')$, obtained from $\mathcal{Q}$ by removing the set $\{u, v\}$, and update $\mathit{Tab}_x(P', \mathcal{Q}', R')$ with $\mathit{Tab}_y(P, \mathcal{Q}, R)$. We don't do anything for the patterns $(P, \mathcal{Q}, R)$ which do not satisfy any of the above conditions.

Next, consider the case, where x is an introduce node with a child y, and assume that $v \in V_x \setminus V_y$ and B is the set of arcs of A_x^V incident to v. For each $B' \subseteq B$ and for each pattern $(P, \mathcal{Q}, R)$ for y such that there is a Q_i in $\mathcal{Q}$ with $N_{B'}(v) \subseteq Q_i$ we let $R' = (V_x, A(R) \cup B')$, $D' = (V(P) \cup \{v\}, A(P) \cup B')$, $P' = \mathrm{Reach}^*(D')$ and we distinguish two cases. If $B' = \emptyset$, then for every $Q_i \in \mathcal{Q}$ we let $\mathcal{Q}'$ be obtained from $\mathcal{Q}$ by adding v to the set Q_i and update $\mathit{Tab}_x(P', \mathcal{Q}', R')$ with $\mathit{Tab}_y(P, \mathcal{Q}, R) + \omega(B)$. Additionally, for $\mathcal{Q}'$ obtained from $\mathcal{Q}$ by adding the set $\{v\}$ we also update $\mathit{Tab}_x(P', \mathcal{Q}', R')$ with $\mathit{Tab}_y(P, \mathcal{Q}, R) + \omega(B)$. If B' is non-empty, then let Q_i be the set of $\mathcal{Q}$ with $N_{B'}(v) \subseteq Q_i$ and let $\mathcal{Q}'$ be obtained from $\mathcal{Q}$ by adding v to the set Q_i. We update $\mathit{Tab}_x(P', \mathcal{Q}', R')$ with $\mathit{Tab}_y(P, \mathcal{Q}, R) + \omega(B) - \omega(B')$.

Finally, consider the case that x is a join node with children y and z. For each pair of patterns $(P_y, \mathcal{Q}_y, R)$ for y and $(P_z, \mathcal{Q}_z, R)$ for z such that $\mathcal{Q}_y$ and $\mathcal{Q}_z$ partition the vertices of $V_y = V_z = V_x$ in the same way we do the following. Let D' be the DAG obtained from the disjoint union of P_y and P_z by identifying the vertices in V_x and $P' = \text{Reach}^*(D')$. Let $\mathcal{Q}'$ be a partition of $V(P')$ such that it partitions V_x in the same way as $\mathcal{Q}_y$ and $\mathcal{Q}_z$ and for each $u \in V(P') \setminus V_x$ add u to a set Q_i which contain a vertex v with (v, u) being an arc of P'. It is not hard to see, that there is always exactly one such set Q_i, as there are no arcs between different sets in P_y and P_z. If some set $Q \in \mathcal{Q}'$ contains more than one vertex of $V(P') \setminus V_x$, then continue with a different pair of patterns. Otherwise, we update $\mathit{Tab}_x(P', \mathcal{Q}', R)$ with $\mathit{Tab}_y(P_y, \mathcal{Q}_y, R) + \mathit{Tab}_z(P_z, \mathcal{Q}_z, R) - \omega(A_x^V) + \omega(A(R))$.

6 Outlook

We have presented two parameterized algorithms for DAG Partitioning, one with parameter solution size k and one with parameter treewidth w. In particular the algorithm for the parameter k seems suitable for implementation; in combination with data reduction, this might allow to solve optimally instances for which so far only heuristics are employed [6].

On the theoretical side, one open question is whether we can use the Strong Exponential Time Hypothesis (SETH) to show that there is no $O((2-\varepsilon)^k \text{poly}(n))$ time algorithm for DAG Partitioning. Another question is whether there is an algorithm solving the problem in $O(2^{O(w \log w)} \cdot n)$ or even $O(2^{O(w)} \cdot n)$ time, where w is the treewidth, as the $O(2^{O(w^2)} \cdot n)$ running time of our algorithm still seems to limit its practical relevance.

References

[1] Alamdari, S., Mehrabian, A.: On a DAG partitioning problem. In: Bonato, A., Janssen, J. (eds.) WAW 2012. LNCS, vol. 7323, pp. 17–28. Springer, Heidelberg (2012)

[2] Bodlaender, H.L., Jansen, B.M.P., Kratsch, S.: Cross-composition: A new technique for kernelization lower bounds. In: Proc. 28th STACS. LIPIcs, vol. 9, pp. 165–176. Dagstuhl Publishing (2011)

[3] Dahlhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., Yannakakis, M.: The complexity of multiterminal cuts. SIAM J. Comput. 23(4), 864–894 (1994)

[4] Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)

[5] Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. System Sci. 63(4), 512–530 (2001)

[6] Leskovec, J., Backstrom, L., Kleinberg, J.M.: Meme-tracking and the dynamics of the news cycle. In: Proc.15th ACM SIGKDD, pp. 497–506. ACM (2009)

[7] Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006)

[8] Xiao, M.: Simple and improved parameterized algorithms for multiterminal cuts. Theory Comput. Syst. 46, 723–736 (2010)

Four Measures of Nonlinearity

Joan Boyar[1,⋆], Magnus Find[1,⋆⋆], and René Peralta[2]

[1] Department of Mathematics and Computer Science, University of Southern Denmark, Denmark
{joan,magnusgf}@imada.sdu.dk

[2] Information Technology Laboratory, National Institute of Standards and Technology, USA
peralta@nist.gov

Abstract. Cryptographic applications, such as hashing, block ciphers and stream ciphers, make use of functions which are simple by some criteria (such as circuit implementations), yet hard to invert almost everywhere. A necessary condition for the latter property is to be "sufficiently distant" from linear, and cryptographers have proposed several measures for this distance. In this paper, we show that four common measures, *nonlinearity, algebraic degree, annihilator immunity*, and *multiplicative complexity*, are incomparable in the sense that for each pair of measures, μ_1, μ_2, there exist functions f_1, f_2 with $\mu_1(f_1) > \mu_1(f_2)$ but $\mu_2(f_1) < \mu_2(f_2)$. We also present new connections between two of these measures. Additionally, we give a lower bound on the multiplicative complexity of collision-free functions.

1 Preliminaries

For a vector $\mathbf{x} \in \mathbb{F}_2^n$ its *Hamming weight* is the number of non-zero entries in $\mathbf{x}$. For $n \in \mathbb{N}$ its Hamming weight, $H^{\mathbb{N}}(n)$ is defined as the Hamming weight of the binary representation of n. We let $B_n = \{f : \mathbb{F}_2^n \to \mathbb{F}_2\}$ be the set of Boolean predicates on n variables.

A Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ can be uniquely represented by its *algebraic normal form* also known as its Zhegalkin polynomial [30]:

$$f(x_1, \ldots, x_n) = \bigoplus_{S \subseteq \{1,2,\ldots,n\}} \alpha_S \prod_{i \in S} x_i$$

where $\alpha_s \in \{0,1\}$ for all S and we define $\prod_{i\in\emptyset} x_i$ to be 1. If $\alpha_S = 0$ for $|S| > 1$, we say that f is *affine*. An affine function f is *linear* if $\alpha_\emptyset = 0$ or equivalently if $f(\mathbf{0}) = 0$. The function f is *symmetric* if $\alpha_S = \alpha_{S'}$ whenever $|S| = |S'|$, that is f only depends on the Hamming weight of the input. The kth *elementary symmetric Boolean function*, denoted Σ_k^n, is defined as the sum of all terms where $|S| = k$.

⋆ Partially supported by the Danish Council for Independent Research, Natural Sciences. Part of this work was done while visiting the University of Waterloo.

⋆⋆ Part of this work was done while visiting the University of Toronto.

For two functions $f, g \in B_n$ the distance d between f and g is defined as the number of inputs where the functions differ, that is

$$d(f,g) = |\{\mathbf{x} \in \mathbb{F}_2^n | f(\mathbf{x}) \neq g(\mathbf{x})\}| .$$

For the rest of this paper, unless otherwise stated, n denotes the number of input variables. We let log denote the logarithm base 2 and ln the natural logarithm.

2 Introduction

Cryptographic applications, such as hashing, block ciphers and stream ciphers, make use of functions which are simple by some criteria (such as circuit implementations) yet hard to invert almost everywhere. A necessary condition for the latter to hold is that the tools of algebra – and in particular linear algebra – be somehow not applicable to the problem of saying something about x given $f(x)$. Towards this goal, cryptographers have proposed several measures for the distance to linearity for Boolean functions. In this paper we consider four such measures. We compare and contrast them, both in general and in relation to specific Boolean functions. Additionally, we propose a procedure to find collisions when the multiplicative complexity is low.

The *nonlinearity* of a function is the Hamming distance to the closest affine function. The nonlinearity of a function on n bits is between 0 and $2^{n-1} - \lceil 2^{n/2-1} \rceil$ [26,6]. Affine functions have nonlinearity 0. Unfortunately, this introduces an overloading of the word "nonlinearity" since it also refers to the more general concept of distance to linear. The meaning will be clear from context.

Functions with nonlinearity $2^{n-1} - 2^{n/2-1}$ exist if and only if n is even. These functions are called *bent*, and several constructions for bent functions exist (see [26,21,14] or the survey by Carlet [6]). For odd n, the situation is a bit more complicated; for any bent function f on $n-1$ variables, the function $g(x_1, \ldots, x_n) = f(x_1, \ldots, x_{n-1})$ will have nonlinearity $2^{n-1} - 2^{(n-1)/2}$. It is known that for odd $n \geq 9$, this is suboptimal [17]. Despite this, no infinite family achieving higher nonlinearity is known. For a Boolean function f, there is a tight connection between the nonlinearity of f and its Fourier coefficients. More precisely the nonlinearity is determined by the largest Fourier coefficient, and for bent functions all the Fourier coefficients have the same magnitude. A general treatment on Fourier analysis, can be found in [24].

The *algebraic degree* (which we from now on will refer to as just the *degree*) of a function is the degree of its Zhegalkin polynomial, that is the largest $|S|$ such that $\alpha_S = 1$. We note that Carlet [5] has compared nonlinearity and degree to two other measures which we do not consider here, algebraic thickness and nonnormality.

The *annihilator immunity* (also known as algebraic immunity[1]) of a function f is the minimum degree of a non-zero function g such that $fg = 0$ or $(f+1)g = 0$.

[1] In this paper we use the term "annihilator immunity" rather than "algebraic immunity", see the remark in [11].

We denote this measure by $AI(f)$. The function g is called an *annihilator.* It is known that $0 \leq AI(f) \leq \lceil \frac{n}{2} \rceil$ for all functions [8,10]. Specific functions are known which achieve the upper bound [11].

The *multiplicative complexity* of a function f, denoted $c_{\wedge}(f)$, is the smallest number of AND gates necessary and sufficient to compute the function using a circuit over the basis (XOR,AND,1) (i.e. using arithmetic over $GF(2)$). Clearly, the multiplicative complexity of f is at least 0 with equality if and only if f is affine. For even n, the multiplicative complexity is at most $2^{\frac{n}{2}+1} - \frac{n}{2} - 2$, and for odd n at most $\frac{3}{2\sqrt{2}}2^{n/2+1} - \frac{n+3}{2}$ [3,22] (see also [16]). Despite this, no specific predicate has been proven to have multiplicative complexity larger than $n-1$.[2]

Nonlinearity, degree and multiplicative complexity all capture an intuitive notion of the degree of "nonlinearity" of Boolean functions. Annihilator immunity is also related to nonlinearity, albeit less obviously.

In [6], it is shown that algebraic degree, annihilator immunity, and nonlinearity are affine invariants. That is, if $L : \{0,1\}^n \to \{0,1\}^n$ is an invertible linear mapping, applying L to the input variables first does not change the value of any of these measures. It is easy to see that multiplicative complexity is also an affine invariant, since L and L^{-1} can be computed using only XOR gates.

Ideally, a measure of nonlinearity should be invariant with respect to addition of affine functions and embedding into a higher dimensional space (e.g. considering $f(x_1, x_2) = x_1x_2$ as a function of three variables). The four measures studied here have these properties with two exceptions.

- Adding an affine function l to f can cause the annihilator immunity to vary by up to 1. That is $AI(f) - 1 \leq AI(f + l) \leq AI(f) + 1$ [7];
- Embedding a function $f : \{0,1\}^n \to \{0,1\}$ in $\{0,1\}^{n+1}$ doubles its nonlinearity. Thus, if one wants to consider nonlinearity of functions embedded in larger spaces, it might be more natural to redefine nonlinearity using a normalized metric instead of the Hamming distance metric. In this paper, we will not use embeddings.

There is a substantial body of knowledge which relates nonlinearity, annihilator immunity, and algebraic degree to cryptographic properties. However, the analogous question with respect to multiplicative complexity remains little studied. Among the few published results is [9], in which Courtois et al. show (heuristically) that functions with low multiplicative complexity are less resistant against algebraic attacks. Here we present evidence that low multiplicative complexity in hash functions can make them prone to second preimage or collision attacks.

Multiplicative complexity also turns out to be important in cryptographic protocols. Several techniques for secure multi-party computation yield protocols with communication complexity proportional to the multiplicative complexity

[2] We have experimentally verified that all predicates on four bits have multiplicative complexity at most three. This is somewhat surprising, as circuit realization of random functions (e.g. $x_1x_2x_3x_4 + x_1x_2x_3 + x_2x_3x_4 + x_1x_3x_4 + x_1x_3 + x_2x_4 + x_1x_4$) would appear to need more than three AND gates. We conjecture that some predicate on five bits will turn out to have multiplicative complexity five.

of the function being evaluated (see, for example, [15,18,23]). Several flavors of one-prover non-interactive cryptographically secure proofs (of knowledge of x given $f(x)$) have length proportional to the multiplicative complexity of the underlying function f (see, for example, [1]).

In this paper we show that very low nonlinearity implies low multiplicative complexity and vice-versa. We also show an upper bound on nonlinearity for functions with very low multiplicative complexity.

For nonlinearity, annihilator immunity, and algebraic degree, there exist symmetric Boolean functions achieving the maximal value among all Boolean functions. However, the only symmetric functions which achieve maximum nonlinearity are the quadratic functions, which have low algebraic degree. In [4] Canteaut and Videau have characterized the symmetric functions with almost optimal nonlinearity. In this paper we analyze the multiplicative complexity and annihilator immunity of these functions.

3 Relations between Nonlinearity Measures

In general, random Boolean functions are highly nonlinear with respect to all these measures:

- In [13], Didier shows that the annihilator immunity of almost every Boolean function is $(1 - o(1))n/2$.
- In [25], Rodier shows that the nonlinearity of almost every function is at least $2^{n-1} - 2^{n/2-1}\sqrt{2n \ln 2}$, which is close to maximum.
- In [5], Carlet observes that almost every function has degree at least $n - 1$.
- In [3], Boyar et al. show that almost every Boolean function has multiplicative complexity at least $2^{n/2} - O(n)$.

If a function f has algebraic degree d, the multiplicative complexity is at least $d-1$ [28]. This is a very weak bound for most functions. However this technique easily yields lower bounds of $n - 1$ for many functions on n variables, and no larger lower bounds are known for concrete functions

Additionally, it has been shown that low nonlinearity implies low annihilator immunity [10]. Still, there are functions optimal with respect to annihilator immunity that have nonlinearity much worse than that of bent functions. An example of this is the majority function, see [11]. Bent functions have degree at most $\frac{n}{2}$ ([26,6]). Since $f \oplus 1$ is an annihilator for f, the annihilator immunity of a function is at most its degree.

4 Incomparability

In this section we show that our four measures are incomparable in the sense that for each pair of measures, μ_1, μ_2, there exist functions f_1, f_2 with $\mu_1(f_1) > \mu_1(f_2)$, but $\mu_2(f_1) < \mu_2(f_2)$. To show this we look at four functions:

Σ_2^n: For even n, the function Σ_2^n is bent [26]. For odd n it has nonlinearity $2^{n-1} - 2^{(n-1)/2}$, which is maximum among the symmetric functions on an odd number of variables [20]. But being a quadratic function, both the algebraic degree and the annihilator immunity are 2 which is almost as bad as for linear functions. The multiplicative complexity is $\lfloor n/2 \rfloor$, which is the smallest possible multiplicative complexity for nonlinear symmetric functions [3].

MAJ_n, which is 1 if and only if at least $n/2$ of the n inputs are 1: In [2] it is shown that when $n = 2^r + 1$, the multiplicative complexity is at least $n-2$. In [11] it is shown that MAJ_n has annihilator immunity $\lceil \frac{n}{2} \rceil$; they also show that it has nonlinearity $2^{n-1} - \binom{n-1}{\lfloor \frac{n}{2} \rfloor}$, which by Stirling's approximation is $2^{n-1} - (1+o(1))\sqrt{\frac{2}{\pi}} \frac{2^{n-1}}{\sqrt{n-1}}$.

$FMAJ_n$, defined as:

$$FMAJ_n(x_1, \ldots, x_n) = MAJ_{\lceil \log n \rceil}(x_1, \ldots, x_{\lceil \log n \rceil}) \oplus x_{\lceil \log n \rceil + 1} \oplus \ldots \oplus x_n.$$

The degree of $FMAJ_n$ is equal to the degree of $MAJ_{\lceil \log n \rceil}$ which is at least $\frac{\lceil \log n \rceil}{2}$, so the multiplicative complexity is at least $\frac{\lceil \log n \rceil}{2} - 1$. Also its multiplicative complexity is equal to that of $MAJ_{\lceil \log n \rceil}$, which is at most $\lceil \log(n) \rceil - H^{\mathbb{N}}(\lceil \log n \rceil) + \lceil \log(\lceil \log n \rceil + 1) \rceil$ [2]. The annihilator immunity of $FMAJ_n$ is at least $\left\lceil \frac{\log n}{2} \right\rceil - 1$, since $MAJ_{\lceil \log n \rceil}$ has annihilator immunity $\left\lceil \frac{\log n}{2} \right\rceil$, and $FMAJ_n$ is just $MAJ_{\lceil \log n \rceil}$ plus a linear function. This can change the annihilator immunity by at most 1 [7].

Σ_n^n : The nonlinearity of Σ_n^n is 1 because it has Hamming distance 1 to the zero function. It has annihilator immunity 1 ($x_1 \oplus 1$ is an annihilator), its algebraic degree is n, and its multiplicative complexity is $n-1$.

Incomparability Examples: From the observations above it can be seen that Σ_2^n has higher nonlinearity than MAJ_n but smaller degree, annihilator immunity, and multiplicative complexity. $FMAJ_n$ has higher degree and annihilator immunity than Σ_2^n but lower multiplicative complexity. Σ_n^n has larger degree than $FMAJ_n$ but smaller annihilator immunity. These examples are shown in Table 1.

Table 1. Incomparability examples. For every pair (f_1, f_2) f_1 scores higher in the measure for the row and f_2 scores higher in the measure for the column.

	NL	MC	deg	AI
NL	-	(Σ_2^n, MAJ_n)	(Σ_2^n, MAJ_n)	(Σ_2^n, MAJ_n)
MC	-	-	$(\Sigma_2^n, FMAJ_n)$	$(\Sigma_2^n, FMAJ_n)$
deg	-	-	-	$(\Sigma_n^n, FMAJ_n)$

Remark: These separations are fairly extreme except with respect to multiplicative complexity, where the values are small compared to those for random functions. This is due to the fact that currently no specific function has been proven to have multiplicative complexity larger than $n-1$. If larger bounds were proven, one could have more extreme separations: Suppose $f : \{0,1\}^{n-1} \to \{0,1\}$ has large multiplicative complexity, degree, nonlinearity and annihilator immunity, and let $g(x_1, \ldots, x_n) = f(x_1, \ldots, x_{n-1}) \cdot x_n$. Then clearly g has high degree, nonlinearity and multiplicative complexity, but annihilator immunity 1, since multiplying by $x_n + 1$ gives the zero function. This is also an example where the annihilator immunity fails to capture the intuitive notion of nonlinearity.

5 Relationship between Nonlinearity and Multiplicative Complexity

In this section we will show that, despite being incomparable measures, the multiplicative complexity and nonlinearity are somehow related. We first show that if a function has low nonlinearity, this gives a bound on the multiplicative complexity. Conversely we show that if the multiplicative complexity is $\frac{n-a}{2}$, the nonlinearity is at most $2^{n-1} - 2^{n/2-a/2-1}$, and this nonlinearity can be achieved by a function with this number of AND gates.

We will use the following theorem due to Lupanov [19] (see Lemma 1.2 in [16]). Given a Boolean matrix A, a *decomposition* is a set of Boolean matrices $B_1, \ldots, B_k$ each having rank 1, satisfying $A = B_1 + B_2 + \ldots + B_k$ where addition is over the reals. For each B_i its weight is defined as the number of non-zero rows plus the number of non-zero columns. The *weight* of a decomposition is the sum of the weights of the B_i's.

Theorem 1 (Lupanov). *Every Boolean $p \times q$ matrix admits a decomposition of weight*

$$(1 + o(1))\frac{pq}{\log p}.$$

Theorem 2. *A function $f \in B_n$ with nonlinearity $s > 1$ has multiplicative complexity at most* $\min\{s(n-1), (2 + o(1))\frac{sn}{\log s}\}$.

Proof. Let L be an affine function with minimum distance to f. Let

$$\epsilon(\mathbf{x}) = f(\mathbf{x}) \oplus L(\mathbf{x}).$$

Note that ϵ takes the value 1 s times. Let $\epsilon^{-1}(1)$ be the preimage of 1 under ϵ. Suppose $\epsilon^{-1}(1) = \{z^{(1)}, \ldots, z^{(s)}\}$ where each $z^{(i)}$ is an n-bit vector. Let $M_i(\mathbf{x}) = \prod_{j=1}^{n}(x_j \oplus z_j^{(i)} \oplus 1)$ be the minterm associated to $z^{(i)}$, that is the polynomial that is 1 only on $z^{(i)}$. By definition

$$\epsilon(\mathbf{x}) = \bigoplus_{i=1}^{s} M_i(\mathbf{x}) = \bigoplus_{i=1}^{s}\prod_{j=1}^{n}(x_j \oplus z_j^{(i)} \oplus 1)$$

Adding the minterms together can be done using only XOR gates and gives exactly the function ϵ. We will give two constructions for the minterms. Using the one with fewest AND gates proves the result.

The first construction simply computes each of the s minterms directly using $n-1$ AND gates for each. For the second construction, define the $s \times 2n$ matrix A where columns $1, 2, \ldots, n$ correspond to $x_1, x_2, \ldots, x_n$ and columns $n+1, \ldots, 2n$

correspond to $(1 \oplus x_1), \ldots, (1 \oplus x_n)$, and row i corresponds to minterm M_i. Let $A_{ij} = 1$ if and only if the literal corresponding to column j is a factor in the minterm M_i. Now consider the rectangular decomposition guaranteed to exist by Theorem 1. For each B_i, all non-zero columns are equal. AND together the literals corresponding to these variables. Call the result Q_i. Now each row can be seen as a logical AND of Q_i's. AND these together for every row to obtain the s results. The number of AND gates used is at most the weight of the decomposition, that is at most $(1 + o(1))\frac{2sn}{\log s}$ AND gates. $\square$

Lemma 1. *Let f have multiplicative complexity $M = \frac{n-a}{2}$, for $a \geq 0$. Then there exists an invertible linear mapping $L : \{0,1\}^n \to \{0,1\}^n$, a Boolean predicate $g \in B_D$ for $D \leq 2M$, and a set $T \subseteq \{1, 2, \ldots, n\}$ such that for $\mathbf{t} = L(\mathbf{x})$, f can be written as*

$$f(x_1, \ldots, x_n) = g(t_1, \ldots, t_D) \oplus \bigoplus_{j \in T} t_j$$

Proof. Let $M = c_\wedge(f) = \frac{n-a}{2}$ for $a \geq 0$. Consider an XOR-AND circuit C with M AND gates computing f, and let $A_1, \ldots, A_M$ be a topological ordering of the AND gates. Let the inputs to A_1 be I_1, I_2 and inputs to A_2 be I_3, I_4, etc. so A_M has inputs I_{2M-1}, I_{2M}, $2M = n - a$. Now the value of f, the output of C, can be written as a sum of some of the AND gate outputs and some of the inputs to the circuit:

$$f = \bigoplus_{i \in Z_{out}} A_i \oplus \bigoplus_{i \in X_{out}} x_i,$$

for appropriate choices of Z_{out} and X_{out}. Similarly for I_j:

$$I_j = \bigoplus_{i \in Z_j} A_i \oplus \bigoplus_{i \in X_j} x_i.$$

Define g as $g = \bigoplus_{i \in Z_{out}} A_i$. Since X_j is a subset of $\{0,1\}^n$, it can be thought of as a vector y_j in the vector space $\{0,1\}^n$ where the ith coordinate is 1 if and only if $i \in X_j$.

Clearly the dimension D of $Y = \mathit{span}(y_1, \ldots y_{2M})$ is at most $2M$. Let $\{y_{j_1}, \ldots y_{j_D}\}$ be a basis of Y. There exists some invertible linear mapping $L : \{0,1\}^n \to \{0,1\}^n$ with $L(x_1, \ldots, x_n) = (t_1, \ldots, t_n)$ having $t_j = y_{i_j}$ for $1 \leq j \leq D$. That is, g depends on just $t_1, \ldots t_D$, and each x_j is a sum of t_l's, hence f can be written as a function of $t_1, \ldots, t_n$ as

$$f = g(t_1, \ldots, t_D) \oplus \bigoplus_{j \in T} t_j \qquad \square$$

Corollary 1. *If a function $f \in B_n$ has multiplicative complexity $M = \frac{n-a}{2}$ for $a \geq 0$, it has nonlinearity at most $2^{n-1} - 2^{n/2-a/2-1}$, and this nonlinearity is achievable.*

Proof. Since nonlinearity is an affine invariant, we can use Lemma 1 and look at the nonlinearity of

$$f = g(t_1, \ldots, t_{2M}) \oplus \bigoplus_{j \in T_{out}} t_j$$

Now the best affine approximation of g agrees on at least $2^{2M-1} + 2^{M-1}$ inputs. Replacing g with its best affine approximation, we see that the nonlinearity of f is at most $2^{n-2M}(2^{2M-1} - 2^{M-1})2^{n-1} - 2^{n/2-a/2-1}$. Furthermore, if g is bent this nonlinearity is met with equality. □

Remark: This shows that Σ_2^n is optimal with respect to nonlinearity among functions having multiplicative complexity $\lfloor n/2 \rfloor$.

6 Low Multiplicative Complexity and One-Wayness

If a function f has multiplicative complexity μ, then it can be inverted (i.e. a preimage can be found) in at most 2^μ evaluations of f. To do this, consider a circuit C for f with μ AND gates. Suppose y has a non-empty preimage under f. Guessing the Boolean value of one input for each AND gate results in a linear system of equations, L. Solve L to obtain a candidate input x and test whether $f(x) = y$. This finds a preimage of y after at most 2^μ iterations. Thus, one-way functions, if they exist, have superlogarithmic multiplicative complexity.

The one-wayness requirements of hash functions include the much stronger requirement of *collision resistance*: it must be infeasible to find two inputs that map to the same output. We next observe that collision resistance of a function f with n inputs and $m < n$ outputs requires f to have multiplicative complexity at least $n - m$.

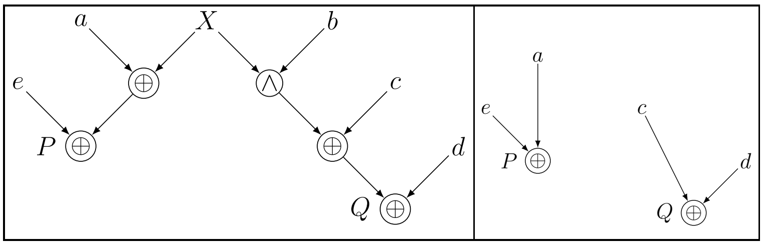


Fig. 1. The circuit to the right is the circuit obtained when X in the left circuits is restricted to the value 0. Notice that only the gates P, Q remain nonredundant.

Let C be a circuit for f. Without loss of generality, we can assume the circuit contains no negations and that we seek two distinct inputs which map to $\mathbf{0}$.[3]

[3] Negations can be "pushed" to the outputs of the circuit without changing the number of AND gates. Once at the outputs, for purposes of finding a collision, negations can be simply removed.

Since there are no negations in the circuit, one such input is $\mathbf{0}$. We next show how to obtain a second preimage of $\mathbf{0}$.

Pick a topologically minimal AND gate and set one of its inputs to 0. This generates one homogeneous linear equation on the inputs to f and allows us to remove the AND gate from the circuit (see Figure 1). Repeating this until no AND gates are left yields a homogeneous system S with at most μ equations, plus a circuit C' which computes a homogeneous linear system with m equations. The system of equations has $2^{n-m-\mu}$ distinct solutions. Thus, if $m + \mu < n$, then standard linear algebra yields non-zero solutions. These are second preimages of $\mathbf{0}$.

We re-state this as a theorem below. The idea of using hyperplane restrictions to eliminate AND gates has been used before, however with different purposes, see e.g. [2,12].

Theorem 3. *Collision resistance of a function f from n to m bits requires that f have multiplicative complexity at least $n - m$.*

It is worth noting that the bound from Theorem 3 does not take into account the position of the AND gates in the circuit. It is possible that fewer linear equations can be used to remove all AND gates. We have tried this on the reduced-round challenges issued by the Keccak designers (Keccak is the winner of the SHA-3 competition, see `http://keccak.noekeon.org/crunchy_contest.html`). These challenges are described in the notation Keccak$[r, c, nr]$ where r is the rate, c the capacity, and nr the number of rounds. For the collision challenges, the number of outputs is set to 160. Each round of Keccak uses $r + c$ AND gates. However, in the last round of Keccak the number of AND gates that affect the output bits is equal to the number of outputs.

We consider circuits for Keccak with only one block (r bits) of input. The circuit for Keccak$[r{=}1440, c{=}160, nr{=}1]$ contains 160 AND gates, yet 96 linear equations will remove them all. Keccak$[r{=}1440, c{=}160, nr{=}2]$ contains 1760 AND gates, yet 1056 linear equations removes them all. Thus, finding collisions is easy, because 1440 is greater than 160 + 1056 (in the one-round case, because 1440 $>$ 160 + 96). These two collision challenges were first solved by Morawiecki (using SAT solvers, see `http://keccak.noekeon.org/crunchy_mails/coll-r2-w1600-20110729.txt`) and, more recently, by Duc et al. (see `http://keccak.noekeon.org/crunchy_mails/coll-r1r2-w1600-20110802.txt`). Our reduction technique easily solves both of these challenges, and yields a large number of multicollisions.

Dinur et al. are able to obtain collisions for Keccak$[r{=}1440, c{=}160, nr{=}4]$ i.e. for four rounds of Keccak (see `http://keccak.noekeon.org/crunchy_mails/coll-r3r4-w1600-20111124.txt`). The technique of Theorem 3 cannot linearize the Keccak circuit for more than two rounds. How to leverage our methods to solve three or more rounds is work in progress.

7 Some Symmetric Boolean Functions with High Nonlinearity

When designing Boolean functions for cryptographic applications, we seek functions with high nonlinearity, simple structure, high annihilator immunity, and high algebraic degree. Bent functions have high nonlinearity. Symmetric functions have simple structure. However, the multiplicative complexity of a symmetric function on n variables is never larger than $n + 3\sqrt{n}$ [3]. The symmetric functions with highest nonlinearity are quadratic ([27] and [20]). But these functions have low algebraic degree, low annihilator immunity, and multiplicative complexity only $\lfloor \frac{n}{2} \rfloor$.

For $n \geq 3$, let $F_n = \bigoplus_{k=3}^{n} \Sigma_k^n$ and $G_n = \Sigma_2^n \oplus \Sigma_n^n$. It is known that there are exactly 8 symmetric functions with nonlinearity exactly 1 less than the largest achievable value. These are $F_n \oplus \lambda$ and $G_n \oplus \lambda$, where $\lambda \in \{0, 1, \Sigma_1^n, \Sigma_1^n + 1\}$ [4]. These functions have many of the criteria sought after for cryptographic functions: they are symmetric, have optimal degree, and almost optimal nonlinearity. We have exactly calculated or tightly bound the multiplicative complexity of these functions. Precise values are important for applications in secure multiparty computations.

Lemma 2. *(Proofs omitted due to space constraints)*

1. $c_\wedge(G_n) = n - 1$.
2. $c_\wedge(F_n) \geq n - 1$ *for* $n > 6$ *and exactly* $n - 1$ *for* $3 \leq n \leq 6$.
3. $c_\wedge(F_n) \leq n - H^{\mathbb{N}}(n) + \lceil \log(n+1) \rceil - 1$.

It turns out that these eight functions have very low annihilator immunity. We consider the variants of F_n functions first and then the variants of G_n.

Lemma 3. *The function* $f = a \oplus b\Sigma_1^n \oplus \bigoplus_{i=3}^{n} \Sigma_i^n$ *has annihilator immunity at most* 2.

Proof. Let $\tilde{f} = b\Sigma_1^n \oplus \bigoplus_{i=3}^{n} \Sigma_i^n$, and let $h = 1 \oplus (1 \oplus b)\Sigma_1^n \oplus \Sigma_2^n$ be the algebraic complement of $\tilde{f}$, [29]. Notice that

$$\tilde{f} \oplus h = \bigoplus_{i=1}^{n} \Sigma_i^n \oplus 1 = (1 \oplus x_1)(1 \oplus x_2) \ldots (1 \oplus x_n)$$

which is 1 if and only if $\mathbf{x} = \mathbf{0}$. That is for $\mathbf{x} \neq \mathbf{0}$, $\tilde{f} = h$, so $1 \oplus h$ clearly annihilates $\tilde{f}$ on all non-zero inputs. Since $\tilde{f}(\mathbf{0}) = 0$, h is an annihilator of $\tilde{f}$ with degree 2, so depending on a, h is an annihilator of f. □

Lemma 4. *The function* $f = a \oplus b\Sigma_1^n \oplus \Sigma_2^n \oplus \Sigma_n^n$ *has annihilator immunity at most* 2.

Proof. Let $\mathbf{1}$ denote the all 1 input vector. For some fixed choice of a, and b, depending on n, either $(a \oplus b\Sigma_1^n \oplus \Sigma_2^n)(\mathbf{1}) = 1$ or $(a \oplus b\Sigma_1^n \oplus \Sigma_2^n)(\mathbf{1}) = 0$. In the first case, the function $h = 1 \oplus a \oplus b\Sigma_1^n \oplus \Sigma_2^n$ is an annihilator of f, and otherwise $h = a \oplus b\Sigma_1^n \oplus \Sigma_2^n$ is an annihilator of $f \oplus 1$. And again, clearly there is no annihilator of degree less than 2. □

8 Conclusion

Four nonlinearity concepts are considered and compared, and new relations between them are presented. The four concepts are shown to be distinct; none is subsumed by any of the others.

We are currently extending the ideas present here for cryptanalyzing functions with low multiplicative complexity. It will be interesting to see if using the topology of the circuit for the cryptographic function will lead to useful heuristics for cryptanalytic attacks, especially for variants of hash function with few rounds.

References

1. Boyar, J., Damgaard, I., Peralta, R.: Short non-interactive cryptographic proofs. Journal of Cryptology 13, 449–472 (2000)
2. Boyar, J., Peralta, R.: Tight bounds for the multiplicative complexity of symmetric functions. Theor. Comput. Sci. 396(1-3), 223–246 (2008)
3. Boyar, J., Peralta, R., Pochuev, D.: On the multiplicative complexity of Boolean functions over the basis $(\wedge, \oplus, 1)$. Theor. Comput. Sci. 235(1), 43–57 (2000)
4. Canteaut, A., Videau, M.: Symmetric Boolean functions. IEEE Transactions on Information Theory 51(8), 2791–2811 (2005)
5. Carlet, C.: On the degree, nonlinearity, algebraic thickness, and nonnormality of Boolean functions, with developments on symmetric functions. IEEE Transactions on Information Theory 50(9), 2178–2185 (2004)
6. Carlet, C.: Boolean functions for cryptography and error correcting codes. In: Crama, Y., Hammer, P.L. (eds.) Boolean Models and Methods in Mathematics, Computer Science, and Engineering, ch. 8, pp. 257–397. Cambridge Univ. Press, Cambridge (2010)
7. Carlet, C., Dalai, D.K., Gupta, K.C., Maitra, S.: Algebraic immunity for cryptographically significant Boolean functions: Analysis and construction. IEEE Transactions on Information Theory 52(7), 3105–3121 (2006)
8. Courtois, N., Meier, W.: Algebraic attacks on stream ciphers with linear feedback. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 345–359. Springer, Heidelberg (2003)
9. Courtois, N., Hulme, D., Mourouzis, T.: Solving circuit optimisation problems in cryptography and cryptanalysis, e-print can be found at http://eprint.iacr.org/2011/475.pdf
10. Dalai, D.K., Gupta, K.C., Maitra, S.: Results on algebraic immunity for cryptographically significant Boolean functions. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 92–106. Springer, Heidelberg (2004)
11. Dalai, D.K., Maitra, S., Sarkar, S.: Basic theory in construction of Boolean functions with maximum possible annihilator immunity. Des. Codes Cryptography 40(1), 41–58 (2006)
12. Demenkov, E., Kulikov, A.S.: An elementary proof of a $3n - o(n)$ lower bound on the circuit complexity of affine dispersers. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 256–265. Springer, Heidelberg (2011)
13. Didier, F.: A new upper bound on the block error probability after decoding over the erasure channel. IEEE Transactions on Information Theory 52(10), 4496–4503 (2006)

14. Dobbertin, H.: Construction of bent functions and balanced Boolean functions with high nonlinearity. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 61–74. Springer, Heidelberg (1995)
15. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, STOC 1987, pp. 218–229. ACM, New York (1987), `http://doi.acm.org/10.1145/28395.28420`
16. Jukna, S.: Boolean Function Complexity: Advances and Frontiers. Springer, Heidelberg (2012)
17. Kavut, S., Maitra, S., Yücel, M.D.: There exist Boolean functions on n (odd) variables having nonlinearity $> 2^{\text{n-1}}$ - $2^{(\text{n-1})/2}$ if and only if n>7. IACR Cryptology ePrint Archive 2006, 181 (2006)
18. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)
19. Lupanov, O.: On rectifier and switching-and-rectifier schemes. Dokl. Akad. 30 Nauk SSSR 111, 1171-1174 (1965)
20. Maitra, S., Sarkar, P.: Maximum nonlinearity of symmetric Boolean functions on odd number of variables. IEEE Transactions on Information Theory 48(9), 2626–2630 (2002)
21. McFarland, R.L.: Sub-difference sets of Hadamard difference sets. J. Comb. Theory, Ser. A 54(1), 112–122 (1990)
22. Nechiporuk, E.I.: On the complexity of schemes in some bases containing nontrivial elements with zero weights (in Russian). Problemy Kibernetiki 8, 123–160 (1962)
23. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R. (ed.) CRYPTO 2012. LNCS, vol. 7417, pp. 681–700. Springer, Heidelberg (2012)
24. O'Donnell, R.: Analysis of Boolean Functions. Book Draft (2012), `http://www.analysisofbooleanfunctions.org`
25. Rodier, F.: Asymptotic nonlinearity of Boolean functions. Des. Codes Cryptography 40(1), 59–70 (2006)
26. Rothaus, O.S.: On "bent" functions. J. Comb. Theory, Ser. A 20(3), 300–305 (1976)
27. Savický, P.: On the bent Boolean functions that are symmetric. Eur. J. Comb. 15(4), 407–410 (1994)
28. Schnorr, C.-P.: The multiplicative complexity of Boolean functions. In: Mora, T. (ed.) AAECC 1988. LNCS, vol. 357, pp. 45–58. Springer, Heidelberg (1989)
29. Zhang, X.-M., Pieprzyk, J., Zheng, Y.: On algebraic immunity and annihilators. In: Rhee, M.S., Lee, B. (eds.) ICISC 2006. LNCS, vol. 4296, pp. 65–80. Springer, Heidelberg (2006)
30. Zhegalkin, I.I.: On the technique of calculating propositions in symbolic logic. Matematicheskii Sbornik 43, 9–28 (1927)

On the Characterization of Plane Bus Graphs[*]

Till Bruckdorfer[1], Stefan Felsner[2], and Michael Kaufmann[1]

[1] Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Germany
{bruckdor,mk}@informatik.uni-tuebingen.de
[2] Institut für Mathematik, Technische Universität Berlin, Germany
felsner@math.tu-berlin.de

Abstract. Bus graphs are being used for the visualization of hyperedges, for example in VLSI design. Formally, they are specified by bipartite graphs $G = (B \cup V, E)$ of bus vertices B realized by single horizontal and vertical segments, and point vertices V that are connected orthogonally to the bus segments without any bend. The decision whether a bipartite graph admits a bus realization is NP-complete. In this paper we show that in contrast the question whether a plane bipartite graph admits a planar bus realization can be answered in polynomial time.

We first identify three necessary conditions on the partition $B = B_V \cup B_H$ of the bus vertices, here B_V denotes the vertical and B_H the horizontal buses. We provide a test whether *good partition*, i.e., a partition obeying these conditions, exist. The test is based on the computation of maximum matching on some auxiliary graph. Given a good partition we can construct a non-crossing realization of the bus graph on an $O(n) \times O(n)$ grid in linear time.

1 Introduction

A classical topic in the area of graph visualization is *orthogonal graph drawing*; related surveys can be found in [3,11,18]. In this drawing model each edge consists of a series of subsequent horizontal or vertical line segments. Applications can be found in e.g. VLSI design, cf. [17,12]. In this application it may also be necessary to model hypergraphs. For example power buses on VLSI chips are often modeled as hyperedges, as well as LANs in computer network visualization. Bus graphs - as being defined later - and their generalizations are a possible approach to model hyperedges. A bus-style representation might also be used when facing the visualization of highly interconnected parts of a given graph. So, cliques can be represented in a compact and comprehensive way using a bus-style model.

The bus graph approach is very much related to the classical topic of rectilinear Steiner trees, where trees are being used to connect subsets of the vertices [8,10]. Related are also works on rectangular drawings, rectangular duals and visibility graphs. The latter graphs are highly related to bus graphs, because connections in bus graphs enforce visibility constraints in realizations.

[*] Our Research is partially supported by EuroGIGA project GraDR 10-EuroGIGA-OP-003.

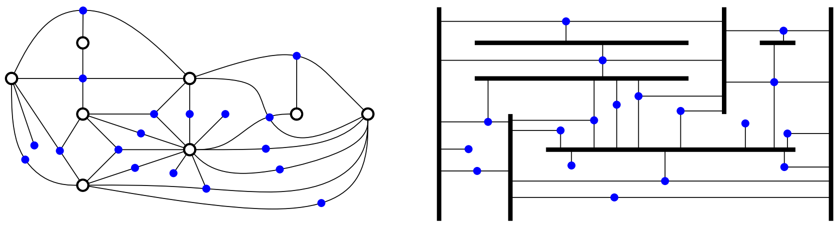


Fig. 1. An example of a plane bus graph with a planar realization

Corresponding key concepts and surveys can be found in [9,15,16] and in chapter 6 of [14]. We will use some of the methods that have been developed in this area.

We are considering the bus graph model introduced by Ada et al. [1] A *bus graph* is a bipartite graph $G = (B \cup V, E)$, where $E \subseteq V \times B$ and $deg(v) \leq 4$ for all $v \in V$. We call vertices in B *bus vertices* and vertices in V *connector vertices.* A *plane bus graph* is a planar bus graph together with a planar embedding. A *realization* of a bus graph is a drawing D, where bus vertices are represented as horizontal or vertical line segments (*bus segments*), connector vertices are drawn as points, and the edges are horizontal or vertical segments (*connections*), i.e. segments connecting perpendicular a point with a bus segment. To distinguish between bus vertices and edges in a realization, the bus segments are drawn with bold lines, see Figure 1. A *planar realization* is a realization without crossings. We always assume to have a connected bus graph, since components can be considered separately.

In [1] a relation of bus graphs to hypergraphs is mentioned: the bipartite adjacency matrix of a bipartite graph can be read as incidence matrix of a hypergraph and vice versa, if vertices are contained in at most four hyperedges. Ada et al. [1] considered the problem to decide, if a bus graph has a realization and showed the NP-completeness. In this paper we consider the problem to decide if a plane bus graph has a planar realization. We show that this question in contrast to the previous result can be decided in polynomial time.

The bus segments will be drawn either vertically or horizontally. So we assign a labeling to the bus vertices that determines if they will be realized either vertically or horizontally. This labeling ensures a planar realization, if it obeys some properties. The paper is structured as follows: In Section 2 we provide necessary properties for the labeling. After that in Section 3 we give a polynomial time algorithm that tests whether a given maximal plane bus graph admits a labeling with these properties, which we call a *good partition.* If it exists, the algorithm also returns a good partition. In Section 4 we modify the approach so that it also works in cases where the bus graph is not maximal. In Section 5 we show how to actually produce a realization of a plane bus graph with a good partition. The approach is based on techniques from [2] and [6].

2 Preliminaries

In this paper we consider *plane bus graphs* $G = (V \cup B, E)$, i.e., planar bus graphs together with a fixed planar embedding. A *diamond* in a plane bus graph is a cycle $z = (b, v, b', v')$ of length four with bus vertices b, b' and connector vertices v, v' such that both v, v' have a third bus neighbour in the bounded region defined by z. A planar realization of a plane bus graph $G = (V \cup B, E)$, induces labels H or V for the bus vertices where H indicates that a bus is represented by a horizontal segment in the realization, while V indicates that the bus is represented by a vertical segment. Let $\Pi = (B_V, B_H)$ denote the *partition* of B according to the label of the bus vertices. We observe three properties of a partition $\Pi = (B_V, B_H)$ corresponding to a planar realization:

(P1) Every connector vertex of degree ≥ 3 has neighbors in both classes.
(P2) A connector vertex v of degree 4 has cyclically alternating neighbors in both classes.
(P3) A diamond has both bus vertices b, b' in the same partition class.

The first two properties are obvious from the nature of a realization. The third property is shown in the following lemma.

Lemma 1. *Let G be a plane bus graph that has a realization inducing the partition (B_V, B_H) of the set of buses B. For any diamond $z = (b, v, b', v')$ the two bus vertices b and b' belong to the same class.*

Proof. Suppose by contradiction that $b \in B_H$ and $b' \in B_V$. The interior of z in the planar bus realization is a polygon with six corners. Four of the corners are at contacts of connector edges and buses and two corners are at the connector vertices. Some of these corners may degenerate so that the polygon actually only has four corners. We account for four corners of size $\pi/2$ each, where the edges meet the buses. The other two corners are at v and v'. Since b is horizontal and b' vertical the angles at v and v' have to be either $\pi/2$ or $3\pi/2$. Because v and v' have an additional bus neighbor in the interior the angle at each of v and v' is at least π. Hence, both these angles are of size $3\pi/2$. The sum of interior angles is at least $4 \cdot \pi/2 + 2 \cdot 3\pi/2 = 5\pi$. A six-gon, however, has a sum of angles of 4π. The contradiction shows that b and b' belong to the same class of the partition $B = B_V \cup B_H$. □

Note that the outer cycle of G is a diamond when the outer face of G has cardinality 4.

A partition $\Pi = (B_V, B_H)$ of the buses of a plane bus graph G is called a *good partition* if it obeys properties (P1), (P2), and (P3).

Let $\Delta = \Delta(G)$ denote the *degree of a plane bus graph G* which is defined as the maximum degree among the connector vertices of G.

In the next section we consider maximal plane bus graphs and test efficiently, if they admit a good partition. The test is constructive, i.e., if the answer is yes, then a good partition is constructed.

3 Maximal Plane Bus Graphs

A *maximal plane bus graph* G is a plane bipartite bus graph, where all its faces have cardinality 4. Let $G = (V \cup B, E)$ be a maximal plane bus graph. We assume, that G has no connector vertices of degree 1 or 2, since they have no influence on the existence of a good partition.

In this section we first assume $\Delta = 3$ for G and give an algorithm to test G if it admits a good partition and if so the algorithm returns a good partition. After that we allow $\Delta = 4$ and reduce this case with a simple modification to the case $\Delta = 3$.

Let $G = (V \cup B, E)$ be a plane maximal bus graph with $\Delta = 3$. The *connector graph* $C_G = (V_C, E_C)$ of G consists of all the connector vertices $V_C = V$ and edges $(v, v') \in E_C$, if v and v' are both incident to the same face of the plane embedding of G. The connector graph is helpful because it allows the translation of the problem of finding a good partition for G to the problem of finding an appropriate perfect matching in C_G, summarized in Proposition 1 and Proposition 2 and illustrated in Figure 2.

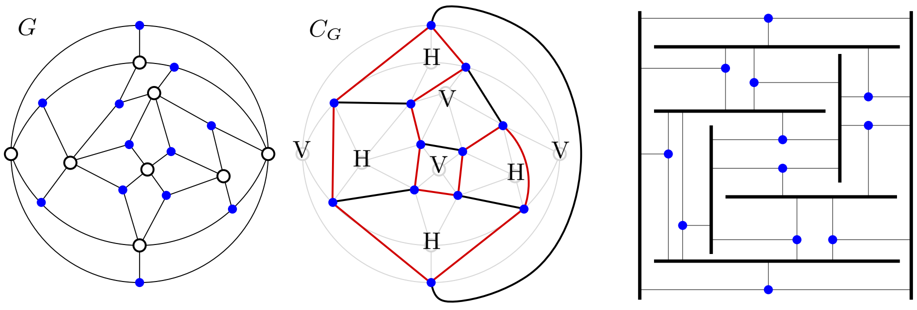


Fig. 2. A maximal plane bus graph, the connector graph with a matching and its good partition, and a corresponding bus representation

The first property (P1) of a good partition Π of G requires that every connector vertex v has two adjacent bus vertices in one partition class and one in the other. If b and b' are neighbors of v in G with the same label, then there is a connector vertex v' sharing a face with v, b, and b', since every face has cardinality 4. When looking at v' the two neighbors b and b' are again the two in a common partition class. Hence, property (P1) of a good partition of G induces a perfect matching on C_G.

Conversely a perfect matching M of C_G induces a labeling of the bus vertices. Removing the matching edges from C_G leaves a 2-regular graph, i.e., a disjoint collection of cycles. The regions defined by this collection of cycles can be 2-colored with colors V and H such that each cycle has regions of different colors on its sides. Let B_V be the set of bus vertices in faces colored with V, and B_H be the set of bus vertices in faces colored with H. This yields a partition satisfying

(P1) because every connector vertex is on a cycle and has a bus neighbour in each of the two faces bounded by the cycle.

Since $\Delta = 3$, the second property (P2) is void.

Consider a diamond $z = (b, v, b', v')$ in G. Each of v, v', has exactly one edge e, e', in C_G, that corresponds to a face of the outside of z. Since (P3) forces equal labels for b, b', the faces represented by e, e' have equally labeled incident bus vertices (b, b') and thus e, e' must be in a matching of C_G. We define the set E_d of *edges forced by diamonds* as the set of edges consisting of the two outside edges e, e' in C_G for each diamond z of G. We have thus shown that a perfect matching M induced by a good partition contains E_d.

Conversely, if E_d is contained in a perfect matching M of C_G, then the bus vertices b, b' of each diamond are in the same partition class and thus G has a partition, that satisfies property (P3). The findings are summarized in the following proposition.

Proposition 1. *Let G be a maximal plane bus graph with $\Delta = 3$ and C_G its connector graph and E_d the set of edges of C_G forced by diamonds. Then G admits a good partition, iff C_G has a perfect matching M, with $E_d \subseteq M$.*

Now we allow $\Delta = 4$ for a maximal plane bus graph G. To transform G into a plane bus graph G' with $\Delta = 3$, we split every connector vertex v of degree 4 into two connector vertices v', v'', both of degree 3 in the following way: let b_1, b_2, b_3, b_4 be the adjacent bus vertices of v in consecutive cyclic order around v. Remove v and its incident edges and connect the new introduced vertices v', v'' with edges (b_1, v'), (b_2, v'), (b_3, v'), (b_3, v''), (b_4, v''), (b_1, v''). The connector graph C_G is obtained from $C_{G'}$ by contracting the edges (v', v'') corresponding to the pairs v', v'' that have been obtained by splitting a vertex of degree 4. Define the set E_s of edges *forced by splits* of $C_{G'}$ as the set of these edges (v', v'').

If G has a partition satisfying property (P2), then we have to ensure alternating labels for the neighbours of v in G. This forces $(v', v'') \in E_s$ to be a matching edge in $C_{G'}$. Conversely if $(v', v'') \in E_s$ is a matching edge, then the common neighbours b_1, b_3 have the same label. Since v', v'' have both degree 3 and two of their neighbours have equal label, the third neighbour (for each of v', v'') has a different label, i.e. b_2 and b_4 have both different label compared to the label of b_1, b_3, hence, v obeys property (P2). So in total a partition Π of G satisfies property (P2), iff the edges E_s are contained in a matching of $C_{G'}$. For notational simplicity we denote the connector graph $C_{G'}$ of the transformed graph G' by C_G. An example for a maximal plane bus graph with its connector graph showing the edges of $E_d \cup E_s$ is shown in Figure 3.

Proposition 2. *Let G be a maximal plane bus graph with $\Delta = 4$ and C_G its connector graph and E_d, E_s the edges of C_G that are forced by diamonds and splits. The graph G admits a good partition, iff C_G has a perfect matching M, with $(E_d \cup E_s) \subseteq M$.*

Proof. The proof almost follows from Proposition 1 and the above considerations. Splitting connector vertices of degree 4, however, may separate diamonds.

We claim that this is no problem. Let v be split into v', v'' as above. Any diamond containing v has a cycle z with bus vertices b_1 and b_3. Condition (P3) for this diamond requires that b_1 and b_3 belong to the same class of a good partition. This requirement, however, is already implied by condition (P2) for the original connector vertex v. That is, separated diamonds do not impose additional conditions on the matching. $\square$

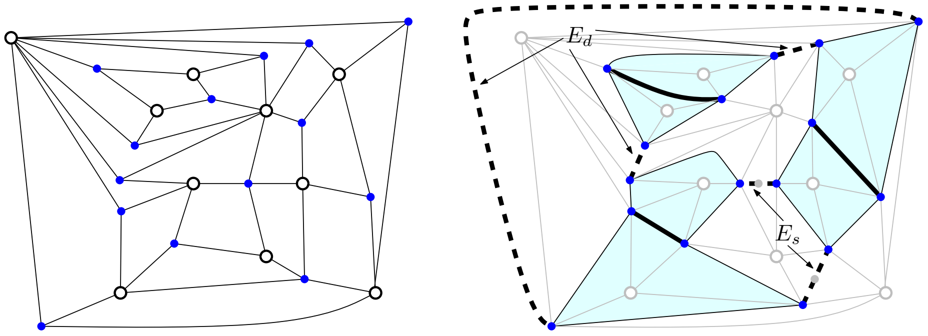


Fig. 3. A maximal plane bus graph and its connector graph with the modifications, where the dotted edges are forced and the fat edges complete the perfect matching

Theorem 1. *Let G be a maximal plane bus graph. A good partition for G can be computed in $O(n^{3/2})$ time, if it exists.*

Proof. By Proposition 2 it suffices to test the connector graph C_G for a perfect matching M that contains $(E_d \cup E_s)$. The extraction of the connector graph C_G from G requires linear time. The set E_s can be computed while constructing C_G. To identify diamonds we consider the dual D_G of the connector graph C_G. The vertices of D_G are the bus vertices of G and edges correspond to faces of G'. Diamonds of G' corresponds to a double edge of D_G the only exception is the diamond bounding the outer face. Double edges of D_G can be found and sorted so that the set E_d can be constructed in $O(n \text{ polylog}(n))$ time. To force $E_d \cup E_s$ we simply delete all vertices incident to these edges from the graph. If a vertex is incident to two edges from the set, then there is no matching. For constructing a maximum matching of a graph there exist several $O(\sqrt{n}m)$ algorithms. For planar graphs this yields the claimed $O(n^{3/2})$ complexity.[1]

Given the perfect matching the corresponding good partition can again be computed in linear time. $\square$

4 Non Maximal Plane Bus Graphs

In this section we consider a plane bus graph G, that is not necessarily maximal. In a first preprocessing step we remove all bus vertices and connector vertices

[1] In [13] a slightly faster randomized algorithm for planar graphs has been proposed.

with degree 1, as well as their incident edge. These objects can easily be integrated in a realization of the remaining graph.

In the following we describe how to extend G to a maximal plane bus graph G^+ containing G as induced subgraph such that G^+ has a good partition iff G has a good partition (Lemma 2). The graph G^+ will be called the *quadrangulation* of G.

Let f be a face with cardinality $2k$ in G and let $b_1, \ldots, b_k$ be the bus vertices of f in clockwise order. To *quadrangulate* f we first place a new bus vertex b_f^* in the inside. The bus vertex b_f^* is then connected to the boundary of f by adding a triangular structure for every consecutive pair b_i, b_{i+1} of bus vertices including the pair b_k, b_1. The triangular structure for b_i, b_{i+1} consists of another new bus vertex c_i and three connector vertices v_i^1, v_i^2, v_i^3 such that $N(v_i^1) = \{b_i, c_i, b_{i+1}\}$, $N(v_i^2) = \{b_{i+1}, c_i, b_f^*\}$, and $N(v_i^1) = \{b_f^*, c_i, b_i\}$. Figure 4 shows an example.

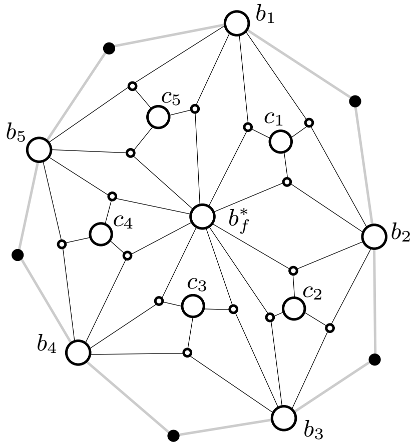


Fig. 4. New vertices and edges added to quadrangulate a face f with cardinality 10

The graph G^+ is obtained from G by quadrangulation every face f with cardinality > 4 including, if necessary, the outer face. The following properties of the quadrangulation G^+ of G are obvious:

- G^+ is planar and has $O(n)$ vertices.
- All diamonds of G^+ are diamonds of G.[2]

In addition we have the following important lemma:

Lemma 2. *Let G be a plane bus graph and G^+ be its quadrangulation. Then G has a good partition, iff G^+ has a good partition.*

[2] Note that the outer face of G^+ has cardinality 4 if the outer face of G has cardinality > 4 this is an additional diamond of G^+. We ignore this diamond and the condition imposed by it on good partitions of G^+.

Proof. The three defining properties (P1), (P2), and (P3) are stable under taking induced subgraphs. Hence, a good partition of G^+ immediately yields a good partition of G.

Now assume that G has a good partition. We aim for a partition of the bus vertices of G^+ that extends the given partition of the bus vertices of G. Since all bus vertices of degree 4 and all diamonds of G^+ already belong to G we don't have to care of (P2), and (P3). The following rules define the labels for the new bus vertices such that (P1) is fulfilled for all new connector vertices:

- Label all central bus vertices b_f^* with V.
- If b_i and b_{i+1} are both labeled H, then the label of c_i is defined to be V. Otherwise the label c_i is H.

It is straightforward to check that this yields a good partition of G^+.

If vertices have been added to the outer face f^* in the quadrangulation process, then we can choose the outer face of G^+ such that it contains $b_{f^*}^*$ and both bus vertices of the (new) outer face are labelled V. This change in the outer face does not affect the plane embedding of G. These considerations imply that when looking for a good partition of G^+ we may disregard the condition implied by the diamond defined by the outer face if the outer face of G had cardinality > 4, c.f. footnote 2. □

Theorem 2. *Let G be a plane bus graph. A good partition for G can be computed in $O(n^{3/2})$ time, if it exists.*

Proof. By Lemma 2 it suffices to test if the quadrangulation G^+ of G has a good partition. Hence we first compute G^+ in linear time. Since G^+ is a maximal plane bus graph we can use the algorithm from Theorem 1 to check whether G^+ has a good partition. The running time is $O(n^{3/2})$ and the algorithm returns a good partition if it exists. □

5 Planar Realizations

In the last two sections we analyzed the complexity of testing and computing a good partition. In this section we assume the existence of a good partition for a plane bus graph G and give a polynomial time algorithm to construct a planar realization for G.

Theorem 3. *Let G be a plane bus graph admitting a good partition. Then G has a planar realization on an $O(n) \times O(n)$ grid. If the good partition is given the realization can be computed in $O(n)$ time.*

Let G be a plane bus graph admitting a good partition. Again we start with some simplifications. First we recursively remove all connector and bus vertices of degree 1 and all connector vertices of degree 2.

Let G^+ be the quadrangulation of G. The assumption about the existence of a good partition of G together with Lemma 2 imply that G^+ has a good partition which can by Theorem 1 be computed efficiently.

Given the good partition of G^+ we split all connector vertices of degree 4 into two connector vertices of degree 3. For simplicity we continue denoting the resulting graph G^+.

The *reduced bus graph* $R^+ = (B^+, E_R)$ of G^+ is the graph on the bus vertices of G^+ with edges (b, b'), iff b, b' are incident to a common face and have different labels. Diamonds with different labeled bus vertices are the only substructure that would create double edges in R^+ but diamonds have identically labeled bus vertices in a good partition. Hence, there are no double edges in R^+. From the three faces incident to a connector vertex exactly two contribute an edge to R^+. It follows that R^+ is a quadrangulation, i.e., all faces have cardinality 4. Another approach to derive this is by observing that the edges of the matching M of Proposition 2 are in bijection with the faces of R^+.

Let Q be a quadrangulation, we call the color classes of the bipartition white and black and name the two black vertices on the outer face s and t. A *separating decomposition* of Q is an orientation and coloring of the edges of Q with colors red and blue such that:

- All edges incident to s are ingoing red and all edges incident to t are ingoing blue.
- Every vertex $v \neq s, t$ is incident to a non-empty interval of red edges and a non-empty interval of blue edges. If v is white, then, in clockwise order, the first edge in the interval of a color is outgoing and all the other edges of the interval are incoming. If v is black, the outgoing edge is the last one in its color in clockwise order (see Figure 5).

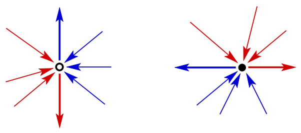


Fig. 5. Edge orientations and colors at white and black vertices

Separating decompositions have been studied in [2], [6], and [5]. In particular it is known that every plane quadrangulation admits a separating decomposition. To us separating decompositions are of interest because of their connection with segment contact representations of the underlying quadrangulation. A proof of the following lemma can be found in [4].

Lemma 3. *A separation decomposition of Q can be used to construct a segment contact representation of Q with vertical and horizontal segments, such that edges $v \to w$ of the separating decomposition correspond to a contact of the segments S_v and S_w where an endpoint of S_v is in the interior of S_w.*

An illustration for the lemma is given in Figure 6.

Identify the two classes V and H of the bipartition of the reduced bus graph R^+ with black and white. Construct a separating decomposition of R^+ and a

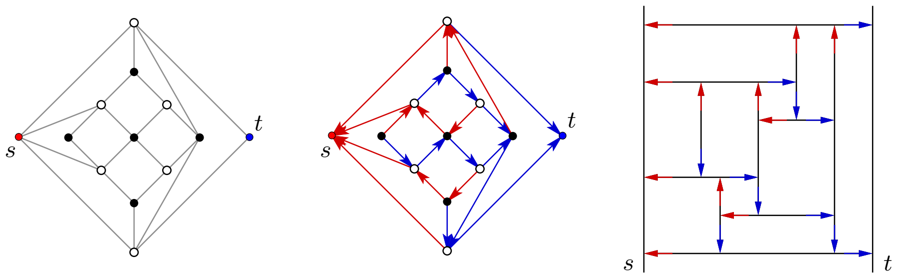


Fig. 6. A quadrangulation Q, a separating decomposition of Q and a corresponding segment contact representation of Q

corresponding segment contact representation. Later the following observation will be important:

($\star$) The rectangles in the segment contact representation correspond bijectively to the faces of R^+. Moreover, vertex b is incident to face f in R^+ iff segment S_b contributes a side of the rectangle R_f corresponding to f.

From the segment contact representation of R^+ we obtain a representation of the bus graph G^+ in two steps. First clip the endpoints of all segments of the representation so that a disjoint collection of segments remains. These segments serve as the bus segments for the representation of the bus graph G^+. It remains to insert the connector vertices and the edges of G^+ into the picture. To this end recall that each connector vertex belongs to a unique face of R^+ and each face of R^+ contains exactly two connector vertices. The two connector vertices contained in a face f can easily be accommodated in the rectangle R_f, because of ($\star$). Figure 7 shows the picture.

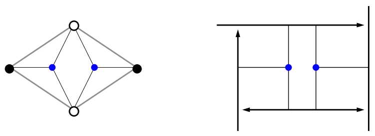


Fig. 7. A face f of R^+ with its two connector vertices and the placement of the two vertices in R_f

At this point we have a representation of the plane bus graph G^+. It remains to transform this into a representation of the original input graph G. These are the steps that have to be done:

- Merge pairs of connector vertices that have been created by splitting a connector vertex of degree 4.
- Delete all bus and connector vertices from the representation that have been introduced to make the bus graph maximal.

- Insert all connector and bus vertices of degree 1 and all connector vertices of degree 2 that had been deleted in the reverse order of the deletion.

This yields a representation of the input graph G.

To complete the proof of Theorem 3 it remains to argue about the complexity. Let $G = (V \cup B, E)$ be the input bus graph with $n = |V| + |B|$. Simple estimates on the basis of Euler's formula show that in going from G to G^+ at most $14|B|$ new vertices have been introduced, hence, G^+ has $n^+ \in O(n)$ vertices. The reduced bus graph R^+ can be computed from the plane G^+ in $O(n^+)$. A separating decomposition of R^+ can also be computed in linear time, details can be found in the PhD thesis of É. Fusy [7]. The segment contact representation of R^+ associated to the separation decomposition is computable in linear time with standard techniques, c.f. [3] or [4]. The number of grid lines needed for the segment contact representation of R^+ is bounded by the number of bus vertices of R^+, i.e., the size of the grid is $O(n) \times O(n)$. The clipping of endpoints increases the number of grid lines by a factor of 3 and the insertion of connector vertices may require an additional grid line for each vertex. The same holds for the reinsertion of vertices of degree 1 and 2. All these steps can be done in linear time and keep the size of the grid in $O(n) \times O(n)$.

6 Conclusion and Future Work

We have considered the class of plane bus graphs, that admit a planar realization and have characterized this class by the existence of a good partition of bus vertices. To test for the existence of a good partition we gave an $O(n^{3/2})$ algorithm based on planar matching. Given a good partition the representation can be computed in linear time.

It would be interesting to extend the approach from plane to planar bus graphs. The problem here is that the connector graphs of different plane embeddings of a planar graph differ.

It is also open to characterize the class of graphs that admit realizations, where connections are allowed to cross apart from the knowledge, that the decision is NP-complete. Another generalization would be to allow connections to cross bus segments or bus segments to cross each other.

References

1. Ada, A., Coggan, M., Marco, P.D., Doyon, A., Flookes, L., Heilala, S., Kim, E., Wing, J.L.O., Préville-Ratelle, L.F., Whitesides, S., Yu, N.: On bus graph realizability. CCCG abs/cs/0609127, 229–232 (2007)
2. De Fraysseix, H., De Mendez, P.O.: On topological aspects of orientations. Discrete Mathematics 229(1-3), 57–72 (2001)
3. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice-Hall (1999)
4. Felsner, S.: Rectangle and square representations of planar graphs. In: Pach, J. (ed.) Thirty Essays in Geometric Graph Theory, Algorithms and Combinatorics, vol. 29. Springer (2012)

5. Felsner, S., Fusy, É., Noy, M., Orden, D.: Bijections for Baxter families and related objects. Journal of Comb. Theory A 18, 993–1020 (2011)
6. Felsner, S., Huemer, C., Kappes, S., Orden, D.: Binary labelings for plane quadrangulations and their relatives. Discrete Mathematics & Theoretical Computer Science 12(3), 115–138 (2010)
7. Fusy, E.: Combinatoire des cartes planaires et applications algorithmiques. Ph.D. thesis, LIX Ecole Polytechnique (2007)
8. Hanan, M.: On Steiner's problem with rectilinear distance. SIAM J. Appl. Math (14), 255–265 (1966)
9. He, X.: On finding the rectangular duals of planar triangular graphs. SIAM J. Comput. 22, 1218–1226 (1993)
10. Hwang, F.W., Richards, D.S., Winter, P.: The Steiner tree problem. Annals of Discrete Mathematics (53) (1992)
11. Kaufmann, M., Wagner, D. (eds.): Drawing Graphs, Methods and Models. LNCS, vol. 2025. Springer, Heidelberg (2001)
12. Lengauer, T.: VLSI theory. In: Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A), pp. 835–868 (1990)
13. Mucha, M., Sankowski, P.: Maximum matchings in planar graphs via gaussian elimination. Algorithmica 45(1), 3–20 (2006)
14. Nishizeki, T., Rahman, M.S.: Planar Graph Drawing. World Scientific (2004)
15. Rosenstiehl, P., Tarjan, R.E.: Rectilinear planar layouts and bipolar orientations of planar graphs. Discrete & Computational Geometry 1, 343–353 (1986)
16. Tamassia, R., Tollis, I.G.: A unified approach a visibility representation of planar graphs. Discrete & Computational Geometry 1, 321–341 (1986)
17. Thompson, C.D.: A Complexity Theory for VLSI. Ph.D. thesis, Carnegie-Mellon University (1980)
18. Zhou, X., Nishizeki, T.: Algorithm for the cost edge-coloring of trees. J. Comb. Optim. 8(1), 97–108 (2004)

Online Multi-dimensional Dynamic Bin Packing of Unit-Fraction Items

Mihai Burcea[⋆], Prudence W.H. Wong, and Fencol C.C. Yung

Department of Computer Science, University of Liverpool, UK
{m.burcea,pwong}@liverpool.ac.uk, ccyung@graduate.hku.hk

Abstract. We study the 2-D and 3-D dynamic bin packing problem, in which items arrive and depart at arbitrary times. The 1-D problem was first studied by Coffman, Garey, and Johnson motivated by the dynamic storage problem. Bar-Noy et al. have studied packing of unit fraction items (i.e., items with length $1/k$ for some integer $k \geq 1$), motivated by the window scheduling problem. In this paper, we extend the study of 2-D and 3-D dynamic bin packing problem to unit fractions items. The objective is to pack the items into unit-sized bins such that the maximum number of bins ever used over all time is minimized. We give a scheme that divides the items into classes and show that applying the First-Fit algorithm to each class is 6.7850- and 21.6108-competitive for 2-D and 3-D, respectively, unit fraction items. This is in contrast to the 7.4842 and 22.4842 competitive ratios for 2-D and 3-D, respectively, that would be obtained using only existing results for unit fraction items.

1 Introduction

Bin packing is a classical combinatorial optimization problem that has been studied since the early 70's and different variants continue to attract researchers' attention (see [7,10,12]). It is well known that the problem is NP-hard [14]. The problem was first studied in one dimension (1-D), and has been extended to multiple dimensions (d-D, where $d \geq 1$). In d-D packing, the bins have lengths all equal to 1, while items are of lengths in $(0, 1]$ in each dimension. The objective of the problem is to pack the items into a minimum number of unit-sized bins such that the items do not overlap and do not exceed the boundary of the bin. The items are oriented and cannot be rotated.

Extensive work (see [7,10,12]) has been done in the offline and online settings. In the offline setting, all the items and their sizes are known in advance. In the online setting, items arrive at unpredictable time and the size is only known when the item arrives. The performance of an online algorithm is measured using competitive analysis [3]. Consider any online algorithm $\mathcal{A}$ with an input I. Let $OPT(I)$ and $\mathcal{A}(I)$ be the maximum number of bins used by the optimal offline algorithm and $\mathcal{A}$, respectively. Algorithm $\mathcal{A}$ is said to be c-competitive if there exists a constant b such that $\mathcal{A}(I) \leq c \cdot OPT(I) + b$ for all I.

[⋆] Supported by EPSRC Studentship.

In some real applications, item size is not represented by arbitrary real numbers in $(0, 1]$. Bar-Noy et al. [2] initiated the study of the *unit fraction bin packing problem*, a restricted version where all sizes of items are of the form $\frac{1}{k}$, for some integer k. The problem was motivated by the window scheduling problem [1,2]. Another related problem is for *power fraction* items, where sizes are of the form $\frac{1}{2^k}$, for some integer k. Bin packing with other restricted form of item sizes includes divisible item sizes [8] (where each possible item size can be divided by the next smaller item size) and discrete item sizes [6] (where possible item sizes are $\{1/k, 2/k, \cdots, j/k\}$ for some $1 \leq j \leq k$). For d-D packing, items of restricted form have been considered, e.g., [16] considered strip packing ([19]) of items with one of the dimensions having discrete sizes and [17] considered bin packing of items where the lengths of each dimension are at most $1/m$, for some integer m. The study of these problems is motivated by applications in job scheduling. As far as we know, unit or power fraction items have not been considered in multi-dimensional packing.

Dynamic Bin Packing. Earlier work concentrated on "static" bin packing, where items do not depart. In potential applications, like warehouse storage, a more realistic setting is the dynamic model, where items arrive and depart dynamically. This natural generalization, known as dynamic bin packing problem, was introduced by Coffman, Garey, and Johnson [9]. The items arrive over time, reside for some period of time, and may depart at arbitrary times. Each item must be packed to a bin from its arrival to its departure. Again, migration to another bin is not allowed, yet rearrangement of items within a bin is allowed. The objective is to minimize the maximum number of bins used over all time. In the offline setting, the sizes, and arrival and departure times of items are known in advance, while in the online setting the sizes and arrival times of items are only known when items arrive, and the departure times are known only when items depart.

Previous Work. The dynamic bin packing problem was first studied in 1-D for general size items by Coffman, Garey and Johnson [9], showing that the First-Fit (FF) algorithm has a competitive ratio lying between 2.75 and 2.897, and a modified First-Fit algorithm is 2.788-competitive. They gave a formula of the competitive ratio of FF when the item size is at most $\frac{1}{k}$. When $k = 2$ and 3, the ratios are 1.7877 and 1.459, respectively. They also gave a lower bound of 2.388 for any deterministic online algorithm, which was improved to 2.5 [5] and then to 2.666 [21]. For unit fraction items, Chan et al. [4] obtained a competitive ratio of 2.4942, which was recently improved by Han et al. to 2.4842 [15], while the lower bound was proven to be 2.428 [4]. Multi-dimensional dynamic bin packing of general size items has been studied by Epstein and Levy [13], who showed that the competitive ratios are 8.5754, 35.346 and $2 \cdot 3.5^d$ for 2-D, 3-D and d-D, respectively. The ratios are then improved to 7.788, 22.788, and 3^d, correspondingly [20]. For 2-D and 3-D general size items, the lower bounds are 3.70301 and 4.85383 [13], respectively. In this case, the lower bounds apply even to unit fraction items.

Table 1. Competitive ratios for general size, unit fraction, and power fraction items. Results obtained in this paper are marked with "[*]".

	1-D	2-D	3-D
General size	2.788 [9]	7.788 [20]	22.788 [20]
Unit fraction	2.4842 [15]	6.7850 [*]	21.6108 [*]
Power fraction	2.4842 [15]	6.2455 [*]	20.0783 [*]

Our Contribution. In this paper, we extend the study of 2-D and 3-D online dynamic bin packing problem to unit and power fraction items. We observe that using the 1-D results on unit fraction items [15], the competitive ratio of 7.788 for 2-D [20] naturally becomes 7.4842, while the competitive ratio of 22.788 for 3-D [20] becomes 22.4842. An immediate question arising is whether we can have an even smaller competitive ratio. We answer the questions affirmatively as follows (see Table 1 for a summary).

– For 2-D, we obtain competitive ratios of 6.7850 and 6.2455 for unit and power fraction items, respectively; and
– For 3-D, we obtain competitive ratios of 21.6108 and 20.0783 for unit and power fraction items, respectively.

We adopt the typical approach of dividing items into classes and analyzing each class individually. We propose several natural classes and define different packing schemes based on the classes[1]. In particular, we show that two schemes lead to better results. We show that one scheme is better than the other for unit fraction items, and vice versa for power fraction items. Our approach gives a systematic way to explore different combinations of classes. One observation we have made is that dividing 2-D items into three classes gives comparable results but dividing into four classes would lead to much higher competitive ratios.

As an attempt to justify the approach of classifying items, we show that, when classification is not used, the performance of the family of any-fit algorithms is unbounded for 2-D general size items. This is in contrast to the case of 1-D packing, where the First-Fit algorithm (without classification) is $O(1)$-competitive [9].

2 Preliminaries

Notations and Definitions. We consider the online dynamic bin packing problem, in which 2-D and 3-D items must be packed into 2-D and 3-D unit-sized bins, respectively, without overflowing. The items arrive over time, reside for some period of time, and may depart at arbitrary times. Each item must be packed into a bin from its arrival to its departure. Migration to another bin is not allowed and the items are oriented and cannot be rotated. Yet, repacking

[1] The proposed classes are not necessarily disjoint while a packing scheme is a collection of disjoint classes that cover all types of items.

Table 2. Types of unit fraction items considered

$T(1,1)$	$T(1,\frac{1}{2})$	$T(\frac{1}{2},1)$	$T(\frac{1}{2},\frac{1}{2})$	$T(1,\le\frac{1}{3})$	$T(\frac{1}{2},\le\frac{1}{3})$	$T(\le\frac{1}{3},\le 1)$

Table 3. The 2-D results of [20] for unit-fraction items

Scheme in [20]		
Classes	Types of items	Competitive ratios
Class A	$T(\le\frac{1}{3},\le 1)$	3 [20]
Class B	$T(\frac{1}{2},1), T(\frac{1}{2},\frac{1}{2}), T(\frac{1}{2},\le\frac{1}{3})$	2 [20]
Class C	$T(1,1), T(1,\frac{1}{2}), T(1,\le\frac{1}{3})$	2.4842 [15]
Overall	All items	7.4842

of items within the same bin is permitted[2]. The *load* refers to the total area or volume of a set of 2-D or 3-D items, respectively. The objective of the problem is to minimize the total number of bins used over all time. For both 2-D and 3-D, we consider two types of input: unit fraction and power fraction items.

A *general size item* is an item such that the length in each dimension is in $(0,1]$. A *unit fraction (UF) item* is an item with lengths of the form $\frac{1}{k}$, where $k \ge 1$ is an integer. A *power fraction (PF) item* has lengths of the form $\frac{1}{2^k}$, where $k \ge 0$ is an integer.

A packing is said to be *feasible* if all items do not overlap and the packing in each bin does not exceed the boundary of the bin; otherwise, the packing is said to *overflow* and is *infeasible.*

Some of the algorithms discussed in this paper repack existing items (and possibly include a new item) in a bin to check if the new item can be packed into this bin. If the repacking is infeasible, it is understood that we would restore the packing to the original configuration.

For 2-D items, we use the notation $T(w,h)$ to refer to the type of items with width w and height h. We use '$*$' to mean that the length can take any value at most 1, e.g., $T(*,*)$ refers to all items. The parameters w (and h) may take an expression $\le x$ meaning that the width is at most x. For example, $T(\frac{1}{2},\le\frac{1}{2})$ refers to the items with width $\frac{1}{2}$ and height at most $\frac{1}{2}$. In the following discussion, we divide the items into seven disjoint types as showed in Table 2.

The bin assignment algorithm that we use for all types of 2-D and 3-D unit and power fraction items is the First-Fit (FF) algorithm. When a new item arrives, if there are occupied bins in which the item can be repacked, FF assigns the new item to the bin which has been occupied for the longest time.

Remark on Existing Result on Unit Fraction Items. Using this notation, the algorithm in [20] effectively classifies unit fraction items into the classes as shown in Table 3. Items in the same class are packed separately, independent of

[2] If rearrangement within a bin is not allowed, one can show that there is no constant competitive deterministic online algorithm.

other classes. The overall competitive ratio is the sum of the competitive ratios of all classes. By the result in [15], the competitive ratio for Class C reduces from 2.788 [9] to 2.4842 [15] and the overall competitive ratio immediately reduces from 7.778 to 7.4842.

Corollary 1. *The 2-D packing algorithm in* [20] *is* 7.4842*-competitive for UF items.*

Remarks on Using Classification of Items. To motivate our usage of classification of items, let us first consider algorithms that do not use classification. In the full paper, we show that the family of any-fit algorithms is unbounded for 2-D general size items (Lemma 1). When a new item R arrives, if there are occupied bins in which R can be packed (allowing repacking for existing items), the algorithms assign R to one of these bins as follows: First-Fit (FF) assigns R to the bin which has been occupied for the longest time; Best-Fit (BF) assigns R to the heaviest loaded bin with ties broken arbitrarily; Worst-Fit (WF) assigns R to the lightest loaded bin with ties broken arbitrarily; Any-Fit (AF) assigns R to any of the bins arbitrarily.

Lemma 1. *The competitive ratio of the any-fit family of algorithms (*First-Fit*,* Best-Fit*,* Worst-Fit*, and* Any-Fit*) for the online dynamic bin packing problem of* 2*-D general size items with no classification of items is unbounded.*

When the items are unit fraction and no classification is used, we can show that FF is not c-competitive for any $c < 5.4375$, BF is not c-competitive for any $c < 9$, and WF is not c-competitive for any $c < 5.75$. The results hold even for power fraction items. These results are in contrast to the lower bound of 3.70301 of unit fraction items for any algorithm [13].

Repacking. To determine if an item can be packed into an existing bin, we will need some repacking. Here we make some simple observations about the load of items if repacking is not feasible. We first note the following lemma which is implied by Theorem 1.1 in [18].

Lemma 2 ([18]). *Given a bin with width u and height v, if all items have width at most $\frac{u}{2}$ and height at most v, then any set of these items with total area at most $\frac{uv}{2}$ can fit into the same bin by using Steinberg's algorithm.*

The implication of Lemma 2 is that if packing a new item of width $w \le \frac{u}{2}$ and height h into a bin results in infeasible packing, then the total load of the existing items is at least $\frac{uv}{2} - wh$.

Lemma 3. *Consider packing of two types of items* $\mathrm{T}(\frac{1}{2}, \le h)$ *and* $\mathrm{T}(1,*)$*, for some* $0 < h < 1$*. If we have an item of type* $\mathrm{T}(1, h')$ *that cannot be packed to an existing bin, then the current load of the bin is at least* $1 - \frac{h}{2} - h'$*.*

Proof. We first pack all items with width 1, including the new type $\mathrm{T}(1, h')$ item, one by one on top of the previous one. For the remaining space, we divide it into two equal halves each with width $\frac{1}{2}$. We then try to pack the $\mathrm{T}(\frac{1}{2}, \le h)$

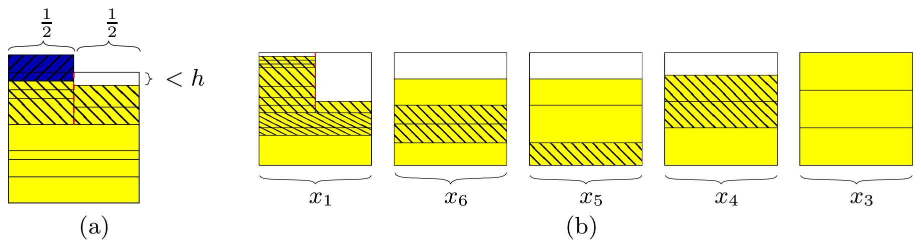


Fig. 1. (a) Infeasible repacking of existing items of types $\mathrm{T}(1, \leq\frac{1}{3})$ and $\mathrm{T}(\frac{1}{2}, \leq\frac{1}{3})$ and a new item of type $\mathrm{T}(1, *)$. The empty space has width $\frac{1}{2}$ and height less than h. (b) Illustration of the proof of Lemma 8. In each set of bins, the shaded items are the item types that do not appear in subsequent bins. For example, items of type $\mathrm{T}(\frac{1}{2}, \leq \frac{1}{3})$ in the first x_1 bins do not appear in the subsequent bins.

Table 4. Values of $\beta \langle x, y \rangle$ for $3 \leq x \leq 6$ and $3 \leq y \leq 6$

$\beta \langle x, y \rangle$	$y = 3$	4	5	6
$x = 3$	1	1	1	1
4	$\frac{3}{4}$	$\frac{5}{6} = \frac{1}{3} + \frac{1}{4} + \frac{1}{4}$	$\frac{5}{6}$	$\frac{11}{12} = \frac{2}{3} + \frac{1}{4}$
5	$\frac{7}{10} = \frac{1}{4} + \frac{1}{4} + \frac{1}{5}$	$\frac{47}{60} = \frac{1}{3} + \frac{1}{4} + \frac{1}{5}$	$\frac{5}{6}$	$\frac{17}{20} = \frac{1}{4} + \frac{3}{5}$
6	$\frac{7}{10}$	$\frac{23}{30} = \frac{1}{6} + \frac{3}{5}$	$\frac{49}{60} = \frac{1}{4} + \frac{1}{6} + \frac{2}{5}$	$\frac{17}{20}$

items into one compartment until it overflows, and then continue packing into the other compartment. The space left in the second compartment has a height less than h, otherwise, the overflow item can be packed there (see Figure 1(a)). As a result, the total load of items is at least $1 - \frac{h}{2}$. Since the new item has a load of h', the total load of existing items is at least $1 - \frac{h}{2} - h'$ as claimed. □

In the case of 1-D packing, Chan et al. [4] have defined the following notion. Let x and y be positive integers. Suppose that a 1-D bin is already packed with some items whose sizes are chosen from the set $\{1, \frac{1}{2}, \ldots, \frac{1}{x}\}$. They defined the notion of the minimum load of such a bin that an additional item of size $\frac{1}{y}$ cannot fit into the bin. We modify this notion such that the set in concern becomes $\{\frac{1}{3}, \frac{1}{4}, \ldots, \frac{1}{x}\}$. We define $\beta \langle x, y \rangle$ to be the minimum load of this bin containing items with length at least $\frac{1}{x}$ and at most $\frac{1}{3}$ such that an item of size $\frac{1}{y}$ cannot be packed into this bin. Precisely,

$$\beta \langle x, y \rangle = \min_{3 \leq j \leq x \text{ and } n_j \geq 0} \{\frac{n_3}{3} + \frac{n_4}{4} + \ldots + \frac{n_x}{x} \mid \frac{n_3}{3} + \frac{n_4}{4} + \ldots + \frac{n_x}{x} > 1 - \frac{1}{y}\}.$$

Table 4 shows the values of this function for $3 \leq x \leq 6$ and $3 \leq y \leq 6$.

Table 5. Classifications of 2-D unit fraction items and their competitive ratios

Classes	Types of items	Competitive ratios
Class 1	$\mathrm{T}(\leq\frac{1}{3}, \leq 1)$	2.8258
Class 2	$\mathrm{T}(1, \leq\frac{1}{3})$, $\mathrm{T}(\frac{1}{2}, \leq\frac{1}{3})$	1.7804
Class 3	$\mathrm{T}(1,1)$, $\mathrm{T}(1,\frac{1}{2})$, $\mathrm{T}(\frac{1}{2},1)$, $\mathrm{T}(\frac{1}{2},\frac{1}{2})$	2.25
Class 4	$\mathrm{T}(1,\frac{1}{2})$, $\mathrm{T}(1,\leq\frac{1}{3})$, $\mathrm{T}(\frac{1}{2},\frac{1}{2})$, $\mathrm{T}(\frac{1}{2},\leq\frac{1}{3})$	2.4593
Class 5	$\mathrm{T}(1,1)$, $\mathrm{T}(\frac{1}{2},1)$	1.5

3 Classification of 2-D Unit Fraction Items

Following the idea in [20], we also divide the type of items into classes. In Table 5, we list the different classes we considered in this paper. We propose two packing schemes, each of which makes use of a subset of the classes that are disjoint. The competitive ratio of a packing scheme is the sum of the competitive ratio we can achieve for each of the classes in the scheme. In this section, we focus on individual classes and in the next section, we discuss the two packing schemes. For each class, we use FF (First-Fit) to determine which bin to assign an item. For each bin, we check if the new item can be packed together with the existing items in the bin; this is done by some repacking procedures and the repacking is different for different classes.

Class 5: $\mathbf{T(1,1), T(\frac{1}{2},1)}$

This is a simple case and we skip the details.

Lemma 4. *FF is* 1.5*-competitive for UF items of types* $\mathrm{T}(1,1)$ *and* $\mathrm{T}(\frac{1}{2},1)$.

Class 3: $\mathbf{T(1,1), T(1,\frac{1}{2}), T(\frac{1}{2},1), T(\frac{1}{2},\frac{1}{2})}$

We now consider Class 3 for which both the width and height are at least $\frac{1}{2}$.

Lemma 5. *FF is* 2.25*-competitive for UF items of types* $\mathrm{T}(1,1), \mathrm{T}(1,\frac{1}{2}), \mathrm{T}(\frac{1}{2},1)$, $\mathrm{T}(\frac{1}{2},\frac{1}{2})$.

Proof. Suppose the maximum load at any time is n. Then OPT uses at least n bins. Let x_1 be the last bin that FF ever packs a $\mathrm{T}(\frac{1}{2},\frac{1}{2})$-item, $x_1 + x_2$ for $\mathrm{T}(1,\frac{1}{2})$ and $\mathrm{T}(\frac{1}{2},1)$, and $x_1 + x_2 + x_3$ for $\mathrm{T}(1,1)$. When FF packs a $\mathrm{T}(\frac{1}{2},\frac{1}{2})$-item to bin-$x_1$, all the $x_1 - 1$ before that must have a load of 1. Therefore, $(x_1 - 1) + \frac{1}{4} \leq n$. When FF packs a $\mathrm{T}(1,\frac{1}{2})$ or $\mathrm{T}(\frac{1}{2},1)$-item to bin-$(x_1 + x_2)$, all the bins before that must have a load of $\frac{1}{2}$. Hence, $\frac{x_1+x_2}{2} \leq n$. When FF packs a $\mathrm{T}(1,1)$-item to bin-$(x_1 + x_2 + x_3)$, the first x_1 bins must have a load of at least $\frac{1}{4}$, the next x_2 bins must have a load of at least $\frac{1}{2}$, and the last $x_3 - 1$ bins must have a load of 1. Therefore, $\frac{x_1}{4} + \frac{x_2}{2} + (x_3 - 1) + 1 \leq n$. The maximum value of $x_1 + x_2 + x_3$ is obtained by setting $x_1 = x_2 = n$ and $x_3 = \frac{n}{4}$. Then, $x_1 + x_2 + x_3 = 2.25n \leq 2.25OPT$. □

Class 2: $\mathbf{T(1, \leq\frac{1}{3}), T(\frac{1}{2}, \leq\frac{1}{3})}$

We now consider items whose width is at least $\frac{1}{2}$ and height is at most $\frac{1}{3}$. For this class, the repack when a new item arrives is done according to the description in the proof of Lemma 3. We are going to show that FF is 1.7804-competitive for Class 2.

Suppose the maximum load at any time is n. Let x_1 be the last bin that FF ever packs a $\mathrm{T}(\frac{1}{2}, \leq\frac{1}{3})$-item. Using the analysis in [9] for 1-D items with size at most $\frac{1}{3}$, one can show that $x_1 \leq 1.4590n$.

Lemma 6 ([9]). *Suppose we are packing UF items of types* $\mathrm{T}(1, \leq\frac{1}{3}), \mathrm{T}(\frac{1}{2}, \leq\frac{1}{3})$ *and the maximum load over time is n. We have $x_1 \leq 1.4590n$, where x_1 is the last bin that FF ever packs a* $\mathrm{T}(\frac{1}{2}, \leq\frac{1}{3})$*-item.*

Lemma 6 implies that FF only packs items of $\mathrm{T}(1, \leq\frac{1}{3})$ in bin-y for $y > 1.459n$. The following lemma further asserts that the height of these items is at least $\frac{1}{6}$.

Lemma 7. *Suppose we are packing UF items of types* $\mathrm{T}(1, \leq\frac{1}{3}), \mathrm{T}(\frac{1}{2}, \leq\frac{1}{3})$ *and the maximum load over time is n. Any item that is packed by FF to bin-y, for $y > 1.459n$, must be of type* $\mathrm{T}(1, h)$*, where* $\frac{1}{6} \leq h \leq \frac{1}{3}$*.*

Proof. Suppose on the contrary that FF packs a $\mathrm{T}(1, \leq \frac{1}{7})$-item in bin-$y$ for $y > 1.459n$. This means that packing the item in any of the first $1.459n$ bins results in an infeasible packing. By Lemma 3, with $h = \frac{1}{3}$ and $h' = \frac{1}{7}$, the load of each of the first $1.459n$ bins is at least $1 - \frac{1}{6} - \frac{1}{7} = 0.69$. Then the total is at least $1.459n \times 0.69 > 1.0067n$, contradicting that the maximum load at any time is n. Therefore, the lemma follows. □

Lemma 8. *FF is 1.7804-competitive for UF items of types* $\mathrm{T}(1, \leq\frac{1}{3})$, $\mathrm{T}(\frac{1}{2}, \leq\frac{1}{3})$.

Proof. Figure 1(b) gives an illustration. Let $(x_1 + x_6), (x_1 + x_6 + x_5), (x_1 + x_6 + x_5 + x_4)$, and $(x_1 + x_6 + x_5 + x_4 + x_3)$ be the last bin that FF ever packs a $\mathrm{T}(1, \frac{1}{6})$-, $\mathrm{T}(1, \frac{1}{5})$-, $\mathrm{T}(1, \frac{1}{4})$-, and $\mathrm{T}(1, \frac{1}{3})$- item, respectively. When FF packs a $\mathrm{T}(1, \frac{1}{6})$-item to bin-$(x_1 + x_6)$, the load of the first x_1 is at least $1 - \frac{1}{6} - \frac{1}{6} = \frac{2}{3}$, by Lemma 3. By Lemma 7, only type $\mathrm{T}(1, k)$-item, for $\frac{1}{6} \leq k \leq \frac{1}{3}$, could be packed in the x_6 bins. These items all have width 1 and thus can be considered as 1-D case. Therefore, when we cannot pack a $\mathrm{T}(1, \frac{1}{6})$-item, the current load must be at least $\beta\,\langle 6, 6\rangle$. Then we have $x_1(\frac{2}{3}) + x_6\,\beta\,\langle 6, 6\rangle \leq n$. Similarly, we have

1. $x_1(\frac{2}{3}) + x_6\,\beta\,\langle 6, 6\rangle \leq n$,
2. $x_1(1 - \frac{1}{6} - \frac{1}{5}) + x_6\,\beta\,\langle 6, 5\rangle + x_5\,\beta\,\langle 5, 5\rangle \leq n$,
3. $x_1(1 - \frac{1}{6} - \frac{1}{4}) + x_6\,\beta\,\langle 6, 4\rangle + x_5\,\beta\,\langle 5, 4\rangle + x_4\,\beta\,\langle 4, 4\rangle \leq n$,
4. $x_1(1 - \frac{1}{6} - \frac{1}{3}) + x_6\,\beta\,\langle 6, 3\rangle + x_5\,\beta\,\langle 5, 3\rangle + x_4\,\beta\,\langle 4, 3\rangle + x_3\,\beta\,\langle 3, 3\rangle \leq n$.

We note that for each inequality, the coefficients are increasing, e.g., for (1), we have $\frac{2}{3} \leq \beta\,\langle 6, 6\rangle = \frac{17}{20}$, by Table 4. Therefore, the maximum value of $x_1 + x_6 + x_5 + x_4 + x_3$ is obtained by setting the maximum possible value of x_6 satisfying (1), and then the maximum possible value of x_5 satisfying (2), and so on. Using Table 4, we compute the corresponding values as $1.4590n$, $0.0322n$, $0.0597n$, $0.0931n$ and $0.1365n$, respectively. As a result, $x_1 + x_6 + x_5 + x_4 + x_3 \leq 1.7804n \leq 1.7804OPT$. □

Table 6. Competitive ratios for 2-D unit fraction items

2DDynamicPackUFS1		
Classes	Types of items	Competitive ratios
Class 1	$\mathrm{T}(\leq\frac{1}{3}, \leq 1)$	2.8258
Class 4	$\mathrm{T}(1, \frac{1}{2})$, $\mathrm{T}(1, \leq\frac{1}{3})$, $\mathrm{T}(\frac{1}{2}, \frac{1}{2})$, $\mathrm{T}(\frac{1}{2}, \leq\frac{1}{3})$	2.4593
Class 5	$\mathrm{T}(1, 1)$, $\mathrm{T}(\frac{1}{2}, 1)$	1.5
Overall	All of the above	6.7850

Class 1: $\mathbf{T}(\leq\frac{1}{3}, \leq 1)$

Items of type $\mathrm{T}(\leq\frac{1}{3}, \leq 1)$ are further divided into three subtypes: $\mathrm{T}(\leq\frac{1}{3}, \leq\frac{1}{3})$, $\mathrm{T}(\leq\frac{1}{3}, \frac{1}{2})$, and $\mathrm{T}(\leq\frac{1}{3}, 1)$. We describe how to repack these items and leave the analysis in the full paper.

1. When the new item is $\mathrm{T}(\leq\frac{1}{3}, \leq\frac{1}{3})$, we use Steinberg's algorithm [18] to repack the new and existing items. Note that the item width satisfies the criteria of Lemma 2.
2. When the new item is $\mathrm{T}(\leq\frac{1}{3}, \frac{1}{2})$ or $\mathrm{T}(\leq\frac{1}{3}, 1)$ and the bin contains $\mathrm{T}(\leq\frac{1}{3}, \leq\frac{1}{3})$-item, we divide the bin into two compartments, one with width $\frac{1}{3}$ and the other $\frac{2}{3}$ and both with height 1. We reserve the small compartment for the new item and try to repack the existing items in the large compartment using Steinberg's algorithm. This idea originates from [20].
3. When the new item is $\mathrm{T}(\leq\frac{1}{3}, \frac{1}{2})$ or $\mathrm{T}(\leq\frac{1}{3}, 1)$ and the bin does not contain $\mathrm{T}(\leq\frac{1}{3}, \leq\frac{1}{3})$-item, we use the repacking method as in Lemma 3 but with the width becoming the height and vice versa. Note that this implies that Lemma 8 applies for these items.

Lemma 9. *FF is* 2.8258*-competitive for UF items of type* $\mathrm{T}(\leq\frac{1}{3}, \leq 1)$.

Class 4: $\mathbf{T}(1, \frac{1}{2}), \mathbf{T}(1, \leq\frac{1}{3}), \mathbf{T}(\frac{1}{2}, \frac{1}{2}), \mathbf{T}(\frac{1}{2}, \leq\frac{1}{3})$

The analysis of Class 4 follows a similar framework as in Class 2. We state the result (Lemma 10) and leave the proof in the full paper.

Lemma 10. *FF is* 2.4593*-competitive for UF items of types* $\mathrm{T}(1, \frac{1}{2})$, $\mathrm{T}(1, \leq\frac{1}{3})$, $\mathrm{T}(\frac{1}{2}, \frac{1}{2})$, $\mathrm{T}(\frac{1}{2}, \leq\frac{1}{3})$.

4 Packing of 2-D Unit Fraction Items

Our algorithm, named as 2DDynamicPackUF, classifies items into classes and then pack items in each class independent of other classes. In each class, FF is used to pack the items as described in Section 3. In this section, we present two schemes and show their competitive ratios.

Table 6 shows the classification and associated competitive ratios for 2D-DynamicPackUFS1. This scheme contains Classes 1, 4, and 5, covering all items.

Table 7. Competitive ratios for 2-D power fraction items. Marked with [*] are the competitive ratios that are reduced as compared to unit fraction items.

2DDynamicPackPF		
Class	Types of items	Competitive ratios
Class 1	T($\leq\frac{1}{4}$, $\leq$1)	2.4995 [*]
Class 2	T(1, $\leq\frac{1}{4}$), T($\frac{1}{2}$, $\leq\frac{1}{4}$)	1.496025 [*]
Class 3	T(1, 1), T(1, $\frac{1}{2}$), T($\frac{1}{2}$, 1), T($\frac{1}{2}$, $\frac{1}{2}$)	2.25
Overall	All items	6.2455

Theorem 1. 2DDynamicPackUFS1 *is* 6.7850*-competitive for* 2*-D UF items.*

Scheme 2DDynamicPackUFS2 has a higher competitive ratio than Scheme 2DDynamicPackUFS1, nevertheless, Scheme 2DDynamicPackUFS2 has a smaller competitive ratio for power fraction items to be discussed in the next section. 2DDynamicPackUFS2 contains Classes 1, 2, and 3, covering all items.

Lemma 11. 2DDynamicPackUFS2 *is* 6.8561*-competitive for* 2*-D UF items.*

5 Adaptations to Other Scenarios

In this section we extend our results to other scenarios.

2-D Power Fraction Items. Table 7 shows a scheme based on 2DDynamicPackUFS2 for unit fraction items and the competitive ratio is reduced to 6.2455.

Theorem 2. 2DDynamicPackPF *is* 6.2455*-competitive for* 2*-D PF items.*

3-D Unit and Power Fraction Items. The algorithm in [20] effectively classifies the unit fraction items as shown in Table 8(a). The overall competitive ratio

Table 8. (a) Competitive ratios for 3-D UF items. [*] This result uses Theorem 1. [**] This result uses Lemma 9. (b) Competitive ratios for 3-D PF items. [*] This result uses Theorem 2. [**] This result uses the competitive ratio of Class 1 2-D PF items.

(a)

3DDynamicPackUF [20]		
Classes	Types of items	Competitive ratios
Class 1	T($>\frac{1}{2}$, $*$, $*$)	6.7850 [*]
Class 2	T($\leq\frac{1}{2}$, $>\frac{1}{2}$, $*$)	4.8258 [**]
Class 3	T($\leq\frac{1}{2}$, $(\frac{1}{3}, \frac{1}{2}]$, $*$)	4
Class 4	T($\leq\frac{1}{2}$, $\leq\frac{1}{3}$, $*$)	6
Overall	All items	21.6108

(b)

3DDynamicPackPF		
Classes	Types of items	Competitive ratios
Class 1	T($>\frac{1}{2}$, $*$, $*$)	6.2455 [*]
Class 2	T($\leq\frac{1}{2}$, $>\frac{1}{2}$, $*$)	4.4995 [**]
Class 3	T($\leq\frac{1}{2}$, $(\frac{1}{4}, \frac{1}{2}]$, $*$)	4
Class 4	T($\leq\frac{1}{2}$, $\leq\frac{1}{4}$, $*$)	5.334
Overall	All items	20.0783

reduces from 22.788 to 21.6108. For power fraction items we slightly modify the classification for 3-D items, such that boundary values of $\frac{1}{3}$ are replaced by $\frac{1}{4}$. Table 8(b) details this classification. The overall competitive ratio reduces to 20.0783. We state the following theorem and leave the proof in the full paper.

Theorem 3. (1) *Algorithm* 3DDynamicPackUF *is* 21.6108*-competitive for UF items and* (2) *algorithm* 3DDynamicPackPF *is* 20.0783*-competitive for PF items.*

6 Conclusion

We have extended the study of 2-D and 3-D dynamic bin packing problem to unit and power fraction items. We have improved the competitive ratios that would be obtained using only existing results for unit fraction items from 7.4842 to 6.7850 for 2-D, and from 22.4842 to 21.6108 for 3-D. For power fraction items, the competitive ratios are further reduced to 6.2455 and 20.0783 for 2-D and 3-D, respectively. Our approach is to divide items into classes and analyzing each class individually. We have proposed several classes and defined different packing schemes based on the classes. This approach gives a systematic way to explore different combinations of classes.

An open problem is to further improve the competitive ratios for various types of items. The gap between the upper and lower bounds could also be reduced by improving the lower bounds. Another problem is to consider multi-dimensional bin packing. For d-dimensional static and dynamic bin packing, for $d \geq 2$, the competitive ratio grows exponentially with d. Yet there is no matching lower bound that also grows exponentially with d. It is believed that this is the case [11] and any such lower bound would be of great interest.

Another direction is to consider the packing of unit fraction and power fraction squares, where all sides of an item are the same length. We note that the competitive ratio for the packing of 2-D unit fraction square items would reduce to 3.9654 compared to the competitive ratio of 2-D general size square items of 4.2154 [13]. For 3-D unit fraction squares, this would reduce to 5.24537 compared to 5.37037 for 3-D general size squares [13].

References

1. Bar-Noy, A., Ladner, R.E.: Windows scheduling problems for broadcast systems. SIAM J. Comput. 32, 1091–1113 (2003)
2. Bar-Noy, A., Ladner, R.E., Tamir, T.: Windows scheduling as a restricted version of bin packing. ACM Trans. Algorithms 3 (August 2007)
3. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press, New York (1998)
4. Chan, J.W.-T., Lam, T.-W., Wong, P.W.H.: Dynamic bin packing of unit fractions items. Theoretical Computer Science 409(3), 521–529 (2008)
5. Chan, J.W.-T., Wong, P.W.H., Yung, F.C.C.: On dynamic bin packing: An improved lower bound and resource augmentation analysis. Algorithmica 53, 172–206 (2009)

6. Coffman Jr., E.G., Courcoubetis, C., Garey, M.R., Johnson, D.S., Shor, P.W., Weber, R.R., Yannakakis, M.: Bin packing with discrete item sizes, Part I: Perfect packing theorems and the average case behavior of optimal packings. SIAM J. Discrete Math. 13, 38–402 (2000)
7. Coffman Jr., E.G., Galambos, G., Martello, S., Vigo, D.: Bin packing approximation algorithms: Combinatorial analysis. In: Du, D.Z., Pardalos, P.M. (eds.) Handbook of Combinatorial Optimization (1998)
8. Coffman Jr., E.G., Garey, M.R., Johnson, D.: Bin packing with divisible item sizes. Journal of Complexity 3, 405–428 (1987)
9. Coffman Jr., E.G., Garey, M.R., Johnson, D.S.: Dynamic bin packing. SIAM J. Comput. 12(2), 227–258 (1983)
10. Coffman Jr., E.G., Garey, M.R., Johnson, D.S.: Approximation algorithms for bin packing: A survey. In: Hochbaum, D.S. (ed.) Approximation Algorithms for NP-Hard Problems, pp. 46–93. PWS Publishing (1996)
11. Coppersmith, D., Raghavan, P.: Multidimensional on-line bin packing: Algorithms and worst-case analysis. Operations Research Letters 8(1), 17–20 (1989)
12. Csirik, J., Woeginger, G.J.: On-line packing and covering problems. In: Fiat, A., Woeginger, G.J. (eds.) On-line Algorithms–The State of the Art, pp. 147–177. Springer (1996)
13. Epstein, L., Levy, M.: Dynamic multi-dimensional bin packing. J. of Discrete Algorithms 8, 356–372 (2010)
14. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
15. Han, X., Peng, C., Ye, D., Zhang, D., Lan, Y.: Dynamic bin packing with unit fraction items revisited. Inf. Process. Lett. 110, 1049–1054 (2010)
16. Jansen, K., Thöle, R.: Approximation algorithms for scheduling parallel jobs: Breaking the approximation ratio of 2. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 234–245. Springer, Heidelberg (2008)
17. Miyazawa, F., Wakabayashi, Y.: Two- and three-dimensional parametric packing. Computers & Operations Research 34(9), 2589–2603 (2007)
18. Steinberg, A.: A strip-packing algorithm with absolute performance bound 2. SIAM J. Comput. 26(2), 401–409 (1997)
19. van Stee, R.: Combinatorial algorithms for packing and scheduling problems. Habilitation thesis, Universität Karlsruhe (June 2008), `http://www.mpi-inf.mpg.de/~vanstee/habil.pdf` (accessed November 2012)
20. Wong, P.W.H., Yung, F.C.C.: Competitive multi-dimensional dynamic bin packing via L-shape bin packing. In: Bampis, E., Jansen, K. (eds.) WAOA 2009. LNCS, vol. 5893, pp. 242–254. Springer, Heidelberg (2010)
21. Wong, P.W.H., Yung, F.C.C., Burcea, M.: An 8/3 lower bound for online dynamic bin packing. In: Chao, K.-M., Hsu, T.-S., Lee, D.-T. (eds.) ISAAC 2012. LNCS, vol. 7676, pp. 44–53. Springer, Heidelberg (2012)

A Greedy Approximation Algorithm for Minimum-Gap Scheduling

Marek Chrobak[1,⋆], Uriel Feige[2], Mohammad Taghi Hajiaghayi[3,⋆⋆], Sanjeev Khanna[4], Fei Li[5,⋆⋆⋆], and Seffi Naor[6]

[1] Department of Computer Science, Univ. of California at Riverside, USA
[2] Department of Computer Science and Applied Mathematics, The Weizmann Institute, Israel
[3] Computer Science Department, Univ. of Maryland, College Park, USA
[4] Department of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, USA
[5] Department of Computer Science, George Mason University, USA
[6] Computer Science Department, Technion, Israel

Abstract. We consider scheduling of unit-length jobs with release times and deadlines to minimize the number of gaps in the schedule. The best algorithm for this problem runs in time $O(n^4)$ and requires $O(n^3)$ memory. We present a simple greedy algorithm that approximates the optimum solution within a factor of 2 and show that our analysis is tight. Our algorithm runs in time $O(n^2 \log n)$ and needs only $O(n)$ memory. In fact, the running time is $O(ng^* \log n)$, where g^* is the minimum number of gaps.

1 Introduction

Research on approximation algorithms up to date has focussed mostly on optimization problems that are ℕℙ-hard. From the purely practical point of view, however, there is little difference between exponential and high-degree polynomial running times. Memory requirements could also be a critical factor, because high-degree polynomial algorithms typically involve computing entries in a high-dimensional table via dynamic programming. An algorithm requiring $O(n^4)$ or more memory would be impractical even for relatively modest values of n because when the main memory fills up, disk paging will considerably slow down the (already slow) execution. With this in mind, for such problems it is natural to ask whether there are faster algorithms that use little memory and produce near-optimal solutions. This direction of research is not entirely new. For example, in recent years, approximate streaming algorithms have been extensively studied for problems that are polynomially solvable, but where massive amounts of data need to be processed in nearly linear time.

⋆ Research supported by NSF grant CCF-1217314.
⋆⋆ Supported in part by NSF CAREER award 1053605, ONR YIP award N000141110662, and a University of Maryland Research and Scholarship Award (RASA). The author is also with AT&T Labs–Research.
⋆⋆⋆ Research supported by NSF grants CCF-0915681 and CCF-1146578.

In this paper we focus on the problem of minimum-gap job scheduling, where the objective is to schedule a collection of unit-length jobs with given release times and deadlines, in such a way that the number of gaps (idle intervals) in the schedule is minimized. This scheduling paradigm was originally proposed, in a somewhat more general form, by Irani and Pruhs [7]. The first polynomial-time algorithm for this problem, with running time $O(n^7)$, was given by Baptiste [3]. This was subsequently improved by Baptiste *et al.* [4], who gave an algorithm with running time $O(n^4)$ and space complexity $O(n^3)$. All these algorithms are based on dynamic programming.

Our Results. We give a simple, greedy algorithm for minimum-gap scheduling of unit-length jobs that computes a near-optimal solution. Our algorithm runs in time $O(n^2 \log n)$, uses only $O(n)$ space, and it approximates the optimum within a factor of 2. More precisely, if the optimal schedule has g^* gaps, our algorithm will find a schedule with at most $2g^* - 1$ gaps (assuming $g^* \geq 1$). The running time can in fact be expressed as $O(ng^* \log n)$; thus, since $g^* \leq n$, the algorithm is considerably faster if the optimum is small. (To be fair, so is the algorithm in [3], whose running time can be reduced to $O(n^3 g^*)$.) The idea of the algorithm is to add gaps one by one, at each step adding the longest gap for which there exists a feasible schedule. Our analysis is based on new structural properties of schedules with gaps, that may be of independent interest.

Related Work. Prior to the paper by Baptiste [3], Chretienne [5] studied versions of scheduling where only schedules without gaps are allowed. The algorithm in [3] can be extended to handle jobs of arbitrary length, with preemptions, although then the time complexity increases to $O(n^5)$. Working in another direction, Demaine *et al.* [6] showed that for p processors the gap minimization problem can be solved in time $O(n^7 p^5)$ if jobs have unit lengths.

The generalization of minimum-gap scheduling proposed by Irani and Pruhs [7] is concerned with computing minimum-energy schedules in the model where the processor uses energy at constant rate when processing jobs, but it can be turned off during the idle periods with some additive energy penalty representing an overhead for turning the power back on. If this penalty is at most 1 then the problem is equivalent to minimizing the number of gaps. The algorithms from [3] can be extended to this power-down model without increasing their running times. Note that our approximation ratio is even better if we express it in terms of the energy function: since both the optimum and the algorithm pay n for job processing, the ratio can be bounded by $1 + g^*/(n + g^*)$. Thus the ratio is at most 1.5, and it is only $1 + o(1)$ if $g^* = o(n)$.

The power-down model from [7] can be generalized further to include the speed-scaling capability. The reader is referred to surveys in [1,7], for more information on the models involving speed-scaling.

2 Preliminaries

We assume that the time axis is partitioned into unit-length time slots numbered $0, 1, \ldots$. By $\mathcal{J}$ we will denote the instance, consisting of a set of unit-length jobs numbered $1, 2, ..., n$, each job j with a given release time r_j and deadline d_j, both integers. Without loss of generality, $r_j \leq d_j$ for each j. By a standard exchange argument, we can also assume that all release times are distinct and that all deadlines are distinct. By $r_{\min} = \min_j r_j$ and $d_{\max} = \max_j d_j$ we denote the earliest release time and the latest deadline, respectively.

A *(feasible) schedule* S of $\mathcal{J}$ is a function that assigns jobs to time slots such that each job j is assigned to a slot $t \in [r_j, d_j]$ and different jobs are assigned to different slots. If j is assigned by S to a slot t then we say that j is *scheduled* in S at time t. If S schedules a job at time t then we say that slot t is *busy*; otherwise we call it *idle*. The *support* of a schedule S, denoted $\mathsf{Supp}(S)$, is the set of all busy slots in S. An inclusion-maximal interval consisting of busy slots is called a *block*. A block starting at $r_{\min}$ or ending at $d_{\max}$ is called *exterior* and all other blocks are called *interior*. Any inclusion-maximal interval of idle slots between $r_{\min}$ and $d_{\max}$ is called a *gap*. Note that if there are idle slots between $r_{\min}$ and the first job then they also form a gap and there is no left exterior block, and a similar property holds for the idle slots right before $d_{\max}$. To avoid this, we will assume that jobs 1 and n are tight jobs with $r_1 = d_1 = r_{\min}$ and $r_n = d_n = d_{\max}$, so these jobs must be scheduled at $r_{\min}$ and $d_{\max}$, respectively, and each schedule must have both exterior blocks. We can modify any instance to have this property by adding two such jobs to it, without changing the number of gaps in the optimum solution.

Throughout the paper we assume that the given instance $\mathcal{J}$ is *feasible*, that is, it has a schedule. Feasibility can be checked by running the greedy earliest-deadline-first algorithm (EDF): process the time slots from left to right and at each step schedule the earliest-deadline pending job, if there is any. Then $\mathcal{J}$ is feasible if and only if no job misses its deadline in EDF. Also, since a schedule can be thought of as a bipartite matching between jobs and time slots, by a simple adaptation of Hall's theorem we obtain that $\mathcal{J}$ is feasible if and only if for any time interval $[t, u]$ we have $|Load(t, u)| \leq u - t + 1$, where $Load(t, u) = \{j : t \leq r_j \leq d_j \leq u\}$ is the set of jobs that must be scheduled in $[t, u]$.

Scheduling with Forbidden Slots. We consider a more general model where some slots in $[r_{\min}, d_{\max}]$ are designated as *forbidden*, namely no job is allowed to be scheduled in them. A schedule of $\mathcal{J}$ that does not schedule any jobs in a set Z of forbidden slots is said to *obey* Z. A set Z of forbidden slots will be called *viable* if there is a schedule that obeys Z. Formally, we can think of a schedule with forbidden slots as a pair (S, Y), where Y is a set of forbidden slots and S is a schedule that obeys Y. However, we will avoid this formalism as the set Y of forbidden slots associated with S will be always understood from context.

All definitions and properties above extend naturally to scheduling with forbidden slots. Now for any schedule S we have three types of slots: *busy*, *idle* and *forbidden*. The *support* is now defined as the set of slots that are either busy or forbidden, and a block is a maximal interval consisting of slots in the support. The support uniquely determines our objective function (the number of gaps), and thus we will be mainly interested in the support of the schedules we consider, rather than in the exact mapping from jobs to slots. The criterion for feasibility generalizes naturally to scheduling with forbidden slots, as follows: a forbidden set Z is viable if and only if $|Load(t,u)| \le |[t,u] - Z|$, holds for all $t \le u$, where $[t,u] - Z$ is the set of non-forbidden slots between t and u (inclusive).

3 Transfer Paths

Let Q be a feasible schedule. Consider a sequence $\boldsymbol{t} = (t_0, t_1, ..., t_k)$ of different time slots such that $t_0, ..., t_{k-1}$ are busy and t_k is idle in Q. Let j_a be the job scheduled by Q in slot t_a, for $a = 0, ..., k-1$. We will say that $\boldsymbol{t}$ is a *transfer path for* Q (or simply a *transfer path* if Q is understood from context) if $t_{a+1} \in [r_{j_a}, d_{j_a}]$ for all $a = 0, ..., k-1$. Given such a transfer path $\boldsymbol{t}$, the *shift operation along* $\boldsymbol{t}$ moves each j_a from slot t_a to slot t_{a+1}. For technical reasons we allow $k = 0$ in the definition of transfer paths, in which case t_0 itself is idle, $\boldsymbol{t} = (t_0)$, and no jobs will be moved by the shift.

Note that if $Z = \{t_0\}$ is a forbidden set that consists of only one slot t_0, then the shift operation will convert Q into a new schedule that obeys Z. To generalized this idea to arbitrary forbidden sets, we prove the lemma below.

Lemma 1. *Let Q be a feasible schedule. Then a set Z of forbidden slots is viable if and only if there are $|Z|$ disjoint transfer paths in Q starting in Z.*

Proof. ($\Leftarrow$) This implication is simple: For each $x \in Z$ perform the shift operation along the path starting in x, as defined before the lemma. The resulting schedule Q' is feasible and it does not schedule any jobs in Z, so Z is viable.

($\Rightarrow$) Let S be an arbitrary schedule that obeys Z. Consider a bipartite graph $\mathcal{G}$ whose vertex set consists of jobs and time slots, with job j connected to slot t if $t \in [r_j, d_j]$. Then both Q and S can be thought of as perfect matchings in $\mathcal{G}$, in the sense that all jobs are matched to some slots. In S, all jobs will be matched to non-forbidden slots. There is a set of disjoint alternating paths in $\mathcal{G}$ (that alternate between the edges of Q and S) connecting slots that are not matched in S to those that are not matched in Q. Slots that are not matched in both schedules form trivial paths, that consist of just one vertex.

Consider a slot x that is not matched in S. In other words, x is either idle or forbidden in schedule S. The alternating path in $\mathcal{G}$ starting at x, expressed as a list of vertices, has the form: $x = t_0 - j_0 - t_1 - j_1 - ... - j_{k-1} - t_k$, where, for each $a = 0, ..., k-1$, j_a is the job scheduled at t_a in Q and at t_{a+1} in S, and t_k is idle in Q. Therefore this path defines uniquely a transfer path $\boldsymbol{t} = (t_0, t_1, ..., t_k)$ for slot x of Q. Note that if t_0 is idle in Q then this path is trivial – it ends at t_0. This way we obtain $|Z|$ disjoint transfer paths for all slots $x \in Z$, as claimed. $\square$

Any set $\mathcal{P}$ of transfer paths that satisfies Lemma 1 will be called a *Z-transfer multi-path for Q*. We will omit the attributes Z and/or Q if they are understood from context. By performing the shifts along the paths in $\mathcal{P}$ we can convert Q into a new schedule S that obeys Z. For brevity, we will write $S = \mathsf{Shift}(Q, \mathcal{P})$.

Next, we would like to show that Q has a Z-transfer multi-path with a regular structure, where each path proceeds in one direction (either left or right) and where different paths do not "cross" (in the sense formalized below). This property is generally not true, but we show that in such a case Q can be replaced by a schedule with the same support that satisfies these properties.

To formalize the above intuition we need a few more definitions. If $\boldsymbol{t} = (t_0, ..., t_k)$ is a transfer path then any pair of slots (t_a, t_{a+1}) in $\boldsymbol{t}$ is called a *hop* of $\boldsymbol{t}$. The length of hop (t_a, t_{a+1}) is $|t_a - t_{a+1}|$. The *hop length* of $\boldsymbol{t}$ is the sum of the lengths of its hops, that is $\sum_{a=0}^{k-1} |t_a - t_{a+1}|$. A hop (t_a, t_{a+1}) of $\boldsymbol{t}$ is *leftward* if $t_a > t_{a+1}$ and *rightward* otherwise. We say that $\boldsymbol{t}$ is *leftward* (resp.*rightward*) if all its hops are leftward (resp. *rightward*). A path that is either leftward or rightward will be called *straight*. Trivial transfer paths are considered both leftward and rightward.

For two non-trivial transfer paths $\boldsymbol{t} = (t_0, ..., t_k)$ and $\boldsymbol{u} = (u_0, ..., u_l)$, we say that $\boldsymbol{t}$ and $\boldsymbol{u}$ *cross* if there are indices a, b for which one of the following four-conditions holds: $t_a < u_{b+1} < t_{a+1} < u_b$ or $u_b < t_{a+1} < u_{b+1} < t_a$, or $t_{a+1} < u_b < t_a < u_{b+1}$, or $u_{b+1} < t_a < u_b < t_{a+1}$. If such a, b exist, we will also refer to the pair of hops (t_a, t_{a+1}) and (u_b, u_{b+1}) as a *crossing*. One can think of the first two cases as "inward" crossings, with the two hops directed towards each other, and the last two cases as "outward" crossings.

Lemma 2. *Let Q be a feasible schedule and let Z be a viable forbidden set. Then there is a schedule Q' such that* (i) *Supp*(Q') = *Supp*(Q), *and* (ii) *Q' has a Z-transfer multi-path $\mathcal{P}$ in which all paths in $\mathcal{P}$ are straight and do not cross.*

Proof. We only show here how to remove crossings. (The complete proof will appear in the full paper.) Consider now two paths in $\mathcal{R}$ that cross, $\boldsymbol{t} = (t_0, ..., t_k)$ and $\boldsymbol{u} = (u_0, ..., u_l)$. We can assume that the hops that cross are (t_a, t_{a+1}) and (u_b, u_{b+1}), where $t_a < t_{a+1}$. We have two cases. If $t_a < u_{b+1} < t_{a+1} < u_b$, then we replace $\boldsymbol{t}$ and $\boldsymbol{u}$ in $\mathcal{R}$ by paths $(t_0, ..., t_a, u_{b+1}, ..., u_l)$ and $(u_0, ..., u_b, t_{a+1}, ..., t_k)$. (See Figure 1 for illustration.) It is easy to check that these two paths are indeed correct transfer paths starting at t_0 and u_0 and ending at u_l and t_k, respectively. The second case is when $u_{b+1} < t_a < u_b < t_{a+1}$. In this case we also need to modify the schedule by swapping the jobs in slots t_a and u_b. Then we replace $\boldsymbol{t}$ and $\boldsymbol{u}$ in $\mathcal{R}$ by $(t_0, ..., t_a, u_{b+1}, ..., u_l)$ and $(u_0, ..., u_b, t_{a+1}, ..., t_k)$.

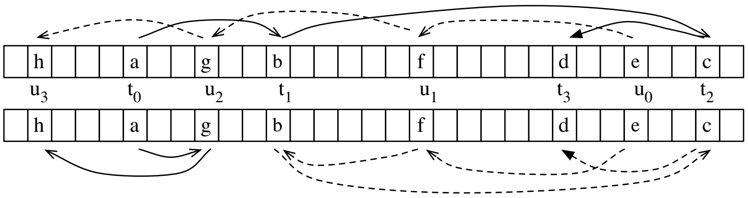


Fig. 1. Removing path crossings in the proof of Lemma 2

Each of the operations above reduces the total hop length of $\mathcal{R}$; thus, after a sufficient number of repetitions we must obtain a set $\mathcal{R}$ of transfer paths without crossings. Also, these operations do not change the support of the schedule. Let Q' be the schedule Q after the steps above and let $\mathcal{P}$ be the final set $\mathcal{R}$ of the transfer paths. Then Q' and $\mathcal{P}$ satisfy the properties in the lemma. $\square$

Note that even if $\mathcal{P}$ satisfies Lemma 2, it is still possible that opposite-oriented paths traverse over the same slots. If this happens, however, then one of the paths must be completely "covered" by a hop of the other path.

Corollary 1. *Assume that $\mathcal{P}$ is a Z-transfer multi-path for Q that satisfies Lemma 2, and let $\boldsymbol{t} = (t_0, ..., t_k)$ and $\boldsymbol{u} = (u_0, ..., u_l)$ be two paths in $\mathcal{P}$, where $\boldsymbol{t}$ is leftward and $\boldsymbol{u}$ is rightward. If there are any indices a, b such that $t_{a+1} < u_b < t_a$ then $t_{a+1} < u_0 < u_l < t_a$, that is the whole path $\boldsymbol{u}$ is between t_{a+1} and t_a. An analogous statement holds if $\boldsymbol{t}$ is rightward and $\boldsymbol{u}$ is leftward.*

4 The Greedy Algorithm

Our greedy algorithm LVG (for Longest-Viable-Gap) is simple: at each step it creates a maximum-length gap that can be feasibly added to the schedule. More formally, we describe this algorithm using the terminology of forbidden slots.

Algorithm LVG: Initialize $Z_0 = \emptyset$. The algorithm works in stages. In stage $s = 1, 2, ...$, we do this: If Z_{s-1} is an inclusion-maximal forbidden set that is viable for $\mathcal{J}$ then schedule $\mathcal{J}$ in the set $[r_{\min}, d_{\max}] - Z_{s-1}$ of time slots and output the computed schedule S_{LVG}. (The forbidden regions then become the gaps of S_{LVG}.) Otherwise, find the longest interval $X_s \subseteq [r_{\min}, d_{\max}] - Z_{s-1}$ for which $Z_{s-1} \cup X_s$ is viable and add X_s to Z_{s-1}, that is $Z_s \leftarrow Z_{s-1} \cup X_s$.

After each stage the set Z_s of forbidden slots is a disjoint union of the forbidden intervals added at stages $1, 2, ..., s$. In fact, any two consecutive forbidden intervals in Z_s must be separated by at least one busy time slot.

In this section we show that the number of gaps in schedule S_{LVG} is within a factor of two from the optimum. More specifically, we show that the number of gaps is at most $2g^* - 1$, where g^* is the minimum number of gaps. (We assume that $g^* \geq 1$, since for $g^* = 0$ S_{LVG} will not contain any gaps.)

Proof Outline. In our proof, we start with an optimal schedule Q_0, namely the one with g^* gaps, and we gradually modify it by introducing forbidden regions computed by Algorithm LVG. The resulting schedule, as it evolves, will be called the *reference schedule* and denoted Q_s. The construction of Q_s will ensure that it obeys Z_s, that the blocks of Q_{s-1} will be contained in blocks of Q_s, and that each block of Q_s contains some block of Q_{s-1}. As a result, each gap in the reference schedule shrinks over time and will eventually disappear.

The idea of the analysis is to charge forbidden regions either to the blocks or to the gaps of Q_0. We show that there are two types of forbidden regions, called *oriented* and *disoriented*, that each interior block of Q_0 can intersect at most one disoriented region, and that introducing each oriented region causes at least one gap in the reference schedule to disappear. Further, each disoriented region intersects a block of Q_0. As a result, the total number of forbidden regions is at most the number of interior blocks plus the number of gaps in Q_0, which add up to $2g^* - 1$.

Construction of Reference Schedules. Let m be the number of stages of Algorithm LVG and $Z = Z_m$. For the rest of the proof we fix a Z-transfer multi-path $\mathcal{P}$ for Q_0 that satisfies Lemma 2, that is all paths in $\mathcal{P}$ are straight and they do not cross. For any s, define $\mathcal{P}_s$ to be the set of those paths in $\mathcal{P}$ that start in the slots of Z_s. In particular, we have $\mathcal{P} = \mathcal{P}_m$.

To formalize the desired relation between consecutive reference schedules, we introduce another definition. Consider two schedules Q, Q', where Q obeys a forbidden set Y and Q' obeys a forbidden region Y' such that $Y \subseteq Y'$. We will say that Q' is an *augmentation* of Q if (a1) $\mathsf{Supp}(Q) \subseteq \mathsf{Supp}(Q')$, and (a2) each block of Q' contains a block of Q. Recall that, by definition, forbidden slots are included in the support. Immediately from the above definition we obtain that if Q' is an augmentation of Q then the number of gaps in Q' does not exceed the number of gaps in Q.

Our objective is now to convert each $\mathcal{P}_s$ into another Z_s-transfer multi-path $\widehat{\mathcal{P}}_s$ such that if we take $Q_s = \mathsf{Shift}(Q_0, \widehat{\mathcal{P}}_s)$ then each Q_s will satisfy Lemma 2 and will be an augmentation of Q_{s-1}. For each path $\boldsymbol{t} = (t_0, ..., t_k) \in \mathcal{P}_s$, $\widehat{\mathcal{P}}_s$ will contain a *truncation* of $\boldsymbol{t}$, defined as a path $\hat{\boldsymbol{t}} = (t_0, ..., t_a, \tau)$, for some index a and slot $\tau \in (t_a, t_{a+1}]$.

We now describe the *truncation process*, an iterative procedure that constructs the reference schedules. The construction runs parallel to the algorithm. Fix some arbitrary stage s, suppose that we already have computed $\widehat{\mathcal{P}}_{s-1}$ and Q_{s-1}, and now we show how to construct $\widehat{\mathcal{P}}_s$ and Q_s. We first introduce some concepts and properties:

– $\mathcal{R}$ is a set of transfer paths, $\mathcal{R} \subseteq \mathcal{P}_s$. It is initialized to $\mathcal{P}_{s-1}$ and at the end of the stage we will have $\mathcal{R} = \mathcal{P}_s$. The cardinality of $\mathcal{R}$ is non-decreasing, but not the set $\mathcal{R}$ itself; that is, some paths may get removed from $\mathcal{R}$ and replaced by other paths. Naturally, $\mathcal{R}$ is a Y-transfer multi-path for Q_0, where Y is the set of starting slots of the paths in $\mathcal{R}$. Y will be initially equal to Z_{s-1} and at the end of the stage it will become Z_s. Since Y is implicitly defined by $\mathcal{R}$, we will not specify how it is updated.
– An any iteration, for each path $\boldsymbol{t} \in \mathcal{R}$ we maintain its truncation $\hat{\boldsymbol{t}}$. Let $\widehat{\mathcal{R}} = \{\hat{\boldsymbol{t}} : \boldsymbol{t} \in \mathcal{R}\}$. At each step $\widehat{\mathcal{R}}$ is a Y-transfer multi-path for Q_0, for Y defined above. Initially $\widehat{\mathcal{R}} = \widehat{\mathcal{P}}_{s-1}$ and when the stage ends we set $\widehat{\mathcal{P}}_s = \widehat{\mathcal{R}}$.
– W is a schedule initialized to Q_{s-1}. We will maintain the invariant that W obeys Y and $W = \mathsf{Shift}(Q_0, \widehat{\mathcal{R}})$. At the end of the stage we will set $Q_s = W$.

Consider now some step of this process. If $\mathcal{R} = \mathcal{P}_s$, we take $Q_s = W$, $\widehat{\mathcal{P}}_s = \widehat{\mathcal{R}}$, and we are done. Otherwise, choose arbitrarily a path $\boldsymbol{t} = (t_0, ..., t_k) \in \mathcal{P}_s - \mathcal{R}$. Without loss of generality, assume that $\boldsymbol{t}$ is rightward. We now have two cases.

(t1) If there is an idle slot τ in W with $t_0 < \tau \leq t_k$, then choose τ to be such a slot that is nearest to t_0. Let a be the largest index for which $t_a < \tau$. Then do this: add $\boldsymbol{t}$ to $\mathcal{R}$, set $\hat{\boldsymbol{t}} = (t_0, ..., t_a, \tau)$, and modify W by performing the shift along $\hat{\boldsymbol{t}}$ (so τ will now become a busy slot in W).

(t2) If no such idle slot exists, it means that there is some path $\boldsymbol{u} \in \mathcal{R}$ whose current truncation $\hat{\boldsymbol{u}}$ ends at $\tau' = t_k$. In this case, we do this: modify W by undoing the shift along $\hat{\boldsymbol{u}}$ (that is, by shifting backwards), remove $\boldsymbol{u}$ from $\mathcal{R}$, add $\boldsymbol{t}$ to $\mathcal{R}$, and modify W by performing the shift along $\boldsymbol{t}$.

Note that any path $\boldsymbol{t}$ may enter and leave $\mathcal{R}$ several times, and each time $\boldsymbol{t}$ is truncated the endpoint τ of $\hat{\boldsymbol{t}}$ gets farther and farther from t_0. It is possible that the process will terminate with $\hat{\boldsymbol{t}} \neq \boldsymbol{t}$. However, if at some step case (t2) applied to $\boldsymbol{t}$, then this truncation step is trivial, in the sense that after the step we have $\hat{\boldsymbol{t}} = \boldsymbol{t}$, and from now on $\boldsymbol{t}$ will never be removed from $\mathcal{R}$. These observations imply that the truncation process always ends.

Lemma 3. *Fix some stage $s \geq 1$. Then* (i) *Q_s is an augmentation of Q_{s-1}.* (ii) $|\mathsf{Supp}(Q_s) - \mathsf{Supp}(Q_{s-1})| = |X_s|$. (iii) *Furthermore, denoting by ξ^0 the number of idle slots of Q_{s-1} in X_s, we can write $|X_s| = \xi^- + \xi^0 + \xi^+$, such that $\mathsf{Supp}(Q_s) - \mathsf{Supp}(Q_{s-1})$ consists of the ξ^0 idle slots in X_s (which become forbidden in Q_s), the ξ^- nearest idle slots of Q_{s-1} to the left of X_s, and the ξ^+ nearest idle slots of Q_{s-1} to the right of X_s (which become busy in Q_s).*

Proof. At the beginning of stage s we have $W = Q_{s-1}$. During the process, we never change a status of a slot from busy or forbidden to idle. Specifically, in steps (t1), for non-trivial paths the first slot t_0 of $\boldsymbol{t}$ was busy and will become forbidden and the last slot τ was idle and will become busy. For trivial paths, $t_0 = t_k$ was idle and will become forbidden. In steps (t2), if $\boldsymbol{t}$ is non-trivial then t_0 was busy and will become forbidden, while t_k was and stays busy. If $\boldsymbol{t}$ is trivial, the status of $t_0 = t_k$ will change from busy to forbidden. In regard to path $\boldsymbol{u}$, observe that $\boldsymbol{u}$ must be non-trivial, since otherwise $\hat{\boldsymbol{u}}$ could not end at t_k. So undoing the shift along $\hat{\boldsymbol{u}}$ will cause u_0 to change from forbidden to busy. This shows that a busy or forbidden slot never becomes idle, so $\mathsf{Supp}(Q_{s-1}) \subseteq \mathsf{Supp}(Q_s)$.

New busy slots are only added in steps (t1), in which case τ is either in X_s, or is a nearest idle slot to X_s, in the sense that all slots between τ and X_s are in the support of W. This implies that Q_s is an augmentation of Q_{s-1}.

To justify (i) and (ii), note that the slots in $X_s - \mathsf{Supp}(Q_{s-1})$ will become forbidden in Q_s and that for each slot $x \in X_s \cap \mathsf{Supp}(Q_{s-1})$ there will be a step of type (t1) in stage s of the truncation process when we will chose a non-trivial path $\boldsymbol{t}$ starting at $t_0 = x$, so in this step a new busy slot will be created. This implies (ii), and together with (i) it also implies (iii). $\square$

Using Lemma 3, we can make the relation between Q_{s-1} and Q_s more specific. Let h be the number of gaps in Q_{s-1} and let $C_0, ..., C_h$ be the blocks of Q_{s-1} ordered from left to right. Thus C_0 and C_h are exterior blocks and all other are interior blocks. Then, for some indices $a \leq b$, the blocks of Q_s are $C_0, ..., C_{a-1}, D, C_{b+1}, ..., C_h$, where the new block D contains X_s as well as all blocks $C_a, ..., C_b$. As a result of adding X_s, in stage s the $b - a$ gaps of Q_{s-1} between C_a and C_b disappear from the reference schedule. For $b = a$, no gap disappears and $C_a \subset D$. In this case adding X_s causes C_a to expand.

Two Types of Regions. We now define two types of forbidden regions, as we mentioned earlier. Consider some forbidden region X_p. If all paths of $\mathcal{P}$ starting at X_p are leftward (resp. rightward) then we say that X_p is *left-oriented* (resp. *right-oriented*). A region X_p that is either left-oriented or right-oriented will be called *oriented*, and if it is neither, it will be called *disoriented*. Recall that trivial paths (consisting only of the start vertex) are considered both leftward and rightward. An oriented region may contain a number of trivial paths, but all non-trivial paths starting in this region must have the same orientation. A disoriented region must contain starting slots of at least one non-trivial leftward path and one non-trivial rightward path.

Charging Disoriented Regions. Let $B_0, ..., B_{g^*}$ be the blocks of Q_0, ordered from left to right. The lemma below establishes some relations between disoriented forbidden regions X_s and the blocks and gaps of Q_0.

Lemma 4. (i) *If B_q is an exterior block then B_q does not intersect any disoriented forbidden regions.* (ii) *If B_q is an interior block then B_q intersects at most one disoriented forbidden region.* (iii) *If X_s is a disoriented forbidden region then X_s intersects at least one block of Q_0.*

Proof. Suppose B_q is the leftmost block, that is $q = 0$, and let $x \in B_0 \cap X_s$. If $\boldsymbol{t} \in \mathcal{P}$ starts at x and is non-trivial then $\boldsymbol{t}$ cannot be leftward, because $\boldsymbol{t}$ ends in an idle slot and there are no idle slots to the left of x. So all paths from $\mathcal{P}$ starting in $B_0 \cap X_s$ are rightward. Thus X_s is right-oriented, proving (i).

Now we prove part (ii). Fix some interior block B_q and, towards contradiction, suppose that there are two disoriented forbidden regions that intersect B_q, say X_s and $X_{s'}$, where X_s is before $X_{s'}$. Then there are two non-trivial transfer paths in $\mathcal{P}$, a rightward path $\boldsymbol{t} = (t_0, ..., t_k)$ starting in $X_s \cap B_q$ and a leftward path $\boldsymbol{u} = (u_0, ..., u_l)$ starting in $X_{s'} \cap B_q$. Both paths must end in idle slots of Q_0 that are not in Z and there are no such slots in $B_q \cup X_s \cup X_{s'}$. Therefore $\boldsymbol{t}$ ends to the right of $X_{s'}$ and $\boldsymbol{u}$ ends to the left of X_s. Thus we have $u_l < t_0 < u_0 < t_k$, which means that paths $\boldsymbol{t}$ and $\boldsymbol{u}$ cross, contradicting Lemma 2.

Part (iii) follows from the definition of disoriented regions, since if X_s were contained in a gap then all transfer paths starting in X_s would be trivial. $\square$

Charging Oriented Regions. This is the most nuanced part of our analysis. We want to show that when an oriented forbidden region is added, at least one gap in the reference schedule disappears. The general idea is that if X_s is left-oriented and G is the nearest gap to the left of X_s, then by the maximality of X_s we have $|X_s| \geq |G|$. So when we process the leftward paths starting in X_s, the truncation process will eventually fill G. As it turns out, this is not actually true as stated, because these paths may end before G and their processing may activate other paths, that might be rightward. Nevertheless, using Lemma 2, we show that either G or the gap H to the right of X_s will be filled.

Lemma 5. *If $X_s = [f_{X_s}, l_{X_s}]$ is an oriented region then at least one gap of Q_{s-1} disappears in Q_s.*

Proof. By symmetry, we can assume that X_s is left-oriented, so all paths in $\mathcal{P}_s - \mathcal{P}_{s-1}$ are leftward. If X_s contains a gap of Q_{s-1}, then this gap will disappear when stage s ends. Note also that X_s cannot be strictly contained in a gap of Q_{s-1}, since otherwise we could increase X_s, contradicting the algorithm. Thus for the rest of the proof we can assume that X_s has a non-empty intersection with exactly one block $B = [f_B, l_B]$ of Q_{s-1}. If B is an exterior block then Lemma 3 immediately implies that the gap adjacent to B will disappear, because X_s is at least as long as this gap. Therefore we can assume that B is an interior block. Denote by G and H, respectively, the gaps immediately to the left and to the right of B. Summarizing, we have $X_s \subset G \cup B \cup H$ and all sets $G - X_s$, $B \cap X_s$, $H - X_s$ are not empty. We will show that at least one of the gaps G, H will disappear in Q_s. The proof is by contradiction; we assume that both G and H have some idle slots after stage s and show that this assumption leads to a contradiction with Lemma 2, which $\mathcal{P}$ was assumed to satisfy.

We first give the proof for the case when $X_s \subseteq B$. From the algorithm, $|X_s| \geq \max(|G|, |H|)$. This inequality, the assumption that G and H do not disappear in Q_s, and Lemma 3 imply together that both gaps shrink; in particular, the rightmost slot of G and the leftmost slot of H become busy in Q_s.

At any step of the truncation process (including previous stages), when some path $\boldsymbol{t} = (t_0, ..., t_k) \in \mathcal{R}$ is truncated to $\hat{\boldsymbol{t}} = (t_0, ..., t_a, \tau)$, all slots between t_0 and τ are either forbidden or busy, so all these slots are in the same block of W. Thus, in stage s, the assumption that G and H do not disappear in Q_s implies that at all steps the paths in $\mathcal{P}_s - \mathcal{R}$ start in B.

Let $\boldsymbol{u}$ be the path whose truncation $\hat{\boldsymbol{u}}$ ends in $f_B - 1$ right after stage s. No transfer path can start in the slots immediately to the left of $f_B - 1$ because they were idle in Q_0. Together with the previous paragraph, this implies that $\boldsymbol{u}$ must be leftward. We can now choose a sequence $\boldsymbol{u}^1, ..., \boldsymbol{u}^p = \boldsymbol{u}$ of transfer paths from $\mathcal{P}_s$ such that $\boldsymbol{u}^1$ is a leftward path starting in X_s (so $\boldsymbol{u}^1$ was in $\mathcal{P}_s - \mathcal{R}$ when stage s started) and, for $i = 1, ..., p-1$, $\boldsymbol{u}^{i+1}$ is the path replaced by $\boldsymbol{u}^i$ in $\mathcal{R}$ at some step of type (t2). Similarly, define $\boldsymbol{v}$ to be the rightward path whose truncation ends in the leftmost slot of H and $\boldsymbol{v}^1, ..., \boldsymbol{v}^q = \boldsymbol{v}$ be the similarly defined sequence for $\boldsymbol{v}$. Our goal is to show that there are paths $\boldsymbol{u}^i$ and $\boldsymbol{v}^j$ that cross, which would give us a contradiction.

The following simple observation follows directly from the definition of the truncation process. Note that it holds even if $\boldsymbol{t}$ is trivial.

Observation 1: Suppose that at some iteration of type (t2) in the truncation process we choose a path $\boldsymbol{t} = (t_0, ..., t_k)$ and it replaces a path $\boldsymbol{t}' = (t'_0, ..., t'_l)$ in $\mathcal{R}$ (because $\hat{\boldsymbol{t}}'$ ended at t_k). Then $\min(t'_0, t'_l) < t_k < \max(t'_0, t'_l)$.

Let $\boldsymbol{u}^g$ be the leftward path among $\boldsymbol{u}^1, ..., \boldsymbol{u}^p$ whose start point u^g_0 is rightmost. Note that $\boldsymbol{u}^g$ exists, because $\boldsymbol{u}^p$ is a candidate for $\boldsymbol{u}^g$. Similarly, let $\boldsymbol{v}^h$ be the rightward path among $\boldsymbol{v}^1, ..., \boldsymbol{v}^q$ whose start point v^h_0 is leftmost.

Claim 1: We have (i) $u^g_0 \geq f_{X_s}$ and (ii) the leftward paths in $\{\boldsymbol{u}^1, ..., \boldsymbol{u}^p\}$ cover the interval $[f_B, u^g_0]$, in the following sense: for each $z \in [f_B, u^g_0]$ there is a leftward path $\boldsymbol{u}^i = (u^i_0, ..., u^i_{k_i})$ such that $u^i_{k_i} \leq z \leq u^i_0$.

Part (i) holds because $\boldsymbol{u}^1$ is leftward and $u_0^1 \geq f_{X_s}$. Property (ii) then follows by applying Observation 1 iteratively to show that the leftward paths among $\boldsymbol{u}^g, ..., \boldsymbol{u}^p$ cover the interval $[f_B, u_0^g]$. More specifically, for $i = g, ..., p-1$, we have that the endpoint $u_{k_i}^i$ of $\boldsymbol{u}^i$ is between u_0^{i+1} and $u_{k_{i+1}}^{i+1}$, the start and endpoints of $\boldsymbol{u}^{i+1}$. As i increases, $u_{k_i}^i$ may move left or right, but it satisfies the invariant that the interval $[u_{k_i}^i, u_0^g]$ is covered by the leftward paths among $\boldsymbol{u}^g, ..., \boldsymbol{u}^i$, and the last value of $u_{k_i}^i$, namely $u_{k_p}^p$, is before f_B. This implies Claim 1.

Claim 2: We have (i) $v_0^h < f_{X_s}$ and (ii) the rightward paths in $\{\boldsymbol{v}^1, ..., \boldsymbol{v}^q\}$ cover the interval $[v_0^h, l_B]$, that is for each $z \in [v_0^h, l_B]$ there is a rightward path $\boldsymbol{v}^j = (v_0^j, ..., v_{l_j}^j)$ such that $v_0^j \leq z \leq v_{l_j}^j$.

The argument is similar to that in Claim 1. We show that if $\boldsymbol{v}^e$ is the first non-trivial rightward path among $\boldsymbol{v}^1, ..., \boldsymbol{v}^q$ then $v_0^e < f_{X_s}$. This $\boldsymbol{v}^e$ exists because $\boldsymbol{v}^q$ is a candidate. The key fact is that $e \neq 1$, because X_s is left-oriented. We have two cases. If $e = 2$ then $\boldsymbol{v}^2$ is a rightward path whose truncation after stage $s-1$ ended in $\tau \in X_s$, and in stage s it was replaced in $\mathcal{R}$ by the trivial path $\boldsymbol{v}^1 = (\tau)$ in a step of type (t2). Then $\boldsymbol{v}_0^2 < f_{X_s}$ and (i) holds. If $e > 2$ then $\boldsymbol{v}^2$ is a non-trivial leftward path, so $v_{l_2}^2 < f_{X_s}$. Then (i) follows from Observation 1, by applying it iteratively to paths $\boldsymbol{v}^2, ..., \boldsymbol{v}^e$, all of which except $\boldsymbol{v}^e$ are leftward.

We now focus on v_0^h. The two claims above imply that $v_0^h < u_0^g$. Since the paths $\boldsymbol{u}^1, ..., \boldsymbol{u}^p$ cover $[f_B, u_0^g]$ and $v_0^h \in [f_B, u_0^g]$, there is a leftward path $\boldsymbol{u}^i$ such that $u_{a+1}^i < v_0^h < u_a^i$, for some index a. Since the rightward paths among $\boldsymbol{v}^1, ..., \boldsymbol{v}^q$ cover the interval $[v_0^h, l_B]$ and $u_a^i \in [v_0^h, l_B]$, there is a rightward path $\boldsymbol{v}^j$ such that $v_b^j < u_a^i < v_{b+1}^j$, for some index b. By these inequalities and our choice of $\boldsymbol{v}^h$, we have $u_{a+1}^i < v_0^h \leq v_0^j \leq v_b^j < u_a^i < v_{b+1}^j$. This means that $\boldsymbol{u}^i$ and $\boldsymbol{v}^j$ cross, giving us a contradiction.

We have thus completed the proof when $X_s \subseteq B$. We now extend it to the general case, when X_s may overlap G or H or both. Recall that both $G - X_s$ and $H - X_s$ are not empty. All we need to do is to show that the idle slots adjacent to $X_s \cup B$ will become busy in Q_s, since then we can choose paths $\boldsymbol{u}$, $\boldsymbol{v}$ and the corresponding sequences as before, and the construction above applies.

Suppose that $X_s \cap G \neq \emptyset$. We claim that the slot $l_{X_s} - 1$, namely the slot of G adjacent to X_s, must become busy in Q_s. Indeed, if this slot remained idle in Q_s then $X_s \cup \{l_{X_s} - 1\}$ would be a viable forbidden region in stage s, contradicting the maximality of X_s. By the same argument, if $X_s \cap H \neq \emptyset$ then the slot of H adjacent to X_s will become busy in Q_s. This immediately takes care of the case when X_s overlaps both G and H.

It remains to examine the case when X_s overlaps only one of G, H. By symmetry, we can assume that $X_s \cap G \neq \emptyset$ but $X_s \cap H = \emptyset$. If $l_B + 1$, the slot of H adjacent to B, is not busy in Q_s, Lemma 3 implies that the nearest $|X_s \cap B|$ idle slots to the left of X_s will become busy. By the choice of X_s we have $|X_s| \geq |G|$, so $|X_s \cap B| \geq |G - X_s|$. Therefore G will disappear in Q_s, contradicting our assumption that it did not. $\square$

Putting everything together now, Lemma 4 implies that the number of disoriented forbidden regions among $X_1, ..., X_m$ is at most $g^* - 1$, the number of interior blocks in Q_0. Lemma 5, in turn, implies that the number of oriented forbidden regions among $X_1, ..., X_m$ is at most g^*, the number of gaps in Q_0. Thus $m \leq 2g^* - 1$. This gives us the main result of this paper.

Theorem 1. *Suppose that the minimum number of gaps in a schedule of $\mathcal{J}$ is $g^* \geq 1$. Then the schedule computed by Algorithm* LVG *has at most $2g^* - 1$ gaps.*

Lower Bound Example. In the full paper we show that for any $k \geq 2$ there is an instance $\mathcal{J}_k$ on which Algorithm LVG finds a schedule with $2k - 1$ gaps, while the optimum schedule has $g^* = k$ gaps. Thus our analysis is tight.

Implementation. In the full paper we show that Algorithm LVG can be implemented in time $O(ng^* \log n)$ and memory $O(n)$, where g^* is the optimum number of gaps. The idea is this: At each step we remove the forbidden regions from the timeline, maintaining the invariant that all release times are different and that all deadlines are different. This can be done in time $O(n \log n)$ per step. With this invariant, the maximum forbidden region has the form $[r_i + 1, d_j - 1]$ for some jobs i, j. We then show how to find such i, j in linear time. Since we have $O(g^*)$ steps, the overall running time will be $O(ng^* \log n)$.

5 Final Comments

Among the remaining open questions, the most intriguing one being whether it is possible to efficiently approximate the optimum solution within a factor of $1 + \epsilon$, for arbitrary $\epsilon > 0$. Ideally, such an algorithm should run in near-linear time. We hope that our results in Section 2, that elucidate the structure of the set of transfer paths, will be helpful in making progress in this direction.

Our 2-approximation result for Algorithm LVG remains valid for scheduling jobs with arbitrary processing times when preemptions are allowed, because then a job with processing time p can be thought of as p identical unit-length jobs. For this case, although Algorithm LVG can be still easily implemented in polynomial time, we do not have an implementation that would significantly improve on the $O(n^5)$ running time from [4].

References

1. Albers, S.: Energy-efficient algorithms. Communications of the ACM 53(5), 86–96 (2010)
2. Albers, S., Antoniadis, A.: Race to idle: new algorithms for speed scaling with a sleep state. In: Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1266–1285 (2012)
3. Baptiste, P.: Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 364–367 (2006)

4. Baptiste, P., Chrobak, M., Dürr, C.: Polynomial time algorithms for minimum energy scheduling. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 136–150. Springer, Heidelberg (2007)
5. Chretienne, P.: On single-machine scheduling without intermediate delays. Discrete Applied Mathematics 156(13), 2543–2550 (2008)
6. Demaine, E.D., Ghodsi, M., Hajiaghayi, M.T., Sayedi-Roshkhar, A.S., Zadimoghaddam, M.: Scheduling to minimize gaps and power consumption. In: Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 46–54 (2007)
7. Irani, S., Pruhs, K.R.: Algorithmic problems in power management. SIGACT News 36(2), 63–76 (2005)

Exponential Complexity of Satisfiability Testing for Linear-Size Boolean Formulas

Evgeny Dantsin and Alexander Wolpert

Department of Computer Science, Roosevelt University
430 S. Michigan Av., Chicago, IL 60605, USA
{edantsin,awolpert}@roosevelt.edu

Abstract. The *exponential complexity* of the satisfiability problem for a given class of Boolean circuits is defined to be the infimum of constants α such that the problem can be solved in time $poly(m)\,2^{\alpha n}$, where m is the circuit size and n is the number of input variables [IP01]. We consider satisfiability of linear Boolean formula over the full binary basis and we show that the corresponding exponential complexities are "interwoven" with those of k-CNF SAT in the following sense. For any constant c, let f_c be the exponential complexity of the satisfiability problem for Boolean formulas of size at most cn. Similarly, let s_k be the exponential complexity of k-CNF SAT. We prove that for any c, there exists a k such that $f_c \leq s_k$. Since the Sparsification Lemma [IPZ01] implies that for any k, there exists a c such that $s_k \leq f_c$, we have $\sup_c\{f_c\} = \sup_k\{s_k\}$. (In fact, we prove this equality for a larger class of linear-size circuits that includes Boolean formulas.) Our work is partly motivated by two recent results. The first one is about a similar "interweaving" between linear-size circuits of constant depth and k-CNFs [SS12]. The second one is that satisfiability of linear-size Boolean formulas can be tested exponentially faster than in $O(2^n)$ time [San10, ST12].

1 Introduction

Assuming $\mathbf{P} \neq \mathbf{NP}$, it is still unknown how to classify $\mathbf{NP}$-complete problems by their complexity. For example, is it possible to test satisfiability of 3-CNFs in subexponential time? The conjecture known as the Exponential Time Hypothesis (ETH) states that it is not possible [IP01]. Or, is it possible to test satisfiability of Boolean circuits exponentially faster than using the trivial enumeration of all assignments? Questions like these seem far away from being resolved, even though this line of research has produced useful insights, see surveys in [DH09, PP10].

Exponential Complexity. A natural approach to the complexity classification of problems in $\mathbf{NP}$ is to use the notion of exponential complexity [IP01, CP09]. In this paper, we restrict ourselves to Boolean satisfiability problems. Let $\mathcal{C}$ be a class of circuits and let $\mathcal{C}$ SAT be the satisfiability problem for circuits of $\mathcal{C}$. The *exponential complexity* of $\mathcal{C}$ SAT is the infimum of constants α such that there is an algorithm that solves this problem in time $poly(m)\,2^{\alpha n}$, where

m is the circuit size and n is the number of input variables, see Section 2 for details. Many deep and interesting results on exponential complexity and related questions are relevant to this paper, but due to the space limit, we mention here only few of them, namely known results on the exponential complexity for k-CNFs, linear-size circuits of constant depth, and linear-size Boolean formulas. To describe these results, we use the following notation (throughout the paper, n denotes the number of variables in a circuit):

– s_k is the exponential complexity of the satisfiability problem for k-CNFs; $s_\infty = \sup_k\{s_k\}$;
– r_c^d is the exponential complexity of the satisfiability problem for circuits of depth at most d and size at most cn; $r_\infty^d = \sup_c\{r_c^d\}$;
– f_c is the exponential complexity of the satisfiability problem for Boolean formulas of size at most cn over the full binary basis; $f_\infty = \sup_c\{f_c\}$.

In these terms, ETH is the conjecture that $s_3 > 0$. Impagliazzo and Paturi [IP01] proved that if ETH is true then the sequence $\{s_k\}$ increases infinitely often. Using the sparsification technique, it was shown that s_k remains the same if the class of k-CNFs is restricted to k-CNFs of linear size [IPZ01]. Known upper bounds on s_k have the form $1 - c/k$, where c is a constant [DH09], and it is a challenging open question whether $s_\infty = 1$ or $s_\infty < 1$. The Strong Exponential Time Hypothesis (SETH) states that $s_\infty = 1$ [IP01, CIP09].

An upper bound on r_c^d was obtained in [CIP09]: this bound is strictly less than 1. How do $\{s_k\}$ and $\{r_c^d\}$ relate? Santhanam and Srinivasan [SS12] answered this question by showing that $\{r_c^d\}$ is "interwoven" with $\{s_k\}$: for any numbers c and d, there is an integer k such that $r_c^d \leq s_k$ and, similarly, in the converse direction. Therefore, $r_\infty^d = s_\infty$ for any d.

How important is a constant limit on the circuit depth in such "interweaving"? For example, how about Boolean formulas, a natural class of circuits of non-constant depth? Santhanam [San10], Seto and Tamaki [ST12] showed that satisfiability of linear-size Boolean formulas over the full binary basis can be tested exponentially faster than in $O(2^n)$ time: $f_c < 1$ for any constant c. This result suggests that we could expect f_c to be "interwoven" with $\{s_k\}$. In this paper, we prove this conjecture.

Our Results. It follows from the Sparsification Lemma [IPZ01] that for any positive integer k, there is a number c such that $s_k \leq f_c$. We prove the "converse": for any number c, there is an integer k such that $f_c \leq s_k$. Therefore, $f_\infty = s_\infty$. In fact, our main result is stronger in the following two aspects.

First, instead of $\{f_c\}$, we consider an analogous sequence of the exponential complexities for linear-size circuits of a more general type than Boolean formulas, see Section 4 for the definition of such circuits. Loosely speaking, a circuit of this type has two properties: (1) all gates have bounded fan-in and (2) each subcircuit has a bounded number of "border gates", i.e., gates that have wires to the "residual part" of the circuit. Any Boolean formula is a circuit of this type. Another special case is the class of circuits whose underlying graphs have bounded minimum vertex separation numbers [BFT09].

Second, we relate circuit satisfiability to k-CNF satisfiability using reductions that preserve satisfying assignments. More exactly, for any $\epsilon > 0$, a circuit ϕ with n variables is transformed in polynomial time into a k-CNF F such that

– F has the same variables as ϕ plus at most ϵn additional variables;
– if an assignment satisfies F then its restriction to the variables of ϕ satisfies ϕ; if an assignment satisfies ϕ, then it has a unique extension to the variables of F that satisfies F.

Thus, taking all assignments to the additional variables, ϕ can be transformed into an equivalent disjunction of subexponentially many k-CNFs. That is, any Boolean function computed by a Boolean formula can be computed by a disjunction of subexponentially many k-CNFs.

Note that the equality $f_\infty = s_\infty$ gives an equivalent statement for SETH: $f_\infty = 1$, see [CDL$^+$12] for some other equivalent statements.

Organization of the Paper. The basic definitions and notation are given in Section 2. Section 3 describes how we reduce circuits to k-CNFs. The reducibility uses the extension rule (well known in proof complexity). Section 4 is about graphs underlying circuits for which we prove an "interweaving" with k-CNFs. In Section 5, we state and prove the main results.

2 Basic Definitions and Notation

Circuit Satisfiability. By a *circuit* we mean a single-output Boolean circuit in its most general version [Vol99] where, in particular, every input is labeled with either a variable or a truth value, but such a labeling is not necessarily one-to-one: distinct inputs may be labeled with the same variable. When talking about circuits, it will sometimes be convenient for us to use graph-theoretic terms instead of terms standard for circuits: for example, vertices and edges instead of nodes and wires, in-degree and out-degree instead of fan-in and fan-out, etc.

The number of nodes in a circuit ϕ is denoted by $|\phi|$. The set of variables labeling the inputs of ϕ is denoted by $var(\phi)$. Let A be an assignment of truth values to the variables of ϕ, i.e., a mapping from $var(\phi)$ to $\{0, 1\}$. The value of ϕ on A is defined in the standard way [Vol99] and is denoted by $\phi(A)$. A circuit ϕ is called *satisfiable* if there is an assignment A such that $\phi(A) = 1$.

Let $\mathcal{C}$ be a set of circuits. The *satisfiability problem* for $\mathcal{C}$ is the problem of determining whether a circuit from $\mathcal{C}$ is satisfiable or not. We write $\mathcal{C}$ SAT to denote the language consisting of all satisfiable circuits from $\mathcal{C}$.

Exponential Complexity. Let $\mathcal{C}$ be a class of circuits. Following Impagliazzo and Paturi [IP01] (see also [CP09] for details), we define the *exponential complexity* of $\mathcal{C}$ SAT to be the infimum of constants α such that $\mathcal{C}$ SAT can be decided by a two-sided error randomized algorithm in time $poly(|\phi|)\, 2^{\alpha n}$ with probability of error $< \frac{1}{3}$:

$$exp\text{-}com(\mathcal{C}\ \mathsf{SAT}) = \inf\{\alpha \mid \mathcal{C}\ \mathsf{SAT} \in \mathbf{BPTIME}(poly(|\phi|)\, 2^{\alpha n})\}.$$

Note that the polynomial $poly(|\phi|)$ in this definition may depend on α.

The fact that the definition above uses randomized algorithms, not deterministic ones, is not so important for this paper. All of our results remain valid if the exponential complexity of $\mathcal{C}$ SAT is defined as a similar measure where randomized algorithms are replaced by deterministic algorithms:

$$\mathit{exp\text{-}com}(\mathcal{C}\ \mathsf{SAT}) = \inf\{\alpha \mid \mathcal{C}\ \mathsf{SAT} \in \mathbf{DTIME}(poly(|\phi|)\, 2^{\alpha n})\}.$$

Boolean Formulas and CNFs. A *Boolean formula* is a circuit in which every gate has fan-in at most 2 and fan-out at most 1. Gates with fan-in 2 may be labeled with arbitrary binary Boolean functions. A *literal* is either a single-gate circuit or a two-gate circuit (where the output is labeled with the negation). A *clause* is either a literal or a circuit obtained from literals by adding wires from their outputs to a new output gate labeled with the disjunction of the corresponding arity. A *conjunctive normal form* (a *CNF* for short) is either a clause or a circuit obtained from clauses by adding wires from their outputs to a new output gate labeled with the conjunction of the corresponding arity. A CNF is called a *k-CNF* if every disjunction in its labeling has arity at most k.

3 Extension Rule for Circuits

In this section, we define circuit transformations based on the *extension rule*. This rule was introduced by Tseitin [Tse68] who used it to attain an exponential speed-up for the length of resolution proofs in propositional logic. A more general form of the rule is well known in proof complexity in connection with extended Frege systems, where the rule is used to abbreviate long formulas, see e.g. [Pud98]. In this general form, the extension rule allows using formulas of the form $z \leftrightarrow F$ in a proof, where F is any formula and z is a new propositional variable that appears neither in the previous part of the proof nor in the formula to be proved.

3.1 Induced Circuits

Let ϕ be a circuit and $G = (V, E)$ be its underlying directed graph. Each vertex $u \in V$ determines the directed graph $G_u = (V_u, E_u)$ where

$$V_u = \{u\} \cup \{w \in V \mid \text{there is a path from } w \text{ to } u \text{ in } G\};$$
$$E_u = \{(v, w) \in E \mid \text{both } v \text{ and } w \text{ are in } V_u\}.$$

This graph G_u is called the *subgraph induced by* u. A vertex $v \in V_u$ is said to be a *border vertex* of G_u if v has an outgoing edge incoming to a vertex outside G_u, i.e., there is an edge $(v, w) \in E$ where $w \in V - V_u$. The set of all border vertices of G_u is called the *border* of G_u and is denoted by $\beta(G_u)$. All other vertices in G_u are called *internal*.

The above terminology and notation are extended to circuits in a natural way: given ϕ and u, the *subcircuit induced by* u is the circuit whose underlying graph is G_u and whose labeling is the same as in ϕ, i.e., for every vertex in V_u, its label in ϕ_u is the same as its label in ϕ. Note that the labeling for ϕ_u is defined correctly since for every vertex in V_u, its in-degree in G_u is the same as its in-degree in G (but its out-degrees in G_u and G may be different). The border of G_u is also referred as the *border* of ϕ_u and it is denoted by $\beta(\phi_u)$.

We want to “decompose” ϕ into two circuits: ϕ_u and a “residual” circuit obtained from ϕ by “contraction” of ϕ_u into a single vertex. This “residual” circuit and its underlying graph are denoted by $\phi \ominus \phi_u$ and $G \ominus G_u$ respectively. They are defined as follows:

- $G \ominus G_u$ is obtained from $G = (V, E)$ by removing all internal vertices of $G_u = (V_u, E_u)$ and removing all edges incident on these internal vertices. Thus, every vertex in $G \ominus G_u$ is either a vertex from $V - V_u$ or a border vertex of G_u.
- If a vertex belongs to $V - V_u$, its label in $\phi \ominus \phi_u$ is the same as in ϕ.
- If a vertex belongs to the border of G_u, it has in-degree 0 in $G \ominus G_u$. To label such border vertices, we use new variables, not occurring in ϕ. Namely, let $\beta(G_u) = \{v_1, \ldots, v_b\}$. We introduce b new variables $z_1, \ldots, z_b$ and we label each vertex v_i with z_i.

In this labeling of $v_1, \ldots, v_b$, each variable z_i “replaces” the subcircuit ϕ_{v_i} induced by v_i in ϕ. To emphasize this fact, we denote the circuit $\phi \ominus \phi_u$ using the standard notation for substitutions in formulas:

$$\phi[z_1/\phi_{v_1}, \ldots, z_b/\phi_{v_b}]. \tag{1}$$

It will be convenient for us to use either of the two notations: $\phi \ominus \phi_u$, which specifies the circuit up to names of new variables, or $\phi[z_1/\phi_{v_1}, \ldots, z_b/\phi_{v_b}]$, which specifies it completely.

3.2 Circuit Transformations

The extension rule is typically used to transform a formula F into an “equivalent” (in a special sense) formula $(z \leftrightarrow S) \wedge F[z/S]$ where S is a subformula of F. Here, we generalize this operation for circuits.

We begin with notation for composition of circuits, namely for circuits made up from other circuits using the Boolean functions $\leftrightarrow$ (equivalence) and $\wedge^m$ (m-ary conjunction). Given a circuit ϕ and a variable z not occurring in ϕ, the circuit denoted by $(z \leftrightarrow \phi)$ is obtained from ϕ by adding two new vertices and two new edges: a vertex v labeled with z, a vertex w labeled with the Boolean function $\leftrightarrow$, an edge from v to w, and an edge from the single output of ϕ to w. Thus, w is the output of the resulting circuit $z \leftrightarrow \phi$. Similarly, given circuits $\phi_1, \ldots, \phi_m$, the circuit denoted by $\phi_1 \wedge \ldots \wedge \phi_m$ is obtained by adding one new vertex and m new edges. The new vertex v is labeled with $\wedge^m$. The m new edges go from the outputs of $\phi_1, \ldots, \phi_m$ to v.

Let ϕ be a circuit and u be a vertex in ϕ. Consider the subcircuit ϕ_u induced by u, its underlying graph G_u, and the border of G_u. Let $\beta(G_u) = \{v_1, \ldots, v_b\}$

and let $z_1, \ldots, z_b$ be new variables not occurring in ϕ. We transform ϕ into the circuit

$$(z_1 \leftrightarrow \phi_{v_1}) \ \wedge \ \ldots \ \wedge \ (z_b \leftrightarrow \phi_{v_b}) \ \wedge \ \phi[z_1/\phi_{v_1}, \ldots, z_b/\phi_{v_b}] \tag{2}$$

and we write $\phi \overset{u}{\mapsto} \psi \wedge \phi'$ to denote this transformation, where ψ denotes the conjunction of the equivalences and ϕ' denotes the last conjunctive term in (2). The following simple lemma expresses the fact that such transformations preserve satisfiability.

Lemma 1. *Suppose that $\phi \overset{u}{\mapsto} \psi \wedge \phi'$. Then ϕ is satisfiable if and only if $\psi \wedge \phi'$ is satisfiable. Moreover,*

- *if an assignment satisfies $\psi \wedge \phi'$, then its restriction to $var(\phi)$ satisfies ϕ;*
- *if an assignment satisfies ϕ, then it has a unique extension to $var(\psi \wedge \phi')$ that satisfies $\psi \wedge \phi'$.*

Proof. It easily follows from the definition of $\phi[z_1/\phi_{v_1}, \ldots, z_b/\phi_{v_b}]$ that any satisfying assignment for (2) restricted to $var(\phi)$ satisfies ϕ. Conversely, any satisfying assignment A for ϕ can be extended to a satisfying assignment for circuit (2) by assigning values $\phi_{v_1}(A), \ldots, \phi_{v_b}(A)$ to the variables $z_1, \ldots, z_b$. Any other extension of A falsifies some of the equivalences in (2). $\square$

3.3 Transformation Sequences

Our purpose is to transform a circuit into a conjunction of "small" circuits. A natural strategy is to apply successive transformations $\phi \overset{u}{\mapsto} \psi \wedge \phi'$ where u is chosen so that ϕ_u is a "small" subcircuit. That is, choose a vertex u_1 in ϕ such that ϕ_{u_1} is "small", then choose a vertex u_2 that induces a "small" subcircuit in $\phi \ominus \phi_{u_1}$, and so on. Below, we describe this approach in more precise terms.

Consider a circuit ϕ, a vertex u in ϕ, and the induced subcircuit ϕ_u. We call ϕ_u a (b,s)-*subcircuit* if $|\beta(\phi_u)| \leq b$ and $|\phi_u| \leq s$. A transformation $\phi \overset{u}{\mapsto} \psi \wedge \phi'$ is called a (b,s)-*transformation* if ϕ_u is a (b,s)-subcircuit.

Let $u_1, \ldots, u_l$ be a sequence of vertices in ϕ. We call it a (b,s)-*transformation sequence for* ϕ if there exist sequences $\{\phi_i\}_{i=0}^{l}$ and $\{\psi_i\}_{i=1}^{l}$ of circuits such that

- ϕ_0 is the circuit ϕ;
- for $i = 1, \ldots, l$,
  - u_i is a vertex in ϕ_{i-1};
  - there is (b,s)-transformation $\phi_{i-1} \overset{u_i}{\mapsto} \psi_i \wedge \phi_i$;
- ϕ_l has at most s vertices.

Lemma 2. *If a circuit ϕ has a (b,s)-transformation sequence σ of length l, then there is a circuit χ such that*

- *χ is a conjunction $\chi_1 \wedge \ldots \wedge \chi_t$, where $t \leq bl + 1$;*
- *each circuit χ_i has at most $s + 2$ vertices;*
- *$var(\phi) \subseteq var(\chi)$ and $|var(\chi)| \leq |var(\phi)| + bl$;*

– *ϕ is satisfiable if and only if χ is satisfiable.*

There is a polynomial-time algorithm that takes as input ϕ, σ and outputs a circuit χ that has the above properties.

Proof. The required algorithm takes ϕ, σ as input and constructs the sequence

$$\phi_0 \overset{u_1}{\mapsto} \psi_1 \wedge \phi_1, \quad \phi_1 \overset{u_2}{\mapsto} \psi_2 \wedge \phi_2, \quad \ldots, \quad \phi_{l-1} \overset{u_l}{\mapsto} \psi_l \wedge \phi_l$$

of (b, s)-transformations, where each ψ_i is a conjunction of b_i equivalences of the form $(z \leftrightarrow \phi_v)$. The number of such equivalences in the conjunction is equal to the number of vertices in the border of ϕ_i in ϕ_{i-1}. We denote this number by b_i and we write t to denote $\sum_{i=1}^{l} b_i + 1$. Next, the algorithm constructs the resulting circuit χ as the t-ary conjunction of circuits $\chi_1, \ldots, \chi_t$ where the first $t-1$ circuits $\chi_1, \ldots, \chi_{t-1}$ are equivalences of the form $(z \leftrightarrow \phi_v)$ and the last circuit χ_t is ϕ_l. Clearly, the algorithm constructs χ in polynomial time. We show that χ has the claimed properties.

By the definition of (b, s)-transformations, $b_i \le b$ for $i = 1, \ldots, l$. Hence, we have $t \le bl + 1$. Also, by the same definition, each circuit ϕ_v in an equivalence $(z \leftrightarrow \phi_v)$ has at most s vertices. Therefore, the equivalence itself has at most $s+2$ vertices. Since the last circuit χ_t has at most s vertices, each circuit χ_i has at most $s+2$ vertices. The number of new variables in χ is equal to the number of the equivalences, $\sum_{i=1}^{l} b_i$, which is at most bl. The remaining property (satisfiability preservation) is easily proved using Lemma 1 and induction on l. □

Corollary 1. *If a circuit ϕ has a (b, s)-transformation sequence σ of length l, then there is a k-CNF F such that*

– *$k \le s + 2$;*
– *the number of clauses is at most $(bl + 1)2^k$;*
– *$var(\phi) \subseteq var(\chi)$ and $|var(\chi)| \le |var(\phi)| + bl$;*
– *ϕ is satisfiable if and only if F is satisfiable.*

There is a polynomial-time algorithm that takes as input ϕ, σ and outputs a k-CNF F that has the above properties.

Proof. The circuit $\chi_1 \wedge \ldots \wedge \chi_t$ from Lemma 2 is transformed into a k-CNF F with the claimed properties as follows. Each circuit χ_i has at most $s + 2$ inputs and, therefore, it represents a Boolean function of at most $s + 2$ variables. Any such function can be computed by a k-CNF F_i where $k \le s + 2$ and the number of clauses is not greater than 2^k. The k-CNF F is the conjunction $F_1 \wedge \ldots \wedge F_t$. □

Remark 1. According to the property of preserving satisfiability in Corollary 1, ϕ is satisfiable if and only if F is satisfiable. This equivalence is sufficient for the use of the corollary in Section 5. However, it is not difficult to see that a stronger form of this equivalence holds: each satisfying assignment for ϕ has a unique extension to $var(F)$ that satisfies F, and for each satisfying assignment for F, its restriction to $var(\phi)$ satisfies ϕ (cf. Lemma 1). The same applies to ϕ and χ in Lemma 2.

4 Graphs with (b, s)-Transformation Sequences

The notion of a (b, s)-transformation sequence is defined in terms of circuits (Section 3.3), but it is easy to see that, in fact, such sequences are determined by underlying graphs, independently of their labeling. Here is an equivalent definition in terms of graphs. Let ϕ be a circuit and $G = (V, E)$ be its underlying graph. Let σ be a sequence of vertices $u_1, \ldots, u_l$ in V. This sequence is a (b, s)-transformation sequence for ϕ if and only if there exist sequences $\{G_i\}_{i=0}^{l}$ and $\{H_i\}_{i=1}^{l}$ of graphs such that

- G_0 is the graph G;
- for $i = 1, \ldots, l$,
  - u_i is a vertex in G_{i-1} and H_i is the subgraph of G_{i-1} induced by u_i;
  - H_i has at most s vertices and the border of H_i in G_{i-1} consists of at most b vertices;
  - G_i is $G_{i-1} \ominus H_i$;
- G_l has at most s vertices.

Since σ is a (b, s)-transformation sequence for any circuit whose underlying graph is G, we refer to σ as a (b, s)-transformation sequence for G. In this section, we describe a class of graphs for which (b, s)-transformation sequences can be found in polynomial time.

The maximum in-degree in a graph G is denoted by *max-in*(G). The following lemma relates the maximum in-degree of graphs to sizes of induced subgraphs. Given a graph, does it have an induced subgraph with an "arbitrary" (in a reasonable sense) number of vertices?

Lemma 3. *Let $G = (V, E)$ be a graph with max-in$(G) \leq b$. For any integer t in the interval $b < t \leq |V|$, there is a vertex $u \in V$ such that the number of vertices of the induced subgraph $G_u = (V_u, E_u)$ is between t and $b(t-1)+1$:*

$$t \;\leq\; |V_u| \;\leq\; b(t-1)+1. \tag{3}$$

Proof. Induction on the depth of G. The basis step (the depth is 0, i.e., G is a single-vertex graph) is trivial. In the inductive step (the depth is positive), we select a vertex v that has two properties:

- v has non-zero in-degree;
- the subgraph G_v induced by v has at least t vertices.

Such a vertex exists, for example the sink of G has the above properties. Let d be the in-degree of v and let $v_1, \ldots, v_d$ be all in-neighbors of v. Consider the subgraphs $G_{v_1}, \ldots, G_{v_d}$ induced by $v_1, \ldots, v_d$ respectively and select an index k such that G_{v_k} has the maximum number of vertices among all these subgraphs. Let m be this maximum, the number of vertices in G_{v_k}.

There are only two options: either $t \leq m$ or $t > m$. If $t \leq m$, then the subgraph G_{v_k} has a required vertex u. This follows from the inductive assumption (the depth of G_{v_k} is less than the depth of G) and the fact that t does not exceed the number of vertices in G_{v_k}. If $t > m$, then the vertex v can be taken as the vertex u required in the claim. Indeed, we have

$$t \ \le \ \text{the number of vertices of } G_v \ \le \ dm + 1 \ \le \ d(t-1) + 1 \ \le \ b(t-1) + 1$$

and, thus, inequality (3) holds. □

For a graph $G = (V, E)$, the *maximum border size* of G is defined to be the maximum of $|\beta(G_u)|$ over all $u \in V$, i.e., the maximum border size of all induced subgraphs of G. We denote it by *max-border*(G).

Lemma 4. *For any graph $G = (V, E)$ with at least two vertices and for any numbers b and s such that*

$$\textit{max-in}(G) \le b, \quad \textit{max-border}(G) \le b, \quad 1 + b^2 < s \le |V| \tag{4}$$

there exists a (b, s)-transformation sequence σ for G whose length is at most

$$\frac{b\,|V|}{s - b^2 - 1}. \tag{5}$$

There is a polynomial-time algorithm that takes as input G and numbers b, s satisfying all inequalities (4), and it outputs a (b, s)-transformation sequence σ for G with upper bound (5) on its length.

Proof. To construct a sequence σ and the corresponding sequences $\{G_i\}_{i=0}^{l}$ and $\{H_i\}_{i=1}^{l}$ of graphs (defined in the beginning of this section), we make l steps. At step i, we transform G_{i-1} into G_i by "cutting off" an induced subgraph H_i from G_{i-1}. A key point is that Lemma 3 can be used to find H_i such that the number of vertices of H_i is bounded from below and from above: this number lies between some t and s, where s given in the input and a value for t will be chosen later. Thus, on one hand, each subgraph H_i has at most s vertices, which is required for σ. On the other hand, when "cutting off" H_i, the number of vertices of G_{i-1} reduces by at least $t - b$. Hence, the total number of steps is at most $\lceil |V|/(t-b) \rceil$. It is clear that given t, this construction of σ takes polynomial time.

It remains to choose a value for t. By Lemma 3, for any t such that $b < t \le |V|$, there is an induced subgraph with the number of vertices between t and $b(t-1)+1$. Therefore, an integer t must be chosen so as to satisfy

$$1 \le b < t \quad \text{and} \quad b(t-1) + 1 \le s.$$

We choose $t = \lceil (s-1)/b \rceil$, which guarantees that the second inequality above is satisfied for any $b \ge 1$ and $s \ge 1$. Then the first constraint holds if $b < (s-1)/b$. Thus, σ can be constructed for any b and s satisfying (4).

The length of σ is the number l of "cutting off" steps. Bound (5) on l follows from the fact that the number of steps does not exceed $|V|/(t-b)$. □

5 Exponential Complexity

In this section, we compare the exponential complexity of the satisfiability problems for certain classes of circuits. We begin with a definition of a suitable reducibility.

Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be classes of circuits. Let $\mathcal{C}_1$ SAT and $\mathcal{C}_2$ SAT be the languages consisting of satisfiable circuits from $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively. We say that $\mathcal{C}_1$ SAT is polynomial-time reducible to $\mathcal{C}_2$ SAT with *an arbitrarily small increase in the number of variables* if for every $\epsilon > 0$, there is a polynomial-time Karp reduction from $\mathcal{C}_1$ SAT to $\mathcal{C}_2$ SAT with the following additional property: for any circuit ϕ with n variables, R_ϵ maps ϕ into a circuit with at most $n + \epsilon n$ variables:

$$|var(R_\epsilon(\phi))| \ \leq \ n + \epsilon n.$$

Lemma 5. *If $\mathcal{C}_1$ SAT is polynomial-time reducible to $\mathcal{C}_2$ SAT with an arbitrarily small increase in the number of variables, then*

$$\textit{exp-com}(\mathcal{C}_1\ \mathsf{SAT}) \ \leq \ \textit{exp-com}(\mathcal{C}_2\ \mathsf{SAT}).$$

Proof. Suppose that $\mathcal{C}_1$ SAT is polynomial-time reducible to $\mathcal{C}_2$ SAT with an arbitrarily small increase in the number of variables. Also, suppose that there is an algorithm that solves $\mathcal{C}_1$ SAT in time $poly(|\phi|)\,2^{\alpha n}$. Then the composition of this algorithm and the reduction gives an algorithm that solves $\mathcal{C}_2$ SAT in time $poly(|\phi|)\,2^{\alpha(1+\epsilon)n}$. Taking $\epsilon \to 0$, we obtain the claim. $\square$

We consider the satisfiability problems for the following classes of circuits:

– **k-CNFs.** For any $k \in \mathbb{N}$, the set of all k-CNFs is denoted by k-CNF. To denote the exponential complexity of k-CNF SAT, we use the same notation as in [IP01]:

$$s_k = \textit{exp-com}(k\text{-}\mathsf{CNF\ SAT}) \quad \text{and} \quad s_\infty = \sup_k\{s_k\}.$$

– **Linear-Size Boolean Formulas.** For any number $c > 0$, let $\mathsf{FORMULA}_c$ denote the set of all Boolean formulas ϕ such that the number of vertices of ϕ is at most cn where $n = |var(\phi)|$. Recall that we consider Boolean formulas over the full basis (Section 2). The exponential complexity of $\mathsf{FORMULA}_c$ SAT is denoted using the following notation:

$$f_c = \textit{exp-com}(\mathsf{FORMULA}_c\ \mathsf{SAT}) \quad \text{and} \quad f_\infty = \sup_c\{f_c\}.$$

– **Linear-Size Circuits with Bounded fan-in and Bounded Border Size.** For any numbers $b \geq 1$ and $c > 0$, let $\mathsf{CIRCUIT}_{b,c}$ denote the set of all circuits ϕ such that
  - any node in ϕ has fan-in at most b;
  - the maximum border size of the graph underlying ϕ is at most b;
  - $|\phi| \leq cn$ where $n = |var(\phi)|$.

The exponential complexity of $\mathsf{CIRCUIT}_{b,c}$ SAT is denoted using the following notation:

$$r_{b,c} = \mathit{exp\text{-}com}(\mathsf{CIRCUIT}_{b,c}\ \mathsf{SAT}) \quad \text{and} \quad r_{b,\infty} = \sup_c\{r_{b,c}\}.$$

Theorem 1. *For any $b \geq 1, c > 0$, there is an integer k such that $r_{b,c} \leq s_k$.*

Proof. Let $b \geq 1, c > 0, \epsilon > 0$ be fixed. We show that there is a polynomial-time algorithm that takes as input a circuit $\phi \in \mathsf{CIRCUIT}_{b,c}$ with n variables and outputs a k-CNF F such that

– $|var(F)| \leq n + \epsilon n$ for sufficiently large n;
– ϕ is satisfiable if and only if F is satisfiable.

By Lemma 5, the existence of such an algorithm implies $r_{b,c} \leq s_k$.

Take $s = \lceil 1 + b^2 + \frac{b^2 c}{\epsilon}\rceil$. By Lemma 4, if $n \geq s$, it takes polynomial time to construct a (b, s)-transformation sequence of length l for the graph underlying ϕ such that

$$l \leq \frac{b|\phi|}{s - b^2 - 1} \leq \frac{bcn}{s - b^2 - 1} \leq \frac{\epsilon n}{b}. \tag{6}$$

Next, by Corollary 1, it takes polynomial time to transform ϕ into a k-CNF F with $k \leq s + 2$ and $|var(F)| \leq n + bl$ such that ϕ is satisfiable if and only if F is satisfiable. Using inequality (6), we have $|var(F)| \leq n + \epsilon n$. □

Theorem 2. *For any integer $k \geq 1$, there is a number c such that $s_k \leq f_c$.*

Proof. It follows from the Sparsification Lemma [IPZ01] that satisfiability of k-CNFs has the same exponential complexity as satisfiability of linear-size k-CNFs. Since there is a trivial polynomial-time transformation of a k-CNF F with n variables and with $|F| \leq cn$ into an equivalent Boolean formula ϕ with the same variables and with $|\phi| \leq c'n$, the claim holds. □

Theorem 3. *For any $b \geq 2$ and $c > 0$, we have $f_c \leq r_{b,c}$.*

Proof. For any Boolean formula ϕ, the maximum fan-in is at most 2 and the maximum border size of the underlying graph is at most 1. Therefore, any Boolean formula ϕ with $|\phi| \leq cn$ is a circuit in $\mathsf{CIRCUIT}_{2,c}$. □

Theorem 4. *For any $b \geq 2$, we have $s_\infty = f_\infty = r_{b,\infty}$.*

Proof. Theorem 1 implies $r_{b,\infty} \leq s_\infty$, Theorem 2 implies $s_\infty \leq f_\infty$, and Theorem 3 implies $f_\infty \leq r_{b,\infty}$. □

References

[BFT09] Bodlaender, H.L., Fellows, M.R., Thilikos, D.M.: Derivation of algorithms for cutwidth and related graph layout parameters. Journal of Computer and System Sciences 75(4), 231–244 (2009)

[CDL$^+$12] Cygan, M., Dell, H., Lokshtanov, D., Marx, D., Nederlof, J., Okamoto, Y., Paturi, R., Saurabh, S., Wahlström, M.: On problems as hard as CNF-Sat. In: Proceedings of the 27th Annual IEEE Conference on Computational Complexity, CCC 2012, pp. 74–84. IEEE Computer Society (2012)

[CIP09] Calabro, C., Impagliazzo, R., Paturi, R.: The complexity of satisfiability of small depth circuits. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 75–85. Springer, Heidelberg (2009)

[CP09] Calabro, C., Paturi, R.: k-SAT is no harder than Decision-Unique-k-SAT. In: Frid, A., Morozov, A., Rybalchenko, A., Wagner, K.W. (eds.) CSR 2009. LNCS, vol. 5675, pp. 59–70. Springer, Heidelberg (2009)

[DH09] Dantsin, E., Hirsch, E.A.: Worst-case upper bounds. In: Handbook of Satisfiability, ch.12, pp. 403–424. IOS Press (2009)

[IP01] Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. Journal of Computer and System Sciences 62(2), 367–375 (2001)

[IPZ01] Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity. Journal of Computer and System Sciences 63(4), 512–530 (2001)

[PP10] Paturi, R., Pudlák, P.: On the complexity of circuit satisfiability. In: Proceedings of the 42nd Annual ACM Symposium on Theory of Computing, STOC 2010, pp. 241–250. ACM (2010)

[Pud98] Pudlák, P.: The length of proofs. In: Buss, S.R. (ed.) Handbook of Proof Theory, pp. 547–637. Elsevier (1998)

[San10] Santhanam, R.: Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In: Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, pp. 183–192 (2010)

[SS12] Santhanam, R., Srinivasan, S.: On the Limits of Sparsification. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part I. LNCS, vol. 7391, pp. 774–785. Springer, Heidelberg (2012)

[ST12] Seto, K., Tamaki, S.: A satisfiability algorithm and average-case hardness for formulas over the full binary basis. In: Proceedings of the 27th Annual IEEE Conference on Computational Complexity, CCC 2012, pp. 107–116. IEEE Computer Society (2012)

[Tse68] Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Slisenko, A.O. (ed.) Studies in Constructive Mathematics and Mathematical Logic, Part II, pp. 115–125 (1968) (in Russian); Reprinted in: Siekmann, J., Wrightson, G. (eds.): Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970, pp. 466–483. Springer (1983)

[Vol99] Vollmer, H.: Introduction to Circuit Complexity: A Uniform Approach. Springer (1999)

Linear Time Distributed Swap Edge Algorithms[⋆]

Ajoy K. Datta[1], Lawrence L. Larmore[1], Linda Pagli[2], and Giuseppe Prencipe[2]

[1] Dept. of Comp. Sc., Univ. of Nevada
{ajoy.datta,lawrence.larmore}@unlv.edu
[2] Dipartimento di Informatica, Università di Pisa
{pagli,prencipe}@di.unipi.it

Abstract. In this paper, we consider the *all best swap edges problem* in a distributed environment. We are given a 2-edge connected positively weighted network X, where all communication is routed through a rooted spanning tree T of X. If one tree edge $e = \{x, y\}$ fails, the communication network will be disconnected. However, since X is 2-edge connected, communication can be restored by replacing e by non-tree edge e', called a *swap edge* of e, whose ends lie in different components of $T - e$. Of all possible swap edges of e, we would like to choose the best, as defined by the application. The *all best swap edges problem* is to identify the best swap edge for every tree edge, so that in case of any edge failure, the best swap edge can be activated quickly. There are solutions to this problem for a number of cases in the literature. A major concern for all these solutions is to minimize the number of messages. However, especially in fault-transient environments, time is a crucial factor. In this paper we present a novel technique that addresses this problem from a time perspective; in fact, we present a distributed solution that works in linear time with respect to the height h of T for a number of different criteria, while retaining the optimal number of messages. To the best of our knowledge, all previous solutions solve the problem in $O(h^2)$ time in the cases we consider.

1 Introduction and Preliminaries

For a communication network, low cost and high reliability can be conflicting goals. For example, a *spanning tree* of a network could have minimum cost, but will not survive even a single link failure. We consider the problem of restoring connectivity when one link of a spanning tree fails.

One recent technique, particularly efficient in case of transient faults, consists in pre-computing a *replacement spanning tree* for each possible link or node failure, by computing the best replacement edge (or edges) which reconnects the tree. A number of studies have been done for this problem, both for the sequential [1–6] and distributed [7–10] models of computation, for different types of spanning trees and failures.

[⋆] This work has been partially supported by MIUR of Italy under projects *ARS technomedia* and *AlgoDEEP* prot. 2008TFBWL4.

In this paper, we consider the *all best swap edges problem* in the distributed setting. We are given a positively weighted 2-edge connected network X of processes, where $w(x, y)$ denotes the weight of any edge $\{x, y\}$ of X, together with a spanning tree T of X, rooted at a process r. Suppose that all communication between processes is routed through T. If one tree edge $e = \{x, p(x)\}$ fails (where $p(x)$ denotes the parent of x in T) we say that x is the *point of failure.* Since X is 2-edge connected, communication can be restored by replacing e by an edge e' of X whose ends lie in different components of $T - e$. We call such an edge e' a *swap edge* of x (or a swap edge of e), and we define $\mathit{SwapEdges}(x)$ (or $\mathit{SwapEdges}(e)$) to be the set of all swap edges of x (refer to the example depicted in Figure 1.(b) and (c)). Of all possible swap edges of x, we would like to choose the best, as defined by the application. The *all best swap edges problem* is to identify the best swap edge for every tree edge, so that in case of any edge failure, the best replacement edge can be activated quickly.

Notation. Given T a spanning tree of X, we refer to an edge of T as a *tree edge*, and any other edge of X as a *cross edge* (see also Figure 1.(a)).

If $x \neq r$ is a process, we denote the set of children of x by $\mathit{Chldrn}(x)$, and the subtree of T rooted at x by T_x; the *level* of a process x is defined to be the *hop-distance* from x to r. We write $x \leq y$ or $y \geq x$ to indicate that x is an ancestor of y, *i.e.,* $y \in T_x$, and $x < y$ or $y > x$ if x is a proper ancestor of y.

If S is any subgraph of X, we let $\mathit{path}_S(x, y)$ denote the shortest (least weight) path through S from x to y, and let $W_S(x, y)$ denote the weighted length of $\mathit{path}_S(x, y)$. (We write simply $\mathit{path}(x, y)$ and $W(x, y)$ if S is understood.)

We will denote by T^* the *augmented tree*, whose nodes consist of all processes of T, together with a node for each directed cross edge of T, which we call an *augmentation node* of T^*. (See Figure 1.(d).) In particular, if $\{y, y'\}$ is a cross edge in T, we will denote by $[y, y']$ and $[y', y]$ its corresponding nodes in T^*; the parent of $[y, y']$ is y. For any process x, define T^*_x to be the subtree of the augmented tree rooted at x; in particular, T^*_x consists of T_x together with all the augmentation nodes $[y, y']$ such that $y \in T_x$.

Related Work and Our Contribution. In [2, 9], several different criteria for defining the "best" swap edge for a tree edge e have been considered. In each case, the best swap edge for e is that swap edge e' for which some penalty function F is minimized. We consider three penalty functions in this paper. In each case, let $T' = T - e + e'$ be the spanning tree of X obtained by deleting e and adding e', where $e = \{x, p(x)\}$ is a tree edge, $y \in T_x$, and $e' = \{y, y'\}$ a swap edge for e.

1. $F_{\text{wght}}(x, y, y') = w(e')$, the weight of the swap edge. Note that if T is a minimum spanning tree of X and e' is that swap edge for e such that $w(e')$ is minimum, then $T' = T - e + e'$ is a minimum spanning tree of $X - e$.
2. $F_{\text{dist}}(x, y, y') = W_{T'}(r, x)$, the distance from the root to the point of failure in T'.
3. $F_{\text{max}}(x, y, y') = \max\{W_{T'}(r, u) : u \in T_x\}$, the maximum distance, in T', from the root to any process in T_x.

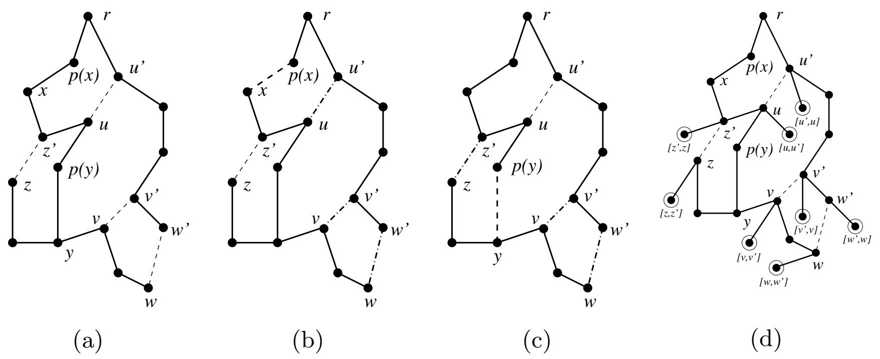


Fig. 1. (a) An example of a network X and its spanning tree T: Tree edges are bold, cross edges are dotted. (b) Failure at x. $\{u, u'\}$, $\{v, v'\}$, and $\{w, w'\}$ are the swap edges of x. (c) Failure at y. $\{v, v'\}$, $\{w, w'\}$, and $\{z, z'\}$ are the swap edges of y. (d) The augmented tree of T; the augmentation nodes are double circled.

If F is any of the above penalty functions, we define $F(x, y, y') = \infty$ for any $\{y, y'\}$ which is not a swap edge of x. The output of the problem is then $M_F(x) = \min\{F(x, y, y') : (y, y') \in T_x^*\}$.

In [7], Flocchini *et al.* give an algorithm for solving the F_{dist} version of all best swap edge problem. In [9], Flocchini *et al.* give a general algorithm for the all best swap edges problem, and then give specific versions of the technique for the $F_{\max}$ version. In [8], the F_{wght} version is solved both for the failure of a link and for the failure of a node and all its incident links.

All the above mentioned distributed solutions have the same general form, and have message complexity $O(n^*)$, with n^* the number of edges of the transitive closure of $T_r \setminus \{r\}$. The time complexity of each is $O(h^2)$, where h is the height of T. In particular, each of those solutions consists of two waves for each level ℓ of T, with $1 \leq \ell \leq h$: A broadcast wave followed by a convergecast wave. In particular, the general schema of previous solutions consists of two nested loops, where the outer loop is indexed by ℓ, and for each ℓ, the inner loop computes $M_F(x)$ for all x at level ℓ using two waves; a top-down wave that computes $F(x, y, y')$ for all $(y, y') \in T_x^*$, and a bottom-up wave that computes $M_F(x)$. Each wave takes $O(h)$ time in the worst case, hence the overall strategy leads to a final cost in time of $O(h^2)$. This is mainly due to the fact that the waves needs to be executed one after the other. In this paper we present a novel technique that finds a solution in linear time, for each of the penalty functions listed above. In particular, our strategy distributes the information and the computation among processes so that the waves can be pipelined. This reduces the final time of the execution to $O(h)$, using the same number of messages as the previous solutions.

As a final remark, we note that in [2], Gfeller *et al.* study the problem of finding the optimal swap edges of a minimum spanning tree having minimum diameter: They provide a distributed algorithm that already works in linear time. The general technique presented here can be also adapted to this case.

The paper is organized as follows. The overall structure of our paradigm is given in Section 2. The various phases are described in Sections 3, 4, and 5. Due to space constraints, some of the proofs are given in the appendix.

2 The Linear Time Solution

In this section, we present the strategy that allows to devise $O(h)$-time distributed algorithms to solve the five versions of the all best swap edges problem introduced in the previous section. We call these algorithms LINEAR$_{\mathrm{dist}}$, LINEAR$_{\mathrm{wght}}$, LINEAR$_{\mathrm{max}}$, respectively. Each can be considered to be a version of a general algorithm, which we call LINEAR, whose structure is given as Algorithm 1. LINEAR is structured in phases; the actual number of phases depends on the specific version of the problem. However, in all cases, the number of phases is at least three: a *preprocessing* phase, a *ranking* phase, and an *optimization* phase. In the last optimization phase, a piece of information, denoted by *up_package*(y, ℓ), is computed in a convergecast wave. The content of this package is different for each of the versions of the problem (details in Section 5).

Algorithm 1. LINEAR

```
1: Preprocessing Phase
2: Ranking Phase
3: If LINEAR$_{\mathrm{max}}$ Then Additional Critical Level Phase(s)
4: Optimization Phase
```

Each of the phases of LINEAR uses at most $O(\delta_x)$ space for each x, where δ_x is the degree of x. The space complexity of LINEAR is thus $O(\delta_x)$ for each x.

Our linear time algorithms make use of the concept of *critical level.* Informally, a critical level function is a function that can be computed top-down, which enables another function – whose computation would otherwise require independent top-down followed by bottom-up waves for all processes – to be computed in a single bottom-up wave for each process, thus allowing the waves to be pipelined. In particular, for each of the versions of the best swap edge problem we consider, one or more critical levels are computed, depending on the specific penalty function. Due to space constraints, all the proofs will be omitted.

The Role of Critical Levels. A *critical level* function is a function Λ on the augmentation nodes of T^* such that $0 \le \Lambda(y, y') \le y.level$, and which aids in the computation of $F(x, y, y')$ for any $x \le y$. More specifically, the computation of $F(x, y, y')$ contains a branch which depends on the comparison between $x.level$ and $\Lambda(y, y')$. For example, the function *rank*, defined in Section 4, has the property that $F(x, y, y') = \infty$ if and only if $rank(y, y') \ge x.level$, where F is any one of the penalty functions defined above.

3 Preprocessing Phase

In the preprocessing phase, which takes $O(h)$ time, each process x computes and retains a set of variables, some of which are the same as in [7, 9]. All the variables listed below are needed for LINEAR$_{\text{max}}$, but only *level*, *index*, and *depth* are needed for LINEAR$_{\text{wght}}$ and LINEAR$_{\text{dist}}$.

1. $x.level$, the *level* of x, which is the hop-distance from r to x.
2. $x.index = (x.pre\_index, x.post\_index)$, the *index* of x, where $x.pre\_index$ is the index of x in the pre-order visit of T, and $x.post\_index$ is the index of x in the reverse postorder of T (see Figure 2(a)).
3. $x.depth = W(r, x)$, the *depth* of x.
4. $x.height = \max \{W(x, u) : u \in T_x\}$, the *height* of x.
5. $x.best\_child$, the *best child* of x, defined to be the process $y \in Chldrn(x)$ such that $w(x, y) + y.height > w(x, z) + z.height$ for any other child z of x. Note that, since we use a strict inequality in this definition, a process can have at most one best child. If $Chldrn(x) = \emptyset$, or if there is more than one choice of y for which $w(x, y) + y.height$ is maximum, $best\_child(x)$ is undefined.
6. $x.eta$, for $x \neq r$, the largest weight of any path in $T_{p(x)} - T_x$ from $p(x)$; that is, $x.eta = \max \{w(p(x), y) + y.height : y \neq x \text{ and } y \in Chldrn(p(x))\}$. If x is the only child of its parent, then $x.eta$ defaults to 0.
7. $x.secondary\_height$, the length of the longest path which does not contain $x.best\_child$ from x to any leaf of T_x. In the case that $x.best\_child$ is undefined, let $x.secondary\_height = x.height$.

Note that all of the above variables can be computed with a constant number of broadcast and convergecast waves, in $O(h)$ total time.

4 Ranking Phase

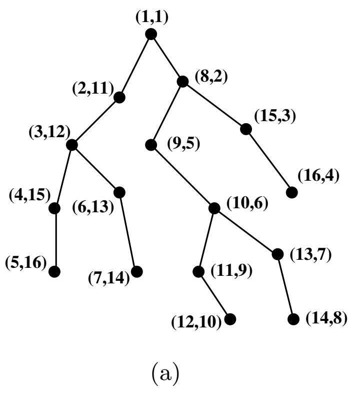
The ranking phase is the same for all versions of the best swap edge problem. In this phase, we compute the *rank* of every cross edge $\{y, y'\}$, defined to be the level of the nearest common ancestor of y and y' in T. This value is stored by both y and y'. Ranks are used to distinguish swap edges of x from other cross edges in T^*_x.

Remark 1.
(a) A process x is an ancestor of y if and only if $x.index \leq y.index$.
(b) If $[y, y'] \in T^*_x$, then $\{y, y'\} \in SwapEdges(x)$ if and only if $x.index \not\leq y'.index$.

From the previous remark, it follows that:

Remark 2. Let $x \neq r$ be a process and $e' = \{z, z'\}$ a cross edge, where $z \in T_x$. Then, e' is a swap edge for x if and only if $rank(z, z') < x.level$.

The ranking phase is given as Algorithm 2. In particular, there is a main loop that cycles over the levels of the tree in increasing order. The phase consists of



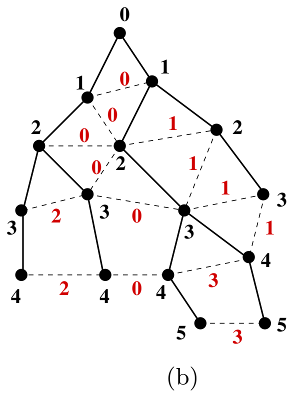


Fig. 2. (a) Processes are labeled with their indices. A process x is an ancestor of y if and only if $x.index \leq y.index$. (b) Levels of processes and ranks of cross edges.

Algorithm 2. Ranking Phase: Rank of every Cross Edge of T is Computed

```
1: For $0 \leq \ell \leq h$ in increasing order Do %Wave $\ell$%
2:     For all $y$ such that $y.level \geq \ell$ in top-down order Do
3:         If $y.level = \ell$ Then $ancestor\_index(y, \ell) \leftarrow y.index$
4:         Else $ancestor\_index(y, \ell) \leftarrow ancestor\_index(p(y), \ell)$
5:         For all cross edges $\{y, y'\}$ Do
6:             If $y'.index \not\geq ancestor\_index(y, \ell)$ Then $rank(y, y') \leftarrow \ell$
```

a top-down *wave* for each $0 \leq \ell \leq h$, denoted by Wave ℓ. For each ℓ, the inner loop computes, for each process y whose level is greater than or equal to ℓ, the value $ancestor\_index(y, \ell)$, which is $x.index$ where x is the ancestor of y at level ℓ. Then, for each $[y, y'] \in T_x^*$, the value ℓ is assigned to $rank(y, y')$ if $y' \notin T_x$, *i.e.*, $y'.index \not\geq ancestor\_index(y.\ell)$ (refer to Remark 1).

The inner loop is executed as a top-down wave; hence the waves can be pipelined, so that the total time of the ranking phase is $O(h)$.

Lemma 1. *If $rank(y, y') = \ell$, then, for all $\ell' \leq \ell$, the computed value of $rank(y, y')$ will be set to ℓ' during Wave ℓ' of Algorithm 2 and thus the final computed value of $rank(y, y')$ will be ℓ.*

5 Optimization Phase

The optimization phase is implemented as a bottom-up wave for each level ℓ. (Refer to Algorithm 1.) In this phase, all best swap edges are computed. In particular, the phase consists of an outer loop, indexed by decreasing values of $1 \leq \ell \leq h$, where each iteration consists of an inner loop which computes $M_F(x)$ for all x at level ℓ. For each x such that $x.level = \ell$, the inner loop consists of a convergecast wave, which computes a set of variables we call $\mathit{up\_package}(y, \ell)$ for each $y \in T_x$; each process y is able to compute $\mathit{up\_package}(y, \ell)$ by using the information computed and stored at y during the earlier phases, as well as the contents of $\mathit{up\_package}(z, \ell)$ received from all $z \in \mathit{Chldrn}(y)$. The final value of $M_F(x)$ is then computed using $\mathit{up\_package}(x, \ell)$. To save space, each up-package is deleted as soon as it is no longer needed. The convergecast waves can be pipelined, and thus the entire optimization phase can be executed in $O(h)$ time.

The specific content of $\mathit{up\_package}(y, \ell)$ depends on the specific version of LINEAR that is solved.

5.1 LINEAR$_{\mathbf{wght}}$ and LINEAR$_{\mathbf{dist}}$

For each $\ell \geq 1$ and each $y \in T$ at level $\geq \ell$, let x be the unique ancestor of y at level ℓ, and let $e = \{x, p(x)\}$. We define $\mathit{Swap\_N}(y, \ell)$ to be the set of all neighbors y' of y such that $\{y, y'\}$ is a swap edge for e. In order to compute this set, the test established by Remark 2 is used.

For both LINEAR$_{\mathrm{wght}}$ and LINEAR$_{\mathrm{dist}}$, $\mathit{up\_package}(y, \ell)$ consists of just the value $\mathit{sbtree\_min}(y, \ell)$, defined as follows. If x is the unique ancestor of y at level ℓ, then

1. In LINEAR$_{\mathrm{wght}}$: $\mathit{sbtree\_min}(y, \ell) = \min\{w(z, z')\}$, such that $(z, z') \in T_y^* \cap \mathit{SwapEdges}(x)$.
2. In LINEAR$_{\mathrm{dist}}$: $\mathit{sbtree\_min}(y, \ell) = \min\{W(x, z) + w(z, z') + z'.\mathit{depth}\}$, such that $(z, z') \in T_y^* \cap \mathit{SwapEdges}(x)$.

At the end of the iteration for ℓ, the value of $M_F(x)$ is set to $\mathit{sbtree\_min}(x, \ell)$ for all x at level ℓ.

The pseudo-code of the optimization phase, for the functions F_{wght} and F_{dist} is given as Algorithm 3 and 4, respectively. In both cases, the waves of the optimization phase are pipelined, permitting the total time complexity of the phase to be $O(h)$.

Concerning the number of messages of both LINEAR$_{\mathrm{wght}}$ and LINEAR$_{\mathrm{dist}}$, note that the information sent along the tree either in the ranking phase or in the optimization phase, is composed of messages of constant size. In the ranking phase, the information consists of node indices, and in the optimization phase of "subtree minimum" values. Thus, the communication complexity, corresponding to the transitive closure of the tree edges, is $O(n^*)$ (limited by $O(n^2)$) in both cases.

5.2 LINEAR$_{\mathbf{max}}$

If $S \subseteq X$ is connected and $x \in S$, define $\mathit{radius}(S, x) = \max\{W_S(x, s) : s \in S\}$, the *radius of* S *based at* s. Note that, we can write $F_{\max}(x, y, y') = \max\{W_{T'}(r, u) : u \in T_x\} = \mathit{radius}(T_x, y) + w(y, y') + y'.\mathit{depth}$ if $\{y, y'\} \in \mathit{SwapEdges}(x)$. Thus, in the case of LINEAR$_{\max}$ we face with the problem of computing $\mathit{radius}(T_x, y)$: This computation is handled by an additional phase before the actual optimization phase. In this phase, we compute a variable called $\mathit{critical\_level}(y)$, for all $y \in T_x$.

Algorithm 3. Algorithm $\text{LINEAR}_{\text{wght}}$

```
1: Preprocessing phase
2: Ranking phase
3: For all $1 \le \ell \le h$ Do %Optimization Phase%
4:     For all $y$ such that $y.level \ge \ell$ in bottom-up order Do
5:         $Swap\_N(y,\ell) \leftarrow \{y' : \{y',y\}$ is a cross edge and $rank(y,y') < \ell\}$
6:         $sbtree\_min(y,\ell) \leftarrow \min \begin{cases} w(y,y') : y' \in Swap\_N(y,\ell) \\ \min\{sbtree\_min(z,\ell) : z \in Chldrn(y)\} \end{cases}$
7:     For all $x$ such that $x.level = \ell$ Do
8:         $M_F(x) = sbtree\_min(x,\ell)$
```

Algorithm 4. Algorithm $\text{LINEAR}_{\text{dist}}$

```
1: Preprocessing phase
2: Ranking phase
3: For all $1 \le \ell \le h$ Do %Optimization Phase%
4:     For all $y$ such that $y.level \ge \ell$ in bottom-up order Do
5:         $Swap\_N(y,\ell) \leftarrow \{y' : \{y',y\}$ is a cross edge and $rank(y,y') < \ell\}$
6:         $sbtree\_min(y,\ell) \leftarrow \min \begin{cases} w(y,y') + depth(y') : y' \in Swap\_N(y,\ell) \\ \min\{w(y,z) + sbtree\_min(z,\ell) : z \in Chldrn(y)\} \end{cases}$
7:     For all $x$ such that $x.level = \ell$ Do
8:         $M_F(x) = sbtree\_min(x,\ell)$
```

Additional Critical Level Phase. For $y \in T_x$, define $\mu(y,x)$ to be the weight of the longest path in T_x from y to any node of $T_x - T_y$. We let $\mu(x,x) = 0$ by default. It follows from these definitions that

$$radius(T_x, y) = \max \begin{cases} y.height \\ \mu(y,x) \end{cases} \tag{1}$$

Since we want LINEAR to use only constant space per process, y can hold only $O(\delta_y)$ values; hence, it could not be possible for y to store all the values $\{\mu(y,x) : x \le y\}$. We tackle this problem by executing in $\text{LINEAR}_{\text{max}}$ an extra phase before the optimization phase (called *critical level phase* in Algorithm 1). In particular, as the convergecast wave moves up the tree, we compute the *critical level* of y, that determines not the actual value of $radius(T_x, y)$, but rather which of the two choices given in Equation (1) is larger, together with enough additional information to calculate the actual value of $M_F(x)$ when the wave reaches x.

We now explain critical levels in greater detail. Let

$$critical\_level(y) = \min\{x.level : y \in T_x \text{ and } radius(T_x, y) = y.height\}.$$

Note that $critical\_level(y) = \min\{x.level : y \in T_x \text{ and } \mu(y,x) \le y.height\}$.

Lemma 2. *For any processes $x' \le x \le y$, $\mu(y,x') \ge \mu(y,x)$.*

Corollary 1. *If $y \in T_x$, then $radius(T_x, y) = y.height$ if and only if $x.level \ge critical\_level(y)$.*

Critical levels are calculated by Algorithm 5. Recall, from Section 3, that $y.eta$, for $y \neq r$, is the largest weight of any path in $T_{p(y)} - T_y$ from $p(y)$; this value is computed during the preprocessing phase. Note that, once again, the waves of the inner loop of Algorithm 5 can be pipelined, so that the total time required for this phase is again $O(h)$.

Algorithm 5. Critical Level Phase

```
1: For $0 \le \ell \le h$ in decreasing order Do %$Wave\ \ell$%
2:     For all $x$ such that $x.level = \ell$ concurrently Do
3:         $\mu(x,x) \leftarrow 0$
4:         For all $y \in T_x - x$ in top down order Do
5:             $\mu(y,x) \leftarrow \max \begin{cases} \mu(p(y),x) + w(y,p(y)) \\ y.eta \end{cases}$
6:             If $\mu(y,x) \le y.height$ Then $critical\_level(y) \leftarrow \ell$
```

Optimization Phase. Before introducing the optimization phase, we need to introduce the notion of *Spine*, which is strictly related to the notion of critical level. (Refer also to Figure 3.)

Definition 1 (Spine). *Given any process x, we define the* Spine *of x:*

$$Spine(x) = \{y \in T_x : radius(T_x, y) = y.height\}.$$

We extend this definition to a specific level ℓ as follows: $Spine(\ell) = \bigcup\{Spine(x) : x.level = \ell\}$.

We will denote by $Others(x)$ the nodes in T_x that are not in $Spine(x)$ (*i.e.*, $Others(x) = T_x - Spine(x)$), and by $Others(\ell) = \bigcup\{Others(x) : x.level = \ell\}$. Furthermore, we define the *base process* of x, denoted by $base(x)$, as the process in $Spine(x)$ of greatest level; again, given a specific level ℓ, we let $Base(\ell) = \{base(x) : x.level = \ell\}$. We define the *tail process* of x as $tail(x) = best\_child(base(x))$ (note that $tail(x)$ might be not defined), and $Tail(\ell) = \{tail(x) : x.level\} = \ell$. Finally, we let $Fan(x) = T_{tail(x)}$ and $Fan(\ell) = \bigcup\{Fan(x) : x.level = \ell\}$; if $tail(x)$ is undefined, we let $Fan(x) = \emptyset$. We now give few properties of $Spine(x)$.

Lemma 3. *For any process x*
(a) *$x \in Spine(x)$.*
(b) *If $y \in Spine(x)$ and $y \neq x$, then $p(y) \in Spine(x)$ and $y = best\_child(p(y))$.*
(c) *$Spine(x)$ is a chain.*

For any $s \in S$, where S is connected, let $longest\_path(S, s)$ denote the simple path of weight $radius(S, s)$ in S starting at s. In the next lemma, we give a characterization of $longest\_path(T_x, y)$.

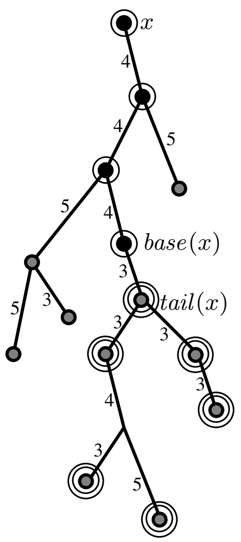


Fig. 3. Def. 1: Black (single circled) nodes are in *Spine*(x), while the light gray nodes are in *Others*(x); *base*(x), *tail*(x) are also shown; nodes in *Fan*(x) are double circled

Lemma 4. *Let $y \in T_x$, and let u be the process of minimum level on longest_path (T_x, y). Then, the following properties hold:*
(a) *$u \in Spine(x)$.*
(b) *If $y \in Fan(x)$, then $longest\_path(T_x, y) = path(y, u) +$ $secondary\_down\_path(u)$, where "+" denotes concatenation of paths.*
(c) *If $y \notin Fan(x)$, then $longest\_path(T_x, y) = path(y, u) + longest\_path(T_u, u)$,*

Let F_ℓ be the forest given by the union of all T_x, where $x.level = \ell$. Thus, $radius(F_\ell, y) = radius(T_x, y)$ if $x.level = \ell$ and $y \in T_x$. The critical level of a process y enables y to determine whether it lies in *Others*(ℓ) for any given ℓ, as shown by the following lemma.

Lemma 5. *$y \in Spine(\ell)$ if and only if $critical\_level(y) \leq \ell \leq y.level$.*

Corollary 2. *Given any y such that $\ell \leq y.level$, the following properties hold:*
(a) *$y \in Others(\ell)$ if and only if $critical\_level(y) > \ell$.*
(b) *$y \in Spine(\ell)$ if and only if $critical\_level(y) \leq \ell$.*
(c) *$y \in Base(\ell)$ if and only if $y \in Spine(\ell)$, and either $best\_child(y) \in Others(\ell)$, or $best\_child(y)$ is undefined.*
(d) *$y \in Tail(\ell)$ if and only if $p(y) \in Base(\ell)$ and $y = best\_child(p(y))$.*

Corollary 2 is used during the optimization phase of LINEAR$_{\max}$ to determine the content of *up_package*(y, ℓ) (See Algorithms 1 and 6). In particular, the optimization phase proceeds bottom-up in the tree, with two nested loops. Let

$$local\_cost(y, \ell) = \min\{w(y, y') + depth(y') : y' \in Swap\_N(y, \ell)\},$$

where *Swap_N*(y, ℓ) is as defined in Section 5.1.

Algorithm 6. Algorithm LINEAR$_{\max}$

```
1: Preprocessing phase
2: Ranking phase
3: Critical Level Phase (Algorithm 5)
4: For all $1 \leq \ell \leq h$ Do %Optimization Phase%
5:     For all $y$ such that $y.level \geq \ell$ in bottom-up order Do
6:         $Swap\_N(y,\ell) \leftarrow \{y' : \{y', y\}$ is a cross edge and $rank(y,y') < \ell\}$
7:         $local\_cost(y,\ell) = \min\{w(y,y') + depth(y') : y' \in Swap\_N(y,\ell)\}$
8:         If $y \in Others(\ell)$ Then
9:             $min\_up\_cost(y,\ell) \leftarrow \min \begin{cases} local\_cost(y,\ell) \\ \min\{min\_up\_cost(z,\ell) + w(y,z) : z \in Chldrn(y)\} \end{cases}$
10:        Else %$y \in Spine(\ell)$%
11:            $min\_normal\_cost(y,\ell) \leftarrow \min \begin{cases} local\_cost(y,\ell) \\ \min\{min\_up\_cost(z,\ell) + w(y,z) : z \in Normal\_Chldrn(y)\} \end{cases}$
12:            If $best\_child(y)$ is defined Then
13:                $z \leftarrow best\_child(y)$
14:                If $z \in Spine(\ell)$ Then
15:                    $min\_fan\_cost(y,\ell) \leftarrow min\_fan\_cost(z,\ell) + w(z,y)$
16:                Else
17:                    $min\_fan\_cost(y,\ell) \leftarrow min\_up\_cost(z,\ell) + w(z,y)$
18:                $sbtree\_min(y,\ell) \leftarrow \min \begin{cases} min\_normal\_cost(y,\ell) + y.height \\ min\_fan\_cost(y,\ell) + secondary\_height(y) \\ sbtree\_min(z,\ell) \end{cases}$
19:            Else %$y = base(x)$, and $tail(x)$ undefined%
20:                $min\_fan\_cost(y,\ell) \leftarrow \infty$
21:                $sbtree\_min(y,\ell) \leftarrow min\_normal\_cost(y,\ell) + y.height$
22:    For all $x$ such that $x.level = \ell$ Do
23:        $M_F(x) = sbtree\_min(x,\ell)$
```

If $y \in Others(\ell)$, then $radius(F_\ell, y)$ is not computed going down in the tree; hence, the only information that needs to be propagated (that is, the content of $up\_package(y,\ell)$) is $min\_up\_cost(y,\ell) = \min\{local\_cost(z,\ell) + W(y,z) : z \in T_y\}$, *i.e.,* the minimum value of $W(path(y,z)) + w(z,z') + depth(z')$ over all $z \in T_y$ such that $\{z,z'\} \in SwapEdges(x)$.

If $y \in Spine(\ell)$, first the value of $min\_normal\_cost(y,\ell)$ is computed: It is equal to $\min\{local\_cost(z,\ell) + W(y,z)\}$, such that $z \in T_y$ and $z \notin T_{best\_child(y)}$. Then, the algorithm branches according to whether $best\_child(y)$ is defined or not. If $z = best\_child(y)$ is defined, then $T_y \cap Fan(\ell) \neq \emptyset$; in this case we compute $min\_fan\_cost(y,\ell) = \min\{local\_cost(z,\ell) + W(y,z) : z \in T_y \cap Fan(\ell)\}$, *i.e.,* the min value of $W(path(y,z)) + w(z,z') + depth(z')$ over all $\{z,z'\} \in SwapEdges(x)$ such that $z \in T_y \cap Fan(\ell)$ (note that the algorithm computes $min\_fan\_cost(y,\ell)$ differently, according to whether $z \in Spine(\ell)$ or not). Finally, the actual cost of the swap edge is computed: $sbtree\_min(y,\ell)$, which is the minimum value of $radius(T_y,z) + w(z,z') + depth(z')$ over all $\{z,z'\} \in SwapEdges(x)$ such that $z \in T_y$ (this is the value that is propagated in $up\_package(y,\ell)$). By Lemma 4, $sbtree\_min(y,\ell)$ is the minimum between the value of $sbtree\_min(z,\ell)$ obtained from z, $min\_normal\_cost(y,\ell) + y.height$, and $min\_fan\_cost(y,\ell) + secondary\_height(y)$. If $z = best\_child(y)$ is not defined, then $T_y \cap Fan(\ell) = \emptyset$.

In this case, $min\_fan\_cost(y, \ell)$ is set to ∞, and $sbtree\_min(y, \ell)$ is set to $min\_normal\_cost(y, \ell)$ $+y.height$.

When the ℓ^{th} wave terminates, it is possible to compute the best swap edge for x: $M_F(x) = sbtree\_min(x, \ell)$ for all x such that $x.level = \ell$. Again, as in the previous cases, the waves are executed in pipeline, and thus the overall time complexity is $O(h)$. Also, in this case, it is not difficult to see that the number of messages used by Algorithm 6 is the same as for the quadratic time versions, *i.e.,* $O(n^*)$.

Theorem 1. *The overall time complexity for* LINEAR *is* $O(h)$.

References

1. Bilò, D., Gualà, L., Proietti, G.: Finding best swap edges minimizing the routing cost of a spanning tree. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 138–149. Springer, Heidelberg (2010)
2. Gfeller, B., Santoro, N., Widmayer, P.: A distributed algorithm for finding all best swap edges of a minimum diameter spanning tree. IEEE Trans. on Dependable and Secure Comp. 8(1), 1–12 (2011)
3. Nardelli, E., Proietti, G., Widmayer, P.: Swapping a failing edge of a single source shortest paths tree is good and fast. Algorithmica 35(1), 56–74 (2003)
4. Nardelli, E., Proietti, G., Widmayer, P.: Nearly linear time minimum spanning tree maintenance for transient node failures. Algorithmica 40(1), 119–132 (2004)
5. Salvo, A.D., Proietti, G.: Swapping a failing edge of a shortest paths tree by minimizing the stretch factor. Theoretical Computer Science 383(1), 23–33 (2007)
6. Das, S., Gfeller, B., Widmayer, P.: Computing best swaps in optimal tree spanners. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 716–727. Springer, Heidelberg (2008)
7. Flocchini, P., Enriques, A.M., Pagli, L., Prencipe, G., Santoro, N.: Point-of-failure shortest-path rerouting: Computing the optimal swap edges distributively. IEICE Transactions 89-D(2), 700–708 (2006)
8. Flocchini, P., Pagli, A.M.E.L., Prencipe, G., Santoro, N.: Distributed minumum spanning tree maintenance for transient node failures. IEEE Trans. on Comp. 61(3), 408–414 (2012)
9. Flocchini, P., Pagli, L., Prencipe, G., Santoro, N., Widmayer, P.: Computing all the best swap edges distributively. J. Parallel Distrib. Comput. 68(7), 976–983 (2008)
10. Pagli, L., Prencipe, G.: Brief annoucement: Distributed swap edges computation for minimum routing cost spanning trees. In: Abdelzaher, T., Raynal, M., Santoro, N. (eds.) OPODIS 2009. LNCS, vol. 5923, pp. 365–371. Springer, Heidelberg (2009)

Decentralized Throughput Scheduling[⋆]

Jasper de Jong, Marc Uetz, and Andreas Wombacher

University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
{j.dejong-3,m.uetz,a.wombacher}@utwente.nl

Abstract. Motivated by the organization of distributed service systems, we study models for throughput scheduling in a decentralized setting. In throughput scheduling, a set of jobs j with values w_j, processing times p_{ij} on machine i, release dates r_j and deadlines d_j, is to be processed non-preemptively on a set of unrelated machines. The goal is to maximize the total value of jobs scheduled within their time window $[r_j, d_j]$. While approximation algorithms with different performance guarantees exist for this and related models, we are interested in the situation where subsets of machines are governed by selfish players. We give a universal result that bounds the price of decentralization: Any local α-approximation algorithm, $\alpha \geq 1$, yields Nash equilibria that are at most a factor $(\alpha+1)$ away from the global optimum, and this bound is tight. For identical machines, we improve this bound to $\sqrt[\alpha]{e}/(\sqrt[\alpha]{e}-1) \approx (\alpha + 1/2)$, which is shown to be tight, too. The latter result is obtained by considering subgame perfect equilibria of a corresponding sequential game. We also address some variations of the problem.

1 Model and Notation

We consider a non-preemptive scheduling problem with unrelated machines, to which we refer as *decentralized throughput scheduling problem* throughout the paper. The input of an instance $I \in \mathcal{I}$ consists of a set of jobs $\mathcal{J}$, a set of machines $\mathcal{M}$, and a set of players $\mathcal{N}$. Each job $j \in \mathcal{J}$ comes with a release time r_j, a deadline d_j, a nonnegative value w_j and a processing time p_{ij} if scheduled on machine $i \in \mathcal{M}$. Machines can process only one job at a time. Job j is feasibly scheduled (on any of the machines) if its processing starts no earlier than r_j and finishes no later than d_j. For any set of jobs $S \subseteq \mathcal{J}$, we let $w(S) = \sum_{j \in S} w_j$ be the total value. Each player $n \in \mathcal{N}$ controls a subset of machines $M_n \subseteq \mathcal{M}$ and aims to maximize the total value of jobs that can be feasibly scheduled on its set of machines M_n. Here M_n, $n \in \mathcal{N}$, is a partition of the set of machines $\mathcal{M}$.

In this paper we are interested in *equilibrium allocations*, which we define as an allocation in which none of the players n can improve the total value of jobs that can be feasibly scheduled on its set of machines M_n by removing some of its jobs and adding some of the yet unscheduled jobs. Here we make the assumption that a player cannot make a claim on jobs that are scheduled on machines of

⋆ Research supported by CTIT, Centre for Telematics and Information Technology, University of Twente, project "Mechanisms for Decentralized Service Systems".

other players. An equilibrium allocation is a (pure) Nash equilibrium (*NE*) in a strategic form game where player n's strategies are all subsets of jobs $S_n \subseteq \mathcal{J}$. If jobs S_n can be feasibly scheduled on machines M_n, then player n's valuation for S_n is $w(S_n) = \sum_{j \in S_n} w_j$, and we let $w(S_n) = -\infty$ otherwise. Furthermore, the utility of player n is $-\infty$ whenever S_n is not disjoint with the sets chosen by all other players. This way, in any strategy profile $(S_n)_{n \in \mathcal{N}}$ that is at Nash equilibrium, the sets S_n, $n \in \mathcal{N}$, are pairwise disjoint.

Our main focus will be the analysis of the price of decentralization, better known as the price of anarchy (PoA) [11], lower bounding the quality of any Nash equilibrium relative to the quality of a globally optimal allocation, *OPT*. Here *OPT* is an allocation maximizing the weighted sum of feasibly scheduled jobs over all players. More specifically, we are interested in the ratio

$$\mathrm{PoA} = \sup_{I \in \mathcal{I}} \sup_{NE \in NE(I)} \frac{w(OPT)}{w(NE)}, \tag{1}$$

where $NE(I)$ denotes the set of all Nash equilibria of instance I. Note that *OPT* is a Nash equilibrium too, hence the price of stability, as proposed in [1], equals 1.

In general, the question whether a strategy profile $(S_n)_{n \in \mathcal{N}}$ is a Nash equilibrium describes an NP-hard optimization problem for each player, even if each player controls a single machine only [14]. Therefore, we also consider a relaxed equilibrium condition: We say an allocation is an α-approximate Nash equilibrium (α-*NE*) if none of the players n can improve the total value of jobs that can be feasibly scheduled on its set of machines M_n by a factor larger than α by removing some of its jobs and adding some of the yet unscheduled jobs. By the existence of constant factor approximation algorithms for (centralized) throughput scheduling, e.g. [3,4], the players are thus equipped with polynomial time algorithms to reach an α-*NE* in polynomial time, for certain constant values α.

As an interesting variant of the model described thus far, we also propose to analyze the price of anarchy for subgame perfect equilibria of an extensive form game as introduced by Selten [12,17]. Here, we make the assumption that players select their subsets of jobs sequentially in an arbitrary but fixed order. In that situation, the n-th player is presented the set of yet unscheduled jobs $\mathcal{J} - \bigcup_{i<n} S_i$, from which he may select a subset S_n once, and is not allowed to revoke this decision later. For the special case where all machines are identical, the resulting subgame perfect equilibria of the extensive form game are provably better than Nash equilibria of the strategic game.

2 Motivation, Related Work and Contribution

Our motivation to study this problem is to analyze the performance of decentralized service systems, where jobs are posted, e.g. on a portal, and service providers can select these on a take-it-or-leave-it basis. The problem can be seen as a stylized version of coordination problems that appear in several application domains. We give three examples: (1) When operating microgrids for decentralized energy production and consumption, the goal is to consume locally produced energy as much as possible. Here, jobs can be defined as the operation of appliances (e.g. operating a washing machine), bounded by a time window and attached with a certain \$-value. Machines, on the other side, are local energy producers like PV-panels or micro CHPs [2,15]. (2) In cloud computing, service providers such as Amazon and Google provide an infrastructure service, that is, provide a virtual machine with a specific service level for a certain period of time. The aim of a federated cloud computing environment, e.g. [6], is to "coordinate load distribution among different cloud-based data centers in order to determine optimal location for hosting application services ". (3) In private car sharing portals like Tamyca or Autonetzer [19], clients post car rental requests for a certain time period, and the price they are willing to pay. Car owners in the vicinity can select requests and rent their car(s). Stripping off the online nature from these applications exactly yields the type of problems we address.

The underlying non-strategic optimization problem is sometimes referred to as throughput scheduling. See for example [3], and follow-up papers, e.g. [4]. In the 3-field notation of [9], the problem reads $\mathrm{R}|r_j|\sum w_jU_j$, where R denotes the unrelated machine model, r_j specifies that there are release dates, and the objective is to minimize the total weight of late jobs. In terms of the optimal objective value this is equivalent to the maximization objective considered here, yet it is standard to revert to the maximization version for the purpose of approximation. Indeed, approximation algorithms for several versions of the maximization problem have been discussed in the literature, e.g., with or without weights, identical or unrelated machines, most notably [3,4]. Special cases that are of particular interest are the single machine case with unit weights and zero release dates, solved in polynomial time by the Moore-Hodgson algorithm [16], and the case with identical machines and unit processing times, which can be cast and solved as an assignment problem [5]. To the best of our knowledge, the decentralized version that we propose here has not been addressed before.

Our contribution lies in the informal claim that the price of decentralization is very moderate: If local decisions of all players are approximately optimal with performance guarantee α, then any equilibrium allocation is not worse than an $(\alpha+1)$-fraction of the global optimum. We improve this to $\approx(\alpha+1/2)$ when all machines are identical, and when we consider only subgame perfect equilibria of a corresponding extensive form game. Along the way, we also obtain some additional insights.

3 A First Encounter

Example 1. There are two players $\mathcal{N}=\{1,2\}$, each controlling exactly one of two related machines $\mathcal{M}=\{1,2\}$, with machine speeds $s_1=1, s_2=\frac{2}{3}$, respectively[1]. There are two jobs $\mathcal{J}=\{1,2\}$ with processing times $p_1=p_2=1$, deadlines $d_1=1, d_2=\frac{3}{2}$ and values $w_1=w_2=1$. Release dates are $r_1=r_2=0$. $\triangleleft$

In this example, when job 1 is allocated to machine 1 and job 2 to machine 2, both jobs can meet their respective deadlines. This is obviously an optimal

[1] This is a special case of the unrelated machine model by letting $p_{ij}=p_j/s_i$.

allocation. However when job 2 is allocated to machine 1, only one job can be scheduled before its deadline. See also Figure 1. Note that both allocations are

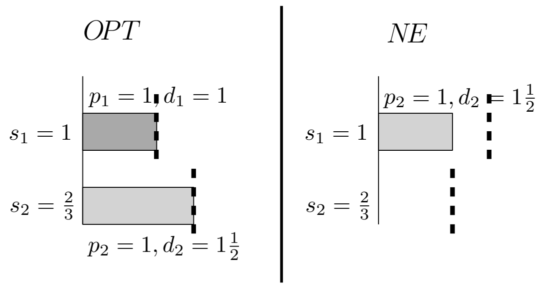


Fig. 1. Optimal solution and Nash equilibrium in the case of related machines

a Nash equilibrium. Now $w(OPT)/w(NE) = 2/1 = 2$ for the second allocation, and we see from this simple example that

$$\mathrm{PoA} \geq 2$$

in (1), even for the case of related machines, unit weights, unit processing times and zero release dates. The strategic form game for Example 1 with both Nash equilibria in boldface is shown in Figure 2. A corresponding extensive form game

		player 2			
		$\emptyset$	{1}	{2}	{1,2}
player 1	$\emptyset$	0,0	$0, -\infty$	$0, 1$	$0, -\infty$
	{1}	1,0	$-\infty, -\infty$	$\mathbf{1, 1}$	$-\infty, -\infty$
	{2}	$\mathbf{1, 0}$	$1, -\infty$	$-\infty, -\infty$	$-\infty, -\infty$
	{1,2}	$-\infty, 0$	$-\infty, -\infty$	$-\infty, -\infty$	$-\infty, -\infty$

Fig. 2. Strategic form game for Example 1 with Nash equilibria

where players select their jobs sequentially, player 1 first, and suppressing the solutions for the trivially inferior strategies $\{1, 2\}$, is shown in Figure 3. Note that each subgame perfect equilibrium of this extensive form game yields an allocation that corresponds to a Nash equilibrium of the strategic form game. Yet the extensive form game has generally more Nash equilibria (here, 3) due to richer strategy spaces of players.

4 Bounds for Approximate Equilibrium Allocations

The players problem to decide if a strategy is at equilibrium is polynomially solvable only for special cases. For instance when jobs have unit values and zero

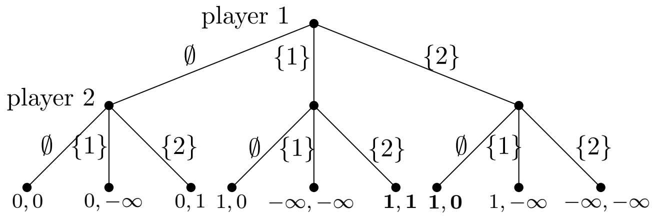


Fig. 3. Extensive form game for Example 1 with subgame perfect equilibria

release dates, and when each player controls exactly one machine, the Moore-Hodgson algorithm [16] maximizes the total number of early jobs. But when players control more than one machine, the players problem is NP-complete as generalization of the makespan minimization problem on parallel machines [8]. When the machines M_n of a player n are identical, and jobs have unit processing times, the players' problem can be cast and solved as an assignment problem [5]. In most other cases, the players' problem is NP-complete. For example, for a player that controls a single machine, when jobs have zero release dates, but arbitrary processing times and weights, the problem is (weakly) NP-hard [13,10]. Adding nontrivial release dates makes the problem strongly NP-hard [14].

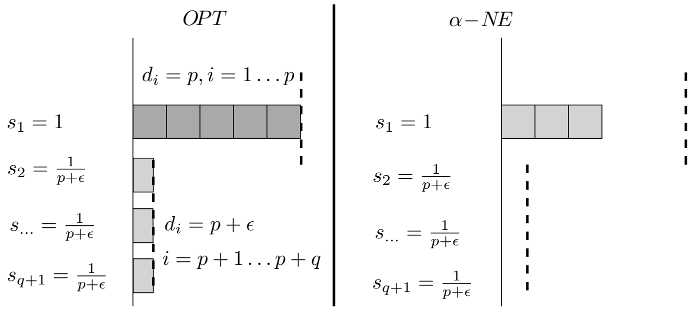
Therefore, we consider a relaxed equilibrium concept, assuming that players strategies are only approximately optimal. This leads to the concept of α-approximate equilibria, which has lately been discussed also in the literature on computing Nash equilibria, for instance in the context of congestion games [18]. Approximate Nash equilibria can also be defined by allowing additive deviations instead of relative deviations, e.g. [7], but given that there exist constant-factor approximation algorithms for throughput scheduling, e.g. [3,4], it appears more reasonable to work with relative bounds here. We say the allocation is an α-approximate Nash equilibrium, or α-*NE*, if no player n can improve the total value of its jobs by a factor larger than α. That said, we obtain the following.

Theorem 1. *The decentralized throughput scheduling problem has PoA* $= \alpha + 1$*, assuming that equilibrium allocations are α-approximate Nash equilibria. The lower bound PoA* $\geq \alpha + 1$ *even holds for the special case of unit values w_j, unit processing times p_j, related machines and zero release dates.*

Proof. First we prove PoA $\leq \alpha + 1$. Take any instance with optimal solution OPT and Nash equilibrium NE[2], and let NE_n and OPT_n, $n \in \mathcal{N}$, be the jobs allocated to player n in NE and OPT, respectively. For any $S \subseteq \mathcal{J}$, let $\overline{S} = \mathcal{J} \setminus S$ be the complement of S in $\mathcal{J}$.

Since all jobs in $\overline{NE}$ are available, and all jobs in OPT_n can be feasibly be scheduled by player n, by the definition of α-approximate Nash equilibrium, we have for all n, $\alpha w(NE_n) \geq w(OPT_n \cap \overline{NE})$. Now we get, by using linearity of the objective function across players,

[2] In a slight abuse of notation, we use OPT and NE to also denote the set of feasibly scheduled jobs in the respective solutions.



OPT
α−NE
$d_i = p, i = 1 \dots p$
$s_1 = 1$
$s_2 = \frac{1}{p+\epsilon}$
$s_{\dots} = \frac{1}{p+\epsilon}$
$s_{q+1} = \frac{1}{p+\epsilon}$
$d_i = p + \epsilon$
$i = p + 1 \dots p + q$

Fig. 4. Optimal solution and α-*NE* in case of related machines

$$\begin{aligned}(\alpha + 1)w(NE) &\geq \alpha w(NE) + w(OPT \cap NE)\\ &= \sum\nolimits_n \alpha w(NE_n) + w(OPT \cap NE)\\ &\geq \sum\nolimits_n w(OPT_n \cap \overline{NE}) + w(OPT \cap NE)\\ &= w(OPT)\,.\end{aligned}$$

To prove PoA $\geq \alpha + 1$ we give a tight example.

Example 2. Consider an instance with unit processing times $p_j = 1$, unit values $w_j = 1$, related machines, and zero release dates. Assume w.l.o.g. that $\alpha = p/q$, $p \geq q$, and assume players deploy an α-approximation each. There are $q + 1$ players $\mathcal{N}$, each controlling one of $q + 1$ machines $\mathcal{M} = \{1, \dots, q + 1\}$ with machine speeds $s_1 = 1$ and $s_2 = s_3 = \cdots = s_{q+1} = 1/(p+\varepsilon)$ for some $0 < \varepsilon < 1$. There are $p + q$ jobs $\mathcal{J} = \{1, \dots, p + q\}$. Jobs $J_1 = \{1, \dots, p\}$ have deadline p. Jobs $J_2 = \{p + 1, \dots, p + q\}$ have deadline $p + \varepsilon$. ◁

Here, machine 1 can schedule at most p jobs. Machines $2, \dots, q+1$ can schedule no jobs from J_1 and only one job from J_2 each. In *OPT* all $p+q$ jobs are feasibly scheduled: jobs J_1 on Machine 1 and each of machines $2, \dots, q + 1$ has one job from J_2. Now consider the α-approximate Nash equilibrium where only q jobs are scheduled: Machine 1 schedules all q jobs from J_2, and machines $2, \dots, q+1$ schedule no job. This is indeed an α-approximate Nash equilibrium, as machine 1 can schedule at most $p = \alpha q$ jobs, and since all jobs from J_2 are scheduled on machine 1, machines $2, \dots, q + 1$ cannot improve from their 0 jobs either. See Figure 4 for an illustration. We conclude that PoA $\geq (p + q)/q = \alpha + 1$. □

Note that $\alpha = 1$ in the special case where the players can verify if a solution is a Nash equilibrium; in that case PoA = 2. Also note that the given upper bound is universal in the sense that it is independent of how the (α-approximate) Nash equilibrium is obtained. It is conceivable that specific algorithms can yield a better bound for the price of anarchy. However, the existence of more complicated counter-examples for specific algorithms is not unlikely either, and we did not take the effort to find them.

5 Subgame Perfect Equilibria

We here propose[3] to analyze the extensive form game in which the players select their subsets of jobs sequentially, and are not allowed to revoke their decisions later. Following Selten [17], an equilibrium of an extensive form game is called (α-approximate) subgame perfect if it induces a (α-approximate) Nash equilibrium in every subgame. The following example shows that indeed, not all (α-approximate) Nash equilibria are (α-approximate) subgame perfect.

Example 3. There are n players each controlling one of n identical machines $\mathcal{M} = \{1, \dots, n\}$, and $2n-1$ jobs $\mathcal{J} = \{1, \dots, 2n-1\}$ with unit weights. Jobs $J_1 = \{1, \dots, n\}$ have processing time $1/n$ and deadline 1. Jobs $J_2 = \{n+1, \dots, 2n-1\}$ have processing time 1 and deadline 1. $\triangleleft$

In *OPT*, machine 1 schedules jobs J_1 and machines $2, \dots, n$ schedule jobs J_2. Consider Nash equilibrium *NE* where each machine schedules one job from J_1. Note that *NE* is indeed an equilibrium: no machine can schedule more than one job without exchanging jobs with another machine. See Figure 5 for an illustration. For this instance $w(OPT)/w(NE) = \frac{2n-1}{n} \to 2$ for $n \to \infty$. This Nash

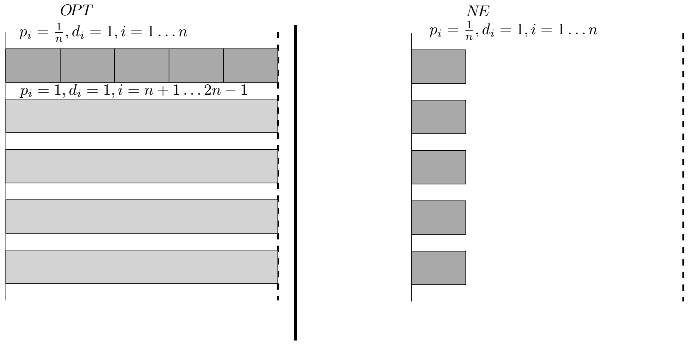

OPT
$p_i = \frac{1}{n}, d_i = 1, i = 1 \dots n$
$p_i = 1, d_i = 1, i = n+1 \dots 2n-1$
NE
$p_i = \frac{1}{n}, d_i = 1, i = 1 \dots n$

Fig. 5. An optimal solution and a Nash equilibrium in case of identical machines

equilibrium is not subgame perfect, however. In any subgame perfect equilibrium, the first player would necessarily schedule all jobs from J_1 on his machine.

This example also shows that the identical machine model does not allow an improvement of the result of Theorem 1. Although non-subgame perfect equilibria might seem unrealistic, the equilibrium obtained in this example is quite reasonable: In a round robin assignment, each player chooses to schedule the most flexible available job(s) first.

Remark: It is not hard to see that each subgame perfect equilibrium of the sequential game proposed here corresponds to an outcome equivalent Nash equi-

[3] Note added in proof: The idea to analyze subgame perfect equilibria of extensive form games rather than Nash equilibria of strategic form games has been proposed also by Paes Leme, R., Syrgkanis, V., Tardos, É. in: The Curse of Simultaneity, Proceedings ICTS, pp. 60–67, ACM (2012).

librium of the (non-sequential) strategic form game that we studied in Section 4. In that sense, our move to subgame perfect equilibria indeed makes sense from the perspective of worst case analysis.

6 Identical Machines

In this section we improve our previous results for the special case of identical machines when considering (α-approximate) subgame perfect equilibria of an extensive form game in which players select their jobs sequentially in any order.

6.1 Identical Machines: Lower Bound

We give a lower bound on the price of anarchy for subgame perfect equilibria.

Theorem 2. *$PoA \geq \sqrt[\alpha]{e}/(\sqrt[\alpha]{e}-1)$ for identical machines, even in the restricted model where we only consider α-approximate subgame perfect equilibria, and for unit processing times, unit weights, and zero release dates.*

Proof. We give a corresponding example.

Example 4. There are n players controlling one of n identical machines $\mathcal{M} = \{1,\ldots,n\}$. There are n^2 jobs $\mathcal{J} = \{1,\ldots,n^2\}$ with unit processing times and unit weights. Jobs have deadlines $\delta \in \{1,\ldots,n\}$ and for each deadline, there are n jobs with this deadline, that is, for all δ, $d_j = \delta$ for $j = 1+(\delta-1)n,\ldots,\delta n$.$\lhd$

We refer to jobs as δ-jobs, $\delta = 1,\ldots,n$. In Figure 6 we see an instance and solution for $n=5$ and $\alpha=2$ (that is, machines use a 2-approximation).

For each of the jobs, the number displayed on it corresponds to its deadline. In *OPT*, every machine schedules n jobs with different deadlines, ordered by increasing deadline. Therefore $w(OPT) = n^2$. We construct an α-approximate subgame perfect equilibrium, say S, as follows. For every machine $i = 1,\ldots,n$ in this order, we find the maximum number of jobs that can be scheduled, say o_i, and let S_i be the $\lceil o_i/\alpha \rceil$ jobs with the largest deadlines (which are the most flexible jobs). For example, for $n=5$ and $\alpha=2$, $w(S) = 3+3+2+2+2 = 12$ as can be seen in Figure 6. We bound $w(S)$ in the following way. In S, denote by $r_\delta(i)$ the fraction of δ-jobs on machine i, relative to the total number of jobs on machine i. Let $r_\delta = \sum_i r_\delta(i)$. In our example, $r_4 = 0 + \frac{1}{3} + 1 + 1 + 0$. Observe that $\sum_\delta r_\delta = n$ for any allocation. In S, any machine scheduling a δ-job, does not schedule any job with deadline $(\delta+2)$ or larger, hence it schedules at most $\lceil (\delta+1)/\alpha \rceil \leq (\delta+1+\alpha)/\alpha$ jobs. Therefore, each job with deadline δ contributes at least $\alpha/(\delta+1+\alpha)$ to r_δ. For any δ for which all n δ-jobs are allocated in S, we get $r_\delta \geq n\alpha/(\delta+1+\alpha)$.

Now, for some $\delta' \geq 0$, by construction of the allocation we have that all n δ-jobs with $\delta = n-\delta',\ldots,n$ are fully scheduled, as well as a fraction of the $(n-(\delta'+1))$-jobs. We get

$$n \geq \sum_{\delta=n-\delta'}^{n} r_\delta \geq \sum_{\delta=n-\delta'}^{n} \frac{n\alpha}{\delta+1+\alpha} \geq \int_{\delta=n-\delta'}^{n} \frac{n\alpha}{\delta+1+\alpha} d\delta\,. \tag{2}$$

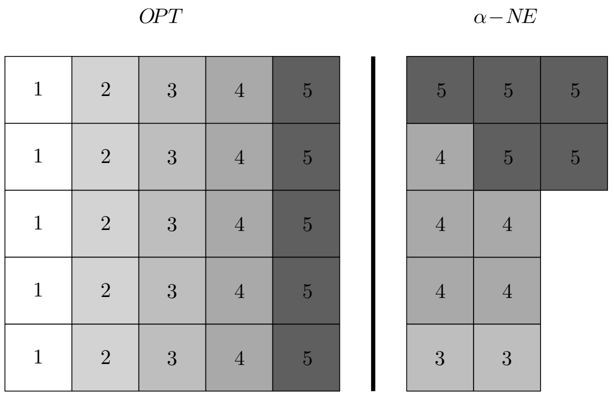


Fig. 6. Optimal solution and 2-approximate subgame perfect equilibrium in case of identical machines. Numbers denote job deadlines.

Because the last term is upper bounded by n, we can derive an upper bound on δ'. In fact, basic calculus shows that

$$\delta' > \frac{(n+1+\alpha)(\sqrt[\alpha]{e}-1)}{\sqrt[\alpha]{e}} \;\Rightarrow\; \int_{\delta=n-\delta'}^{n} \frac{n\alpha}{\delta+1+\alpha} d\delta > n\,,$$

which together with (2) yields that $\delta' \leq \frac{(n+1+\alpha)(\sqrt[\alpha]{e}-1)}{\sqrt[\alpha]{e}}$. Because only δ-jobs with $\delta \geq n-(\delta'+1)$ are scheduled, we conclude that

$$w(S) \leq (\delta'+1)n \leq \frac{(n+1+\alpha+\frac{\sqrt[\alpha]{e}}{\sqrt[\alpha]{e}-1})(\sqrt[\alpha]{e}-1)}{\sqrt[\alpha]{e}} \cdot n\,.$$

We see that

$$\frac{w(OPT)}{w(S)} \geq \frac{n\sqrt[\alpha]{e}}{(n+1+\alpha+\frac{\sqrt[\alpha]{e}}{\sqrt[\alpha]{e}-1})(\sqrt[\alpha]{e}-1)} \to \frac{\sqrt[\alpha]{e}}{\sqrt[\alpha]{e}-1} \quad \text{for } n \to \infty\,,$$

and the claim follows. □

Note that the lower bound construction assumes that players choose the most flexible jobs first, which seems reasonable. The bound also holds for the case with unit processing times, where we may assume that the players use optimal strategies [5], that is $\alpha = 1$. For that case, the result shows that the price of anarchy can be as high as $e/(e-1) \approx 1.58$.

6.2 Identical Machines: Upper Bound

To derive a matching upper bound for identical machines, when considering only subgame perfect equilibria, we use a proof idea from Bar-Noy et al. [3] in their analysis of k-GREEDY, but need a nontrivial generalization to make it work for the case where players control multiple machines.

Assume there are n players and m identical machines, and each player i controls m_i machines. Denote by S_i the set of jobs selected by player i, and $S = \bigcup_i S_i$ the total set of jobs scheduled in an α-approximate subgame perfect equilibrium. The following lemma lower bounds the total weight collected by i.

Lemma 1. *We have for all players i*

$$w(S_i) \geq \frac{m_i}{m\alpha} w\left(OPT\left(\mathcal{J} \setminus \bigcup\nolimits_{j<i} S_j\right)\right) .$$

where $OPT(W)$ denotes an optimal solution for any given set of jobs W and m machines.

Proof. Let $W := \mathcal{J} \setminus \bigcup_{j<i} S_j$. Let OPT_i denote the maximum weight set of jobs that can be scheduled by player i. Observe that $w(OPT_i) \geq (m_i/m)OPT(W)$. This follows because player i could potentially select the jobs scheduled on the m_i most valuable machines from $OPT(W)$, as all machines are identical. Now, by definition $w(S_i) \geq w(OPT_i)/\alpha \geq m_i w(OPT(W))/(m\alpha)$. Here, the first inequality holds because we assume an α-approximate Nash equilibrium, and in particular no player will choose a subset of jobs that is not disjoint from the subsets selected earlier. □

We are now ready to prove the following.

Theorem 3. *$PoA \leq \sqrt[\alpha]{e}/(\sqrt[\alpha]{e} - 1)$ for identical machines and α-approximate subgame perfect equilibria.*

Proof. Due to space limitations, we skip some technicalities of the proof, but give the main idea here. Let $\gamma := m\alpha$, and recall that $w(OPT) = w(OPT(\mathcal{J}))$ denotes the value of the optimal solution. We use Lemma 1, to get

$$w(S_i) \geq \frac{m_i}{\gamma} w\left(OPT\left(\mathcal{J} \setminus \bigcup\nolimits_{j<i} S_j\right)\right) \geq \frac{m_i}{\gamma}\left(w(OPT) - \sum\nolimits_{j<i} w(S_j)\right) ,$$

where the latter inequality holds because $w(OPT) - \sum_{j<i} w(S_j)$ represents the value of a feasible solution for the jobs $\mathcal{J} \setminus \bigcup_{j<i} S_j$. Add $\sum_{j=1}^{i-1} w(S_j)$ to both sides to get

$$\sum_{j=1}^{i} w(S_j) \geq \frac{m_i w(OPT)}{\gamma} + \frac{\gamma - m_i}{\gamma} \sum_{j=1}^{i-1} w(S_j) . \tag{3}$$

We prove by induction on i that

$$\sum_{j=1}^{i} w(S_j) \geq \frac{\gamma^{m'_i} - (\gamma-1)^{m'_i}}{\gamma^{m'_i}} w(OPT) ,$$

where $m'_i = \sum_{j=1}^{i} m_j$. When $i = 1$, we can show by induction on m_1 that $w(S_1) \geq \frac{\gamma^{m_1} - (\gamma-1)^{m_1}}{\gamma^{m_1}} w(OPT)$. Assume the claim holds for $i - 1$. Applying the induction hypothesis to (3) we get

$$\sum_{j=1}^{i} w(S_j) \geq \frac{m_i w(OPT)}{\gamma} + \frac{\gamma - m_i}{\gamma} \cdot \frac{\gamma^{m'_{i-1}} - (\gamma-1)^{m'_{i-1}}}{\gamma^{m'_{i-1}}} w(OPT) .$$

This can be used to prove the inductive claim, using basic but careful calculus. Hence we get for $i = n$ (see also [3, Thm 3.3])

$$w(S) = \sum_{j=1}^{n} w(S_j) \geq \frac{\gamma^m - (\gamma-1)^m}{\gamma^m} w(OPT) .$$

We get

$$\text{PoA} \leq \frac{\gamma^m}{\gamma^m - (\gamma-1)^m} = \frac{(m\alpha)^m}{(m\alpha)^m - (m\alpha-1)^m} \leq \frac{\sqrt[\alpha]{e}}{\sqrt[\alpha]{e} - 1} , \tag{4}$$

where the the last inequality follows because the right hand side is exactly the limit for $m \to \infty$, and the series $b_m = (m\alpha)^m/((m\alpha)^m - (m\alpha-1)^m)$ is monotone in m, with $b_1 = \alpha \leq \sqrt[\alpha]{e}/(\sqrt[\alpha]{e} - 1)$. □

Theorems 2 and 3 yield PoA $= \sqrt[\alpha]{e}/(\sqrt[\alpha]{e} - 1)$ when considering only α-approximate subgame perfect equilibria. Basic calculus shows that

$$\alpha + \frac{1}{2} \leq \sqrt[\alpha]{e}/(\sqrt[\alpha]{e} - 1) \leq \alpha + \frac{1}{e-1}$$

for $\alpha \geq 1$. Also, for $\alpha \to \infty$ this value approaches $\alpha + \frac{1}{2}$. Note that for $\alpha = 1, \text{PoA} = e/(e-1) \approx 1.58$.

Concluding Remarks

We briefly mention some more results for the the case $\alpha = 1$, that is, the case of Nash equilibrium allocations. Due to space limitations, any details are deferred to a full version of this paper.

First, we can show that the bound PoA $= \sqrt{e}/(\sqrt{e} - 1)$ for identical machines with unit processing times, unit weights and zero release dates holds without requiring that the Nash equilibria are subgame perfect. Next, we can generalize our results to a setting where bundle costs are not additive: When $w(J) \neq \sum_{j \in J} w_j$, but if we know that that $\sum_{j \in J} w_j/\beta \leq w(J) \leq \beta \sum_{j \in J} w_j$ for all $J \subseteq \mathcal{J}$ and for some parameter $\beta \geq 1$, then we can show that PoA $= \beta^4 + \beta^2$. (Note that $\beta^4 + \beta^2 = 2$ for $\beta = 1$.) Also, when we allow players to afterwards trade one single job, or even a set of jobs (for money), we can show that this does not improve the PoA substantially.

The most challenging next step from an application viewpoint is to consider online settings. When the goal is (constant) competitive ratios for online-time models, however, we will most probably need to revert to preemptive scheduling models.

Acknowledgements. Thanks to Johann Hurink for some helpful discussions on the context, and Rudolf Müller and Frits Spieksma for very helpful remarks.

References

1. Anshelevich, E., Dasgupta, A., Kleinberg, J., Tardos, E., Wexler, T., Roughgarden, T.: The price of stability for network design with fair cost allocation. In: Proc. 45th Symp. Foundations of Computer Science (FOCS), pp. 295–304. IEEE (2004)
2. Bakker, V., Bosman, M.G.C., Molderink, A., Hurink, J.L., Smit, G.J.M.: Demand side load management using a three step optimization methodology. In: 1st IEEE Int. Conf. on Smart Grid Communications, pp. 431–436. IEEE (2010)
3. Bar-Noy, A., Guha, S., Naor, J., Schieber, B.: Approximating the throughput of multiple machines in real-time scheduling. SIAM J. Comp. 31(2), 331–352 (2001)
4. Berman, P., Dasgupta, B.: Multi-phase Algorithms for Throughput Maximization for Real-Time Scheduling. J. Combinatorial Optimization 4(3), 307–323 (2000)
5. Brucker, P.: Scheduling Algorithms, 4th edn. Springer, Berlin (2004)
6. Buyya, R., Ranjan, R., Calheiros, R.N.: InterCloud: Utility-oriented federation of cloud computing environments for scaling of application services. In: Hsu, C.-H., Yang, L.T., Park, J.H., Yeo, S.-S. (eds.) ICA3PP 2010, Part I. LNCS, vol. 6081, pp. 13–31. Springer, Heidelberg (2010)
7. Daskalakis, C., Mehta, A., Papadimitriou, C.: Progress in Approximate Nash Equilibria. In: Proceedings 8th ACM EC, pp. 355–358. ACM (2007)
8. Garey, M.R., Johnson, D.S.: "Strong" NP-completeness results: motivation, examples, and implications. Journal of the ACM 25(3), 499–508 (1978)
9. Graham, R., Lawler, E., Lenstra, J., Rinnooy Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics 5(2), 287–326 (1979)
10. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103 (1972)
11. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
12. Kuhn, H.W.: Extensive Games and the Problem of Information. In: Contribution to the Theory of Games. Annals of Math. Studies, 28, vol. II, pp. 193–216 (1953)
13. Lawler, E.L., Moore, J.M.: A functional equation and its application to resource allocation and sequencing problems. Management Science 16, 77–84 (1969)
14. Lenstra, J.K., Rinnooy Kan, A.H.G., Brucker, P.: Complexity of Machine Scheduling Problems. Annals Disc. Math. 1, 343–362 (1977)
15. Molderink, A., Bakker, V., Bosman, M.G.C., Hurink, J.L., Smit, G.J.M.: Management and Control of Domestic Smart Grid Technology. IEEE Transactions on Smart Grid 1(2), 109–119 (2010)
16. Moore, J.M.: An n job, one machine sequencing algorithm for minimizing the number of late jobs. Management Science 15, 102–109 (1968)
17. Selten, R.: A simple model of imperfect competition, where 4 are few and 6 are many. International Journal of Game Theory 2, 141–201 (1973)
18. Skopalik, E., Vöcking, B.: Inapproximability of pure Nash equilibria. In: Proc. 40th Symp. on Theory of Computing (STOC), pp. 355–364. ACM (2008)
19. `http://www.tamyca.com` and `http://www.autonetzer.com`

New Results on Stabbing Segments with a Polygon

José Miguel Díaz-Báñez[1,⋆,§], Matias Korman[2,⋆⋆,§], Pablo Pérez-Lantero[3,⋆⋆⋆], Alexander Pilz[4,†], Carlos Seara[2,‡,§], and Rodrigo I. Silveira[2,‡,§]

[1] Dept. Matemática Aplicada II, Universidad de Sevilla, Spain
[2] Dept. Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Spain
[3] Escuela de Ingeniería Civil en Informática, Universidad de Valparaíso, Chile
[4] Institute for Software Technology, Graz University of Technology, Austria

Abstract. We consider a natural variation of the concept of *stabbing* a segment by a simple polygon: a segment is stabbed by a simple polygon $\mathcal{P}$ if at least one of its two endpoints is contained in $\mathcal{P}$. A segment set S is stabbed by $\mathcal{P}$ if every segment of S is stabbed by $\mathcal{P}$. We show that if S is a set of pairwise disjoint segments, the problem of computing the minimum perimeter polygon stabbing S can be solved in polynomial time. We also prove that for general segments the problem is NP-hard. Further, an adaptation of our polynomial-time algorithm solves an open problem posed by Löffler and van Kreveld [Algorithmica 56(2), 236–269 (2010)] about finding a maximum perimeter convex hull for a set of imprecise points modeled as line segments.

1 Introduction

Let S be a set of n straight line segments (*segments* for short) in the plane. The problem of stabbing S with different types of stabbers (in the computer science literature) or transversals (in the mathematics literature) has been widely studied during the last two decades.

Rappaport [14] considered the case in which the stabber is a simple polygon. Specifically, he studied the following problem: a simple polygon $\mathcal{P}$ is a *polygon*

[⋆] Partially supported by project MEC MTM2009-08652 and Excellence Project P09-TIC-4840 (Junta de Andalucía).

[⋆⋆] With the support of the Secretary for Universities and Research of the Ministry of Economy and Knowledge of the Government of Catalonia and the European Union.

[⋆⋆⋆] Partially supported by grant CONICYT, FONDECYT/Iniciación 11110069 and MEC MTM2009-08652.

[†] Recipient of a DOC-fellowship of the Austrian Academy of Sciences at the Institute for Software Technology, Graz University of Technology, Austria.

[‡] Partially supported by projects MINECO MTM2012-30951 and Gen. Cat. DGR2009GR1040. R.S. was also funded by FP7 Marie Curie Actions Individual Fellowship PIEF-GA-2009-251235.

[§] Partially supported by the ESF EUROCORES program EuroGIGA -ComPoSe IP04-MICINN Project EUI-EURC-2011-4306.

transversal of S, if we have $\mathcal{P} \cap s \neq \emptyset$ for all $s \in S$; that is, every segment in S has at least one point in $\mathcal{P}$. A simple polygon $\mathcal{P}$ is a *minimum polygon transversal* of S if $\mathcal{P}$ is a polygon transversal of S and all other transversal polygons have equal or larger perimeter. Rappaport observed that such a polygon always exists, is convex, and may not be unique. He gave an $O(3^m n + n \log n)$ time algorithm for computing one, where m is the number of different segment directions. Several approximation algorithms are known [6,8], but determining if the general problem can be solved in polynomial time is still an intriguing open problem.

Arkin et al. [2] considered a similar problem: S is *stabbable* if there exists a convex polygon whose boundary $\mathcal{C}$ intersects every segment in S; the closed convex chain $\mathcal{C}$ is then called a (convex) *transversal* or *stabber* of S. Note that in this variation there is not always a solution. Arkin et al. [2] proved that deciding whether S is stabbable is NP-hard.

In this paper we also consider the problem of stabbing the set S by a simple polygon, but with a different criterion that is between the two criteria above. More concretely, we use the following definition:

Definition 1. *A segment $s \in S$ is stabbed by a simple polygon $\mathcal{P}$ if* at least *one of the two endpoints of s is contained in $\mathcal{P}$. The set S is stabbed by $\mathcal{P}$ if every segment of S is stabbed by $\mathcal{P}$.*

With this definition we study the Minimum Perimeter Stabbing Polygon (MPSP) problem, defined as finding a simple polygon $\mathcal{P}$ of minimum perimeter that stabs a given set S of segments. The MPSP problem is radically different from the two problems above, those studied by Rappaport [14] and Arkin et al. [2], because for the MPSP only the endpoints of the segments play a role in the solution. Indeed, an alternative way to describe the input to the MPSP problem is by saying that the input are *pairs of points* instead of segments. However, as we will show in this paper, the segments play an important role in establishing the difficulty of the problem, hence we stick to the original definition.

Moreover, the difference with the problem of Rappaport [14] is that in his definition $\mathcal{P}$ can have both endpoints of a segment of $s \in S$ not in $\mathcal{P}$ (provided that the interior of s is stabbed by $\mathcal{P}$), whereas we force one of the endpoints to be in $\mathcal{P}$. One of the common properties of both problems is that the optimal solution is a convex polygon and that it always exists (the convex hull of S is always a stabbing polygon).

On the other hand, a difference with the definition used by Arkin et al. is that in the MPSP problem a segment of S can be fully contained in $\mathcal{P}$, with both endpoints in the interior of $\mathcal{P}$, while this is not allowed in the problem studied by Arkin et al. Therefore, we can say that our problem is *between* the two mentioned ones.

Related Work. Prior to the paper by Rappaport [14], Meijer and Rappaport [12] solved the same problem for a set of n parallel segments in optimal $\Theta(n \log n)$ time. Mukhopadhyay et al. [13] considered a similar problem in which the segments are all vertical, and proposed an $O(n \log n)$ time algorithm to find a minimum-area convex polygon transversal of S. For parallel segments, Goodrich and Snoeyink [7] gave an $O(n \log n)$ time algorithm that decides whether a convex transversal exists.

Several similar problems have been considered in the context of *data imprecision* by Löffler and van Kreveld [10,11]. Their input is a set of imprecise points, where each point is specified by a region in which the point may lie. The output is the smallest and the largest possible convex hulls, measured by perimeter and by area. Among the results obtained in [10], we cite those where regions are segments. For maximum-area convex hulls, the problem can be solved in $O(n^3)$ time if the segments are parallel, or when they are pairwise disjoint with endpoints in convex position. The problem is NP-hard for general segments.

The minimum-perimeter and minimum-area convex hulls problems for parallel segments coincide with the problems studied by Meijer and Rappaport [12] and Mukhopadhyay et al. [13], respectively. Notice also that the setting we consider is in fact a constrained version of the problems studied by Löffler and van Kreveld [10], in which each imprecise point is specified by a pair of points.

Pairs of points are also the input to the problems studied by Arkin et al. [1], who studied the 1-center and 2-center problems for pairs of points. In the former problem, the goal is to find a disk of smallest radius containing at least one point from each pair. The latter one aims at finding two disks of smallest size such that each pair has one point in each disk. Arkin et al. [1] presented algorithms for these problems that run in $O(n^2 \text{polylog } n)$ and $O(n^3 \log^2 n)$ time, respectively.

In a more general setting, Daescu et al. [4] studied the complexity of the problem of given a k-colored point set, finding a convex polygon of minimum perimeter containing at least one point from each color. Note that the MPSP problem is the special case in which $2n$ points are colored with n colors and each color is used twice. They proved that their problem is NP-hard if k is part of the input of the problem.

Our Results. We show in Section 2 that if S is a set of pairwise disjoint segments, the MPSP problem for S can be solved in polynomial time. We then show how the algorithm can be adapted to solve the following maximization problem: Select exactly one point on each segment in S such that the perimeter (or area) of the convex hull of the selected points is maximized. This problem was stated as open [10], and is also the solution to the maximization variant of the transversal problem [10]. In Section 3 we show that for general segments the MPSP problem is NP-hard. We complement the NP-hardness by showing that the MPSP problem is Fixed Parameter Tractable (FPT).

Note throughout the paper that optimization on the perimeter requires comparing sums of radicals (specifically, the sum of Euclidean distances). It is not known whether this problem is in NP [3], and therefore the NP-hardness result does not imply NP-completeness for the decision version of the problem. For the same reason, we assume the real RAM as the underlying computational model in our algorithms. Since our algorithms are combinatorial and only the cost function depends on the geometry of the problem instance, the methods in Section 2 are also applicable for optimizing the area (which is in NP).

Due to lack of space, several proofs have been deferred to the full version [5].

2 Solving the Problem for Pairwise Disjoint Segments

In this section we show that if the segments in S are pairwise disjoint, then the MPSP problem can be solved in polynomial time. Given any two points p and q in the plane, let pq denote the segment joining p and q. For any simple polygon $\mathcal{P}$ let $\partial\mathcal{P}$ denote the boundary of $\mathcal{P}$. Consider all possible bitangents of S, i.e., let B be the set of all segments not contained in S spanned by two endpoints of segments in S. Note that the elements of B might cross each other and might also cross the segments in S. A polygon C^* with minimum perimeter that contains at least one endpoint of every segment of S is spanned by endpoints of segments in S, and its edges are elements of B.

Arkin et al. [2] describe a dynamic programming approach to decide whether a set of pairwise disjoint segments admits a convex transversal (the vertices of the transversing polygon are restricted to a given set of candidate points). They use constant-size polygonal chains that separate subproblems and are not crossed by segments; therefore the subproblems are independent. We adapt their approach to produce an algorithm for the MPSP problem. The main difference (apart from the fact that no candidate points are needed) is that segments actually *can* cross the separating chains. However, we show below that they can be handled in a way that leads to polynomial running time. Afterwards, we discuss how to adapt this approach for the maximization variation.

Triangulating a Combination of Segments and a Polygon. The following way of triangulating a combination of segments and a polygon is crucial for the algorithm, and motivates the structure of the subproblems used in the dynamic programming algorithm.

Let $\mathcal{Q}$ be a simple polygon and let S_c be a set of pairwise disjoint segments of which each crosses $\partial\mathcal{Q}$ exactly once. Note that throughout this section we distinguish between a segment *intersecting* (having a point in common) and *crossing* (having an *interior* point in common with) another segment or set. Let X be the interior of $\mathcal{Q}$ and let X' denote the set we get after removing the 1-dimensional domains of S_c from X, i.e., $X' = X \setminus \bigcup_{s \in S_c} s$. Then X' is an open region whose closure is $\mathcal{Q}$. Note that the vertices of X' are the union of: (i) the vertices of $\mathcal{Q}$, (ii) the endpoints of edges in S_c that are in the interior of $\mathcal{Q}$, and (iii) the points where elements of S_c cross $\partial\mathcal{Q}$. Further, note that X' might not be connected if there is a segment of S_c that has one endpoint on $\partial\mathcal{Q}$ and the other one outside $\mathcal{Q}$ (e.g., the longest segment in Fig. 1, left).

We now triangulate X' (i.e., partition it into triangles that are spanned only by vertices of X', see Fig. 1). The triangulation T of X' behaves like the triangulation of a collection of simple polygons (imagine the 1-dimensional parts not in X' where the segments of S_c enter $\mathcal{Q}$, i.e., $X \setminus X'$, to be slightly "split", as in Fig. 1, center). Note that the vertices of T are exactly the vertices of X'. Each edge in T that is not part of $\partial\mathcal{Q}$ or part of a segment in S_c partitions X' into two sets (note that each set need not be connected). We call such edges *chords* (gray edges in Fig. 1, right). Chords are the equivalent of *diagonals* of simple polygons (interior edges that subdivide the polygon into two smaller polygons). Further,

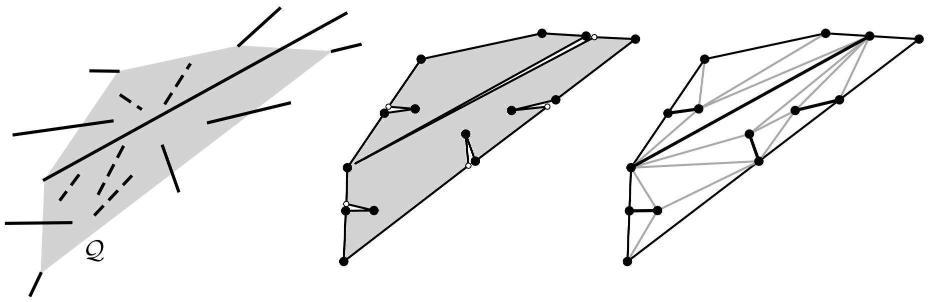


Fig. 1. Left: an optimal polygon $\mathcal{Q}$, only the solid edges are in S_c. Center: schematic view of X' as a collection of simple polygons. Right: a triangulation of X', gray edges are chords. The segments fully contained in the polygon (shown dashed) are ignored by the triangulation.

X' might also be separated by an edge that is part of a segment in S_c (like the longest edge in Fig. 1). We call such a segment a *separating segment*. Keep in mind that there are chords that have one or both of their endpoints not on the endpoint of a segment or at a vertex of $\mathcal{Q}$, but at the crossing of a segment with $\partial\mathcal{Q}$. In any case, a chord or a separating segment defines a polygonal path from one point on an edge of $\mathcal{Q}$ to another point on an edge of $\mathcal{Q}$. Following [2], we will use these polygonal paths of at most three edges, called *bridges*, to define our subproblems to obtain a solution when taking the MPSP C^* as $\mathcal{Q}$. One may think of the approach being similar to the classic dynamic programming algorithm for minimum weight triangulations of simple polygons (see, e.g., [9]), but with a major difference: we do not know the boundary of the triangulated region beforehand.

Subproblems. Every subproblem is defined by an ordered pair (a, b) of directed bitangents of B and a polygonal chain β of at most three edges, the *bridge*, which connects a and b. When evaluating a subproblem (a, b, β), we assume that a and b are edges of C^* and that C^* equals $\mathcal{Q}$ in the discussion above (for some choice of S_c to be defined later). Therefore, the bridge β is part of a triangulation of X' and separates X'; β is either a part of a separating segment or consists of a chord (called the *chord of* β) and at most two parts of segments of S_c. See Fig. 2 for examples of bridges. Note that a bridge might have a chord that is not a bitangent of B (like the second from the left in Fig. 2). Further, note that a bridge can only be crossed by a segment through the chord, since the segments are pairwise disjoint by definition.

Let the directed bitangents be $a = a_1a_2$ and $b = b_1b_2$. Given a directed bitangent $a = a_1a_2$ we write $\overline{a}$ for the directed bitangent a_2a_1. W.l.o.g. let a_1 and b_1 be on the x-axis and a_2 and b_2 be above it. Also, let b be to the left of the directed line through a_1 and a_2. See Fig. 3 for an illustration.

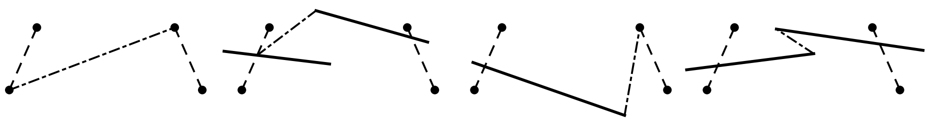


Fig. 2. Examples of bridges. The two bitangents defining the subproblem are shown dashed, chords are dash-dotted, and segments from S_c are shown solid.

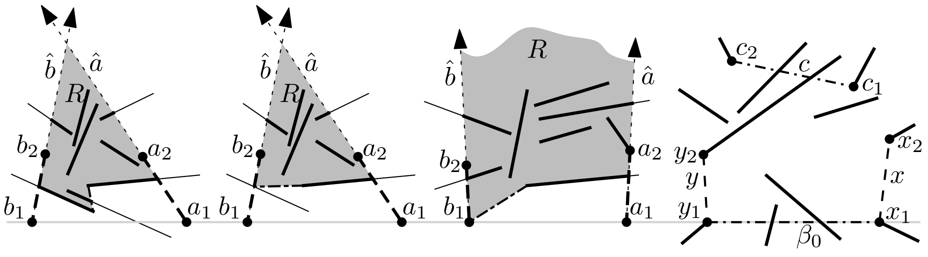


Fig. 3. Examples of subproblems. Rightmost: example for the initial pair.

Solution of a Subproblem. We define the solution of a subproblem as follows. Let $C^*_{a,b,\beta}$ be a polygon of minimum perimeter that: (i) contains a and b as two of its boundary edges, (ii) contains at least one endpoint of each segment in S, and (iii) contains both endpoints of every segment of S that properly crosses the chord of β. The importance of the third condition will become clear later.

Let $C_{a,b,\beta}$ be the polygonal chain on $\partial C^*_{a,b,\beta}$ starting at a_1, counterclockwise traversing $\partial C^*_{a,b,\beta}$ and ending at b_1. Note that $C_{a,b,\beta}$ is an open polygonal chain, as opposed to $C^*_{a,b,\beta}$, which is a simple polygon.

The solution of a subproblem (a, b, β) is $C_{a,b,\beta}$, and its cost is the length of that chain. The base case occurs when $a_2 = b_2$, and has cost equal to the sum of the lengths of a and b. Note throughout the construction that this is the only way a and b can intersect. In general, a and b form a quadrilateral $a_2a_1b_1b_2$. If the quadrilateral is not convex, we discard the subproblem (i.e., we assign it a cost of $+\infty$). The general case where it is convex is discussed next.

Outline of the Algorithm. From now on we assume that a and b define a convex quadrilateral. The outline of the algorithm is as follows. We guess a pair $x, y \in B$ such that $y_2y_1x_1x_2$ are four consecutive vertices of C^*. Hence, after $O(|S|^4)$ guesses we have found x and y such that $\partial C^* = C_{x,y,\beta_0} \cup y_1x_1$ with $\beta_0 = x_1y_1$. Suppose we are given the solution $\mathcal{Q} = C^*$. Let X' be defined as above, and let S_c be the set of segments in S that cross C_{x,y,β_0} (which does not include the ones that cross β_0). Let Δ_0 be the triangle of a triangulation T of X' that has $\beta_0 = y_1x_1$ as one side. The subproblem (x, y, β_0) will be solved by guessing the third endpoint of Δ_0 and the edge c of C_{x,y,β_0} that is incident to Δ_0 or that is crossed by a segment whose endpoint is incident to Δ_0. In the most general case, this gives two new subproblems $(x, \overline{c}, \beta_1)$ and (c, y, β_2), where each of β_1 and β_2 contains one side of Δ_0 that is not part of β_0 (we will consider the other cases in detail below). See Fig. 3, right.

Let $\hat{a}$ be the ray through a_2 starting at a_1. Let $\hat{b}$ be defined analogously. For every subproblem (a, b, β), only a part of the elements of S is relevant. Consider the (possibly unbounded) maximal region to the left of a and to the right of b (recall that a and b are directed). The bridge β disconnects that region into two parts. The *subproblem region* $R_{a,b,\beta}$ is the part "above" β (i.e., the part adjacent to $\hat{a} \setminus a$ and $\hat{b} \setminus b$; the bridge might not be x-monotone).

The subproblem region is marked gray in Fig. 3. Only the segments that have at least one endpoint in $R_{a,b,\beta}$ are relevant for finding $C_{a,b,\beta}$. We distinguish between three different types of such segments: (1) Segments that are entirely inside $R_{a,b,\beta}$ are *complete.* (2) Segments that share more than one point with $R_{a,b,\beta}$ but are not complete are *cut.* (3) A segment with infinitely many points on the bridge is neither cut nor complete. We say that a point is *inside* $C_{a,b,\beta}$ when it is contained in the closure of the region bounded by $C_{a,b,\beta}$ and β.

If there is a segment that is entirely to the right of a or to the left of b, then the choice of a and b cannot give a solution and such a subproblem is assigned $+\infty$ as cost. We also do this if a segment intersected by $\hat{a}$ or $\hat{b}$ does not have an endpoint inside the subproblem region.

Note that if a segment in a valid subproblem intersects $\hat{a}$ or $\hat{b}$, then we know which of its endpoints must be inside $C_{a,b,\beta}$, while we do not know that for the cut segments that intersect the chord of the bridge. However, we will choose our subproblems in a way such that all endpoints of cut segments in the subproblem region will be inside $C_{a,b,\beta}$; the reason for that will become clear in the proof of Lemma 3, but the reader should keep this in mind as an essential part of the method. For complete segments, we need to decide which endpoint to select.

Lemma 1. *Given a subproblem instance (a, b, β), let t be the chord of β, or its only edge if β is a single edge (which may be a chord itself, or part of a separating segment). Let X be the region bounded by $C_{a,b,\beta} \cup \beta$, and let $X' = X \setminus \bigcup_{s \in S_c} s$, for S_c the set of segments of S that are crossed by chain $C_{a,b,\beta}$. Then either t is an edge of $C_{a,b,\beta}$, or there exists a triangle Δ such that:*

1. *The interior of Δ is completely contained in X'.*
2. *The edge t is an edge of Δ.*
3. *The apex of Δ (i.e., the vertex not on t) is either (i) an endpoint of a segment in S_c inside X, (ii) an endpoint of a segment in S that is a vertex of $C_{a,b,\beta}$, or (iii) an intersection point between a segment in S_c and $C_{a,b,\beta}$.*

Proof. Arbitrarily triangulate X'. If t is not on the boundary, then the triangle Δ incident to t inside the subproblem region fulfills the properties. See Fig. 4. □

Lemma 2. *Let Δ be the triangle of Lemma 1. Any segment of S that has a non-empty intersection with the interior of Δ either has both its endpoints inside $C_{a,b,\beta}$ or crosses t; in the latter case the endpoint that is inside $R_{a,b,\beta}$ is also inside $C_{a,b,\beta}$.*

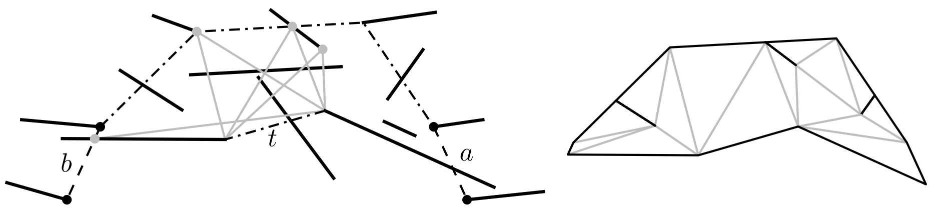


Fig. 4. Illustration of Lemma 1. Left: four possibilities for Δ shown in gray. $C_{a,b,\beta}$ is dash-dotted, with the defining bitangents dashed. Right: a triangulation of X'.

Proof. This follows from the properties of Δ in Lemma 1: A segment intersecting the interior of Δ is not part of S_c but has a non-empty intersection with X. Therefore, either both of its endpoints are inside $C_{a,b,\beta}$, or it enters X via t and therefore has its relevant endpoint inside $C_{a,b,\beta}$ by definition. See Fig. 4. □

Getting Smaller Subproblems. Let A be the set of points that are either endpoints of S or crossing points of a segment and a bitangent (recall that no segment of S is an element of B). Hence, A contains all the points that are possible apices for a triangle Δ of Lemma 1. Note that one may construct subproblems where every possible apex of Δ is an endpoint of a segment in S_c, as well as subproblems where every possible apex is on a point where a segment crosses $C_{a,b,\beta}$. Further, note that $|A| \in O(|S|^3)$ since $|B| = 4\binom{|S|}{2}$.

Consider again the subproblem (a, b, β). As in Lemma 1, let t be the chord of β if a chord exists, or let t otherwise be the only edge of β. Let a_β be the intersection point of a with the bridge β; b_β is defined analogously. For each subproblem (a, b, β) that is not a base case (i.e., $a_2 \neq b_2$), one of the following cases applies, allowing to get one or two smaller subproblems. During the execution of the algorithm we will consider both cases.

Case 1: t is an Edge of the Solution, i.e., an Edge of $C_{a,b,\beta}$. This happens when t is a chord that does not intersect the interior of the quadrilateral defined by a and b. This case is only valid if no segment crosses t, as we require all the endpoints in $R_{a,b,\beta}$ of segments crossing t to be inside $C_{a,b,\beta}$. In that case we get at most two new subproblems $(a, \overline{t}, \beta_1)$ and (t, b, β_2), where β_1 is the edge $a_\beta t_1$ and β_2 is the edge $t_2 b_\beta$. However, note that one of $(a, \overline{t})$ or (t, b) (or both) might intersect at a_2 or b_2, respectively, and therefore form a base case.

Case 2: t is Not an Edge of the Solution. Then there is a triangle adjacent to t as in Lemma 1. We will guess the apex of the triangle. For every point d in $A \cap R_{a,b,\beta}$ consider the triangle Δ_d that d forms with t. We only consider d if Δ_d is completely inside $R_{a,b,\beta}$, and where the interior of Δ_d does not intersect any segment that intersects a or b. It follows from Lemma 1 that one of the triangles tested leads to a subdivision of the optimal solution. We get the following two subcases, see Fig. 5.

Case 2.1: d is a Point Where a Bitangent and a Segment Cross. Let c be the bitangent that contains d. If c equals a or b, then we get one new subproblem (a, b, β'), with β' containing a side of Δ_d as a chord (Fig. 5a). Otherwise, we get

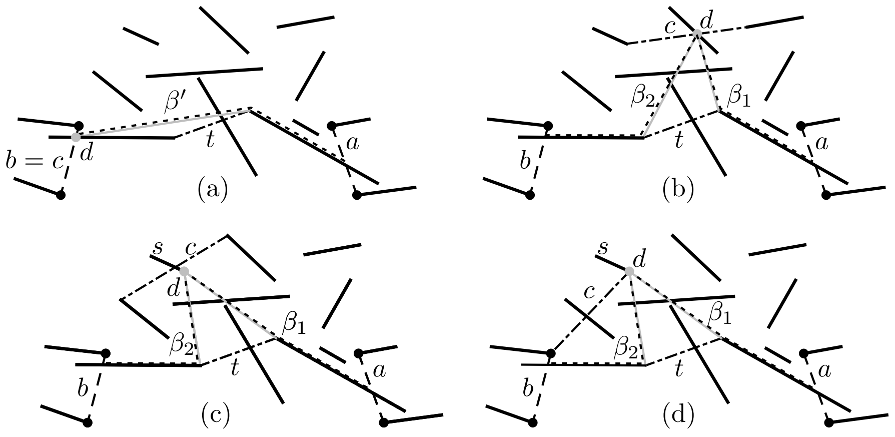


Fig. 5. Case 2. The new bridges are dotted. (a)-(b) Case 2.1. (c)-(d) Case 2.2.

two new subproblems, $(a, \overline{c}, \beta_1)$ and (c, b, β_2), where β_1 and β_2 both contain a side of Δ_d (Fig. 5b).

Case 2.2: d is an Endpoint of a Segment. Let s be the segment that has d as its endpoint. Choose a point x where s intersects some bitangent c. Then, for every possible choice of x (which implies the choices of c), we get two new subproblems $(a, \overline{c}, \beta_1)$ and (c, b, β_2), as in the previous case; note that for both new bridges, $x = d$ is possible. The degenerate case where c equals a or b can be handled as in the previous case. See Fig. 5c-d.

Lemma 3. *Given any valid subproblem (a, b, β), there is a pair of subproblems among the ones above such that the union of their solutions is equal to $C_{a,b,\beta}$.*

Proof. Consider the edge t of Lemma 1. If t is a chord and part of $C_{a,b,\beta}$, then it will be considered in Case 1. Otherwise, consider the triangle Δ inside $C_{a,b,\beta}$. All segments that are intersected by the interior of Δ are either completely contained in $C_{a,b,\beta}$ or enter through t (if it is a chord) and therefore have their relevant endpoint inside $C_{a,b,\beta}$ (cf. Lemma 2). Hence, when the choice of Δ_d coincides with Δ, the two subproblems can be combined into $C_{a,b,\beta}$; the only segments that are part of both subproblems intersect the interior of Δ, and we know that both endpoints will have to be inside the chain that results from the combination of the solutions of the subproblems. Since all possibilities of Δ_d are checked, the subproblem combination of minimum cost is guaranteed to be $C_{a,b,\beta}$. □

This last lemma now implies that we actually find the optimal solution. Note that it is easy to construct a pair of bitangents and a bridge (a, b, β) that is part of the optimal solution but for which $C_{a,b,\beta}$ is not part of C^*. However, as mentioned in the outline of the algorithm, we choose the initial problem (x, y, β_0) in a way that $\partial C^* = C_{x,y,\beta_0} \cup \beta_0$. All segments crossing $\beta_0 = x_1 y_1$ need to have their endpoint above β_0 inside the solution, and the algorithm actually produces a triangulation of X' when taking C^* as $\mathcal{Q}$ and S_c being the segments that cross ∂C^* but do not cross β_0.

Recall that we initialize the algorithm using a brute-force approach: that is, we consider all $O(|S|^4)$ possible choices for two defining bitangents and a bridge a_1b_1. Every subproblem contains less edges of the complete graph on all endpoints of S, and for every subproblem we need polynomial time. The number of subproblems can be bounded by the choices for c and d. Therefore, dynamic programming can be applied to obtain a polynomial-time algorithm.[1]

Theorem 1. *Given a set S of pairwise disjoint segments, a Minimum Perimeter Stabbing Polygon (MPSP)—i.e. a minimum perimeter polygon containing at least one endpoint of each segment in S—can be computed in polynomial time.*

Maximization for Pairwise Disjoint Segments. Our previous algorithm relies on the fact that the result has minimum perimeter: this automatically prevents two endpoints of the same segment from being vertices of the resulting polygon. However, making the algorithm slightly more sophisticated, we can solve in polynomial time a maximization version of the problem, stated open by Löffler and van Kreveld [10]: *select exactly one point on each segment in S such that the perimeter (or area) of the convex hull of the selected points is maximized.* This result is based on the fact that for the maximum area or perimeter transversal, one needs to consider only the endpoints of the segments [10, Lemmata 1 and 8]. The proof can be found in the full version [5].

Theorem 2. *There exists a polynomial-time algorithm that selects exactly one point on each segment in S such that the perimeter (or area) of the convex hull of the selected points is maximized over all possible selections.*

3 Hardness of the General Version

In this section we prove that the MPSP problem is NP-hard by reducing 3-SAT to it. Our reduction is similar to the ones used in [2,4,10].

Theorem 3. *The MPSP problem is NP-hard.*

Proof (Sketch). We only present here the main construction, the rest of the proof is given in the full version [5]. Let a 3-SAT instance consist of n variables $x_1, \dots, x_n$ and m clauses $C_1, \dots, C_m$. We reduce this instance to the following one of the MPSP problem. We draw a circle and place variable gadgets in the left semicircle, clause gadgets in the right semicircle, and segment connectors joining variable gadgets with clause gadgets. See Fig. 6a.

For each variable x_i, $i \in [1..n]$, we put points T_i and F_i on the circle and place three segments: segment T_iF_i, and two zero-length segments a_i and b_i, so that

[1] A straightforward analysis of the running time results in $O(|S|^9)$, which probably can be improved. In any case, it is worth stressing that our main contribution is that the problem can be solved in polynomial time, more than the running time itself.

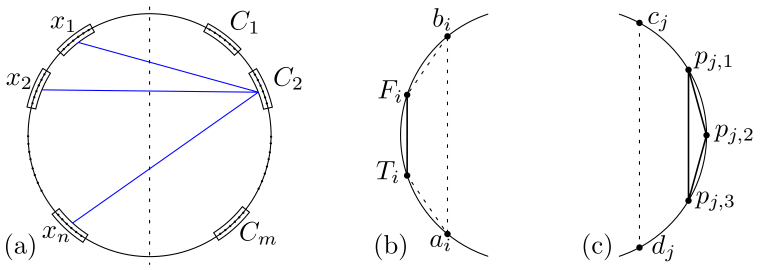


Fig. 6. (a) Overview of the reduction from 3-SAT. Variable gadgets (b) are to the left and clause gadgets (c) to the right.

T_iF_i is parallel to the line containing both a_i and b_i. Refer to Fig. 6b. Furthermore, trapezoids with vertices a_i, T_i, F_i, b_i, for all $i \in [1..n]$, are congruent. Let $P_v := |a_iT_i| + |T_ib_i| = |a_iF_i| + |F_ib_i|$ and $P'_v := |a_iT_i| + |T_iF_i| + |F_ib_i|$ (where $|pq|$ denotes the length of the segment pq).

For each clause C_j, $j \in [1..m]$, we first place two zero-length segments c_j and d_j. We select the three points $p_{j,1}$, $p_{j,2}$, and $p_{j,3}$, dividing evenly the smallest arc of the circle joining c_j and d_j into four arcs, and then we place three other segments: $p_{j,1}p_{j,2}$, $p_{j,2}p_{j,3}$, and $p_{j,3}p_{j,1}$. See Fig. 6c. The convex pentagons with vertices $d_j, c_j, p_{j,1}, p_{j,2}, p_{j,3}$, for all $j \in [1..m]$, are congruent. Let $P_c := |c_jp_{j,1}| + |p_{j,1}p_{j,2}| + |p_{j,2}d_j| = |c_jp_{j,1}| + |p_{j,1}p_{j,3}| + |p_{j,3}d_j| = |c_jp_{j,2}| + |p_{j,2}p_{j,3}| + |p_{j,3}d_j|$ and $P'_c := |c_jp_{j,1}| + |p_{j,1}p_{j,2}| + |p_{j,2}p_{j,3}| + |p_{j,3}d_j|$. We further ensure that $m(P'_c - P_c) < P'_v - P_v$. This condition will be necessary in the problem reduction.

For each clause C_j, $j \in [1..m]$, we add segments called *connectors* as follows. Let x_i be the variable involved in the first literal of C_j. If x_i appears in positive form then we add the segment $T_ip_{j,1}$. Otherwise the segment $F_ip_{j,1}$ is added. We proceed analogously with the variable in the second literal and point $p_{j,2}$, and with the variable in the third literal and point $p_{j,3}$.

Consider the set of segments added at variable gadgets, clause gadgets, and connectors as an instance of the MPSP problem. Observe that any optimal polygon $\mathcal{P}_{opt}$ for this instance satisfies the following conditions:

(a) $\mathcal{P}_{opt}$ contains as vertices points a_i and b_i for all variables x_i, $i \in [1..n]$, and points c_j and d_j for all clauses C_j, $j \in [1..m]$.
(b) For each variable x_i, $i \in [1..n]$, $\mathcal{P}_{opt}$ contains exactly one of T_i and F_i as vertex between a_i and b_i.
(c) In the clause gadget of each clause C_j, $j \in [1..m]$, if the selected endpoint of at least one connector is not in the gadget as a vertex of $\mathcal{P}_{opt}$, then exactly two points among $p_{j,1}$, $p_{j,2}$, and $p_{j,3}$ are vertices of $\mathcal{P}_{opt}$. Otherwise, all three are vertices of $\mathcal{P}_{opt}$.

In the full version [5], we show that any polygon satisfying conditions (a)-(c) induces a valid variable assignment that satisfies the formula if and only if the polygon has minimum perimeter. Further, we give an exact construction for the segment endpoints of the gadgets. □

Observe that the same reduction with minor modifications applies for the case of minimizing the area of the output polygon. Moreover, our proof shows that the problem remains NP-hard even if the endpoints of all the segments are in convex position. On the other hand, the $\sqrt{2}$-approximation algorithm of Daescu et al. [4] gives the same approximation ratio for our MPSP problem.

It is worth mentioning that the MPSP problem is FPT on the number k of segments that intersect other segments. Namely, let $S' \subseteq S$ be the set of segments of S that do not intersect any segment of S. Consider the 2^k instances of the MPSP problem such that each consists of the elements of S' joint with exactly one endpoint (i.e., a segment of length zero) of each element of $S \setminus S'$. All these instances can be solved in $O(2^k P(n))$ time, for the polynomial time $P(n)$ of Theorem 1, since each instance consists of pairwise disjoint segments. The optimal solution for S is among the $O(2^k)$ solutions found for those instances.

References

1. Arkin, E.M., Díaz-Báñez, J.M., Hurtado, F., Kumar, P., Mitchell, J.S.B., Palop, B., Pérez-Lantero, P., Saumell, M., Silveira, R.I.: Bichromatic 2-center of pairs of points. In: Fernández-Baca, D. (ed.) LATIN 2012. LNCS, vol. 7256, pp. 25–36. Springer, Heidelberg (2012)
2. Arkin, E.M., Dieckmann, C., Knauer, C., Mitchell, J.S., Polishchuk, V., Schlipf, L., Yang, S.: Convex transversals. Comput. Geom. (2012) (article in press)
3. Blömer, J.: Computing sums of radicals in polynomial time. In: FOCS 1991, pp. 670–677. IEEE Computer Society (1991)
4. Daescu, O., Ju, W., Luo, J.: NP-completeness of spreading colored points. In: Wu, W., Daescu, O. (eds.) COCOA 2010, Part I. LNCS, vol. 6508, pp. 41–50. Springer, Heidelberg (2010)
5. Díaz-Báñez, J.M., Korman, M., Pérez-Lantero, P., Pilz, A., Seara, C., Silveira, R.I.: New results on stabbing segments with a polygon. CoRR abs/1211.1490 (2012)
6. Dumitrescu, A., Jiang, M.: Minimum-perimeter intersecting polygons. Algorithmica 63(3), 602–615 (2012)
7. Goodrich, M.T., Snoeyink, J.: Stabbing parallel segments with a convex polygon. Computer Vision, Graphics, and Image Processing 49(2), 152–170 (1990)
8. Hassanzadeh, F., Rappaport, D.: Approximation algorithms for finding a minimum perimeter polygon intersecting a set of line segments. In: Dehne, F., Gavrilova, M., Sack, J.-R., Tóth, C.D. (eds.) WADS 2009. LNCS, vol. 5664, pp. 363–374. Springer, Heidelberg (2009)
9. Klincsek, G.T.: Minimal triangulations of polygonal domains. Ann. Discrete Math. 9, 121–123 (1980)
10. Löffler, M., van Kreveld, M.J.: Largest and smallest convex hulls for imprecise points. Algorithmica 56(2), 235–269 (2010)
11. Löffler, M., van Kreveld, M.J.: Largest bounding box, smallest diameter, and related problems on imprecise points. Comput. Geom. 43(4), 419–433 (2010)
12. Meijer, H., Rappaport, D.: Minimum polygon covers of parallel line segments. In: CCCG 1990, pp. 324–327 (1990)
13. Mukhopadhyay, A., Kumar, C., Greene, E., Bhattacharya, B.K.: On intersecting a set of parallel line segments with a convex polygon of minimum area. Inf. Proc. Lett. 105(2), 58–64 (2008)
14. Rappaport, D.: Minimum polygon transversals of line segments. Int. J. Comput. Geometry Appl. 5(3), 243–256 (1995)

Improving the H_k-Bound on the Price of Stability in Undirected Shapley Network Design Games

Yann Disser[1], Andreas Emil Feldmann[2], Max Klimm[1], and Matúš Mihalák[3]

[1] Institut für Mathematik, Technische Universität Berlin, Germany
{disser,klimm}@math.tu-berlin.de

[2] Combinatorics and Optimization Department, University of Waterloo, Canada
andreas.feldmann@uwaterloo.ca

[3] Institute of Theoretical Computer Science, ETH Zurich, Switzerland
matus.mihalak@inf.ethz.ch

Abstract. In this paper we show that the price of stability of Shapley network design games on undirected graphs with k players is at most $\frac{k^3(k+1)/2-k^2}{1+k^3(k+1)/2-k^2}H_k = \big(1-\Theta(1/k^4)\big)H_k$, where H_k denotes the k-th harmonic number. This improves on the known upper bound of H_k, which is also valid for directed graphs but for these, in contrast, is tight. Hence, we give the first non-trivial upper bound on the price of stability for undirected Shapley network design games that is valid for an arbitrary number of players. Our bound is proved by analyzing the price of stability restricted to Nash equilibria that minimize the potential function of the game. We also present a game with $k = 3$ players in which such a restricted price of stability is 1.634. This shows that the analysis of Bilò and Bove (Journal of Interconnection Networks, Volume 12, 2011) is tight. In addition, we give an example for three players that improves the lower bound on the (unrestricted) price of stability to 1.571.

Keywords: undirected Shapley network design game, price of stability, potential-optimal price of stability, potential-optimal price of anarchy.

1 Introduction

Infrastructure networks are the lifelines of our civilization. Through generations a tremendous effort has been undertaken to cover the earth's surface with irrigation canal systems, sewage lines, road networks, railways, and – more recently – data networks. Some of these infrastructures are initiated and planned by a central authority that designs the network and decides on its topology and dimension. Many networks, however, arise as an outcome of actions of selfish individuals who are motivated by their own connectivity requirements rather than by optimizing the overall network design. A prominent example of the latter phenomenon is the rise of the Internet. In order to quantify the efficiency of networks, it is crucial to understand the processes that govern their formation. Anshelevich et al. [1] proposed a particularly elegant model for such processes, which is now known as the Shapley network design game or the network design game with fair cost allocation (for an overview of other models for network formation, see [2]).

The Shapley network design game is played by k players $1, 2, \ldots, k$ on a graph $G = (V, E)$ with positive edge-costs $c_e \in \mathbb{N}$. Each player i has associated with it a source-target pair $s_i, t_i \in V$ of vertices that she needs to connect with a simple path in G. The choice of such a path is called a *strategy* of the player, and a collection consisting of one strategy for each player is called a *strategy profile*. The cost c_e of every edge e is shared equally among the players using it. Each player i aims at choosing a path of smallest possible (individual) cost to herself. This cost is defined as the sum of the cost shares for player i along the path. Players are selfish in that they only care about their own costs. In particular, they do not care about the social cost, defined as the sum of all players' individual costs and denoted by $\text{cost}(P)$ for a strategy profile P.

A *Nash equilibrium* of a Shapley network design game is a strategy profile in which no player i can switch to an s_i-t_i path that yields her a smaller individual cost. To quantify the effect of the selfish behavior, it is natural to compare the social cost of a Nash equilibrium of the game with the smallest social cost among all possible strategy profiles [1,2]. Several quantifications of selfish behavior have been studied, based on whether we restrict ourselves to a specific set of Nash equilibria, and whether we compare the worst or best such equilibrium in terms of social cost. In this paper, we adopt the notion of the *price of stability*, introduced by Anshelevich et al. [1]. Denoting by $\mathcal{N}$ the set of all Nash equilibria and by O a strategy profile that minimizes the social cost of a game, the price of stability of the game is defined as the ratio $\min_{\mathrm{N} \in \mathcal{N}} \text{cost}(\mathrm{N})/\text{cost}(\mathrm{O})$.

Anshelevich et al. [1] observed that Shapley network design games always have a Nash equilibrium by showing that they belong to the class of congestion games. For these games, the existence of a Nash equilibrium is always guaranteed, as shown by Rosenthal [3]. Rosenthal's existence proof relies on a potential function argument. That is, he showed that there exists a function Φ that maps strategy profiles to real numbers and has the property that if any one player changes her strategy unilaterally, then the value of Φ changes by the exact same value as the cost of the player. This observation, together with the finiteness of the space of all strategy profiles, implies the existence of a Nash equilibrium. In particular, any *potential minimum*, i.e., a strategy profile that globally minimizes the potential function, is a Nash equilibrium. The potential function of a game is unique up to an additive constant (see Monderer and Shapley [4]). Using the special form of the potential function for Shapley network design games, Anshelevich et al. [1] showed that the price of stability of any game is at most $H_k = \sum_{i=1}^{k} \frac{1}{i}$, the k-th *harmonic number* (which is of order $\log k$). This upper bound is tight for games played on directed graphs. That is, there are Shapley network design games on directed graphs [1] for which the price of stability is arbitrarily close to H_k.

The situation is different for *undirected* Shapley network design games, i.e., games played on undirected graphs. As the same potential arguments remain valid, the price of stability of any game is still at most H_k. Yet, the largest known price of stability (asymptotically) is a constant, more precisely $348/155 \approx 2.245$ (see Bilò et al. [5]). This leaves the question of the worst-possible price of stability in undirected Shapley network design games with k players wide open. Remarkably, the largest known price of stability, as provided by Bilò et al., does not come from a simple example, but from a complicated construction. Previously known worst-case games had a price of stability of $4/3 \approx 1.333$ [1], $12/7 \approx 1.714$ [6], and $42/23 \approx 1.8261$ [7]. Despite numerous attempts [8,1,5,6,7,9] to narrow the gap of the bounds on the price of stability, there has been little progress in terms of numerical results. It is generally believed that the price of stability is smaller than H_k, and we confirm this belief in this paper. For small values of k some smaller upper bounds are known. For $k = 2$ players, the price of stability is at most $4/3 < H_2 = 3/2$ and this is tight [1,7]. Bilò and Bove [8] analyzed the case of $k = 3$ players and showed that the price of stability of any such game is at most $1.634 < H_3 = 1.83\bar{3}$. For this case, however, a considerable gap remains, as the worst example known has a price of stability of $74/48 \approx 1.542$ [7]. Thus, already for $k = 3$ players, the exact worst-case price of stability is unknown.

For several special cases, one can derive better upper bounds on the price of stability. If all players share the same terminal then the price of stability is at most $O(\log k/\log\log k)$ [9]. If in addition every vertex is the source of at least one player, then the price of stability further degrades to $O(\log\log k)$ [6].

Many of the mentioned upper bounds are not only valid for the best Nash equilibrium of a game, but also for a very specific one – the potential minimum. Potential minima have desirable stability properties. For example, they are reached by certain learning dynamics for players that do not always play rationally (see Blume [10]). This motivates to explicitly study the ratio between the cost of a potential minimum and that of a profile minimizing the social cost – a *social optimum.* To stress the described stability properties of potential minima, Asadpour and Saberi [11] called this ratio the *inefficiency ratio of stable equilibria.* Kawase and Makino [12] called the very same ratio the *potential-optimal price of anarchy.* They also define the *potential-optimal price of stability* of a game in the obvious way as the ratio between the cost of a best potential minimum and that of a social optimum. They prove that the potential-optimal price of anarchy of undirected Shapley network design games is at most $O(\sqrt{\log k})$ for the special case where all players share the same terminal node, and where every vertex is the source of at least one player. They give a construction of a game with potential-optimal price of anarchy $\Omega(\sqrt{\log\log k})$.

Our Contribution

Our main result shows that the price of stability in undirected Shapley network design games is at most $\frac{k^3(k+1)/2-k^2}{1+k^3(k+1)/2-k^2}H_k = (1-\Theta(1/k^4))H_k$. Thus, we provide the first general upper bound that shows that the price of stability for k players is strictly smaller than H_k. To prove this upper bound, we generalize the techniques of Christodoulou et al. [7] to any number of players. In short, similar to Christodoulou et al., we obtain a set of inequalities relating the cost of any Nash equilibrium to the cost of a social optimum. We then combine these in a non-trivial way to obtain the claimed upper bound, additionally assuming that the

Nash equilibrium has a smaller potential than the social optimum. Interestingly, the resulting upper bound is tight for the case of $k = 2$ players.

As an additional contribution, we provide an example of a game with $k = 3$ players in which the potential-optimal price of stability is 1.634. Thus, we show that the upper bound on the potential-optimal price of anarchy given by Bilò and Bove [8] is tight. This result implies that for three players the upper bound on the price of stability cannot be further improved via potential-minimizers. This is in contrast to the directed case, for which a simple inequality relates the cost of the potential minimum to that of a social optimum, giving a tight bound on the price of stability. We believe that this observation provides an insight as to why the undirected case is much harder to tackle than the directed one.

We note that in our tight example for the potential-optimal price of stability/anarchy, the social optimum is also a Nash equilibrium, and thus the example provides no new lower bounds on the price of stability. Our third contribution however is a new lower bound on the price of stability. We provide an example of a game with three players and price of stability 1.571, which improves on the previous best lower bound of $74/48 \approx 1.542$ [7].

2 Problem Definition and Preliminaries

Let $G = (V, E)$ be an undirected graph with a positive *cost* $c_e > 0$ for every edge $e \in E$. The *Shapley network design game* is a strategic game of k players. Every player $i \in \{1, \dots, k\}$ has a dedicated pair of vertices $s_i, t_i \in V$, that we call her *source* and *target*, respectively. The *strategy space* of player i is the collection $\mathcal{P}_i$ of all paths $P_i \subseteq E$ between s_i and t_i. Every such path P_i is called a *strategy.* A *strategy profile* P is a tuple $(P_1, \dots, P_k)$ of k strategies, $P_i \in \mathcal{P}_i$. Given a strategy profile P, we say that player i *plays* strategy P_i in P. The *cost to player* i in a strategy profile $P = (P_1, \dots, P_k)$ is $\mathrm{cost}_i(P) = \sum_{e \in P_i} \frac{c_e}{k_e}$, where k_e denotes the number of paths P_i in P such that $e \in P_i$. That is, k_e is the number of players that use edge e. The goal of every player is to minimize her cost. A *Nash equilibrium* is a strategy profile $\mathrm{N} = (\mathrm{N}_1, \dots, \mathrm{N}_k)$, $\mathrm{N}_i \in \mathcal{P}_i$, such that no player i can improve her cost by playing a different strategy. That is, for every i and every $\mathrm{N}'_i \in \mathcal{P}_i$, it holds that $\mathrm{cost}_i(\mathrm{N}) \leq \mathrm{cost}_i(N'_i, N_{-i})$, where (N'_i, N_{-i}) is a shorthand for $(\mathrm{N}_1, \dots, \mathrm{N}_{i-1}, \mathrm{N}'_i, \mathrm{N}_{i+1}, \dots, \mathrm{N}_k)$. With a slight abuse of terminology we will identify the game played on the graph G with the graph itself and we write $\mathcal{N}(G)$ to denote the set of Nash equilibria of G.

Observe that the edges of any strategy profile P induce a graph $(V, \cup_i P_i)$, which we call the *underlying network.* We denote the edge set of this graph by $E(P)$. The *social cost*, or simply the *cost* of a strategy profile P, denoted as $\mathrm{cost}(P)$, is the sum of the players' individual costs. Observe that the social cost is equal to the total cost of the edges in the played strategies, i.e., $\mathrm{cost}(P) = \sum_{i=1}^{k} \mathrm{cost}_i(P) = \sum_{e \in E(P)} c_e$.

A strategy profile that minimizes the social cost is called the *social optimum.* The *price of stability* of a game G, denoted by $\mathrm{PoS}(G)$, is defined as the cost of the best Nash equilibrium of G divided by the cost of a social optimum $\mathrm{O}(G)$

of G. That is, $\mathrm{PoS}(G) = \min_{\mathrm{N} \in \mathcal{N}(G)} \mathrm{cost}(\mathrm{N})/\mathrm{cost}(\mathrm{O}(G))$. The *price of anarchy* of G, $\mathrm{PoA}(G)$ for short, is obtained by replacing min by max in this definition.

A Shapley network design game is a potential game [1,3]. That is, there is a function $\Phi : \mathcal{P}_1 \times \ldots \times \mathcal{P}_k \to \mathbb{R}$ such that, for every strategy profile P, whenever any player i changes her strategy from P_i to P_i', then $\mathrm{cost}_i(P) - \mathrm{cost}_i(P_i', P_{-i}) = \Phi(P) - \Phi(P_i', P_{-i})$. Rosenthal's (exact) potential function has the form $\Phi(P) = \sum_{e \in E(P)} \sum_{i=1}^{k_e} \frac{c_e}{i} = \sum_{e \in E(P)} H_{k_e} \cdot c_e$, where H_j denotes the j-th Harmonic number $\sum_{i=1}^{j} \frac{1}{i}$. The potential function is unique up to an additive constant [4].

Motivated by the particular stability properties of potential minima, Kawase and Makino [12] introduced two notions to quantify the inefficiency of potential minimizers. For a game G let $\mathcal{F}(G)$ denote the set of potential minimizers of G, i.e., strategy profiles of G that minimize the potential function of G. The *potential-optimal price of stability* of G is then defined as $\mathrm{POPoS}(G) = \min_{\mathrm{N} \in \mathcal{F}(G)} \mathrm{cost}(\mathrm{N})/\mathrm{cost}(\mathrm{O}(G))$ and the *potential-optimal price of anarchy* is defined as $\mathrm{POPoA}(G) = \max_{\mathrm{N} \in \mathcal{F}(G)} \mathrm{cost}(\mathrm{N})/\mathrm{cost}(\mathrm{O}(G))$. Since $\mathcal{F}(G) \subseteq \mathcal{N}(G)$, clearly, for any game G, $\mathrm{PoS}(G) \leq \mathrm{POPoS}(G) \leq \mathrm{POPoA}(G) \leq \mathrm{PoA}(G)$.

For a fixed number of players $k \geq 2$, we are interested to bound the *worst-case* price of stability of games with k players. For a formal definition, let $\mathcal{G}(k)$ denote the set of all games with k players. The price of stability of undirected Shapley network design games with k players is defined as $\mathrm{PoS}(k) = \sup_{G \in \mathcal{G}(k)} \mathrm{PoS}(G)$. $\mathrm{POPoS}(k)$ and $\mathrm{POPoA}(k)$ are defined analogously.

Using Rosenthal's potential function Φ, we can bound the potential-optimal price of anarchy (and, thus, the price of stability) from above by H_k as follows (cf. [1]). Let O be a social optimum. For a potential minimum N, we have $\Phi(\mathrm{N}) \leq \Phi(\mathrm{O})$. Using this together with $\mathrm{cost}(\mathrm{N}) \leq \Phi(\mathrm{N})$ and $\Phi(\mathrm{O}) \leq H_k \cdot \mathrm{cost}(\mathrm{O})$ we obtain $\mathrm{cost}(\mathrm{N}) \leq H_k \cdot \mathrm{cost}(\mathrm{O})$, as claimed.

We define N^i and O^i, for $i \in \{1, \ldots, k\}$, to be the sets of edges of N and O that are used by exactly i players, respectively. Thus, $E(\mathrm{N}) = \bigcup_{i=1}^{k} \mathrm{N}^i$ and $E(\mathrm{O}) = \bigcup_{i=1}^{k} \mathrm{O}^i$. For a set of edges $M \subseteq E$, we will denote by $|M|$ the total cost of the edges in M, i.e., $|M| = \sum_{e \in M} c_e$. This allows us to express the value of the potential function for N and O by $\Phi(\mathrm{N}) = \sum_{j=1}^{k} H_j |\mathrm{N}^j|$ and $\Phi(\mathrm{O}) = \sum_{j=1}^{k} H_j |\mathrm{O}^j|$, respectively.

3 A General Upper Bound

In this section we derive an upper bound on the price of stability of *any* undirected Shapley network design game. The main idea to show our upper bound follows that of Christodoulou et al. [7], which they used for deriving an upper bound of $33/20 = 1.65$ for the case of $k = 3$ players. By the definition of Nash equilibria, no player can change her chosen s_i-t_i path in such a profile and thereby improve her cost. We will consider a specific change of strategy for each player, which gives us an inequality that relates the costs of the edges used in the Nash equilibrium to those used in the social optimum. We then combine this inequality in a non-trivial way with another that is gained from the fact that we consider a potential minimum. This allows us to obtain an upper bound of $\frac{k^3(k+1)/2-k^2}{1+k^3(k+1)/2-k^2} H_k = (1-\Theta(1/k^4))H_k$ on the price of stability.

Consider a Nash equilibrium N and a social optimum O of a given Shapley network design game with k players. In undirected graphs, if O^k is non-empty any player i can use paths in the underlying network of the optimum O to connect its terminals to the source and the target of another player j. By additionally using the path of player j in the Nash equilibrium N, we obtain a valid strategy for player i. This is the specific alternative strategy of player i which we use to relate the cost of the Nash equilibrium to the social optimum.

We will additionally use the following observation about the structure of social optima. Observe that any social optimum O forms a forest (because from any cycle we could remove an edge and thereby decrease the cost). If O^k is non-empty, i.e., some edges are shared among all players in the optimum, the edges of $E(\mathrm{O}) \setminus \mathrm{O}^k$ form two trees such that every player has one terminal in each of the trees. Let O^+ denote the edge set of the larger tree, and O^- denote the edge set of the smaller one (measured in terms of the social cost). We have $\mathrm{O}^+ \cup \mathrm{O}^- = E(\mathrm{O}) \setminus \mathrm{O}^k$ and, by the definition, $|\mathrm{O}^+| \geq |\mathrm{O}^-|$. Every tree has a closed walk that visits every vertex at least once and every edge exactly twice (for example, a depth-first traversal). We consider such a walk in the tree given by the edges in O^+, and use it to order the players. Without loss of generality, if $\mathrm{O}^k \neq \emptyset$, the players are numbered such that there is a closed walk in O^+ that visits the terminals in the order given by the players' numbers, while using each edge exactly twice. We say that the players are in *major-tree order.*

Consider the edges of a Nash equilibrium which are not used by all k players. The following lemma bounds the cost of these edges with respect to the cost of a social optimum.

Lemma 1. *Given a game G, let $N = (N_1, \ldots, N_k)$ be a Nash equilibrium and $O = (O_1, \ldots, O_k)$ a social optimum with $O^k \neq \emptyset$. Then,*

$$\sum_{j=1}^{k-1} |N^j| \leq (k^2(k+1)/2 - k) \sum_{j=1}^{k-1} |O^j|.$$

Proof. For every player i, we construct an s_i-t_i path P_i (cf. Figure 1) with the property that every edge on P_i is either in N_{i+1} or not in O^k. In the following we understand indices modulo k, i.e. $k+1 \equiv 1$ and $0 \equiv k$. Let u, v be the first and last vertex on O_i that are also on O_{i+1}. These vertices are well defined as O^k is non-empty, and thus $\mathrm{O}_i \cap \mathrm{O}_{i+1}$ is non-empty. Assume that u lies before v on O_{i+1} (otherwise, exchange s_{i+1} and t_{i+1} in the following). Let P_i be the path from s_i to t_i that first follows O_i until u, then O_{i+1} (backwards) from u to s_{i+1}, then N_{i+1} from s_{i+1} to t_{i+1}, then O_{i+1} (backwards) from t_{i+1} to v, and finally O_i from v to t_i. In case P_i contains cycles, we skip them to obtain a simple path. It is easy to verify that every edge on P_i either lies on N_{i+1} or is not in $\mathrm{O}_i \cap \mathrm{O}_{i+1}$. Thus, P_i has the desired property. In the following, let Q_i denote the set of edges of P_i that are also contained in $E(\mathrm{O})$.

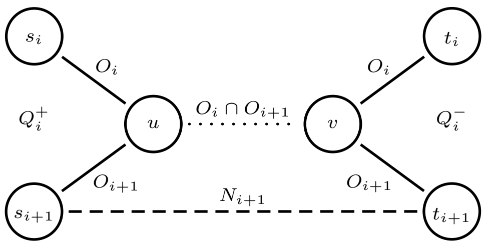


Fig. 1. Constructing the path P_i with O_i, O_{i+1}, N_{i+1} (dashed line), and Q_i (continuous lines). Note that $\mathrm{O}_i \cap \mathrm{O}_{i+1}$ (dotted line) is not part of P_i.

Since N is a Nash equilibrium, player i cannot improve her cost by choosing path P_i. Therefore, $\mathrm{cost}_i(\mathrm{N}) \le \mathrm{cost}_i(P_i, \mathrm{N}_{-i})$. If k_e is the number of players using edge e in N, this inequality amounts to $\sum_{e \in \mathrm{N}_i} \frac{c_e}{k_e} \le \sum_{e \in P_i \cap \mathrm{N}_i} \frac{c_e}{k_e} + \sum_{e \in P_i \setminus \mathrm{N}_i} \frac{c_e}{k_e+1}$. By the properties of P_i, the right-hand side of this inequality can be upper bounded by $\sum_{e \in Q_i} c_e + \sum_{e \in \mathrm{N}_{i+1} \cap \mathrm{N}_i} \frac{c_e}{k_e} + \sum_{e \in \mathrm{N}_{i+1} \setminus \mathrm{N}_i} \frac{c_e}{k_e+1}$. By shifting all terms not depending on Q_i to the left-hand side of the resulting inequality, we get

$$\sum_{e \in \mathrm{N}_i \setminus \mathrm{N}_{i+1}} \frac{c_e}{k_e} - \sum_{e \in \mathrm{N}_{i+1} \setminus \mathrm{N}_i} \frac{c_e}{k_e+1} \le \sum_{e \in Q_i} c_e. \tag{1}$$

Similar to the path P_i we can define a path $\widehat{P}_i$ with respect to player $i-1$. That is, $\widehat{P}_i$ uses the edges of O_i, O_{i-1}, and N_{i-1} to connect s_i to t_i and does not contain any edges from $\mathrm{O}_i \cap \mathrm{O}_{i-1}$. Let $\widehat{Q}_i$ denote the set of edges of $\widehat{P}_i$ also contained in $E(\mathrm{O})$. Using the same arguments as above on $\widehat{P}_i$ we can derive an analogous inequality as (1) for edges in N_i, N_{i-1}, and $\widehat{Q}_i$. Adding this inequality to (1) and then summing over all i gives

$$\sum_{i=1}^{k} \left(\sum_{e \in \mathrm{N}_i \setminus \mathrm{N}_{i+1}} \frac{c_e}{k_e} - \sum_{e \in \mathrm{N}_{i+1} \setminus \mathrm{N}_i} \frac{c_e}{k_e+1} + \sum_{e \in \mathrm{N}_i \setminus \mathrm{N}_{i-1}} \frac{c_e}{k_e} - \sum_{e \in \mathrm{N}_{i-1} \setminus \mathrm{N}_i} \frac{c_e}{k_e+1} \right) \le \sum_{i=1}^{k} \left(\sum_{e \in Q_i} c_e + \sum_{e \in \widehat{Q}_i} c_e \right). \tag{2}$$

We bound the left-hand side and the right-hand side of (2) separately, starting with the left-hand side. Since indices are modulo k, we may shift the index in the second and the fourth sum on the left-hand side. This lets us combine the first and the fourth sum to $\sum_{e \in \mathrm{N}_i \setminus \mathrm{N}_{i+1}} \frac{c_e}{k_e} - \sum_{e \in \mathrm{N}_i \setminus \mathrm{N}_{i+1}} \frac{c_e}{k_e+1} = \sum_{e \in \mathrm{N}_i \setminus \mathrm{N}_{i+1}} \frac{c_e}{k_e(k_e+1)}$, and analogously the second and the third sum to $\sum_{e \in \mathrm{N}_i \setminus \mathrm{N}_{i-1}} \frac{c_e}{k_e(k_e+1)}$, to obtain

$$\sum_{i=1}^{k} \sum_{e \in \mathrm{N}_i \setminus \mathrm{N}_{i+1}} \frac{c_e}{k_e(k_e+1)} + \sum_{i=1}^{k} \sum_{e \in \mathrm{N}_i \setminus \mathrm{N}_{i-1}} \frac{c_e}{k_e(k_e+1)}.$$

Each of the two resulting sums counts each edge in $E(\mathrm{N}) \setminus \mathrm{N}^k$ at least once. This is because for any edge e used by at least one player but not all of them, there is a pair of players with consecutive indices (modulo k) such that e is used in N by one of the players but not the other. Thus, we can lower bound the above term by

$$2 \sum_{e \in E(\mathrm{N}) \setminus \mathrm{N}^k} \frac{c_e}{k_e(k_e+1)} = \sum_{j=1}^{k-1} \frac{2}{j(j+1)} |\mathrm{N}^j| \geq \frac{2}{k(k-1)} \sum_{j=1}^{k-1} |\mathrm{N}^j|,$$

which gives a lower-bound on the left-hand side of Inequality (2).

The right-hand side of Inequality (2) can be bounded by exploiting the major-tree order of the players. We first only bound the sum depending on Q_i. We denote the two parts of Q_i that lie in the larger and smaller parts of $E(\mathrm{O}) \setminus \mathrm{O}^k$ by $Q_i^+ = Q_i \cap \mathrm{O}^+$ and $Q_i^- = Q_i \cap \mathrm{O}^-$, respectively. Note that, by construction of P_i, there are no edges of O^k in Q_i. Thus, we get

$$\sum_{i=1}^{k} \sum_{e \in Q_i} c_e = \sum_{i=1}^{k} \left(\sum_{e \in Q_i^+} c_e + \sum_{e \in Q_i^-} c_e \right).$$

By the defining property of the major-tree order, each edge in O^+ is counted exactly twice in the above sum, while each edge of O^- is counted at most k times. At the same time, the weight of the edges in O^- amounts to at most half the total weight of $E(\mathrm{O}) \setminus \mathrm{O}^k$. Hence,

$$\begin{aligned}\sum_{i=1}^{k} \left(\sum_{e \in Q_i^+} c_e + \sum_{e \in Q_i^-} c_e \right) &\leq 2 \sum_{e \in \mathrm{O}^+} c_e + k \sum_{e \in \mathrm{O}^-} c_e = 2 \sum_{e \in E(\mathrm{O}) \setminus \mathrm{O}^k} c_e + (k-2) \sum_{e \in \mathrm{O}^-} c_e \\ &\leq (k/2+1) \sum_{e \in E(\mathrm{O}) \setminus \mathrm{O}^k} c_e .\end{aligned}$$

Analogously, we can derive a corresponding bound for the sum depending on $\widehat{Q}_i$. Since the sum over all costs of edges in $E(\mathrm{O}) \setminus \mathrm{O}^k$ is exactly $\sum_{i=1}^{k-1} |\mathrm{O}^j|$, we can bound the right-hand side of Inequality (2) by $(k+2) \sum_{i=1}^{k-1} |\mathrm{O}^j|$. Together the two derived bounds for the left-hand side and the right-hand side of Inequality (2) give the claimed inequality. □

The following lemma encapsulates some technical calculations that allows to derive an upper bound on the price of stability using Lemma 1.

Lemma 2. *For a game G with social optimum O, let N be a Nash equilibrium with $\Phi(N) \leq \Phi(O)$ and let $\beta > 0$ be such that $\sum_{j=1}^{k-1} |N^j| \leq \beta \sum_{j=1}^{k-1} |O^j|$. Then,*

$$\sum_{j=1}^{k} |N^j| \leq \frac{\beta k}{1+\beta k} H_k \cdot \sum_{j=1}^{k} |O^j|.$$

Proof. We take $1 < \alpha < H_k$ and compute

$$\sum_{j=1}^{k} |\mathrm{N}^j| \le \alpha |\mathrm{N}_k| + \sum_{j=1}^{k-1} |\mathrm{N}^j| = \frac{\alpha}{H_k} \cdot \sum_{j=1}^{k} H_j |\mathrm{N}^j| + \sum_{j=1}^{k-1} \left(1 - \alpha \frac{H_j}{H_k}\right) |\mathrm{N}^j|$$
$$\le \frac{\alpha}{H_k} \cdot \sum_{j=1}^{k} H_j |\mathrm{N}^j| + \sum_{j=1}^{k-1} \left(1 - \frac{\alpha}{H_k}\right) |\mathrm{N}^j|.$$

Using that $\Phi(\mathrm{N}) \le \Phi(\mathrm{O})$ and the condition of the lemma, we introduce β to get

$$\sum_{j=1}^{k} |\mathrm{N}^j| \le \frac{\alpha}{H_k} \cdot \sum_{j=1}^{k} H_j |\mathrm{O}^j| + \left(1 - \frac{\alpha}{H_k}\right) \cdot \beta \sum_{j=1}^{k-1} |\mathrm{O}^j|$$
$$= \alpha |\mathrm{O}^k| + \sum_{j=1}^{k-1} \left[\alpha \frac{H_j}{H_k} + \beta \left(1 - \frac{\alpha}{H_k}\right)\right] |\mathrm{O}^j|$$
$$\le \alpha |\mathrm{O}^k| + \sum_{j=1}^{k-1} \left[\alpha \frac{H_{k-1}}{H_k} + \beta \left(1 - \frac{\alpha}{H_k}\right)\right] |\mathrm{O}^j|. \tag{3}$$

For $\alpha = \frac{\beta k}{1+\beta k} H_k$ we have

$$\alpha \frac{H_{k-1}}{H_k} + \beta \left(1 - \frac{\alpha}{H_k}\right) = \frac{\beta k}{1+\beta k} H_{k-1} + \beta \left(1 - \frac{\beta k}{1+\beta k}\right)$$
$$= \frac{\beta k}{1+\beta k} \left(H_k - \frac{1}{k}\right) + \frac{\beta}{1+\beta k} = \alpha. \tag{4}$$

From (3) and (4) it follows that

$$\sum_{j=1}^{k} |\mathrm{N}^j| \le \alpha |\mathrm{O}^k| + \alpha \sum_{j=1}^{k-1} |\mathrm{O}^j| = \frac{\beta k}{1+\beta k} H_k \sum_{j=1}^{k} |\mathrm{O}^j|,$$

which concludes the proof. □

The above lemmas can be put together in order to show the following theorem.

Theorem 3. *The potential-optimal price of anarchy POPoA(k) for Shapley network design games with $k \ge 2$ players is at most $\frac{k^3(k+1)/2-k^2}{1+k^3(k+1)/2-k^2} H_k$.*

Proof. Let N be a potential minimum. If $\mathrm{O}^k \ne \emptyset$, we may combine Lemma 1 and Lemma 2 to obtain $\frac{\mathrm{cost(N)}}{\mathrm{cost(O)}} \le \frac{k^3(k+1)/2-k^2}{1+k^3(k+1)/2-k^2} H_k$. If, on the other hand, $\mathrm{O}^k = \emptyset$, then obviously also $|\mathrm{O}^k| = 0$. We show that then $\mathrm{cost(N)} \le H_{k-1}\mathrm{cost(O)}$ (which has been observed before for $k = 3$, e.g., by Christodoulou et al. [7]). We can express the potential functions of N and O using N^j and O^j and use the fact that $\Phi(\mathrm{N}) \le \Phi(\mathrm{O})$ to obtain

$$\sum_{j=1}^{k} |\mathrm{N}^j| \le \sum_{j=1}^{k} H_j |\mathrm{N}^j| \le \sum_{j=1}^{k} H_j |\mathrm{O}^j| = \sum_{j=1}^{k-1} H_j |\mathrm{O}^j| \le H_{k-1} \sum_{j=1}^{k} |\mathrm{O}^j|.$$

Hence, in this case we get $\frac{\text{cost(N)}}{\text{cost(O)}} \leq H_{k-1}$, which is lower than the first bound. □

We obtain the following corollary. Note that the bound is tight for $k = 2$.

Corollary 4. *The price of stability* $PoS(k)$ *for Shapley network design game with* $k \geq 2$ *players is at most* $\frac{k^3(k+1)/2-k^2}{1+k^3(k+1)/2-k^2}H_k$.

4 Three-Player Games

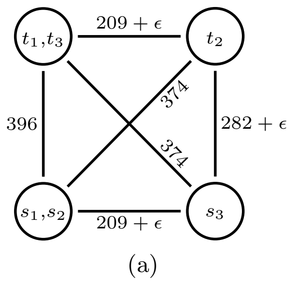
The upper bound of $\frac{k^3(k+1)/2-k^2}{1+k^3(k+1)/2-k^2}H_k$ on the price of stability presented in the previous section is valid for an arbitrary number of players. For the special case of $k = 3$ players, it evaluates to $165/92 \approx 1.793$. For this case, however, better bounds are known. Bilò and Bove proved that the price of stability does not exceed $286/175 \approx 1.634$. For the proof of their result they combine inequalities that are valid for any potential minimum of the game. Thus, their proof implies that also the potential-optimal price of anarchy (and thus the potential-optimal price of stability) is at most 286/175. As the main result of this section, we will show that this result is tight. That is, there is a three-player game such that the cost of the best potential minimum is 286/175 times the cost of the social optimum.

Theorem 5. *For three players, the potential-optimal price of stability and the potential-optimal price of anarchy are* $POPoA(3) = POPoS(3) = 286/175 \approx 1.634$.

Proof. The upper bound of 286/175 on the potential-optimal price of anarchy was proved by Bilò and Bove [8, Theorem 3.1]. They derive the bound for the potential-optimal price of anarchy, but only explicitly state the implied bound for the (regular) price of stability.

Consider the three player game in Figure 2(a), and let $\epsilon > 0$ be sufficiently small. The potential-optimal price of stability of this example approaches 286/175 when ϵ tends to 0, which establishes tightness of the upper bound. It also shows that the potential-optimal price of stability and the potential-optimal price of anarchy coincide for the class of games with three players.

Obviously, any strategy profile in the example has to use at least three edges to connect all terminal pairs. The three cheapest edges already connect all terminal pairs and thus constitute the social optimum O of cost $700 + 3\epsilon$. It is easy to verify that the underlying networks of Nash equilibria in the example do not contain cycles, since at least one edge of each cycle would be abandoned by all players. Hence, all Nash equilibria in the example use exactly three edges. We show that the *unique* potential minimum N uses the three edges not used in O (note that O itself is a Nash equilibrium). This profile has both a potential-function value and cost equal to $396 + 2 \cdot 374 = 1144$, since every edge is used by one player only. In contrast, the social optimum has a potential-function value of $2H_2 \cdot (209 + \epsilon) + H_3 \cdot (282 + \epsilon) > 1144$. If the edge $\{t_2, s_3\}$ is used in a Nash equilibrium, the other two edges have to be used by at least two players each. For profiles other than O, this gives a potential function value of at least



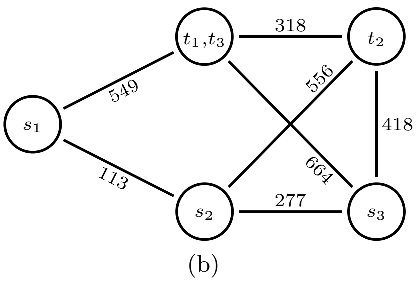


Fig. 2. (a) A three-player game with POPoS and POPoA approaching $286/175 \approx 1.634$ for $\epsilon \to 0$. (b) A three-player game with PoS $= 1769/1126 \approx 1.571$.

$H_2 \cdot (209+\epsilon+374)+(282+\epsilon) > 1156$. If the edges with cost 396 and $(282+\epsilon)$ are both unused, all three players use an edge of cost 374 and the resulting potential value is either $H_3 \cdot 374 + H_2 \cdot (209+\epsilon)+(209+\epsilon) > 1208$ or $H_2 \cdot 374+374+(209+\epsilon) > 1144$. Equilibria that use one edge each of costs $(209+\epsilon)$, 374, 396 have a potential function value of $H_2 \cdot 396 + 374 + (209+\epsilon) > 1177$. And finally, the profile using both cheap edges together with the one of cost 396 has a potential of $H_3 \cdot 396 + 2 \cdot (209+\epsilon) > 1144$. We conclude that the potential minimum is as claimed. For ϵ tending to 0, the ratio between the cost of N and O approaches 286/175. □

Our result in particular implies that it is impossible to push the upper bound on the price of stability for three-player Shapley network design games on undirected networks below 286/175 by using inequalities that are only valid for global minima of the potential function. Note that the example in Figure 2(a) has a price of stability of 1, since its social optimum is itself a Nash equilibrium.

So far, the best lower bound on the price of stability for three-player games was $74/48 \approx 1.542$ [7]. We can slightly improve this bound by presenting a game with three players whose price of stability is $1769/1126 \approx 1.571$. Consider the network with 5 vertices shown in Figure 2(b). By exhaustive enumeration of all strategy profiles, one can verify that only the strategy profile in which each player i uses the edge (s_i, t_i) is a Nash equilibrium. The social optimum uses all other edges and has a cost of 1126, while the unique Nash equilibrium has a cost of 1769. This establishes the claimed lower bound on the price of stability in undirected Shapley network design games with three players.

5 Conclusions

We gave an upper bound for the price of stability for an arbitrary number of players k in undirected graphs. Our bound is smaller than H_k for every k and tight for two players. For three players, we showed that the upper bound of $286/175 \approx 1.634$ by Bilò and Bove [8] is tight for both the potential-optimal price of stability and anarchy. We also improved the lower bound to $1769/1126 \approx 1.571$ for the price of stability in this case.

Asymptotically, a wide gap remains between the upper bound of the price of stability that is of order $\log k$, and the best known lower bound construction by Bilò et al. [5] that approaches the constant $348/155 \approx 2.245$. It is unclear where the correct answer lies within this gap, in particular since bounds of $O(\log k / \log\log k)$ [9] and $O(\log\log k)$ [6] emerge for restrictions of the problem. Our bound approaches H_k for a growing number of players k. It would already be interesting to know whether the price of stability is asymptotically below H_{k-c} or $H_{k/c}$ for some (or even any) positive constant $c \in \mathbb{N}$.

Acknowledgements. This work has been partially supported by the Swiss National Science Foundation (SNF) under the grant number 200021_143323/1. We thank Julian Mestre for pointing out a mistake in the proof of Lemma 1 in an earlier version of this paper.

References

1. Anshelevich, E., Dasgupta, A., Kleinberg, J.M., Tardos, É., Wexler, T., Roughgarden, T.: The price of stability for network design with fair cost allocation. SIAM Journal on Computing 38, 1602–1623 (2008)
2. Tardos, É., Wexler, T.: Network formation games. In: Nisan, N., Roughgarden, T., Tardos, É., Vazirani, V. (eds.) Algorithmic Game Theory. Cambridge University Press (2007)
3. Rosenthal, R.W.: A class of games possessing pure-strategy Nash equilibria. International Journal of Game Theory 2, 65–67 (1973)
4. Monderer, D., Shapley, L.S.: Potential games. Games and Economic Behavior 14, 124–143 (1996)
5. Bilò, V., Caragiannis, I., Fanelli, A., Monaco, G.: Improved lower bounds on the price of stability of undirected network design games. In: Kontogiannis, S., Koutsoupias, E., Spirakis, P.G. (eds.) SAGT 2010. LNCS, vol. 6386, pp. 90–101. Springer, Heidelberg (2010)
6. Fiat, A., Kaplan, H., Levy, M., Olonetsky, S., Shabo, R.: On the price of stability for designing undirected networks with fair cost allocations. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 608–618. Springer, Heidelberg (2006)
7. Christodoulou, G., Chung, C., Ligett, K., Pyrga, E., van Stee, R.: On the price of stability for undirected network design. In: Proc. 7th International Workshop on Approximation and Online Algorithms, pp. 86–97 (2009)
8. Bilò, V., Bove, R.: Bounds on the price of stability of undirected network design games with three players. Journal of Interconnection Networks 12, 1–17 (2011)
9. Li, J.: An upper bound on the price of stability for undirected shapley network design games. Information Processing Letters 109, 876–878 (2009)
10. Blume, L.E.: The statistical mechanics of best-response strategy revision. Games and Economic Behavior 11, 111–145 (1995)
11. Asadpour, A., Saberi, A.: On the inefficiency ratio of stable equilibria in congestion games. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 545–552. Springer, Heidelberg (2009)
12. Kawase, Y., Makino, K.: Nash equilibria with minimum potential in undirected broadcast games. In: Rahman, M. S., Nakano, S.-I. (eds.) WALCOM 2012. LNCS, vol. 7157, pp. 217–228. Springer, Heidelberg (2012)

Complexity of Barrier Coverage with Relocatable Sensors in the Plane*

Stefan Dobrev[1], Stephane Durocher[2], Mohsen Eftekhari[3], Konstantinos Georgiou[4], Evangelos Kranakis[5], Danny Krizanc[6], Lata Narayanan[3], Jaroslav Opatrny[3], Sunil Shende[7], and Jorge Urrutia[8]

[1] Inst. of Math., Slovak Academy of Sci., Slovakia
[2] Dept. of Comp. Sci., U. of Manitoba, Canada
[3] Dept. of Comp. Sci. and Software Eng., Concordia U., Canada
[4] Dept. of Combin. and Opt., U. Of Waterloo, Canada
[5] School of Comp. Sci., Carleton U., Canada
[6] Dept. of Math. and Comp. Sci., Wesleyan U., USA
[7] Dept. of Comp. Sci., Rutgers U., USA
[8] Inst. of Math., UNAM, Mexico

Abstract. We consider several variations of the problems of covering a set of barriers (modeled as line segments) using sensors that can detect any intruder crossing any of the barriers. Sensors are initially located in the plane and they can *relocate* to the barriers. We assume that each sensor can detect any intruder in a circular area centered at the sensor. Given a set of barriers and a set of sensors located in the plane, we study three problems: the feasibility of barrier coverage, the problem of minimizing the largest relocation distance of a sensor (MinMax), and the problem of minimizing the sum of relocation distances of sensors (MinSum). When sensors are permitted to move to arbitrary positions on the barrier, the problems are shown to be NP-complete. We also study the case when sensors use *perpendicular* movement to one of the barriers. We show that when the barriers are parallel, both the MinMax and MinSum problems can be solved in polynomial time. In contrast, we show that even the feasibility problem is NP-complete if two perpendicular barriers are to be covered, even if the sensors are located at integer positions, and have only two possible sensing ranges. On the other hand, we give an $O(n^{3/2})$ algorithm for a natural special case of this last problem.

1 Introduction

The protection of a region by sensors against intruders is an important application of sensor networks that has been previously studied in several papers. Each sensor is typically considered to be able to sense an intruder in a circular region around the sensor. Previous work on region protection using sensors can

* Supported by VEGA 2/0136/12, NSERC Discovery, and SEP-CONACYT grants.

be classified into two major classes. In the first body of work, called *area coverage*, the monitoring of an entire region is studied [7, 9], and the presence of an intruder can be detected by a sensor anywhere in the region, either immediately after an appearance of an intruder, or within a fixed time delay. In the second body of work, called *barrier coverage*, a region is assumed to be protected by monitoring its perimeter called barrier [1, 2, 4, 5, 8], and an intruder is detected when crossing the barrier. Clearly, the second approach is less costly in terms of the number of sensors required, and it is sufficient in many applications.

There are two different approaches to barrier coverage in the literature. In the first approach, a barrier is considered to be a narrow strip of fixed width. Sensors are dispersed randomly on the barrier, and the probability of barrier coverage is studied based on the density of dispersal. Since random dispersal may leave gaps in the coverage, Yan et al. [11] propose using several rounds of random dispersal for complete barrier coverage. In the second approach, several papers assume that sensors, once dispersed, are mobile, and can be instructed to relocate from the initial position to a final position on the barrier in order to achieve complete coverage [4, 5]. Clearly, when a sufficient number of sensors is used, this approach always guarantees complete coverage of the barrier. The problem therefore is assigning final positions to the sensors in order to minimize some aspect of the relocation cost. The variations studied so far include minimizing the *maximum* relocation distance (*MinMax*), the *sum* of relocation distances (*MinSum*), or minimizing the *number* of sensors that relocate (*MinNum*).

Most of the previous work is set in the one-dimensional setting: the barriers are assumed to be one or more line segments that are part of a line $\mathcal{L}$, and furthermore, the sensors are initially located on the same line $\mathcal{L}$. In [4] it was shown that there is an $O(n^2)$ algorithm for the MinMax problem in the case when the sensor ranges are identical. The authors also showed that the problem becomes NP-complete if there are two barriers. A polynomial time algorithm for the MinMax problem is given in [3] for arbitrary sensor ranges for the case of a single barrier, and an improved algorithm is given for the case when all sensor ranges are identical. In [5], it was shown that the MinSum problem is NP-complete when arbitrary sensor ranges are allowed, and an $O(n^2)$ algorithm is given when all sensing ranges are the same. Similarly as in the MinSum problem, the MinNum problem is NP-complete when arbitrary sensor ranges are allowed, and an $O(n^2)$ algorithm is given when all sensing ranges are the same [10].

In this paper we consider the algorithmic complexity of several natural generalizations of the barrier coverage problem with sensors of arbitrary ranges. We generalize the work in [3, 4, 5, 10] in two significant ways. First, we assume that the initial positions of sensors are points in the two-dimensional plane and are not necessarily collinear. This assumption is justified since in many situations a dispersal of sensors on the barrier might not be practical. Second, we consider multiple barriers that are parallel or perpendicular to each other. This generalization is motivated by barrier coverage of the perimeter of an area. We use standard cost measures such as Euclidean or rectilinear distance between initial and final positions of sensors.

1.1 Preliminaries and Notation

Throughout the paper, we assume that we are given a set of sensors $S = \{s_1, s_2, \dots, s_n\}$ located in the plane in positions $p_1, p_2, \dots, p_n$, where $p_i = (x_i, y_i)$ for some real values x_i, y_i. The sensing ranges of the sensors are $r_1, r_2, \dots, r_n$, respectively. A sensor s_i can detect any intruder in the closed circular area around p_i of radius r_i. We assume that sensor s_i is mobile and thus can relocate itself from its initial location p_i to another specified location p'_i. A *barrier* b is a closed line segment in the plane. Given a set of barriers $\mathcal{B} = \{b_1, b_2, \dots, b_k\}$, and a set of sensors S in positions $p_1, p_2, \dots, p_n$ in the plane, of sensing ranges $r_1, r_2, \dots, r_n$, the *barrier coverage* problem is to determine for each s_i its final position p'_i on one of the barriers, so that all barriers are covered by the sensing ranges of the sensors. We call such an assignment of final positions a *covering assignment.* Figure 1 shows an example of a barrier coverage problem and a possible covering assignment. Sometimes we are also interested in optimizing some measure of the movement of sensors involved to achieve coverage.

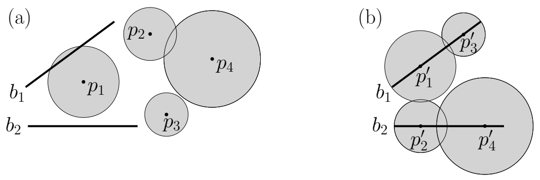


Fig. 1. (a) A given barrier coverage problem (b) a possible covering assignment

We are interested in the algorithmic complexity of three problems:

Feasibility Problem: Given a set of sensors S located in the plane at positions $p_1, p_2, \dots, p_n$, and a set of barriers $\mathcal{B}$, determine if there exists a valid covering assignment, i.e. determine whether there exist final positions $p'_1, p'_2, \dots, p'_n$ on the barriers such that all barriers in $\mathcal{B}$ are covered.

MinMax Problem: Given a set of sensors S located in the plane at positions $p_1, p_2, \dots, p_n$, and a set of barriers $\mathcal{B}$, find final positions $p'_1, p'_2, \dots, p'_n$ on the barriers so that all barriers in $\mathcal{B}$ are covered and $\max_{1 \le i \le n}\{d(p_i, p'_i)\}$ is minimized.

MinSum Problem: Given a set of sensors S located in the plane at positions $p_1, p_2, \dots, p_n$, and a set of barriers $\mathcal{B}$, find final positions $p'_1, p'_2, \dots, p'_n$ on the barriers so that all barriers in $\mathcal{B}$ are covered, and $\sum_{i=1}^{n} d(p_i, p'_i)$ is minimized.

1.2 Our Results

Our results are summarized in Table 1. Throughout the paper, we consider the barrier coverage problem with sensors of arbitrary ranges, initially located at arbitrary locations in the plane. In Section 2, we assume that sensors can move to arbitrary positions on any of the barriers. While feasibility is trivial in the case of one barrier, it is straightforward to show that it is NP-complete for even two barriers. The NP-completeness of the MinSum problem for one barrier follows trivially from the result in [5]. In this paper, we show that the MinMax problem is NP-complete even for a single barrier. We show that this holds both when the cost measure is Euclidean distance and when it is rectilinear distance.

In light of these hardness results, in the rest of the paper, we consider a more restricted but natural movement. We assume that once a sensor has been ordered to relocate to a particular barrier, it moves to the closest point on the line containing the barrier. We call this *perpendicular movement* [1]. Section 3 considers the case of one barrier and perpendicular movement, while Section 4 considers the case of perpendicular movement and multiple *parallel* barriers. We show that all three of our problems are solvable in polynomial time. Finally, in Section 5, we consider the case of perpendicular movement and two barriers perpendicular to each other. We show that even the feasibility problem is NP-complete in this case. The NP-completeness result holds even in the case when the given positions of the sensors have integer values and the sensing ranges of sensors are limited to two different sensing ranges. In contrast, we give an $O(n^{1.5})$ algorithm for finding a covering assignment for a natural restriction of the problem that includes the case when all sensors are located in integer positions and the sensing ranges of all sensors are of diameter 1.

Table 1. Summary of our results

Barriers	Movement	Feasibility	MinMax	MinSum
1 barrier	Arbitrary final positions	$O(n)$	NPC	NPC [5]
2 barriers	Arbitrary final positions	NPC	NPC	NPC
1 barrier	Perpendicular	$O(n)$	$O(n \log n)$	$O(n^2)$
k parallel barriers	Perpendicular	$O(kn)$	$O(kn^{k+1})$	$O(kn^{k+1})$
2 perpendicular barriers	Perpendicular	NPC	NPC	NPC

2 Arbitrary Final Positions

In this section, we assume that sensors are allowed to relocate to any final positions on the barrier(s). We consider first the case of a single barrier b. Without loss of generality, we assume that b is located on the x-axis between $(0, 0)$ and $(L, 0)$ for some L. The feasibility of barrier coverage in this case is simply a matter of checking if $\Sigma_{i=1}^{n} 2r_i \geq L$. For the MinSum problem, it was shown in [5] that even if the initial positions of sensors are on the line containing the barrier, the problem is NP-complete; therefore the more general version of the problem studied here is clearly NP-complete. Recently, it was shown in [3] that if the initial positions of sensors are on the line containing the barrier, the MinMax problem is solvable in polynomial time. The complexity of the MinMax problem for general initial positions in the plane has not yet been studied and thus we proceed to study the complexity of the MinMax problem when initial positions of sensors can be anywhere on the plane, and the final positions can be anywhere on the barrier. See Figure 2 for an example of the initial placement of sensors.

[1] Note that it is possible for a sensor that is not located on the barrier to cover part of the barrier. However, we require final positions of sensors to be on the barrier.

Theorem 1. *Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of sensors of ranges $r_1, r_2, \ldots, r_n$ initially located in the plane at positions $p_1, p_2, \ldots, p_n$. Let the barrier b be a line segment between $(0, 0)$ and $(L, 0)$. Given an integer k, the problem of determining if there is a covering assignment such that the maximum relocation distance (Euclidean/rectilinear) of the sensors is at most k is NP-complete.*

Proof. The problem is trivially in NP; we give here a reduction from the 3-partition problem. We are given a multiset $A = \{a_1 \geq a_2 \geq \cdots \geq a_n\}$ of $n = 3m$ positive integers such that $B/4 < a_i < B/2$ for $1 \leq i \leq n$ and $\sum_{i=1}^{n} a_i = mB$ for some B. The problem is to decide whether A can be partitioned into m triples $T_1, T_2, \ldots, T_m$ such that the sum of the numbers in each triple is equal to B. We create an instance of the barrier coverage problem as follows: Let $L = mB+m-1$ so that the barrier b is a line segment from $(0, 0)$ to $(L, 0)$, and let $k = L + 1$. Create a sensor s_i of radius $a_i/2$ for every $1 \leq i \leq 3m$ positioned at $-a_i/2$. In addition, create $m - 1$ sensors $s_{3m+1}, s_{3m+2}, \ldots, s_{4m-1}$ of range $1/2$ located at positions $(B+1/2, k), (2B+3/2, k), (3B+5/2, k), \ldots, ((m-1)B+(2m-3)/2, k)$. See Figure 2 for an example. Since $L = \sum_{i=1}^{n} 2r_i$, all sensors must move to the barrier. It is easy to verify that there is a partition of S into m triples $T_1, T_2, \ldots, T_m$, the sum of each triple being B, if and only if there is a solution to the movement of the sensors such that the three sensors corresponding to triple T_i are moved to fill the ith gap in the barrier b and all moves are at most of length k. □

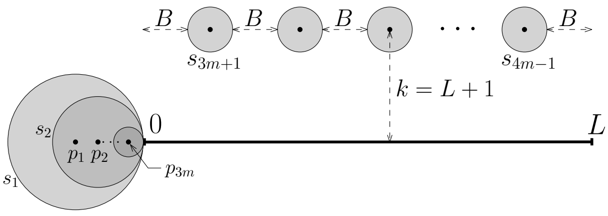


Fig. 2. Reduction from 3-partition to the MinMax problem

It is easy to see that when there are two barriers to be covered, even feasibility of coverage is NP-complete. This can be shown by reducing the Partition problem to an appropriate 2-barrier coverage problem, as in [4]. It follows that k-barrier coverage is also NP-complete.

3 Perpendicular Movement: One Barrier

In this section, we assume that all sensors use only perpendicular movement. Without loss of generality, let the barrier b be the line segment between $(0, 0)$ and $(L, 0)$ and let the set of n sensors $s_1, s_2, \ldots, s_n$ be initially located at positions $p_1, p_2, \ldots, p_n$ respectively, where $p_i = (x_i, y_i)$ and $x_1 - r_1 \leq x_2 - r_2 \leq \cdots \leq x_n - r_n$. For simplicity we assume all points of interest (sensor locations, left/right endpoints of sensor ranges and barriers) are distinct. Since the y-coordinate of all points on the barrier are the same, we sometimes represent the barrier or a segment of the barrier by an interval of x-coordinates. For technical reasons, we denote the segment of the barrier between the points $(i, 0)$ and $(j, 0)$ by the *half-open interval* $[i, j)$. Similarly, we consider the interval on the barrier that a sensor can cover to be a half-open interval.

We first show a necessary and sufficient condition on the sensors for the barrier to be covered. We say that the sensor s at position $p = (x, y)$ is a *candidate sensor* for $q = (x', 0)$ on the barrier if $x - r \leq x' < x + r$. Alternatively we say *s potentially covers* the point q. Clearly, the barrier b can be covered only if every point on the barrier has a candidate sensor. Conversely, if every point has a candidate sensor, the problem can be solved in linear time by simply moving all sensor nodes down to the barrier.

We give a dynamic programming formulation for the MinSum problem. We denote the set of sensors $\{s_i, s_{i+1}, \ldots, s_n\}$ by S_i. If the barrier is an empty interval, then the cost is 0. If no sensor is a candidate for the left endpoint of the barrier, or if the sensor set is empty while the barrier is a non-empty interval, then clearly the problem is infeasible and the cost is infinity. If not, observe that the optimal solution to the MinSum problem either involves moving sensor s_1 to the barrier or it doesn't. In the first case, the cost of the optimal solution is the sum of $|y_1|$, the cost of moving the first sensor to the barrier, and the optimal cost of the subproblem of covering the interval $[x_1 + r_1, L)$ with the remaining sensors $S_2 = S - \{s_1\}$. In the second case, the optimal solution is the optimal cost of covering the original interval $[0, L)$ with S_2. The recursive formulation is given below:

$$cost(S_i, [a, L)) = \begin{cases} 0 & \text{if } L < a \\ \infty & \text{if } x_i - r_i > a \text{ or } (S_i = \emptyset \text{ and } L > a) \\ min \begin{cases} |y_i| + cost(S_{i+1}, [x_i + r_i, L)), \\ cost(S_{i+1}, [a, L)) \end{cases} & \text{otherwise} \end{cases}$$

Observe that a subproblem is always defined by a set S_i and a left endpoint to the barrier which is given by the rightmost x-coordinate covered by a sensor. Thus the number of possible subproblems is $O(n^2)$, and it takes constant time to compute $cost(S_i, [a, L))$ given the solutions to the sub-problems. Using either a tabular method or memoization, the problem can be solved in quadratic time. The same dynamic programming formulation works for minimizing the maximum movement, except that in the case when the i-th sensor moves to the barrier in the optimal solution, the cost is the maximum of $|y_i|$ and $cost(S_{i+1}, [x_i + r_i, L))$ instead of their sum. A better approach is to check the feasibility of covering the barrier with the subset of sensors at distance at most d from the barrier in $O(n)$ time and find the minimum value of d using binary search on the set of distances of all sensors to the barrier. This yields an $O(n \log n)$ algorithm for MinMax.

Theorem 2. *Let $s_1, s_2, \ldots, s_n$ be n sensors initially located in the plane at positions $p_1, p_2, \ldots, p_n$ respectively, and let b be a barrier between $(0, 0)$ and $(L, 0)$. The MinSum problem using only perpendicular movement can be solved in $O(n^2)$ time, and the MinMax problem can be solved in $O(n)$ time.*

4 Perpendicular Movement: Multiple Parallel Barriers

In this section, we study the problem of covering multiple *parallel* barriers. We assume that sensors can relocate to any of the barriers, but will use perpendicular movement. Without loss of generality, we assume all barriers are parallel to the x-axis. Since there are k barriers, there are k points on barriers with the same x-coordinate. We therefore speak of sensors being *candidates for x-coordinates*: a sensor s in position $p = (x, y)$ with sensing range r is a candidate sensor for x-coordinate x' if $x - r \leq x' < x + r$. Clearly, such a sensor is a candidate for a point q on a border with x-coordinate x'. We say an interval $I = [a, b)$ of x-coordinates is *k-coverable* if every x-coordinate in the interval has k candidate sensors; such an interval of x-coordinates can exist on multiple barriers.

For simplicity, we explain the case of two barriers; the results for the feasibility and MinSum problems generalize to k barriers. Assume without loss of generality that the two barriers to be covered are b_1 between $(0, 0)$ and $(L, 0)$ and b_2 between $(0, W)$ and (L, W) and the set of n sensors $s_1, s_2, \ldots, s_n$ to be initially located at positions $p_1, p_2, \ldots, p_n$ respectively, and is ordered by the $x_i - r_i$ values as in Section 3. Sensors may only move in a vertical direction. We assume that the sensing radii of sensors are smaller than half the distance W between the two barriers, and thus it is impossible for a sensor to simultaneously cover two barriers. See Figure 3 for an example of such a problem.

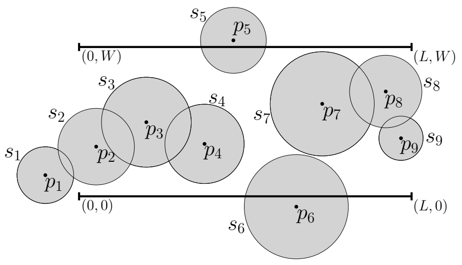


Fig. 3. An example of a barrier coverage problem with two parallel barriers

We first show a necessary and sufficient condition on the sensors for the two barriers to be covered. Clearly, since the sensing radius of every sensors is smaller than half of the distance between the two barriers, the barrier coverage problem for the two parallel barriers b_1 and b_2 above is solvable by a set of sensors S *only if* the interval $[0, L)$ is 2-coverable by S. We proceed to show that 2-coverability is also a sufficient condition, and give a $O(n)$ algorithm for finding a covering assignment for two parallel barriers. To simplify the proof of the main theorem, we first prove a lemma that considers a slightly more general version of the two parallel barrier problem.

Lemma 1. *Let $s_1, s_2, \ldots, s_n$ be sensors located at positions $p_1, p_2, \ldots, p_n$ respectively where $p_i = (x_i, y_i)$ and $x_1 - r_1 \leq x_2 - r_2 \leq \cdots x_n - r_n$. Let b_1 between*

$(0,0)$ and $(L,0)$ and b_2 between (P,W) and (Q,W), where $0 \leq P < L \leq Q$, be two parallel barriers to be covered. If intervals $[0,P)$ and $[L,Q)$ are 1-coverable, and interval $[P,L)$ is 2-coverable, then a covering assignment that uses only perpendicular movement of the sensors can be obtained in $O(n)$ time.

Proof. Omitted due to lack of space. □

The above lemma establishes that complete coverage of two parallel barriers b_1 between $(0,0)$ and $(L,0)$ and b_2 between $(0,W)$ and (L,W) can be achieved if and only if the interval of x-coordinates $[0,L]$ is 2-covered, and a covering assignment can be found in linear time. It is easy to see that the lemma can be generalized for k barriers to show that the feasibility problem can be solved in $O(kn)$ time. We proceed to study the problem of minimizing the sum of movements required to perform barrier coverage.

The dynamic programming formulation given in Section 3 can be generalized for the case of two barriers. The key difference is that in an optimal solution, sensor s_i may be used to cover a part of barrier b_1 or barrier b_2 or neither. Let $xcost(S_i,[a_1,L),[a_2,L))$ denote the cost of covering the interval $[a_1,L)$ of the barrier b_1 and the interval $[a_2,L)$ of the second barrier with the sensor set $S_i = \{s_i, s_{i+1}, \ldots, s_n\}$. The optimal cost is given by the formulation below:

$$xcost(S_i,[a_1,L),[a_2,L)) =$$
$$= \begin{cases} cost(S_i,[a_2,L)) & \text{if } L < a_1 \\ cost(S_i,[a_1,L)) & \text{if } L < a_2 \\ \infty & \text{if } x_i - r_i > min\{a_1,a_2\} \text{ or } (S_i = \emptyset \text{ and } L > min\{a_1,a_2\}) \\ min \begin{cases} |y_i| + xcost(S_{i+1},[x_i+r_i,L),[a_2,L)), \\ |W-y_i| + xcost(S_{i+1},[a_1,L),[x_i+r_i,L)), \\ xcost(S_{i+1},[a_1,L),[a_2,L)) \end{cases} & \text{otherwise} \end{cases}$$

It is not hard to see that the formulation can be generalized to k barriers; a sensor s_i may move to any of the k barriers with the corresponding cost being added to the solution. Observe that a subproblem is now given by a set S_i, and a left endpoint to each of the barriers. The total number of subproblems is $O(n^{k+1})$ and the time needed to compute the cost of a problem given the costs of the subproblems is $O(k)$. Thus, the time needed to solve the problem is $O(kn^{k+1})$.

Theorem 3. *Let $s_1, s_2, \ldots, s_n$ be n sensors initially located in the plane at positions $p_1, p_2, \ldots, p_n$ respectively where $p_i = (x_i, y_i)$ and $x_1 - r_1 \leq x_2 - r_2 \leq \cdots x_n - r_n$. The MinSum problem for k parallel barriers using only perpendicular movement can be solved in $O(kn^{k+1})$ time.*

Clearly a very similar formulation as above can be used to solve the MinMax problem in $O(kn^{k+1})$ time as well. However, the approach used for a single barrier can be generalized for two barriers, as shown in the theorem below:

Theorem 4. *Let $s_1, s_2, \ldots, s_n$ be n sensors initially located in the plane at positions $p_1, p_2, \ldots, p_n$ respectively, and let b_1 between $(0,0)$ and $(L,0)$ and b_2*

between $(0, W)$ and (L, W) be the two parallel barriers to be covered. The Min-Max problem for the 2 parallel barriers using only perpendicular movement can be solved in $O(n \log n)$ time.

Proof. Omitted due to lack of space. □

5 Perpendicular Movement: Two Perpendicular Barriers

In this section, we consider the problem of covering two perpendicular barriers. Once again, we assume that sensors can relocate to either of the two barriers, but will use perpendicular movement. In contrast to the case of parallel barriers, we show here that even the feasibility problem in this case is NP-complete. Figure 4 illustrates an example of such a problem. For simplicity we assume that b_1 is a segment on the x-axis between $(0, 0), (L_1, 0)$ and b_2 is a segment on the y-axis between $(0, 0), (0, L_2)$. Since the sensors can only employ perpendicular movement, the only possible final positions on the barriers for a sensor s_i in position $p_i = (x_i, y_i)$ are $p'_i = (0, y_i)$ or $p'_i = (x_i, 0)$.

We first show that the feasibility problem for this case is NP-complete by giving a reduction from the monotone 3-SAT problem [6]. Recall that a Boolean 3-CNF formula $f = c_1 \wedge c_2 \wedge ... \wedge c_m$ of m clauses is called *monotone* if and only if every clause c_i in f either contains only unnegated literals or only negated literals. In order to obtain a reduction into a barrier coverage problem with two perpendicular barriers, we first put a monotone 3-SAT formula in a special form as shown in the lemma below.

Lemma 2. *Let $f = f_1 \wedge f_2$ be a monotone 3-CNF Boolean formula with n clauses where f_1 and f_2 only contain unnegated and negated literals respectively, and every literal appears in at most m clauses. Then f can be transformed into a monotone formula $f' = f'_1 \wedge f'_2$ such that f'_1 and f'_2 have only unnegated and negated literals respectively, and f' has the following properties:*

1. *f and f' are equisatisfiable, i.e. f' is satisfiable if and only if f is satisfiable.*
2. *All clauses are of size two or three.*
3. *Clauses of size two contain exactly one variable from f and one new variable.*
4. *Clauses of size three contain only new variables.*
5. *Each new literal appears exactly once: either in a clause of size two or in a clause of size three.*
6. *Each x_i appears exactly in m clauses of f'_1, and exactly in m clauses of f'_2.*
7. *f' contains at most $4n + mn$ clauses.*
8. *The clauses in f'_1 (respectively f'_2) can be ordered so that all clauses containing the literal x_i ($\overline{x_i}$) appear before clauses containing the literal x_j (respectively $\overline{x_j}$) for $i < j$, and all clauses of size three are placed last.*

Proof. Omitted due to lack of space. □

We give an example that illustrates the reduction and the ordering specified in Property 7.

Example 1: Consider 3-CNF formula

$$f = (x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4) \wedge (\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4)$$

An equisatisfiable formula f' satisfying the properties of Lemma 2 is:

$$\begin{aligned}
f' = &(x_1 \vee x_{1,1}) \wedge (x_1 \vee x_{1,3}) \wedge (x_1 \vee y_{1,1}) \wedge (x_2 \vee x_{2,2}) \wedge (x_2 \vee x_{2,3}) \\
\wedge\ &(x_2 \vee y_{2,1}) \wedge (x_3 \vee x_{3,1}) \wedge (x_3 \vee x_{3,2}) \wedge (x_3 \vee x_{3,3}) \wedge (x_4 \vee x_{4,1}) \\
\wedge\ &(x_4 \vee x_{4,2}) \wedge (x_4 \vee y_{4,1}) \wedge (x_{1,4} \vee x_{2,4} \vee x_{4,4}) \wedge (x_{2,5} \vee x_{3,5} \vee x_{4,5}) \\
\wedge\ &(\overline{x_1} \vee \overline{x_{1,4}}) \wedge (\overline{x_1} \vee \overline{z_{1,1}}) \wedge (\overline{x_1} \vee \overline{z_{1,2}}) \wedge (\overline{x_2} \vee \overline{x_{2,4}}) \wedge (\overline{x_2} \vee \overline{x_{2,5}}) \wedge (\overline{x_2} \vee \overline{z_{2,1}}) \\
\wedge\ &(\overline{x_3} \vee \overline{x_{3,5}}) \wedge (\overline{x_3} \vee \overline{z_{3,1}}) \wedge (\overline{x_3} \vee \overline{z_{3,2}}) \wedge (\overline{x_4} \vee \overline{x_{4,4}}) \wedge (\overline{x_4} \vee \overline{x_{4,5}}) \wedge (\overline{x_4} \vee \overline{z_{4,1}}) \\
\wedge\ &(\overline{x_{1,1}} \vee \overline{x_{3,1}} \vee \overline{x_{4,1}}) \wedge (\overline{x_{2,2}} \vee \overline{x_{3,2}} \vee \overline{x_{4,2}}) \wedge (\overline{x_{1,3}} \vee \overline{x_{2,3}} \vee \overline{x_{3,3}})
\end{aligned}$$

Theorem 5. *Let $s_1, s_2, \ldots, s_n$ be n sensors initially located in the plane at positions $p_1, p_2, \ldots, p_n$ respectively, and let b_1 between $(0,0)$ and $(L_1, 0)$ and b_2 between $(0,0)$ and $(0, L_2)$ be the two perpendicular barriers to be covered. Then the problem of finding a covering assignment using perpendicular movement for the two barriers is NP-complete.*

Proof. It is easy to see that any given covering assignment can be verified in polynomial time. Given a monotone 3-SAT formula f, we use the construction described in Lemma 2 to obtain a formula $f' = f_1' \wedge f_2'$ satisfying the properties stated in Lemma 2 with clauses ordered as described in Property 7. Let f_1 have i_1 clauses, and f_2 have i_2 clauses, and assume the clauses in each are numbered from $1, \ldots, i_1$ and $1, \ldots, i_2$ respectively. We create an instance P of the barrier coverage problem with two barriers b_1, the line segment between $(0,0)$ and $(2i_1, 0)$ and b_2, the line segment between $(0,0)$, and $(0, 2i_2)$.

For each variable x_i of the original formula f we have a sensor s_i of sensing range m located in position $p_i = ((2i-1)m, (2i-1)m)$, i.e., on the diagonal. Figure 4 illustrates the instance of barrier coverage corresponding to the 3-SAT formula from Example 5 above. Each of the variables $x_{i,j}, y_{i,j}, z_{i,j}$ is represented by a sensor of sensing range 1, denoted $s_{i,j}$, $s'_{i,j}$, and $s''_{i,j}$ respectively, and is placed in such a manner that the sensors corresponding to variables associated with the same s_i collectively cover the same parts of the two barriers as covered by sensor s_i. Furthermore, sensors corresponding to variables that appear in the same clause of size three cover exactly the same segment of a barrier. A sensor corresponding to a new variable $x_{i,j}$ that occurs in the pth clause in f_1' and in the qth clause in f_2' is placed in position $(2p-1, 2q-1)$. For example the sensor $s_{1,3}$ corresponding to the variable $x_{1,3}$ appears in the second clause of f_1' and the fifteenth clause of f_2', and hence is placed at position $(3, 29)$. Similarly, the sensor $s_{2,4}$ corresponding to the variable $x_{2,4}$ appears in the thirteenth clause of f_1' and the fourth clause of f_2', and hence is placed at position $(25, 7)$. A sensor corresponding to variable $y_{i,j}$ which occurs in the ℓth clause in f_1' is placed in position $(2\ell-1, -1)$ and sensor corresponding to variable $z_{i,j}$ which occurs in the ℓth clause of f_2' is placed in position $(-1, 2\ell-1)$. Observe that in this assignment of positions to sensors, for any i, there is a one-to-one correspondence between

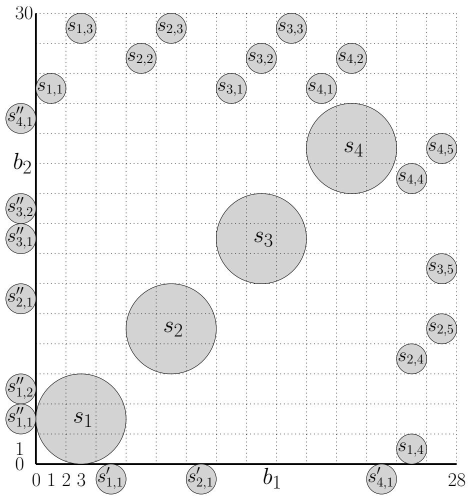


Fig. 4. Barrier coverage instance corresponding to Example 1

the line segments of length 2 in b_1 and b_2 and clauses in f'_1 and f'_2 respectively. In particular, the sensors that potentially cover the line segment from $(2i-2,0)$ to $(2i,0)$ on the barrier b_1 correspond to variables in clause i of f'_1. Similarly, the sensors that potentially cover the line segment from $(0,2i-2)$ to $(0,2i)$ on the barrier b_2 correspond to variables in clause i of f'_2. It is easy to verify that f' is satisfiable iff for the corresponding instance P there exists a covering assignment assuming perpendicular movement. $\square$

Since any instance of monotone 3-SAT problem can be transformed into one in which no literal occurs more than 4 times, it follows from the proof that the problem is NP-complete even when the sensors are in integer positions and the diameter of sensors are limited to two different sizes 1 and $m \geq 4$. It is also clear from the proof that the perpendicularity of the barriers is not critical. The key issue is that the order of intervals covered by the sensors in one barrier has no relationship to those covered in the other barrier. In the case of parallel barriers, this property does not hold. The exact characterization of barriers for which a polytime algorithm is possible remains an open question.

We now turn our attention to restricted versions of barrier coverage of two perpendicular barriers where a polytime algorithm is possible. For S a set of sensors, and barriers b_1, b_2, we call (S, b_1, b_2) a *non-overlapping arrangement* if for any two sensors $s_i, s_j \in S$, the intervals that are potentially covered by s_1 and s_2 on the barrier b_1 (and b_2) are either the same or disjoint. This would be the case, for example, if all sensor ranges are of the same diameter equal to 1 and the sensors are in integer positions. We show below that for a non-overlapping arrangement, the problem of finding a covering assignment is polynomial.

Theorem 6. *Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of n sensors initially located in the plane at positions $p_1, p_2, \ldots, p_n$ and let b_1 and b_2 be two perpendicular barriers to be covered. If (S, b_1, b_2) form a non-overlapping arrangement, then there exists an $O(n^{1.5})$ algorithm that finds a covering assignment, using only perpendicular movement or reports that none exists.*

Proof. Omitted due to lack of space. □

6 Conclusions

It is known that the MinMax barrier coverage problem when the sensors are initially located on the line containing the barrier is solvable in polynomial time [3]. In contrast, our results show that the same problem becomes NP-complete when sensors of arbitrary ranges are initially located in the plane and are allowed to move to any final positions on the barrier. It remains open whether this problem is polynomial in the case when there is a fixed number of possible sensor ranges. If sensors are restricted to use perpendicular movement, the feasibility, MinMax, and MinSum problems are all polytime solvable for the case of k parallel barriers. However, when the barriers are not parallel, even the feasibility problem is NP-complete, even when sensor ranges are restricted to two distinct values. It would be therefore interesting to study approximation algorithms for MinMax and MinSum for this case. Characterizing the problems for which barrier coverage is achievable in polytime remains an open question.

References

[1] Balister, P., Bollobas, B., Sarkar, A., Kumar, S.: Reliable density estimates for coverage and connectivity in thin strips of finite length. In: Proceedings of MobiCom 2007, pp. 75–86 (2007)

[2] Bhattacharya, B., Burmester, M., Hu, Y., Kranakis, E., Shi, Q., Wiese, A.: Optimal movement of mobile sensors for barrier coverage of a planar region. Theoretical Computer Science 410(52), 5515–5528 (2009)

[3] Chen, D.Z., Gu, Y., Li, J., Wang, H.: Algorithms on minimizing the maximum sensor movement for barrier coverage of a linear domain. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 177–188. Springer, Heidelberg (2012)

[4] Czyzowicz, J., Kranakis, E., Krizanc, D., Lambadaris, I., Narayanan, L., Opatrny, J., Stacho, L., Urrutia, J., Yazdani, M.: On minimizing the maximum sensor movement for barrier coverage of a line segment. In: Ruiz, P.M., Garcia-Luna-Aceves, J.J. (eds.) ADHOC-NOW 2009. LNCS, vol. 5793, pp. 194–212. Springer, Heidelberg (2009)

[5] Czyzowicz, J., Kranakis, E., Krizanc, D., Lambadaris, I., Narayanan, L., Opatrny, J., Stacho, L., Urrutia, J., Yazdani, M.: On minimizing the sum of sensor movements for barrier coverage of a line segment. In: Nikolaidis, I., Wu, K. (eds.) ADHOC-NOW 2010. LNCS, vol. 6288, pp. 29–42. Springer, Heidelberg (2010)

[6] Gold, E.M.: Complexity of automaton identification from given data. Information and Control 37(3), 302–320 (1987)

[7] Huang, C.F., Tseng, Y.C.: The coverage problem in a wireless sensor network. In: Proceedings of WSNA, pp. 115–121 (2003)

[8] Kumar, S., Lai, T.H., Arora, A.: Barrier coverage with wireless sensors. In: Proceedings of MobiCom 2005, pp. 284–298 (2005)

[9] Meguerdichian, S., Koushanfar, F., Potkonjak, M., Srivastava, M.B.: Coverage problems in wireless ad-hoc sensor networks. In: Proceedings of INFOCOM 2001, vol. 3, pp. 1380–1387 (2001)

[10] Mehrandish, M., Narayanan, L., Opatrny, J.: Minimizing the number of sensors moved on line barriers. In: Proceedings of IEEE WCNC 2011, pp. 1464–1469 (2011)

[11] Yan, G., Qiao, D.: Multi-round sensor deployment for guaranteed barrier coverage. In: Proceedings of IEEE INFOCOM 2010, pp. 2462–2470 (2010)

Succinct Permanent Is *NEXP*-Hard with Many Hard Instances[⋆]
(Extended Abstract)

Shlomi Dolev[1], Nova Fandina[1], and Dan Gutfreund[2]

[1] Department of Computer Science
Ben Gurion University of the Negev, Israel
[2] IBM Research, Tel Aviv, Israel
{dolev,fandina}@cs.bgu.ac.il, danny.gutfreund@gmail.com

Abstract. The main motivation of this work is to study the average case hardness of the problems which belong to high complexity classes. In more detail, we are interested in provable hard problems which have a big set of hard instances. Moreover, we consider efficient generators of these hard instances of the problems. Our investigation has possible applications in cryptography. As a first step, we consider computational problems from the $NEXP$ class.

We extend techniques presented in [7] in order to develop efficient generation of hard instances of exponentially hard problems. Particularly, for any given polynomial time (deterministic/probabilistic) heuristic claiming to solve $NEXP$ hard problem our procedure finds instances on which the heuristic errs. Then we present techniques for generating hard instances for (super polynomial but) sub exponential time heuristics.

As a concrete example the Succinct Permanent mod p problem is chosen. First, we prove the $NEXP$ hardness of this problem (via randomized polynomial time reduction). Next, for any given polynomial time heuristic we construct hard instance. Finally, an efficient technique which expands one hard instance to exponential set (in the number of additional bits added to the found instance) of hard instances of the Succinct Permanent mod p problem is provided.

1 Introduction

Computationally hard problems play important role in the theory of computer science. The main implications of such problems are in modern cryptography,

⋆ Partially supported by the Israeli Ministry of Science (Russia Israel grant), Rita Altura Trust Chair in Computer Sciences, Lynne and William Frankel Center for Computer Sciences, Israel Science Foundation (grant number 428/11), Cabarnit Cyber Security MAGNET Consortium, Grant from the Technion's Institute for Future Defense Technologies Research named for the Medvedi, Shwartzman and Gensler families, and the Israeli Internet Association.

where the hardness of the problem is necessary in order to build secure cryptographic primitives. Yet for the most of the current cryptographic schemes the worst case hardness does not suffice, rather the hardness on the average of the problem is employed in construction. In addition, some of these schemes require the possibility to explicitly (and efficiently) construct hard instances of the problem. For example, the implementation of the Merkle's Puzzles technique [10] may use the instances of such problems as puzzles. Moreover, the current state of complexity theory does not enable to prove the hardness of the problems which are usually used in cryptography.

Thus, the main aim of this work is to investigate computational problems which are provably hard (against deterministic/probabilistic polynomial time solvers), and to develop technique for efficient generation of hard instances of these problems. Since the central motivation comes from cryptography, we focus our attention on providing specific problems of desired type which are appropriate for practical tasks. As a first step towards achieving general results we propose to study a concrete problem. Particularly, the succinct permanent modulo a prime problem is chosen to be considered through the paper.

The scheme of succinct representation of graphs (or matrices) was proposed by Galperin and Wigderson [6]. The authors suggest to represent graph with the short boolean circuit which computes the neighborhood relation of any two given vertices. Further, authors consider and analyze various computational problems assuming the succinctly represented instances. The general result was stated and proved by Papadimitriou and Yannakakis [12].

Theorem 1. *Let A be some computational problem on graphs. Then if $3SAT$ can be reduced via reduction-projection to A, then $Succ - A$ problem is $NEXP$ time hard.*

The *permanent* is a candidate for a problem with a big fraction of hard instances. The reason is the random self-reducibility property of the permanent problem: solutions of a small random set of instances solve the given instance [9]. Namely, assuming the hardness of the permanent on the worst (randomized) case implies the hardness of the permanent on the average case.

Therefore, according to the above arguments, we investigate the permanent problem assuming that instances are given in succinct representation.

Related Work. There are many results showing an equivalence of the worst case and average case hardness for problems of high complexity classes, basically $PSPACE$ and above [8,3,15,14]. The most recent work on this line of research is by Trevisan and Vadhan [15]. The authors show that if $BPP \neq EXP$, then EXP has problems which cannot be solved by any BPP algorithm on even $(1/2 + 1/poly(n))$ fraction of the inputs. Meaning that these problems are hard for all efficient algorithms with respect to uniform distribution over the inputs. These hard on average problems are obtained by encoding (using error correcting schemes in the style used in probabilistically checkable proofs) the entire truth table of another hard in the worst case problem, which we call the *source problem*. Thus, roughly speaking, ensuring the identification of the value of a certain bit in the encoded truth table requires knowledge of the solutions of all instances of the source problem. And hence, every instance (bit in the encoded truth table) is as hard to compute as the worst case instance of the source problem. Therefore, revealing any bit in the encoded truth table has similar hardness, implying hardness on the average case over the chosen indices.

Unfortunately, these languages may have the following property. After investing an exponential effort on solving some instance of the truth table encoding language, one have, in some sense, the information about solutions of all the instances of the source problem. Thus, it may happen, that there is an efficient way to compute the solutions of all instances (of the same length) of the new encoded bits, using the solution of the truth table encoding language, and the information regarding solutions of the instances of the source problem (similar to the idea of the rolling hash function).

This property of the reduction can be a drawback when using it in cryptographic applications. In other words, if many instances of the problem are used (perhaps as part of cryptographic primitives), one may invest time to solve some small set of the encoded bit instances, and thus to nullify the hardness of all other encoded bit instances at once.

For example, it may turn out that it is impossible to use the obtained hard on average bits as puzzles in the Merkle's puzzle scheme. In this scheme, two parties, Alice and Bob, wish to establish a shared secret key. The only way Alice and Bob can communicate is by using a public insecure channel. The proposed protocol in [10] is based on the concept of puzzles. A puzzle is the cryptogram that is meant to be broken. When it is broken (or solved), the information that was "en-puzzled" is revealed.

In one of the variants of the Merkle's scheme in order to agree on the secret key, Alice creates n pairs of puzzles (namely, $2n$ puzzles totally). We denote one pair of puzzles by $\langle P_I, P_S \rangle$. Alice sends all the pairs to Bob over the insecure channel. Then Alice and Bob each randomly choose and solve $O(\sqrt{n})$ pairs of the puzzles. They both invest the required amount of time to solve these puzzles. The size of the puzzles should be tuned to the capabilities of Alice and Bob and to a reasonable time that is needed to establish a key versus the required security parameter. After Alice and Bob finish the work, they send each other the solutions of the first puzzle in each pair of puzzles — the solutions of P_I. By the birthday paradox they both have, with high probability, at least one common pair of puzzles selection. Therefore, they have sent each other at least one the same string of bits which denotes the solution of the P_I part of the commonly selected puzzle. Now, Alice and Bob have the same secret key. Namely, the solution of the appropriate puzzle P_S.[1]

The security of this scheme is based on the lack of ability of the adversary, Eve, to know which puzzles Alice and Bob have chosen to solve. Therefore, to find out the secret, Eve has to solve (in the worst case) all (in fact $O(n)$) of the puzzles P_I that Alice sent. Suppose it takes n time to solve one puzzle, then to agree

[1] Another possibility is to solve only the P_I puzzles of all $\sqrt{n}$ chosen puzzles, and after detection of the common puzzle to solve an appropriate P_S puzzle.

on the key Alice and Bob will spend $O(n\sqrt{n})$ time, while Eve will spend $O(n^2)$ time. In addition, the important property of the puzzle on which the security of the scheme is based is its (proven) computational hardness. Furthermore, an essential property of the puzzles which is implicitly employed in the scheme is independence of solutions of different puzzles. Namely, the solution of one puzzle does not reveal solution of the another puzzle(s).

Suppose one implements a puzzle as the instance of the hard on average problem obtained from the truth table encoding reduction. In this case, the puzzles are bits in the encoded truth table. Therefore, finding a solution of some puzzle implies the need to solve all the (hard) instances of the source problem. Namely, the solutions of all the puzzles are based on the knowledge of the same bits and (it may turn out that) there is an efficient way to compute solutions of all the rest of the puzzles. Hence, Alice and Bob do not have an advantage over Eve.

Therefore, our interest is focused on providing an exponential hard on average problem which can be used in various cryptographic applications. For example when implementing the above scheme we want to be sure that there will not exist a polynomial time algorithm which solves the instances sent by Alice to Bob within the same or less time than it takes the communicating parties to establish a key (under the $BPP \neq NEXP$ assumption).[2]

Our Contribution. Following our goal we first prove that the succinct permanent modulo a prime problem is $NEXP$-hard (via randomized polynomial time reduction).

We then present general technique of constructing hard instances for a given polynomial time (deterministic) heuristic that claims to solve the $NEXP$ time hard problem. This construction does not assume any complexity assumption. In addition, we provide a new technique that can be interesting as an independent result. Specifically, we show how to efficiently generate interactively (in the standard model of multi-prover interactive proofs where the provers are computationally unlimited) a hard instance of a (deterministic) heuristic whose running time is larger than the verifier's (in particular, for super polynomial but sub exponential time heuristics). Both these techniques, though, are developed to work against one-sided error heuristics.

Furthermore, we consider randomized polynomial and super polynomial time heuristics and establish the following result. Assuming $BPP \neq NEXP$ and given any polynomial or (super polynomial) time randomized one-sided error heuristic claiming to solve $NEXP$ hard problem, there is an efficient (polynomial time) randomized procedure that generates a small set of the instances of the problem such that the heuristic errs on one of them with the high probability.

[2] We note that a similar approach can be used to cope with an adversary that can use randomized algorithms, by choosing instances that are hard in average, in the double exponential class or beyond. In such a class there is a need for exponential time even if all possible randomized selections are examined. Presburger Arithmetic may qualify for such a case, especially given the result on hardness of many instances that was obtained by Rybalov in [13].

As a consequence, our main result states that assuming $BPP \neq NEXP$ and given any polynomial time (deterministic or randomized) one-sided error heuristic, there is an efficient procedure that generates instances of the succinct permanent modulo a prime problem, such that the heuristic errs on them. Moreover, we present an efficient procedure which for any given hard instance of the succinct permanent modulo a prime produces exponentially many sets of hard instances. We then discuss the possibility of the existence of the following property of the generated sets: the solutions of the instances of some set do not reveal information concerning the solutions of the instances of another set.

Organization. Our first step in the work is to establish complexity result of the succinct permanent modulo a prime problem. In Section 2 we prove that the decision problem of whether the value of a permanent of a matrix is zero when the matrix is given in a succinct representation is *NEXP* time hard. We use the obtained result to prove that computing the permanent modulo a prime number is *NEXP* time hard (via a randomized reduction). Due to space constraints, we provide only key points of the proofs. Full proofs of all the statements appear in [4]. In Section 3 we turn to the general problem of constructing a hard instance of the $NEXP$ hard problem for any given heuristic. We present a polynomial search of finding hard instances in the case the heuristic is deterministic polynomial time algorithm (for the randomized polynomial time heuristic we make an assumption of $BPP \neq NEXP$ relation), and present a polynomial search that uses two provers in the case that the heuristic is deterministic super polynomial (or exponential, assuming *EXP* $\neq$ *NEXP*).

Finally, we present a procedure that expands any given hard instance of the succinct permanent modulo a prime to exponential number of sets of the instances. Each set consists of hard on average instances, where the exponential growth is relative to the number of bits added to the input.

2 The (Worst Case) Hardness of the Succinct Permanent mod p

In this section we present succinct permanent related problems and build chain of reductions between them in order to establish the computationally hardness of the problem of our interest.

2.1 Zero Succinct Permanent

We introduce the *Zero Succinct Permanent* problem and establish its complexity hardness.

Definition 1. *The Zero Succinct Permanent problem is defined by the following input and output:*
input: An $O(\log^k n)$ sized boolean circuit C which succinctly represents an $n \times n$ integer matrix A (with positive and negative polynomially bounded values) where k is some constant integer.
output: permanent$(A) == 0$.

Theorem 2. *Zero Succinct Permanent is $NEXP$ time hard.*

Proof. In [12] the authors have shown that the Succ-3SAT decision problem is *NEXP* time hard. We reduce this problem to the Zero Succinct Permanent on basis of the techniques presented in [16]. Technical details are presented in [4]. □

2.2 Succinct Permanent Modulo a Prime

In this section we define the problem we will focus on and establish its computational hardness.

Definition 2. *The Zero Succinct Permanent* mod *p problem is defined by the following input and output:*
input: An $O(\log^k n)$ sized boolean circuit C which succinctly represents an $n \times n$ integer matrix A (with positive and negative polynomially bounded values) where k is some constant integer.
p is a prime number, s.t. $p = O(n^k)$, given in a binary representation.
output: permanent(A) mod *p in binary representation.*

To prove the hardness of the defined problem, it is enough to prove the decision version of it. Namely, the problem that decides whether the permanent of a succinctly represented integer matrix is equal to zero mod p. We call this problem Zero Succinct Permanent mod p. In the previous section, we proved that Zero Succinct Permanent Problem (the same problem without modulo operation) is *NEXP* time hard in the worst case. In [4] we build a polynomial time randomized reduction from Zero Succinct Permanent to Zero Succinct Permanent mod p. Given an instance of the Zero Succinct Permanent problem, the reduction calls (a polynomial number of calls) an oracle of the Zero Succinct Permanent mod p problem on the randomly chosen instances in order to decide whether the permanent of the input matrix is zero. Using a Chinese Reminder Theorem randomized algorithm outputs correct answer with probability 1 if the permanent of the input matrix is zero. In case the permanent of the input matrix is non-zero, the probability of a correct answer is at least $\frac{1}{2}$.

Note that we can assume that the input encoded matrices have only positive values, from the field Z_p (when p is also given as a part of the input). Given an input that encodes a matrix with negative values, we can add an additional small boolean circuit that performs appropriate arithmetical operations to obtain an equivalent mod p positive valued matrix. The permanent value of the new matrix under modulo operation is not changed.

3 Finding Hard Instance for a Given Heuristic

The hardness of the $NEXP$-complete problems we have discussed above is a *worst case* hardness. Namely, given any polynomial time deterministic (or nondeterministic) algorithm claiming to solve the problem, there are infinitely many integers

n, such that the algorithm errs on solving at least one instance of length n. This is due to the fact $P \subset NEXP$ ($NP \subset NEXP$). An interesting question is whether we can efficiently produce hard instances of the problem. In [7], the authors present a technique that provides hard distributions of the inputs for heuristics (deterministic and randomized) attempting to solve NP-complete problems. However, to produce a hard instance of some length n, their technique consumes more time than is required for heuristics to solve this instance.

In this section, we will adapt the technique in [7] to provide a hard distribution for heuristics that attempt to solve $NEXP$-complete problems. Obviously, this technique inherits the disadvantage mentioned above. To overcome this obstacle, we use the idea of two-prover interactive protocols that was proposed and discussed in [2]. We present a new method to generate hard distributions of $NEXP$ problems against superpolynomial time heuristics.

To generate hard instances for the Succinct Permanent mod p problem, it is enough to show how to efficiently generate a hard distribution of any specific $NEXP$-complete problem. To obtain a hard distribution of any other complete language, we apply many-one reduction. In particular, we are considering hard distributions of the Succ-3SAT problem.

This section is organized as follows: first, we discuss polynomial time heuristics, both deterministic and randomized; next, we observe the case of superpolynomial time heuristics.

3.1 Polynomial Time Heuristics

Deterministic Case. Assume we are given deterministic polynomial time algorithm B claiming to solve the Succ-3SAT problem. The goal is to generate hard instances for heuristic B. However, the result we have established so far is considering some special type of heuristics, namely, heuristics that have only one-sided error. Suppose we are given algorithm B such that if B answers "yes" on the input x, then it is assumed that indeed it holds that x is in the language. From now, we assume that the heuristic trying to solve the Succ-3SAT problem satisfies the above requirement. Next, we describe a technique that generates a set of instances of the problem that B fails to solve. We use an idea that was proposed by Gutfreund, Shaltiel and Ta-Shma [7]. In their paper, they describe a procedure that outputs a set of hard instances of the 3SAT problem. We modify their technique in order to apply it to our case and formulate the following lemma, that is a $NEXP$ version of Lemma 3.1 of [7].

Lemma 1. *There is a deterministic procedure R, a polynomial $q()$ and a constant d, such that the procedure R gets three inputs: integers* n *and* a *and a description of a deterministic machine B, such that B claims to solve the Succ-3SAT problem and B has one-sided error. The procedure R runs in time n^{da^2} and outputs at most two boolean circuits where the length of each circuit is either* n *or $q(n^a)$. Furthermore, if B is an algorithm that runs in time bounded by n^a on inputs of length* n *(for some constant* a*), then for infinitely many input lengths* n*, the invocation of $R(n, a, B)$ gives a set F of boolean circuits, such that there exists $C \in F$ with Succ-3SAT$(C) \neq B(C)$.*

Proof. We know that Succ-3SAT has no deterministic polynomial time algorithm. Therefore, there are infinitely many natural numbers n, such that the heuristic B errs on the input of the size n. Next we consider the statement denoted as Φ_n: *'there exists a boolean circuit C of length n, such that Succ-3SAT(C)=1 and $B(C) \neq 1$'* .

We define a language $Err_B = \{\Phi_n \mid n \in \mathbb{N} \text{ and } \Phi_n \text{ is true}\}$. Note, that Err_B is not empty (due to our assumption of one-sided error of the heuristic), and the length of Φ_n is a polynomial on the terms of n. The first observation is that $Err_B \in NEXP$. Indeed, there is a deterministic exponential (in n^a) time verifier such that given as a certificate the boolean circuit C, representing a 3SAT instance ϕ_C, and an assignment α (of an exponential length on the terms of n) for ϕ_C, checks whether it is the case that both α is a satisfying assignment for ϕ_C and $B(C) \neq 1$.

Therefore, we can reduce Err_B to the Succ-3SAT problem using the property of the Cook-Levin reduction noted by the authors in [12]. A result of the Cook-Levin procedure is a boolean formula that reflects the movements of an exponential time Turing machine verifying a membership of the language Err_B. We call this formula Ψ_n. The variables of this formula are x, α, z, such that x variables describe a boolean circuit, α variables describe an assignment for the formula represented by circuit x, and z are the auxiliary variables added by the reduction. And the following holds: for any (x, α, z) that satisfies Ψ_n, x satisfies Φ_n, and α is a certificate for that. Furthermore, Ψ_n has a highly regular structure. In fact, it can be shown, that there is a polynomial time (on the terms of n^a) algorithm X_{Ψ_n}, such that given any two binary strings of length $c \times n$ computes a clause-literal relation of the formula Ψ_n in polynomial in n^a time. Namely, for every statement Φ_n, we match a polynomial sized (in terms of n^a) boolean circuit X_{Ψ_n} that encodes a 3SAT formula Ψ_n. We chose $q()$ to be a polynomial that is large enough so that $q(n^a)$ is bigger than the length of X_{Ψ_n}. Finally, we have reduced the language Err_B into the instances of the Succ-3SAT problem. Namely, for every Φ_n (polynomially sized on n), there is X_{Ψ_n} (polynomially sized on n^a), such that $\Phi_n \in Err_B$ if and only if $X_{\Psi_n} \in Succ{-}3SAT$.

Searching Procedure. The hard instance for B is obtained by applying the following searching technique. Assume n is an integer such that B fails to solve an instance of the length n. Run B on X_{Ψ_n}. If B answers *"no"* then it errs on X_{Ψ_n} and we have found a hard instance. If B answers *"yes"*, we start a searching process that will hopefully end with the boolean circuit C of the size n, such that B errs on it. The process sequentially runs B on the inputs obtained from X_{Ψ_n} by partial assignment of the variables of Ψ_n that describe a boolean circuit (x variables).

The process is as follows:

- Define $\Psi_n^i = \Psi_n(\alpha_1, \ldots, \alpha_i, x_{i+1} \ldots x_n)$, where $\alpha_1, \ldots \alpha_i$ is a partial assignment for variables of Ψ_n that describe a boolean circuit. $\Psi_n^0 = \Psi_n$.
- Suppose we have fixed a partial assignment, namely $B\left(X_{\Psi_n^i}\right) =$ *"yes"*. Then define:

$\Psi_n^i, 1 = \Psi_n(\alpha_1, \ldots, \alpha_i, 1, x_{i+2} \ldots x_n)$.
$\Psi_n^i, 0 = \Psi_n(\alpha_1, \ldots, \alpha_i, 0, x_{i+2} \ldots x_n)$.

– Run $B\left(X_{\Psi_n^i,1}\right)$. If it answers "yes", define $\Psi_n^{i+1} = \Psi_n^i, 1$.
  Else, run $B\left(X_{\Psi_n^i,0}\right)$. If it answers "yes", define $\Psi_n^{i+1} = \Psi_n^i, 0$.
  Else, B errs on one of the two $X_{\Psi_n^i,1}$, $X_{\Psi_n^i,0}$. Output them.
– At the end, we hold the whole assignment $C = \alpha_1 \ldots \alpha_n$. C is a circuit B errs on. Output it.

Note, that for each i, Ψ_n^i defines a $NEXP$ language. Therefore, we can reduce it in polynomial time to $X_{\Psi_n^i}$ instances.

In the worst case, the searching process will stop when the whole assignment for x variables is found. In every step of the process, we run machine B on the input of length poly(n^a). Since the number of x variables is polynomial in n and the running time of B is n^a, the total time procedure R runs on the inputs n, a, B is poly(n^{a^2}). □

Randomized Case. The main motivation of this section is to produce a hard distribution of instances of the Succinct Permanent mod p problem. Since the reduction we have built for this problem from Succ-3SAT is randomized, we have to consider producing hard instances for polynomial time randomized heuristics. We assume that the $NEXP$ class of problems is hard for *BPP*. Namely, we assume that *BPP* $\subset NEXP$. Informally, we want to prove that if *BPP* $\neq NEXP$, then for any polynomial time randomized algorithm trying to solve the Succ-3SAT problem, it is possible to efficiently produce two instances of that problem such that with a high probability the algorithm errs on one of them. For that, again, we follow the technique of generating a hard distribution of the instances for randomized heuristics that was proposed in [7]. Again we discuss the heuristics with one-sided error. Namely, the heuristics such that the answer "yes" on the input x implies that x belongs to the language with probability 1.

Next, we provide the results from [7] (without their proofs) and combine them with our observations to get the proof for the following lemma, which is the $NEXP$ analogous lemma to Lemma 4.1 of [7].

Lemma 2. *Assume that $NEXP \neq BPP$. For every constant $c > \frac{1}{2}$, there is a randomized procedure R, a polynomial $q()$ and a constant d such that the procedure R gets three inputs: integers* n*,* a *and a description of a randomized machine B that has one-sided error. The procedure R runs in time n^{da^2} and outputs at most two boolean circuits where the length of each circuit is either n or $q(n^a)$. Furthermore, if B is a randomized algorithm that runs in time bounded by n^a on inputs of length n then for infinitely many input lengths n, invoking $R(n, a, B)$ results with probability $1 - \frac{1}{n}$ in a set F of boolean circuits such that there exists $C \in F$ with $Succ - 3SAT(C) \neq B_c(C)$.*

$B_c : \{0,1\}^* \to \{0,1,*\}$ is a deterministic function associated with the randomized machine B in the following way. Given some probabilistic machine M and some function $c(n)$ over integers, such that $\frac{1}{2} < c(n) \leq 1$, $M_c : \{0,1\}^* \to \{0,1,*\}$ is defined as follows: $M_c(x) = 1$ ($M_c(x) = 0$) if M accepts (rejects) x with probability at least $c(|x|)$ over its coins; otherwise $M_c(x) = *$.

Proof. We follow the proof of the Lemma 4.1 in [7].

First, we transform the randomized algorithm B (of the lemma) into a randomized algorithm $\overline{B}$ by amplification.

The algorithm $\overline{B}$*:* Given an input x of length n, algorithm $\overline{B}$ uniformly chooses n^2 independent strings $v_1, \ldots, v_{n^2}$ each of them of length n^a. For every $1 \leq i \leq n^2$, the algorithm calls $B(x, v_i)$ and outputs the majority vote of the answers.

Next, the authors of [7] prove the following statement:

Lemma 3. *Let* $\frac{1}{2} < c \leq 1$ *be some constant. With probability at least* $1 - 2^{-n}$ *for a randomly chosen* $u \in \{0,1\}^{n^{a+2}}$*, it holds that for every* x *of length* n *and* $b \in \{0,1\}$*:*

$$\overline{B}(x, u) = b \Rightarrow B_c(x) \in \{b, *\}$$

Now, the randomized heuristic $B(x)$ can be replaced with deterministic algorithm $\overline{B}(x, u)$, where u is a randomly chosen string. Note, that the heuristic $\overline{B}$ has one sided-error. To find incorrect instances for randomized algorithm B, we find incorrect instances for deterministic machine $\overline{B}(x, u)$. Using Lemma 3, we conclude that with high probability one of the instances is incorrect for B as well.

Randomized Searching Procedure. As in the deterministic case, we start the searching procedure by defining the following language. Consider the statement denoted as $\Phi_{n,u}$: *"there exists a boolean circuit* C *of length* n *such that Succ-3SAT(C)=1 and* $\overline{B}(C, u) \neq 1$*"* , for every integer n and $u \in \{0,1\}^{n^{a+2}}$. We define the language $Err_{\overline{B}} = \left\{ \Phi_{n,u} \mid n \in \mathbb{N}\ , u \in \{0,1\}^{n^{a+2}} \mathit{and}\ \Phi_{n,u}\ \mathit{is\ true} \right\}$. As in the deterministic case, due to the assumption of the one-sided error of the heuristic B, it follows that $Err_{\overline{B}}$ is not empty. In addition, it is clear thats it is a $NEXP$ language. In fact, it can be easily shown, that for infinitely many n, except for probability $\frac{1}{2n}$ for random u we have that $\Phi_{n,u} \in Err_{\overline{B}}$. Applying the same arguments as in the deterministic case, we reduce an instance of $Err_{\overline{B}}$ to the boolean circuit $X_{\Psi_{n,u}}$, with the properties as noted in the deterministic case. Next, we describe the randomized procedure R of Lemma 3.

The procedure R chooses at random strings $u \in \{0,1\}^{n^{a+2}}$ and $u' \in \{0,1\}^{q(n^a)^{a+2}}$. Then it runs a searching procedure from the deterministic case on the input $X_{\Psi_{n,u}}$ with the deterministic heuristic $\overline{B}$ with the random choices defined by u', including the following change: when there is a call to $B(x)$, the procedure calls $\overline{B(x, u')}$.

Following that procedure, R outputs at most two instances and the following holds: for infinitely many n, R outputs a set of instances, such that with probability at least $1 - \frac{1}{n}$ there is an instance C in the set, such that $B_c(C) \neq Succ - 3SAT(C)$.

The analysis of the running time of the randomized searching procedure R is the same as in the deterministic case. □

3.2 Superpolynomial Time Heuristics

Suppose we are given a superpolynomial (deterministic) time algorithm claiming to solve the Succ-3SAT problem. From the hierarchy theorems of complexity theory, we know that such an algorithm cannot solve all instances of the Succ-3SAT problem correctly. Hence, we would like to efficiently generate a distribution of the inputs that the heuristic fails to solve. Note, that in this case, we cannot just use the previous technique, as we have no time to run the heuristic in order to identify its answer on the given instance. Namely, it is not efficient (not polynomial time) to run the searching procedure in this case. The idea is to use an interactive two provers protocol, in order to efficiently identify, whether a particular instance is accepted or rejected by the given superpolynomial time heuristic.

According to [2], it holds that for any $NEXP$ language L, there is a randomized polynomial time verifier machine V and infinitely powerful machines $P_1, \dots P_k$ such that the following holds:

1. If $x \in L$ then Pr $[P_1, \dots P_k$ cause V to accept $x] > 1 - 2^n$.
2. If $x \notin L$ then Pr $[P'_1, \dots P'_k$ cause V to accept $x] < 2^{-n}$ for any provers $P'_1, \dots P'_k$.

Let B denote some superpolynomial time deterministic algorithm with one-sided error claiming to solve Succ-3SAT problem. We use an interactive proof system for the language L_B – the language of the heuristic B. Namely, L_B is the set of all instances such that B answers "yes" on them (and by the assumption of one-sided error, B does not mistake on the instances from this set). Note, that L_B is a $NEXP$ language. We start with the language Err_B that was defined in the previous section. Strictly following the proof of the Lemma 1, we reduce this language to the Succ-3SAT problem in order to get the set of the instances $\{X_{\Psi_n}\}$.

The idea is to use the scheme of the searching process as before, but with the following change. Every time the searching procedure calls the heuristic B on the input x, we run the verifier-provers protocol with the input x. According to the decision V makes, the process outputs the instance such that B errs on it with high probability or continues to search for such inputs. In that way, we have a randomized polynomial time procedure that outputs at most two instances of the problem, such that B errs on one of them with a high probability. Thus, in the standard model of interactive proofs, the above procedure efficiently searches and finds the hard instances for the heuristic B. Note, that in that model, the running time of the searching procedure does not include the time required by the provers.

For the randomized superpolynomial heuristic (under an assumption that $NEXP$ is hard for such class of heuristics), we use the same scheme as in the randomized polynomial case. Namely, first, by the amplification argument, we replace the randomized machine with a deterministic one (defined by a random string of choices), and then we use the idea of the two provers protocol.

4 Succinct Permanent Modulo a Prime Has Many Hard Instances

The number of hard instances of the Succinct Permanent $\bmod\ p$ grows exponentially with the input size. In Section 2.2, we proved that there exists (for each sufficiently large n) at least one hard instance of size $O(\log^k n)$ of the Succinct Permanent mod p problem that requires a given polynomial time heuristic to work more than polynomial time to compute the solution correctly. In Section 3 we showed how to find a hard instance for any given heuristic. In this section, we use any given hard succinct permanent instance for a given heuristic (of size $O(\log n^k)$) to generate a set of $O(n)$ hard succinct permanent instances. The generated set is a combination of random self-reducible sets that can efficiently solve the given hard succinct instance. The number of instances in the set is exponential in the additional bits used to enlarge the succinct representation.

Given a boolean circuit C and a prime number p as inputs to the Succinct Permanent mod p. Consider a matrix $A = M(C)$ that is represented by C. Let the first row of A be: $(x_{11}\ x_{12}\ \dots\ x_{1\log n}\ x_{1\log n+1}\ \dots\ x_{1n})$. Then,

$$permanent(A) = x_{11} \times per_1 + \ldots + x_{1\log n} \times per_{\log n} + \ldots + x_{1n} \times per_n$$

where per_j is a permanent of the Adj_{1j} (adjoint) of the matrix A. We can rewrite it as follows: $permanent(A) = x_{11} \times per_1 + x_{12} \times per_2 + \ldots + x_{1\log n} \times per_{\log n} + X$.

The idea is to build $O(n)$ circuits representing matrices, that are obtained from A by replacing the first $\log n$ entries in a manner that allows computing a permanent of A modulo p in polylogarithmic time, given permanent results modulo p of $\log n$ randomly chosen circuits from the set.

One of the possibilities to obtain such a construction is to use the features of the Vandermonde matrix.

Note that in the reduction algorithm, we can use the primes p', that are required by the Chinese Reminder theorem, such that $p' \geq n+1$, without changing the complexity result. Therefore, we can assume that the hard instance is C, p, s.t. the prime p satisfies $p \geq n+1$.

For each $1 \leq i \leq p$ and $a_i \in Z_p$ define: $r_i = (a_i\ {a_i}^2\ {a_i}^3\ \dots\ {a_i}^{\log n})$ Note, that it is necessary that $p > n$ in order to construct a set of size that is exponential in the input size. Let R_i be an $n \times n$ matrix obtained from A by replacing the first $\log n$ entries of the first row of A with the vector r_i. We show that given the value of $permanent(R_i) \bmod p$ of any $\log n + 1$ matrices from $R_1, \ldots R_p$, there is a polynomial time algorithm that computes $permanent(A) \bmod p$.

Let a be a vector of the first $\log n$ entries of the first row of the matrix A. For simplicity, suppose we are given

$$permanent(R_1) \bmod p \equiv z_1, \ldots permanent(R_{\log n+1}) \bmod p \equiv z_{\log n+1}$$

To compute a permanent of A modulo p, one should compute the values of $X \bmod p$, $per_1 \bmod p$, $per_2 \bmod p$, $\ldots per_{\log n} \bmod p$ We build a system of linear equations. The system contains $\log n + 1$ equations, each with $\log n + 1$

variables. All the constants in the system are from the field Z_p, namely, positive integer numbers. The matrix representation of the system is as follows:

$$\begin{pmatrix} 1 & a_1 & {a_1}^2 & \dots & {a_1}^{\log n} \\ 1 & a_2 & {a_2}^2 & \dots & {a_2}^{\log n} \\ \vdots & \vdots & & & \vdots \\ 1 & (a_{\log n+1}) & (a_{\log n+1})^2 & \dots & (a_{\log n+1})^{\log n} \end{pmatrix}$$

The vector of variables of the system is: $(X\ per_1\ per_2\ \dots\ per_{\log n})$, and the vector of answers is $(z_1\ z_2\ \dots\ z_{\log n}\ z_{\log n+1})$.

Since the matrix is a subset of a Vandermonde matrix, there exists a unique solution to the system. Since all computations are over the field Z_p, the time needed to solve the system is polynomial in the size of the succinctly represented instance. To complete the description of the technique, we should clarify that for each matrix R_i, there exists a succinct circuit representation. We construct circuit $C(R_i)$ by combining the circuit C (the succinct circuit representation of the matrix A) with succinct circuit Row_i that contains $\log n$ inputs and $\log n$ outputs. $Row_i(k)$ outputs a binary representation of ${a_i}^k \mod p$, for $a_i \in Z_p$, $1 \leq k \leq \log n$.

In principle, the above technique is not restricted only to the first row of the matrix. The building procedure is valid if we choose some row of the matrix (or some column) and some $\log n$ entries of the chosen row (or column) — the permanent of the matrix can be computed by each of its rows (or columns). Therefore, using this observation, we speculate that the succinct permanent problem is not in the *LEARN* class, which is defined by Impagliazzo and Wigderson in [8]. That is, having an oracle for the answers of $\log n$ or less instances of the same generated set do not reveal the answers of the hard instances of the set of another generated set.[3] We emphasize, however, that there is an efficient algorithm such that given the answers for at least $logn + 1$ instances from the same generated set provides in polynomial time answers for any other instance in the same set. Hence, instances should be carefully chosen from different sets. Such a strategy will not base the hardness on a small set of hard instances as done in the encoding truth tables technique of [14].

Note that we introduce a new framework in which one would like to avoid the dependencies of instances in the manner that a solution to one instance (say, after an exhaustive search for a private key) reveals a solution to other (private key) instances. Not only there is a need to introduce provable hard on average instances, it is also important to ensure *non revealing instance solutions.*

Acknowledgments. With pleasure, we thank Mihalis Yannakakis and Salil Vadhan for useful inputs.

[3] Our speculation is based on the fact that the result of one minor does not totally reveal the result of another minor of the permanent, otherwise we may use this property to solve the permanent problem in polynomial time starting with a matrix of all zeros and adding (non zero) lines and columns one after the other calculating the delta in the permanent value.

References

1. Agrawal, M., Kayal, N., Saxena, N.: PRIMES is in P. Ann. of Math 2, 781–793 (2002)
2. Babai, L., Fortnow, L., Lund, C.: Non-Deterministic Exponential Time has Two-Prover Interactive Protocols. Computational Complexity 1, 3–40 (1991)
3. Babai, L., Fortnow, L., Nisan, N., Wigderson, A.: BPP Has Subexponential Time Simulations Unless EXPTIME has Publishable Proofs. Computational Complexity 3, 307–318 (1993)
4. Dolev, S., Fandina, N., Gutfreund, D.: Succinct Permanent is NEXP-hard with Many Hard Instances. Electronic Colloquium on Computational Complexity (ECCC) 19, 86 (2012)
5. Dolev, S., Korach, E., Uzan, G.: Magnifying Computing Gaps Establishing Encrypted Communication over Unidirectional Channels (Extended Abstract). In: Masuzawa, T., Tixeuil, S. (eds.) SSS 2007. LNCS, vol. 4838, pp. 253–265. Springer, Heidelberg (2007)
6. Galperin, H., Wigderson, A.: Succinct representations of graphs. Inf. Control 56, 183–198 (1984)
7. Gutfreund, D., Shaltiel, R., Ta-shma, A.: If NP Languages are Hard on the Worst-Case, Then it is Easy to Find Their Hard Instances. Computational Complexity 16(4), 412–441 (2007)
8. Impagliazzo, R., Wigderson, A.: Randomness vs time: derandomization under a uniform assumption. J. Comput. Syst. Sci. 63, 672–688 (2001)
9. Lipton, R.: New directions in testing. In: Distributed Computing and Cryptography. DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 2, pp. 191–202 (1991)
10. Merkle, R.C.: Secure Communications Over Insecure Channels. CACM 21(4), 294–299 (1978)
11. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1994)
12. Papadimitriou, C.H., Yannakakis, M.: A note on succinct representations of graphs. Inf. Control 71, 181–185 (1986)
13. Rybalov, A.N.: Generic Complexity of Presburger Arithmetic. In: Diekert, V., Volkov, M.V., Voronkov, A. (eds.) CSR 2007. LNCS, vol. 4649, pp. 356–361. Springer, Heidelberg (2007)
14. Sudan, M., Trevisan, L., Vadhan, S.P.: Pseudorandom Generators without the XOR Lemma. J. Comput. Syst. Sci. 62, 236–266 (2001)
15. Trevisan, L., Vadhan, S.: Pseudorandomness and average-case complexity via uniform reductions. Computational Complexity 16(4), 331–364 (2007)
16. Valiant, L.G.: The complexity of computing the permanent. Theoretical Computer Science 8(2), 189–201 (1979)

Money-Based Coordination of Network Packets[⋆]

Pavlos S. Efraimidis and Remous-Aris Koutsiamanis

Dept. of Electrical and Computer Engineering
Democritus Univ. of Thrace, 67100 Xanthi, Greece
{pefraimi,akoutsia}@ee.duth.gr

Abstract. In this work, we apply a common economic tool, namely money, to coordinate network packets. In particular, we present a network economy, called PacketEconomy, where each flow is modeled as a population of rational network packets, and these packets can self-regulate their access to network resources by mutually trading their positions in router queues. We consider a corresponding Markov model of trade and show that there are Nash equilibria (NE) where queue positions and money are exchanged directly between the network packets. This simple approach, interestingly, delivers significant improvements for packets and routers.

1 Introduction

It is known that a large number of independent flows is constantly competing on the Internet for network resources. Without any central authority to regulate its operation, the available network resources of the Internet are allocated by independent routers to the flows in a decentralized manner. Internet flows may submit at any time an arbitrary amount of packets to the network and then adjust their packet rate with an appropriate flow control algorithm, like the AIMD-based algorithms for TCP-flows. The apparent lack of coordination between the independent flows leads the Internet to an "anarchic" way of operation and gives rise to issues and problems that can be addressed with concepts and tools from algorithmic game theory.

Two representative works on applying game theory to network problems are [15,19]. Certain game-theoretic approaches to congestion problems of the Internet, and especially the TCP/IP protocol suite, are discussed in [20,1,8,6]. A combinatorial perspective on Internet congestion problems is given in [12]. The focus of the above works and the present paper is on sharing the network resources between selfish flows. In this work, however, we propose an economy where packets belonging to selfish flows may interact directly with each other.

⋆ The research leading to these results has received funding from the European Union Seventh Framework Programme [FP7/2007-2013] under grant agreement no 264226. [Project Title: Space Internetworking Center-SPICE]. This paper reflects only the views of the authors. The Union is not liable for any use that may be made of the information contained.

The use of economic tools like pricing, tolls and taxes as a means to regulate the operation of networks and/or to support quality of service (QoS) functionalities in the presence of selfish flows is, for example, discussed in [18,9,4,3,16,17]. In particular, the Paris Metro Pricing approach - using pricing to manage traffic in the Paris Metro - is adapted to computer networks in [18]. A smart market for buying priority in congested routers is presented in [16]. In [4,3] taxes are used to influence the behavior of selfish flows in a different network model. An important issue identified in [3] is that taxes may cause disutility to network users unless the collected taxes can be feasibly returned to the users. In our economic model this issue is naturally solved; trades take place between the flows, so the money is always in the possession of the flows.

In this work, we apply a common economic tool, namely money, to coordinate network packets. This is in contrast to much of the existing literature, which aims to impose charges on Internet traffic, and to our knowledge, this is the first work to propose economic exchanges directly between packets. In particular, we present a network economy, called PacketEconomy, where ordinary network packets can trade their positions in router queues. The role of money in this approach is to facilitate the trades between the network packets. Queue positions and money are exchanged directly between the packets while the routers simply carry out the trades. We show that, in this economy, packets can self-regulate their access to network resources and obtain better services at equilibrium points.

In their seminal work, Kiyotaki and Wright [13] examine the emergence of money as a medium of exchange in barter economies. Subsequently, Gintis [10,11] generalizes the Kiyotaki-Wright model by combining Markov chain theory and game theory. Inspired by the above works, we propose the PacketEconomy where money is used as a coordination mechanism for network packets and prove that there are Nash equilibria where trades are performed to the benefit of all the flows. In the PacketEconomy, specialization - the reason for the emergence of money as per Adam Smith ([21, Chapter 4], cited in [13]) - originates from the diverse QoS requirements of network flows. In particular, the various types of PacketEconomy flows differ in their tolerance for packet delays.

Contribution. The main contributions of this work are:

- A new game-theoretic model representing network packets as populations of rational agents. In this model, a network flow is represented as a population of in-flight packets that can make bilateral trades with other packets.
- Application of bilateral trades and virtual money at a microeconomic level to support better coordination of rational network packets.
- Application of an interesting combination of ergodic Markov chains and strategic games within the context of network games.

Outline. We describe in Section 2 the PacketEconomy and analyze in Section 3 a representative scenario of it. The effect of trades is discussed in Section 4. Concluding remarks are given in Section 5. Due to lack of space, some proofs have been omitted and can be found in the long version of this work [5].

2 An Economy for Packets

The PacketEconomy is comprised of a network model with selfish flows, a queue that supports packet trades, a currency and a specific economic goal. The solution concept is the Nash equilibrium (NE), i.e., a profile of the game in which no player has anything to gain by changing only his/her own strategy unilaterally.

The Network Model. We assume a one-hop network with a router R and a set of N flows, as shown in Figure 1. This setting is equivalent to the common dumbbell topology used for the analysis of many network scenarios, including the seminal paper of Chiu and Jain [2] on the AIMD algorithm. The router R has a FIFO DropTail queue with a maximum capacity of q packets and operates in rounds. In each round, the first packet (the packet at position 0 of the queue) is served. At the end of the round, the first packet reaches its destination. Packets that arrive at the router are added to the end of the queue.

Packet Trades. At the beginning of each round all packets in the queue are shifted one position ahead. A packet that enters the queue in this round, occupies the first free (after the shift) position at the end of the queue. After the shift, the packet that has reached position zero is served, while the other packets in the router queue are simply waiting. These idle packets can engage in trades. During each router round a fixed number b of trading periods take place. In each trading period the idle packets are matched randomly in pairs with a predefined pairing scheme. Each packet pair can perform a trade, as shown in Figure 2, provided the negotiation performed between them leads to an agreement. The way the trades take place at a microeconomic level between paired packets resembles the models of [10,13] where agents meet in random pairs and can make trades.

Packet Delay. The delay d_p of a packet p that starts at position k of the zero-based queue and does not make any trade is $k+1$ rounds (Figure 3a). If, however, the packet engages in trades and buys a total of r_b router rounds and sells r_s router rounds, then its delay d_p, including the time to be served, becomes $d_p = k + 1 + r_s - r_b$ rounds. A packet may have an upper bound $d_{p,\max}$ on its

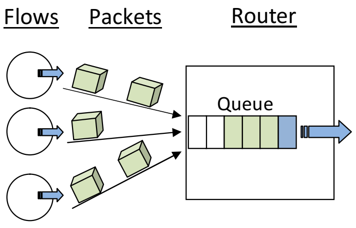


Fig. 1. The network model with the flows, their packets, the router, and the queue

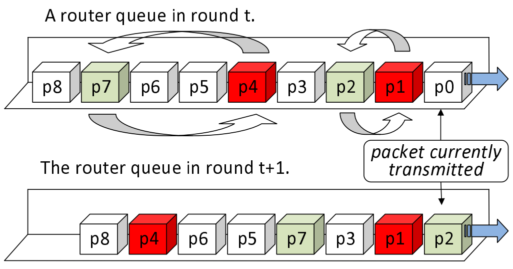

A router queue in round t.
p8
p7
p6
p5
p4
p3
p2
p1
p0
packet currently transmitted
The router queue in round t+1.
p8
p4
p6
p5
p7
p3
p1
p2

Fig. 2. The state of a router queue in two successive rounds. In round t, two trades take place; one between the packet pair (p1,p2) and one between the pair (p4,p7).

delay; for delays larger than $d_{p,\max}$ the value of the packet becomes zero and the packet will not voluntarily accept such delays (that is, it will not sell).

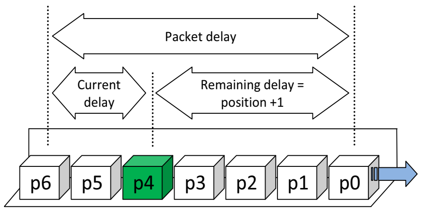
Details. The router operates in rounds and can serve one packet in each one. All packets are assumed to be of the same size and no queue overflows occur. In generating the random packet pairs, the use of predefined pairing reduces the computational burden and avoids stable marriage problems. We make the plausible assumption that flows with different QoS preferences are competing for the network resources. We also make the assumption that the preferences of each flow can be expressed with a utility function for its packets. Thus, packets with different utility functions will, in general, co-exist in the router queue.

Packet Values. For each packet p there is a flow-specific decreasing function $v_p(d)$ which determines the value of p, given its delay d. The value function of each flow must be encoded onto each packet. Thus, its computational requirements should be low in order not to overload the router. A class of simple value functions are $v_p(d) = \max\{v_{\max} - c_p \cdot d, 0\}$ where c_p is the cost per unit of delay (Figure 3b). The value of a packet can be calculated anytime during the packet's journey via the $v_p(d)$ function.

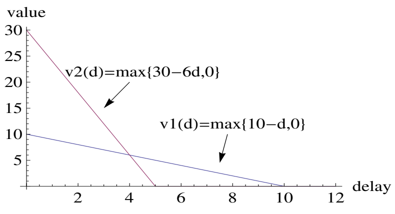
In the PacketEconomy every packet has its compensatory price p. For prices lower than p, the packet is ready to buy better queue positions and for prices higher than p it is ready to sell its position, provided that the extra delay will not cause it to exceed its maximum delay limit.

Inventories. Every time a packet is delivered in time, wealth is created for the flow that owns the packet. Each packet p has an inventory $I_p(t)$ containing two types of indivisible goods or resources; the packet delay $d_p(t)$ and the money account $a_p(t)$. Note that delay bears negative value, whereas money represents positive value. We assume positive integer constants s_a, s_b and s_d, such that $a_p(t) \in \{-s_a, \ldots, s_b\}$ and $d_p(t) \in \{0, \ldots, s_d\}$. The inventory also contains the current position $pos_p(t)$ of the packet in the queue if it is waiting in the queue. When the packet reaches its destination, the contents of the inventory of the packet are used to determine its utility. This utility is then reimbursed to the flow that owns the packet and a new packet of the same flow enters the queue.

Benefit and Utility. Every packet has two types of resources that bear value, the packet value and the budget of a packet. We define the notion of the packet



(a) Packet delay terminology for p4



(b) Two simple packet value functions

Fig. 3. Delays and Packet Values

benefit as the sum of the value of a packet plus/minus its budget. Then we use the benefit concept to define the utility function of the packet. For rate-based flows (see below), the utility of a packet is equal to its benefit. For window-based flows the utility function is the benefit rate (benefit per round).

Trades. The objective of each packet is to maximize its utility. Thus, when two packets are paired in a trading period, their inventories and their trading strategies are used to determine if they can agree on a mutually profitable trade, in which one packet offers money and the other offers negative delay. The obvious prerequisite for a trade to take place is that both packets prefer their post-trade inventories to their corresponding pre-trade inventories. For this to be possible, there must be "surplus value" from a potential trade. In this case, both packets can benefit, i.e., increase their utility, if they come to an agreement.

Flow Types and the Cost of Delay. The delay that a packet experiences has a negative impact on its utility. The value is a non-increasing function of the delay. Window-based flows employ a feedback-based mechanism, the congestion window, which determines the maximum number of packets that the flow may have in-flight. Every packet that is in-flight occupies one of the available positions in the congestion window of a window-based flow. The more a packet delays its arrival, the longer the following packet will have to wait to use the occupied window position. Therefore, the impact of packet delays for window-based flows is twofold; the decreased value of the delayed packet and the reduced packet rate. On the other hand, for rate-based flows which submit packets with some given rate, the only consequence due to packet delays is the reduced packet value.

Assume a rate-based packet p with balance α_1 and delay $d_1 < d_{p,\max} - d_\epsilon$, for some d_ϵ. When a trade changes the delay from d_1 to $d_2 = d_1 + d_\epsilon$, then this also changes the value of the packet from $v(d_1)$ to $v(d_2)$. The difference between these two values determines the compensatory price ρ for the packet.

$$\rho = v(d_1) - v(d_2) = v(d_1) - v(d_1 + d_\epsilon) = c_p d_\epsilon \ . \tag{1}$$

At this price, the utility of the packet remains unchanged after the trade. A packet would agree to sell for a price $\rho_s > \rho$, or to buy for $\rho_b < \rho$.

For window-based flows, however, the price estimation needs more attention. Assume a window-based packet with delay $d_1 < d_{p,\max} - d_\epsilon$ and account balance α_1. Before the trade, the utility (benefit rate) is $r_1 = (v_1 + \alpha_1)/d_1$. If the packet agrees to trade its position and to increase its delay by d_ϵ, then the utility is $r_2 = (v_2 + \alpha_2)/d_2$. Then, by setting $r_1 = r_2$ we obtain the compensatory price ρ for the trade.

$$\frac{v_1 + \alpha_1}{d_1} = \frac{v_2 + \alpha_2}{d_2} \Rightarrow \frac{V - c_p d_1 + \alpha_1}{d_1} = \frac{V - c_p(d_1 + d_\epsilon) + (\alpha_1 + \rho)}{d_1 + d_\epsilon} \Rightarrow$$

$$\rho = (V + \alpha_1)\frac{d_\epsilon}{d_1} \ . \tag{2}$$

The above expression for the price ensures that the utility function of the packet remains unchanged. A packet would agree to sell its position, for a price $\rho_s > \rho$, or to buy a position ($d_\epsilon < 0$) for $\rho_b < \rho$. Unless otherwise specified, the final trading price when a trade takes place will be the average of the ρ_s of the seller packet and ρ_b of the buyer packet. We illustrate the PacketEconomy approach in a representative scenario.

3 Equilibria with Monetary Trades

A Representative Scenario. We examine a simple scenario that produces an interesting configuration. It consists of a set of N window-based flows f_i, for $i \in \{1 \ldots N\}$, each with a constant window size w_i, and $\sum_i w_i = q$. When a packet is served by the router it is immediately replaced by an identical packet submitted by the same flow. This is a simplifying but plausible assumption. In reality, when a flow packet arrives at its destination, a small size acknowledgment packet (ACK) is submitted by the receiver. When the sending flow receives the ACK it submits a new identical packet that immediately enters the queue. We assume $b = 1$ trading period per round but in general b can be any integer $b > 0$.

Failure States. For each packet, there is a small probability p_f for an extra delay of d_f rounds, where d_f is a discrete random variable in $\{1, 2, \ldots, q-1\}$. These delays correspond to potential packet failures of real flows, and occur between the service of a packet and the submission of its replacement. By convention, the delay d_f is added to the delay of the packet that has just been served. If more than one packets enter the queue at the same time (synchronized due to delays), their order in the queue is decided upon uniformly at random. A packet that does not participate in any trade and does not suffer delay due to failure will experience a total delay of q rounds.

Packet States and Strategies. The state $\tau_p(t)$ of a packet p in round t is a pair $\tau_p(t) = (I_p(t), rel_p(t))$, where $I_p(t)$ is the inventory of the packet and $rel_p(t)$, which is meaningful only in failure states, is the remaining number of failure rounds for the packet. The state of all packets of the economy in round t determines the state of the whole economy $\tau(t) = \prod_{p=0}^{q-1} \tau_p(t)$. From a packet's point of view, a trade is simply an exchange of its inventory state (budget, delay and position) with a new one. Consequently, a pure strategy of a packet is a complete ordering of the possible states of its inventory. In each round, the packets that are waiting move by default one position ahead and, thus, enter a new inventory state. We assume that the packet ignores the impact of its state and strategy on the state of the packet population. In every trading period the packet assumes the same stationary state of the economy.

Definition 1. *Let $\tau(t)$ be the state of the economy in round t.*

Lemma 1. *$\tau(t)$ is an ergodic Markov chain.*

Proof. Assume $b = 1$ trading period per round. In each round, the economy moves to a new state with transition probabilities that depend only on the current state and the strategies of the packets. Let σ_p be a pure strategy of each packet p of a flow and σ be a pure strategy profile of the whole economy. Then, there is a corresponding transition probability matrix P^{σ} for the economy. Let σ_m be a mixed strategy profile of the whole economy. Then the corresponding transition probability P^{σ_m} of the economy for σ_m is an appropriate convex combination of the transition matrices of the supporting pure strategies. In case of multiple trading periods per round ($b > 1$), the economy makes b state transitions per round.

The number of potential states for a packet is finite and, consequently, the number of states for the whole economy is also finite.

Definition 2. *A zero state τ_0 is a state of the economy in which all packets have zero budget and each packet p has delay $d_p(t_0) = pos_p(t_0) + 1$, where t_0 is the current round of the router.*

Assume that in round t the packet at position 0 fails for $q-1$ rounds, in round $t+1$ the next packet at position 0 fails for $q-2$ rounds etc. Then after q rounds all new packets will simultaneously enter the queue. Each packet will have zero budget and by definition their ordering will be random. This also means that for each packet p, $d_p(t) = pos_p(t) + 1$. Thus, in round $t + q$ the economy will be in a zero state. The probability for this to happen is strictly positive and thus each zero state τ_0 is recurrent. Since the number of states of the economy is finite, the states that are attainable from zero states like τ_0 form a (finite in size) class of irreducible states. Moreover, each zero state is aperiodic, and thus each of the states of the class of attainable states is also aperiodic. It is known that any finite, irreducible, and aperiodic Markov chain is ergodic.

Lemma 2. *(Proof omitted) For each pure strategy profile σ of the economy, there is a unique stationary distribution π_σ of the economy.*

An interesting argument which can now be applied is that given the stationary distribution of the economy, each trading period becomes a finite state game.

Lemma 3. *(Proof omitted) For every idle packet, each trading period of the economy corresponds to a finite strategic game.*

Theorem 1. *(Proof omitted) A NE exists where packets perform trades.*

Pipelined Shuffling. A core operation of the PacketEconomy is the random pairing of the packets that takes place in each trading period to generate the trading pairs. We present a new parallel algorithm that can support the random pairing procedure in real time. The new algorithm (Algorithm 1) is a parallel, or better, a pipelined version of the random shuffling algorithm of Fisher-Yates, which is also known as Knuth shuffling [7,22,14]. We call the new algorithm *Pipelined Shuffling*. Its core is a pipeline of q instances $0, 1, \ldots, q-1$ of the Fisher-Yates algorithm. At time t, instance k is at step $t+k \bmod q$ of the random shuffling algorithm.

Theorem 2. *The Pipelined Shuffling algorithm delivers a random shuffle every $O(1)$ parallel time steps on a q processors EREW PRAM.*

Algorithm 1. Pipelined Shuffling

```
1: procedure Shuffle(int[] a)
       for i from 0 to q-2 do {
           j = random int in $i \le j \le q-1$;
           exchange a[j] and a[i]}
2: end procedure
1: procedure ParallelShuffle(int[][] A)
       for i from 0 to q-1 do in parallel {
           processor i: wait for i periods;
           processor i: while (true) {Shuffle(A[i]);}
2: end procedure
```

The Scheduling Problem. The underlying algorithmic problem of the PacketEconomy is a scheduling problem of network packets. From the router's point of view, this problem is a single machine scheduling problem with a max weighted total wealth objective.

Definition 3. *Max-Total-Wealth Scheduling (MTW). A set of n jobs j_i, for $i = 1, \ldots, n$. Job j_i has processing time p_i, release date r_i, deadline d_i and weight w_i. Let c_i be the completion time of job i in a schedule. The objective is to find a non-preemptive schedule that maximizes the total wealth $W = \sum_i w_i \cdot \max(d_i - c_i, 0)$.*

The release date r_i is the time when packet i enters the queue and the deadline d_i is the time when the value of the packet becomes zero. For MTW on a network router the following assumptions hold: a) The queue discipline is work-preserving, meaning a non-empty router is never left idle, b) the number of packets in the queue at any time is bounded by a constant (the maximum queue size), and c) the packet sizes may differ by at most a constant factor. In this work, we assume that all packets are of the same size.

The complexity of the MTW problem depends on the assumptions made. It is not hard to show that without deadlines, even the online version of MTW can be optimally solved; there is a 1-competitive algorithm for MTW without deadlines [5]. Moreover, MTW with deadlines can be offline solved in polynomial time as a linear assignment problem.

However, due to the on-line nature and the finite queue size of the PacketEconomy router, the above conventional scheduling algorithms do not seem to naturally fit the MTW problem of the PacketEconomy. Especially for window-based flows, where packet transmission is a closed loop, the order in which the queued packets are served influences, if not determines, the next packet that will enter the queue. Thus, even the online assumption may not be appropriate. A different approach to study the scheduling problem of the PacketEconomy is to consider the (average) packet rate of the flows, as shown in the following example.

Example 1. Assume a scenario with window-based flows and 5 economy packets and 5 business packets. There is a deadline of 40 rounds on the maximum delay of the economy packets. Moreover, all business packets have to be treated equally. The same holds for the economy packets. Consider the scenario where each economy packet will be served with a rate of 1/40 packets/round and delay of 40 rounds and the business flows share the remaining bandwidth; each business packet is served at a rate of 7/40 packets/round and delay 40/7 rounds. This is an upper bound on the rate of total wealth for the router for this scenario.

4 The Effect of Trades

The NE of the representative scenario shows that, in principle, money can be used at a microeconomic level to coordinate network packets. By definition, the flows of the scenario can only benefit through the use of money; each trade is a weak Pareto improvement for the current state of the economy. In this section we further examine the effect of trades.

In the PacketEconomy, each packet can increase its utility by making trades. To show the potential of the approach, consider a packet of maximum priority that pays enough to make any trade that reduces its delay. In the analysis, we will assume that the probability of packet failures is very low, and thus ignore it. We focus on window-based flows, present an exact calculation for the average delay of this packet and then derive simpler, approximate bounds.

Lemma 4. *The average delay $E[d]$ of the packet is*

$$E[d] = \sum_{s=1}^{q} s \left(\frac{1}{q-2}\right)^{s} (s-1)\frac{(q-2)!}{(q-s-1)!} \ . \tag{3}$$

Proof. Let $rand(L, U, s)$ be a uniformly random integer in $\{L, L+1, ..., U\}\backslash\{s\}$ and $pos(p)$ the current position of packet p. Then, the probability $Pr[d > s]$ is

$$= \prod_{k=1}^{s} Pr[rand(1, q-1, pos(p)) \geq s-k+1] = \frac{q-s-1}{q-2} \cdot \frac{q-s}{q-2} \cdots \frac{q-2}{q-2} \Rightarrow$$

$$Pr[d > s] = \left(\frac{1}{q-2}\right)^{s} \cdot \frac{(q-2)!}{(q-s-2)!} \ , \text{ and}$$

$$Pr[d = s] = Pr[d \leq s] - Pr[d \leq s-1] = \left(\frac{1}{q-2}\right)^{s} (s-1)\, \frac{(q-2)!}{(q-s-1)!} .$$

Applying the definition of the expected value completes the proof.

Lemma 5. *(Proof omitted). Let $X^d_{\min}$ be the minimum of $n > 0$ discrete uniform random variables (RV) in $[L, U]$ and $X^c_{\min}$ be the minimum of n continuous uniform RV in $[L, U]$. Then*

$$E[X^d_{\min}] \leq E[X^c_{\min}] \leq E[X^d_{\min}] + 1 \ . \tag{4}$$

Lemma 6. *The average delay of the packet does not exceed* $\frac{-1+2b+2\sqrt{2b(q-2)}}{2b}$. *For* $b=1$ *the bound is* $\frac{1}{2}+\sqrt{2(q-2)}$.

Proof. A packet that enters at position $q-1$ has been served when it advances at least q positions. Note that each random trading partner corresponds to a uniform random number in $[1, q-1]$. To admit a more elegant mathematical treatment we prefer the continuous distribution. Lemma 5 makes this possible.

Assume that a packet has just entered the queue at position $q-1$. Let b be the number of trading periods per router round. Assume that the packet spends at least k rounds in the queue until it reaches position 1. During these k rounds the packet will make bk random draws and will make k single position advancements. From Lemma 7 we obtain that the average value of the minimum of the bk random draws is

$$\frac{1}{bk+1}(q-2) .$$

Lemma 7. *(Proof omitted) Let* $X_1, X_2, \ldots, X_k$ *be continuous uniform random variables in* $[0, U]$ *and let* $X_{\min} = \min_{i=1,\ldots,k} X_i$. *Then* $E[X_{\min}] = \frac{1}{k+1}U$.

Note that the average number of rounds and draws until it achieves its best draw is $(k+1)/2$ and $(bk+1)/2$, respectively. We will add one to the value of the average minimum draw, because the minimum position that can be traded is position 1. Position 0 is the one that is currently being served.

Now, assume that after the k rounds and bk draws the packet advances for h additional rounds until it reaches position 1. From position 1 it needs a final round to proceed to position 0 and be served. Thus, the total delay of the packet is $k+h+1$, and

$$\frac{1}{bk+1}(q-2) + 1 - (k+1)/2 - h - 1 \leq 0 .$$

We solve for k and obtain that the larger of the two roots of k is

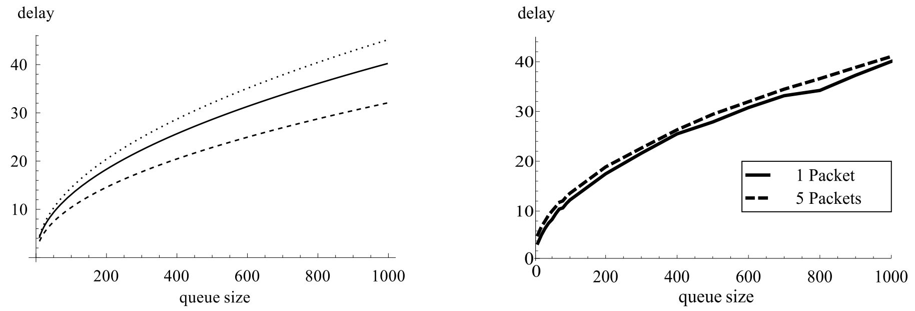
$$k = \frac{-1-b-2bh+\sqrt{(1+b+2bh)^2+4b(2q-5-2h)}}{2b} . \tag{5}$$

The total delay $k+h+1$ is minimized at $h = (1-b)/(2b)$. Substituting $h = (1-b)/(2b)$ in Equation 5 gives that the minimum value of $k+h+1$ is

$$k+h+1 = \frac{b-1+2\sqrt{2b(q-2)}}{2b} .$$

The average delay cannot be larger then the above value. This completes the proof of Lemma 6.

The above lemma can be generalized to the case where only one packet in every $c > 0$ packets in the queue is ready to sell its position. We simply assume b/c trading periods per round. Then, the average delay of the business packet is not larger than $1-(c/2)+\sqrt{2c(q-2)}$. Similarly, we can show:



(a) The exact average delay (inner line) and the lower and upper bounds on the average delay

(b) Experimental measurement of the delay for the cases of one business packet and five business packets.

Fig. 4. Delay of the business packet with respect to the queue size

Lemma 8. *(Proof omitted) The average delay of the packet is at least* $(-1 + 2b + \sqrt{1 - 8b + 4bq})/(2b)$*. For* $b = 1$ *the bound is* $(1/2) + \sqrt{4q - 7}$*.*

This lemma too, can be generalized to the case where only one packet in every $c > 0$ packets in the queue is ready to sell its position. In this case the average delay of the business packet is not less than $\frac{1}{2}(2 - c) + \frac{1}{2}\sqrt{c^2 - 8c + 4qc}$.

In Figure 4, analytical and experimental results for the delay of the business packet are presented. In the long version of this work [5], we use the lemmas of this section to analyze the packet delays and the social wealth of a PacketEconomy instance for the cases of no trades, ideal trades, and PacketEconomy trades.

5 Conclusion

We presented an economy for network packets and showed the existence of NE where money circulates to the benefit of the flows. The basic computational step of the PacketEconomy can be executed in $O(1)$ parallel time on fairly simple multi-core hardware, making it appropriate for modern network router demands.

There are several other issues that have to be addressed for such a model to be of practical importance. For example, a greedy flow may submit economy packets to the network simply to collect money. A realistic economic model has to anticipate such scenarios and address them with appropriate rules. One approach could be to have the router restricting the final budget of any packet to be non-positive, or more effectively, impose router-entry costs on every packet depending on the current load.

Overall, we examined how money can be used at a microeconomic level as a coordination tool between network packets and we believe that our results show that the PacketEconomy approach defines an interesting direction of research for network games. We are currently examining the use of fiat money and the implemention of the PacketEconomy in a realistic network context.

Acknowledgements. The first author, is grateful to Paul Spirakis for inspiring discussions on the intersection of Algorithms and Game Theory. We also wish to thank Vasilis Tsaoussidis for insightful discussions on using budgets in Internet router queues.

References

1. Akella, A., Seshan, S., Karp, R., Shenker, S., Papadimitriou, C.: Selfish Behavior and Stability of the Internet: a Game-Theoretic Analysis of TCP. In: SIGCOMM 2002, pp. 117–130 (2002)
2. Chiu, D.-M., Jain, R.: Analysis of the increase/decrease algorithms for congestion avoidance in computer networks. Comp.Netw.ISDN 17(1), 1–14 (1989)
3. Cole, R., Dodis, Y., Roughgarden, T.: How much can taxes help selfish routing? In: EC 2003, pp. 98–107. ACM (2003)
4. Cole, R., Dodis, Y., Roughgarden, T.: Pricing network edges for heterogeneous selfish users. In: STOC 2003, pp. 521–530. ACM (2003)
5. Efraimidis, P.S., Koutsiamanis, R.-A.: On money as a means of coordination between network packets. CoRR, abs/1208.3747 (2012)
6. Efraimidis, P.S., Tsavlidis, L., Mertzios, G.B.: Window-games between TCP flows. Theoretical Computer Science 411(31-33), 2798–2817 (2010)
7. Fisher, R.A., Yates, F.: Statistical tables for biological, agricultural and medical research, 3rd edn. (1948)
8. Gao, X., Jain, K., Schulman, L.J.: Fair and efficient router congestion control. In: SODA 2004, pp. 1050–1059 (2004)
9. Gibbens, R.J., Kelly, F.P.: Resource pricing and the evolution of congestion control. Automatica 35, 1969–1985 (1999)
10. Gintis, H.: A markov model of production, trade, and money: Theory and artificial life simulation. Comput. Math. Organ. Theory 3(1), 19–41 (1997)
11. Gintis, H.: Game Theory Evolving: A Problem-Centered Introduction to Modeling Strategic Interaction. Princeton University Press (2000)
12. Karp, R., Koutsoupias, E., Papadimitriou, C., Shenker, S.: Optimization problems in congestion control. In: FOCS 2000, p. 66. IEEE Computer Society (2000)
13. Kiyotaki, N., Wright, R.: On money as a medium of exchange. Journal of Political Economy 97(4), 927–954 (1989)
14. Knuth, D.E.: The Art of Computer Programming, 2nd edn. Seminumerical Algorithms, vol. 2. Addison-Wesley Publishing Company (1981)
15. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
16. MacKie-Mason, J.K., Varian, H.R.: Pricing the internet. In: Public Access to the Internet, pp. 269–314. Prentice Hall (1993)
17. McKnight, L.W., Bailey, J.P. (eds.): Internet Economics. MIT Press (1997)
18. Odlyzko, A.: Paris metro pricing for the internet. In: EC 1999, pp. 140–147. ACM (1999)
19. Papadimitriou, C.: Algorithms, games, and the internet. In: ACM STOC 2001, pp. 749–753 (2001)
20. Shenker, S.J.: Making greed work in networks: a game-theoretic analysis of switch service disciplines. IEEE/ACM Trans. Netw. 3(6), 819–831 (1995)
21. Smith, A.: An Inquiry into the Nature and Causes of the Wealth of Nations. Project gutenberg ebook edition (1776)
22. Wikipedia. Fisher-Yates shuffle (entry) (2011) (accessed February 06, 2011)

Truthful Many-to-Many Assignment with Private Weights[⋆]

Bruno Escoffier[1,2], Jérôme Monnot[2,1], Fanny Pascual[3], and Olivier Spanjaard[3]

[1] PSL, Université Paris-Dauphine, 75775 Paris Cedex 16, France
[2] CNRS, LAMSADE UMR 7243
{escoffier,monnot}@lamsade.dauphine.fr
[3] LIP6-CNRS, UMR 7606, Université Pierre et Marie Curie (UPMC)
4 Place Jussieu, F-75252 Paris Cedex 05, France
{fanny.pascual,olivier.spanjaard}@lip6.fr

Abstract. This paper is devoted to the study of truthful mechanisms without payment for the many-to-many assignment problem. Given n agents and m tasks, a mechanism is truthful if no agent has an incentive to misreport her values on the tasks (agent a_i reports a score w_{ij} for each task t_j). The one-to-one version of this problem has already been studied by Dughmi and Ghosh [4] in a setting where the weights w_{ij} are public knowledge, and the agents only report the tasks they are able to perform. We study here the case where the weights are private data. We are interested in the best approximation ratios that can be achieved by a truthful mechanism. In particular, we investigate the problem under various assumptions on the way the agents can misreport the weights.

Keywords: Algorithmic game theory, truthful mechanism without payment, approximation algorithm, many-to-many assignment problem.

1 Introduction

We study here many-to-many assignment problems, where a set of tasks $\{t_1, \dots, t_m\}$ are assigned to a set of agents $\{a_1, \dots, a_n\}$. Let $x_{ij} = 1$ if task t_j is assigned to agent a_i. We tackle the case where one assigns p tasks per agent and q agents per task, i.e. $\sum_j x_{ij} = p$ and $\sum_i x_{ij} = q$. A weight $w_{ij} \geq 0$ can represent the interest shown by agent a_i to perform task t_j, but also an attendance level, a bandwidth, etc. The aim of the optimization problem is to assign the tasks to the agents so as to maximize some social objective, e.g. $\sum_{i,j} w_{ij}x_{ij}$ (in an *utilitarian* setting) or $\min_i \sum_j w_{ij}x_{ij}$ (in an *egalitarian* setting). This type of problem can occur in various circumstances:

– *Assignment of Papers to Reviewers for a Conference.* Consider a program committee chair who has to assign the m submitted papers (the tasks) to n reviewers (the agents). Each reviewer is assigned p papers and each paper

[⋆] This research has been supported by the project ANR-09-JCJC-0066-01 “Combinatorial Optimization with Competing Agents” (COCA).

is reviewed by q reviewers. In order to perform the assignment, during the bidding phase, for each paper t_j each reviewer a_i indicates an interest score w_{ij} on the conference system.

– *Assignment of Public Resources to Non-profit Associations.* Consider a town council who allocates m time slots/room locations (the tasks) to n sports or cultural associations (the agents). There are q time slots in a week (m/q room locations) and each association has the possibility of being assigned p time slots/room locations per week (once for all). Furthermore, for each time slot t_j, each association a_i indicates the expected number w_{ij} of members that would attend the activity.
– *Assignment of Time Slots on an Earth Observation Satellite.* Consider an earth observation satellite shared among n institutions (the agents). In the daily management of the satellite, there are m time slots (the tasks) in a day and a limited number q of possible accesses to the same slot. For a given institution, there is a maximum of p accesses per day and one access per time slot. Furthermore, for each time slot t_j, each institution a_i indicates the required bandwidth w_{ij}.

In all these circumstances, based on the bids, the tasks are assigned to the agents so as to maximize the social welfare (depending on the social objective). Once the bids are known, the optimization problems involved are solvable in polynomial time, except for the many-to-many case in the egalitarian setting (strongly NP-hard by reduction from the 3-partition problem, see Section 4). These optimization problems have also been investigated from the fairness viewpoint in at least two of the three situations described above, namely for the fair paper assignment problem (see e.g. [5,6,11]) and for the fair sharing of a common satellite resource (see e.g. [10]). Nevertheless, since the weights are private data, an agent might have an incentive to misreport them in order to increase her individual satisfaction (the individual satisfaction of an agent a_i is $\sum_j w_{ij}x_{ij}$).

The aim of the present article is precisely to study *truthful* mechanisms for many-to-many assignment problems. A mechanism is truthful if no agent i can benefit by misreporting her weights w_{ij}. At the same time, the mechanism should guarantee a reasonable approximation to the optimal social objective. Note that we focus on mechanisms without payment, since payments are not always possible or desirable [15] (as in most examples above).

Truthful mechanisms dedicated to the one-to-one version of the problem have already been studied by Dughmi and Ghosh [4] for the utilitarian setting in a restricted case where, for each pair (i, j), it is public knowledge that the value of task t_j for agent a_i is either w_{ij} or 0, but agent a_i holds private which of those is the case for each t_j. The authors justify considering such a discrete valuation model by observing that, assuming the weights are private, no mechanism can outperform the trivial one which allocates the tasks uniformly at random, achieving an approximation ratio of n (as a consequence of classical results of Vickrey concerning single item auctions [16]). Nevertheless, we believe that some interesting results are still possible with private weights.

First, the result given by Dughmi and Ghosh [4] does not hold in the egalitarian setting (in this case, returning a random assignment yields an approximation ratio of $n!$). Second, for both the utilitarian and the egalitarian settings, positive results are possible if one assumes some restrictions on the way the agents misreport their preferences. We consider two types of restrictions: the case where the agents cannot overbid (i.e. they cannot bid weights that are larger than their true weights), and the case where the agents cannot underbid (i.e. they cannot bid weights that are smaller than their true weights). The former case occurs for instance in the assignment of public resources to non-profit associations, as described above: as soon as the actual attendance level at the various activities is controlled afterwards, an association cannot overbid, at the risk of losing credibility. The latter case occurs for instance in the assignment of time slots on an earth observation satellite, as described above: as soon as the bandwidth allocated to an institution is equal to the indicated weight, there is no interest for an institution to bid a value under the required bandwidth (an undersized bandwith is equivalent to a bandwidth 0 for the institution). The rationale for these restrictions is based on the assumption that the agents underbid in order to avoid a task, and conversely they overbid in order to obtain a task. This assumption holds for all the mechanisms we propose. Note that the restriction on the way the agents misreport their preferences is related to the idea of so-called *mechanisms with verification* [13]. Mechanisms with verification have been studied both with payment [14,9] and without payment [1,8].

Our Contribution. In Section 2, we study the case where the weights on the edges are unrestricted. In particular, we provide an n-approximate randomized mechanism for the egalitarian setting and we show that there is no $(n/q - \varepsilon)$-algorithm for this problem. In Section 3, we study the case where the agents cannot overbid. This assumption does not change anything for the egalitarian setting, but it enables to design a 2-approximate truthful mechanism for the utilitarian setting (and we show this is the best ratio that can be achieved by a truthful algorithm). In Section 4, we study the case where the agents cannot underbid. Conversely to the previous case, this assumption does not change anything for the utilitarian setting, but it enables to design an optimal truthful mechanism for the egalitarian setting. Our results are summarized in Table 1, where we indicate lower (LB) and upper (UB) bounds on the approximation ratio of a truthful mechanism. Notation "det" (resp. "rand") stands for deterministic (resp. randomized) mechanisms. When no approximation guarantee can be achieved by a truthful mechanism, we write $LB = \infty$. Note that all positive results are based on polynomial-time algorithms, except the optimal truthful mechanism for agents that do not underbid in the egalitarian setting.

Preliminaries. As customary, we view the many-to-many assignment problem as a maximum weight b-matching problem in a complete bipartite graph G with vertex set $V = (A, T)$ (A for agents, T for tasks). Given degree constraints $b : V \to \mathbb{N}$ for the vertices, a *b-matching* is a set of edges M such that for all $v \in V$ the number of edges in M incident to v, denoted by $\deg_M(v)$, is at most $b(v)$. Every agent a_i has to perform (at most) $b(a_i) = p$ tasks, and every

Table 1. An overview of our results

	utilitarian setting	*egalitarian setting*
no restriction	det: $LB = \infty$ rand: $LB = (\frac{n}{q} - \varepsilon)$, $UB = \frac{n}{q}$	det: $LB = \infty$ rand: $LB = (\frac{n}{q} - \varepsilon)$, $UB = n$
no overbidding	det: $LB = 2 - \varepsilon$ [4], $UB = 2$ rand: $LB = (\frac{4}{3} - \varepsilon)$, $UB = 2$	det: $LB = \infty$ rand: $LB = (\frac{n}{q} - \varepsilon)$, $UB = n$
no underbidding	det: $LB = \infty$ rand: $LB = (\frac{n}{q} - \varepsilon)$, $UB = \frac{n}{q}$	det: $UB = 1$

task t_j has to be assigned to (at most) $b(t_j) = q$ agents. There are $n = |A|$ agents and $m = |T|$ tasks, and we assume that $np = mq$ (otherwise some agents would perform less than p tasks or some tasks would be assigned to less than q agents). The weight $w_{ij} \geq 0$ on edge $\{a_i, t_j\}$ represents the interest (value) of agent a_i for task t_j. Note that, since $np = mq$ and by positivity of the weights and completeness of the graph, constraints $\deg_M(v) \leq b(v)\ \forall v$ are equivalent to $\deg_M(v) = b(v)\ \forall v$ (if some constraints are not saturated, it is always possible to add (an) edge(s) to the b-matching to saturate them). Let $M(i)$ denote the set of indices of the p tasks assigned to agent a_i in the b-matching M. The aim of each agent a_i is to maximize $\sum_{j \in M(i)} w_{ij}$. A deterministic mechanism (algorithm) is truthful if no agent a_i has an incentive to misreport the weights w_{ij} ($j = 1, \ldots, n$): $\sum_{j \in M(i)} w_{ij} \geq \sum_{j \in M'(i)} w_{ij}$ where M (resp. M') is the b-matching returned by the mechanism if agent a_i truthfully reports (resp. misreports) her weights (*ceteris paribus*). For a randomized mechanism, we require truthfulness in expectation: $\mathbb{E}[\sum_{j \in M(i)} w_{ij}] \geq \mathbb{E}[\sum_{j \in M'(i)} w_{ij}]$.

We recall that our aim is to obtain truthful mechanisms returning a b-matching M which maximizes one of the following social objective functions:

- $w(M) = \sum_{i=1}^{n} \sum_{j \in M(i)} w_{ij}$ (in the utilitarian setting),
- or $w(M) = \min_{i \in \{1,\ldots,n\}} \sum_{j \in M(i)} w_{ij}$ (in the egalitarian setting).

A deterministic (resp. randomized) mechanism is said to be c-approximate if, for any instance, $w(M^*)/w(M) \leq c$ (resp. $w(M^*)/\mathbb{E}[w(M)] \leq c$) where M is the b-matching returned by the mechanism, M^* is an optimal b-matching and $\mathbb{E}[w(M)]$ is the expected weight of M.

2 If Agents Have No Restriction on the Way They Bid

In this section, we show that no performance guarantee can be achieved by a truthful deterministic mechanism, in both the utilitarian setting and the egalitarian setting. Furthermore, we show that there is are n-approximate randomized truthful mechanisms (in both settings).

2.1 Utilitarian Setting

We show that no positive result can be expected for a deterministic mechanism.

Theorem 1. *In the utilitarian setting, for any $c > 1$, there is no c-approximate truthful deterministic mechanism, even if $n = m = 2$.*

Proof. Let $c > 1$, $n = m$ and $p = q = 1$ (the b-matching problem reduces to a matching problem). Assume that there is a c-approximate truthful mechanism and consider the graph with two agents a_1, a_2 and two tasks t_1, t_2. Let $\gamma > c$ and $w_{11} = \gamma$, $w_{12} = 0$, $w_{21} = 1$ and $w_{22} = 0$. There are only two perfect matchings $M_1 = \{\{a_1, t_1\}, \{a_2, t_2\}\}$ and $M_2 = \{\{a_1, t_2\}, \{a_2, t_1\}\}$ whose weights are γ and 1 respectively. Since $\gamma > c$, the mechanism must return M_1.

Now, take the situation where $w_{21} = \gamma^2$. The weights of M_1 and M_2 are then γ and γ^2. Again, a c-approximate mechanism must return M_2 since otherwise $\frac{w(M^*)}{w(M_1)} = \frac{\gamma^2}{\gamma} > c$. Consequently, in the first situation, agent a_2 has incentive to declare a false weight $w_{21} = \gamma^2$ in order to get task t_1, with value 1 instead of 0 (with the initial task t_2). ∎

Note that as a consequence there is no $f(n, m)$-approximate deterministic mechanism, for any function f.

In the randomized case, as mentioned earlier, Dughmi and Ghosh [4] have observed that no truthful mechanism can perform better than the mechanism returning a random assignment in the one-to-one case (whose approximation ratio is n since each edge of an optimal matching has a probability $1/n$ to belong to the returned matching). Their statement is based on a reference to a seminal paper by Vickrey [16], whose scope is much broader than the simple assignment problem. This result generalizes to the many-to-many case. First, note that the approximation ratio of the mechanism returning a random b-matching is $n/q = m/p$ since each edge of an optimal b-matching has a probability $q/n = p/m$ to belong to the returned b-matching. The following theorem shows that this is the best we can do:

Theorem 2. *In the utilitarian setting, for any n, any q, any $\varepsilon > 0$, there is no $(\frac{n}{q} - \varepsilon)$-approximate truthful randomized mechanism.*

Proof. Let γ such that $1/\gamma < 1/(n/q - \epsilon) - q/n$. Consider the graph with n agents and n tasks where $w_{11} = \gamma + \frac{1}{q} - 1$, $w_{i1} = 1/q$ for $i \geq 2$, and all the other weights are 0 (note that $p = q$ since $n = m$). Note that the optimal solution has value $w_{11} + (q-1) \times 1/q = \gamma$. Let p_i be the probability that edge (a_i, t_1) is taken in the matching returned by the randomized mechanism. Then since any b-matching without edge (a_1, t_1) (resp. with edge (a_1, t_1)) has value at most 1 (resp. γ), we get that the expected value of the solution is at most $p_1\gamma + (1 - p_1) \leq p_1\gamma + 1$. To achieve ratio $n/q - \varepsilon$, we need $p_1\gamma + (1 - p_1) \geq \gamma/(n/q - \varepsilon)$ and thus $p_1\gamma + 1 \geq \gamma/(n/q - \varepsilon)$. Hence $p_1 \geq 1/(n/q - \varepsilon) - 1/\gamma > q/n$ (by the choice of γ). Since $\sum p_i = q$ (t_1 is adjacent to q edges in the returned b-matching), there exists i such that $p_i < q/n$, say $i = 2$ wlog (note that $i \neq 1$ since $p_1 > q/n$).

Now, consider the same situation as before but with $w_{21} = \gamma^2$. By using the same argument as above, we see that the mechanism has to choose edge (a_2, t_1) with probability greater than q/n.

Then, in the first situation, if agent 2 truthfully reports her weight w_{21} she gets t_1 with probability $p_2 < q/n$, while if she reports a weight γ^2 for this edge she gets t_1 with probability greater than q/n. The mechanism is not truthful. ■

2.2 Egalitarian Setting

Similarly to the utilitarian setting, we show that no positive result can be expected for a deterministic mechanism.

Theorem 3. *In the egalitarian setting, for any $c > 1$, there is no c-approximate truthful deterministic mechanism, even if $n = 2$, $m = 2$ and $p = q = 1$.*

Proof. Let $c > 1$, $n = m = 2$ and $p = q = 1$. By contradiction, assume that there is a c-approximate truthful deterministic mechanism and consider the graph with two agents a_1, a_2 and two tasks t_1, t_2. Let $\gamma > c$ and $w_{11} = \gamma + 1$, $w_{12} = \gamma$, $w_{21} = \gamma + 1$ and $w_{22} = \gamma$. There are only two perfect matchings $M_1 = \{\{a_1, t_1\}, \{a_2, t_2\}\}$ and $M_2 = \{\{a_1, t_2\}, \{a_2, t_1\}\}$. Wlog, assume that the mechanism returns matching M_1.

Now, consider the same situation as before, but $w_{22} = 1$. The weight of matching M_1 (resp. M_2) is 1 (resp. γ), where the weight of M_1 (resp. M_2) is the minimal weight of an edge in M_1 (resp. M_2). Since the mechanism is c-approximate, it must return matching M_2 (because $\gamma > c$).

Then, in the first situation, agent a_2 has incentive to bid a false weight $w_{22} = 1$ in order to get task t_1, with value $\gamma + 1$ instead of γ (with the initial task t_2). Therefore, the mechanism is not truthful. ■

Note that as a consequence there is no $f(n, m)$-approximate truthful deterministic mechanism, for any function f.

In the randomized case, note that, in contrast to the utilitarian setting, returning a random assignment is not n/q-approximate anymore for the egalitarian setting (consider an instance where $p = q = 1$, and thus $n = m$, where all the weights are equal to 0, except weights $w_{i\,i} = 1$ for $i = 1, \dots, n$: there is only one matching over $n!$ that has weight 1, all the others having weight 0). However, we now show that it is possible to get an n-approximation thanks to the following truthful randomized mechanism:

> Let M^* denote an optimal b-matching for the egalitarian setting. Return with probability $1/n$, for each $k \in \{0, \dots, n-1\}$, the b-matching where tasks with indices in $M^*((i + k) \bmod (n))$ are assigned to agent a_i ($i = 1, \dots, n$).

Theorem 4. *In the egalitarian setting, the above mechanism is truthful and n-approximate.*

Proof. This mechanism is truthful since, whatever the bids, every agent a_i has the same expected value $\sum_{k=1}^{n} \frac{1}{n} \sum_{j \in M^*(k)} w_{ij} = \frac{q}{n} \sum_j w_{ij}$ (since each task is assigned to q agents in M^*). For $k = 0$, the mechanism returns M^*. The b-matching M^* is thus returned with probability $1/n$, and the mechanism is n-approximate. ■

Note that this mechanism is also a truthful n-approximate mechanism for the utilitarian setting. The following theorem shows that no truthful mechanism can have an approximation ratio better than n/q (the previous truthful mechanism achieves therefore the best possible ratio when $q = 1$):

Theorem 5. *In the egalitarian setting, for any n, any q and any $\varepsilon > 0$, there is no $(\frac{n}{q} - \varepsilon)$-approximate truthful randomized mechanism, even if $p = q$.*

Proof. Let us consider a truthful randomized mechanism. Consider the graph with n agents and n tasks ($p = q$) where $w_{i1} = \gamma \geq 1$ for $i = 1, \ldots, n$, and all the other weights are $1/q$ (see Figure 1, Situation 1). Since t_1 is assigned to q agents, there exists an agent, say a_1 wlog., such that the randomized mechanism returns a b-matching containing edge $\{a_1, t_1\}$ with probability smaller than or equal to q/n.

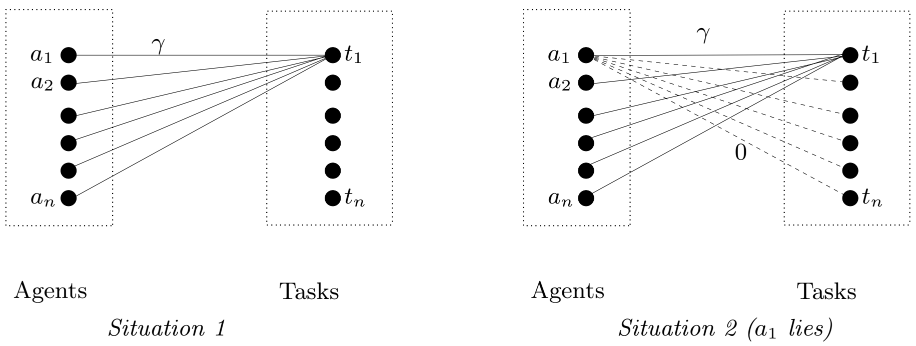


Fig. 1. Illustration of Theorem 5. The sharp edges have weight γ, the dotted edges have weight 0, and the unrepresented edges have weight $1/q$.

Now, consider the same situation as before but $w_{11} = \gamma$ and $w_{1j} = 0$ for $j = 2, \ldots, n$ (see Figure 1, Situation 2). Let p_1 be the probability that the randomized mechanism returns a b-matching containing edge $\{a_1, t_1\}$ for this instance. The expected weight of the returned b-matching is $p_1 \times 1 + (1 - p_1) \times 0 = p_1$ (the only non-zero b-matchings are the ones where edges $\{a_1, t_1\}$ is chosen).

Since the mechanism is truthful, $p_1 \leq q/n$ (otherwise, in situation 1, agent 1 would have incentive to bid false values in order to be in situation 2). Thus, in situation 2, since $p_1 \leq q/n$, the expected weight of the returned b-matching is at most q/n, while the optimal matching has weight 1. The mechanism is therefore at most n/q-approximate. ■

3 If Agents Do Not Overbid

In this section, we assume that the agents cannot bid weights that are strictly larger than their true weights. This assumption does not change anything in the egalitarian setting, since the agents do not overbid in the situation used to establish the lower bound in the unrestricted case. However, we can provide a 2-approximate deterministic truthful mechanism for the utilitarian setting.

Sort the edges by non-increasing weights. Let $M = \emptyset$ and $(e_1, \dots, e_m)$ denote the sorted list of the edges. For i from 1 to m, if $M \cup \{e_i\}$ is a b-matching then $M = M \cup \{e_i\}$. Return M.

This greedy b-matching algorithm has been introduced by Avis [2] for the case $p = q = 1$, who has shown that it is 2-approximate. Mestre [12] has shown that this approximation ratio also holds for b-matchings.

Note that the tie-breaking rule (to decide an ordering over edges of equal weight) matters for the truthfulness of the mechanism. A convenient and simple tie-breaking rule is the following one: if $\{a_i, t_j\}$ and $\{a_{i'}, t_{j'}\}$ are such that $w_{ij} = w_{i'j'}$, $\{a_i, t_j\}$ is ranked before $\{a_{i'}, t'_j\}$ if $i < i'$ or if $i = i'$ and $j < j'$.

Now, we are able to show that the greedy b-matching algorithm is truthful.

Theorem 6. *In the utilitarian setting, if the agents cannot overbid, the greedy b-matching algorithm is a truthful 2-approximate mechanism.*

Proof. We have already said that it returns a 2-approximate matching. We now show that it is also truthful.

By contradiction, assume that agent a_i has incentive to lie on k weights. Let M (resp., M') be the b-matching returned by the algorithm with weights w (resp. w'), i.e., when agent a_i does not (resp. does) misreport her weights. By abuse of notation, we denote by $M(i)$ (resp. $M'(i)$) the set of edges incident to a_i in M (resp. M'). Let $M(i) \setminus M'(i) = \{\{a_i, t_{\pi(1)}\}, \dots, \{a_i, t_{\pi(k)}\}\}$, where $\{a_i, t_{\pi(j)}\}$ is the j^{th} edge of $M(i) \setminus M'(i)$ examined by the algorithm for weights w.

One proceeds as follows to decompose $M \varDelta M' = (M \setminus M') \cup (M' \setminus M)$ into k edge-disjoint cycles $C_1, \dots, C_k$ alternating one edge in $M \setminus M'$ and one edge in $M' \setminus M$, each cycle C_j including edge $\{a_i, t_{\pi(j)}\}$: initially, $C_1 = \emptyset$; note that $\{a_i, t_{\pi(1)}\}$ is necessarily the first edge in $M \varDelta M'$ examined by the algorithm for weights w (provided the agents cannot overbid); insert $\{a_i, t_{\pi(1)}\}$ in C_1; then extend C_1 with the first edge in $M'(t_{\pi(1)}) \setminus M(t_{\pi(1)})$ examined by the algorithm; this latter edge concerns a given agent $a_{i'}$; extend C_1 with the first edge in $M(i') \setminus M'(i')$; and so on until a cycle is created. This is cycle C_1. One iterates to obtain the other cycles $C_2, \dots, C_k$.

Let $\{a_i, t_{\mu(j)}\}$ denote the single edge in $C_j \cap M'(i)$. Since $\{a_i, t_{\mu(j)}\}$ is examined after $\{a_i, t_{\pi(j)}\}$ in cycle C_j, one has $w_{i\,\mu(j)} \leq w_{i\,\pi(j)}$. By summing up these inequalities one obtains $\sum_{j \in M'(i)} w_{ij} \leq \sum_{j \in M(i)} w_{ij}$. Thus agent a_i has no incentive to underbid, and the mechanism is truthful. ∎

The following theorem, due to Dughmi and Ghosh in the simpler case where $p = q = 1$, shows that no truthful deterministic mechanism can have a better approximation ratio:

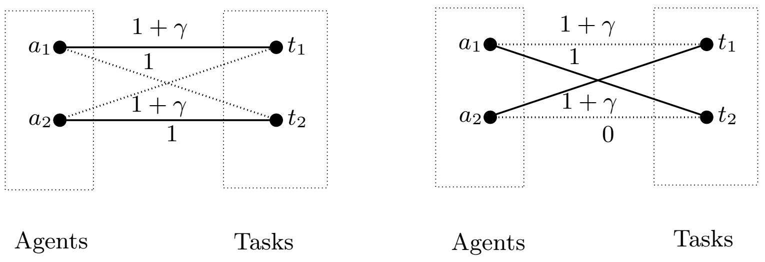
Theorem 7 ([4]). *In the utilitarian setting, for any $\varepsilon \in (0, 1)$, there is no $(2 - \varepsilon)$-approximate truthful deterministic mechanism for the utilitarian setting, even if $n = 2$ and $p = q = 1$.*

A similar proof makes possible to establish the following negative result for randomized mechanisms:

Theorem 8. *In the utilitarian setting, for any $\varepsilon \in (0, 1/3)$, there is no $(4/3-\varepsilon)$-approximate truthful randomized mechanism for the utilitarian setting, even if $n = 2$ and $p = q = 1$.*

Proof. Let $\varepsilon \in (0, 1/3)$. Consider a $\gamma > 0$ such that $\frac{2+\gamma}{3/2+\gamma} > \frac{4}{3} - \varepsilon$.

By contradiction, suppose that there is a $(4/3 - \varepsilon)$-approximate truthful mechanism and consider the graph with 2 agents a_1, a_2 and 2 tasks t_1, t_2. Let $w_{11} = w_{21} = 1 + \gamma$ and $w_{12} = w_{22} = 1$. There are only two perfect matchings $M_1 = \{\{a_1, t_1\}, \{a_2, t_2\}\}$ and $M_2 = \{\{a_1, t_2\}, \{a_2, t_1\}\}$ with same weights and by symmetry, wlog., consider that the mechanism returns the matching M_1 with a probability $p_1 \geq 1/2$ (see Figure 2 case (a). In solid (resp., dotted) lines is the matching M_1 (resp., M_2)).



Case (a): M_1 the matching returned Case (b): agent a_2 lies on weight w_{22}

Fig. 2. Illustration of Theorem 8

Now, take the situation where $w_{22} = 0$ (see Figure 2, case (b)). The weights of matching M_1 (resp. M_2) is $1 + \gamma$ (resp. $2 + \gamma$). By truthfulness, the mechanism should return matching M_1 with probability at least p_1 (otherwise in the initial situation agent 2 would have incentive to bid a false value for edge $\{a_2, t_2\}$). The expected weight of the matching returned by the mechanism in situation 2 is thus $p_1(1 + \gamma) + (1 - p_1)(2 + \gamma) = 2 + \gamma - p_1$. Since $p_1 \geq 1/2$, this weight is smaller than or equal to $3/2 + \gamma$. Since the weight of the optimal matching is $2+\gamma$, the expected approximation ratio of the mechanism is larger then or equal to $\frac{w(M^*)}{w(M_1)} = \frac{2+\gamma}{3/2+\gamma} > \frac{4}{3} - \varepsilon$. Thus the mechanism is not $(4/3 - \varepsilon)$-approximate. ■

4 If Agents Do Not Underbid

In this section, we assume that the agents cannot bid weights that are strictly smaller than their true weights. This assumption does not change anything in the utilitarian setting, since the agents do not underbid in the situations used to establish the lower bound in the unrestricted case.

Let us then consider the egalitarian setting. We will show that there is an optimal truthful mechanism. As a first remark, the optimization problem of finding an optimal b-matching is strongly **NP**-hard: take the 3-partition problem where given a set of $3n$ positive integers $\{s_1, \ldots, s_{3n}\}$ of total sum nB we ask whether there exist n subsets of weight B, each of them consisting of exactly three elements. Build the graph with n agents and $3n$ tasks (one task t_i for each integer s_i), the edges adjacent to a task t_i being weighted by s_i. It is easy to see that the answer to the 3-partition problem is yes if and only if there exists a b-matching of (egalitarian) value B.

So an optimal truthful mechanism is not polynomial time (unless **P**=**NP**), even if $p = 3$. We consider the following mechanism, where P is an upper bound on the optimal value (say P equals the sum of the p maximum edge weights adjacent to agent a_1 for instance).

```
Set $W \leftarrow P$. While no solution has been found:

  – Consider for each agent $a_i$ the set $S_i(W)$ of all sets $T'$ of $p$
    tasks such that $\sum_{t_j \in T'} w_{ij} \geq W$.
  – Use an algorithm A to determine whether there exists a way
    to assign to each $a_i$ a set of tasks in $S_i(W)$ such that each
    task is assigned to $q$ agents.
    If a solution is found, output it. Otherwise, set $W \leftarrow W - 1$.
```

Note that the mechanism terminates: there necessarily exists a feasible assignment for $W = 0$ since the graph is then complete.

Stated like this, the mechanism might not be truthful. More precisely, to make it truthful we have to be more careful on the way sets in $S_i(W)$ are assigned to agents. Informally, by overbidding an agent may add new sets of tasks in $S_i(W)$. We have to be sure that by doing this she will not be better of. To ensure this, we shall use an algorithm A which has the following *stability* property:

Stability: if A assigns on an instance I (defined by the set of agents and the set $S_j(W)$ for each agent a_j) the set of tasks $M(i) \in S_i(W)$ to agent a_i, then on an instance I' where $S'_i(W) \supseteq S_i(W)$ (and $S'_j(W) = S_j(W)$ for each $j \neq i$), A assigns to a_i either $M(i)$ or a set in $S'_i(W) \setminus S_i(W)$.

It is not difficult to see that there exists a stable algorithm A. Indeed, just compute (if any) an assignment of maximal total weight, where the weights of sets in $S_1(W), \ldots, S_n(W)$ are such that any two assignments have different weights, these weights depending only on n (not on W)[1]. This way, when sets are added to $S_i(W)$, if the (unique) maximum weight assignment has not changed a_i receives $M(i)$, otherwise the new maximum weight assignment uses a set in $S'_i(W) \setminus S_i(W)$ for a_i, thus ensuring stability.

Theorem 9. *In the egalitarian setting, if the agents cannot underbid, the mechanism with a stable algorithm A is both optimal and truthful.*

[1] These weights can be defined as follows: the weight of a set $M(k)$ in $S_k(W)$ of tasks is $2^{\alpha_{M(k)}}$ where $\alpha_{M(k)} \in \mathbb{N}$ is different for any two sets of tasks (for instance give a weight $v_{ij} = 2^{i(n-1)+(j-1)}$ to each edge (i, j) in G and define $\alpha_{M(k)}$ as the total weight $\sum_{j \in M(k)} v_{kj}$).

Proof. The b-matching returned by the algorithm at the iteration W has value at least W (by construction). Since we consider W by decreasing value, the mechanism is clearly optimal. For truthfulness, consider a graph G with weights w. Suppose that the mechanism has found a b-matching M when the threshold is W, assigning the set of tasks $M(i)$ to agent a_i. We have $\sum_{j \in M(i)} w_{ij} \geq W$. Suppose that agent a_i reports weights $w'_{ij}, j = 1, \cdots, n$ where $w'_{ij} \geq w_{ij}$. On this new instance G weighted by w' (where $w'_{kj} = w_{kj}$ for $k \neq i$), suppose that the mechanism has found a b-matching M' for threshold W'. Since no underbid is allowed, $W' \geq W$. Hence, two cases may occur.

- If $W' > W$, then necessarily in M' agent a_i receives a set $M'(i)$ where $\sum_{j \in M'(i)} w_{ij} \leq W$. Otherwise, a b-matching would have been found in G weighted by w with threshold (at least) $W + 1$.

- If $W' = W$, then the set $S'_i(W)$ (of the possible sets of p tasks for a_i according to weights w') is obtained from $S_i(W)$ (with weights w) by adding all the subsets of weight less than W according to w, but at least W according to w'. Hence, by the stability property, in M' agent a_i either receives $M(i)$, or one of the added sets whose weight (according to w) is smaller than W. In both cases, $\sum_{j \in M'(i)} w_{ij} \leq \sum_{j \in M(i)} w_{ij}$.

In both cases, agent a_i is not better off, and the mechanism is truthful. ■

In the particular case $p = q = 1$, the problem consists in finding an egalitarian matching and is polynomial time. Our mechanism reduces indeed to finding at each step W if there exists a perfect matching (polynomial time problem) in the graph consisting of edges of weight at least W (using a stable perfect matching algorithm); this is the threshold method [7], which is thus truthful by Theorem 9.

5 Final Remarks

Concerning the unrestricted case, note that the simple truthful mechanism that consists in considering the agents in a random order, and assigning to each of them her p preferred tasks among the available ones (random round robin), is n-approximate in the utilitarian setting[2]. Despite the fact that its performance guarantee is the same as the mechanism returning a random assignment, the random round robin mechanism should be preferred since it is likely to return much better b-matchings in practice.

In order to strengthen the result concerning the case where the agents do not underbid, it would be interesting to investigate what approximation can be achieved truthfully in polynomial time. Besides, another natural research direction is to investigate the group strategyproofness of the mechanisms presented in the paper:

[2] Let W_i be the total weight of the p edges of maximum weight which are adjacent to a_i. The expected cost of the solution returned is at least $\sum_{i=1}^{n}(1/n)W_i$, and the mechanism is clearly n-approximate. The bound is tight when all the edges have weight 0, except the edges adjacent to the same p tasks which have weight $M >> 1$ for the edges adjacent to a_1, and weight 1 for the edges adjacent to the other agents.

a mechanism is *group strategyproof* if for every group of agents $A' \subseteq A$ and every weights w' such that $w'_{ij} = w_{ij}$ if $i \notin A'$, one has $\sum_{j \in M'(i)} w_{ij} \leq \sum_{j \in M(i)} w_{ij}$, where M (resp. M') is the b-matching returned by the mechanism for weights w (resp. w'). A third research direction is to study other restrictions regarding the possible ways that the agents can lie, while still showing positive approximation guarantees. For instance, what happens if $\sum_j w_{ij}$ is a constant for all the agents ? (Think of a conference system that gives the same capital of points to each reviewer) Another possible domain restriction is the following: when attributing time slots to agents, one can assume that the preferences of the agent are single-peaked [3], i.e. each agent has an ideal time slot and her preferences decrease when moving away from the ideal point. It would be interesting to study dedicated truthful mechanisms taking advantage of these domain restrictions.

Acknowledgments. We wish to thank the anonymous reviewers for their fruitful comments on an earlier version of the paper.

References

1. Angel, E., Bampis, E., Pascual, F.: Truthful algorithms for scheduling selfish tasks on parallel machines. Theoretical Computer Science 369(1-3), 157–168 (2006)
2. Avis, D.: A survey of heuristics for the weighted matching problem. Networks 13, 475–493 (1983)
3. Black, D.: The Theory of Committees and Elections. Cambridge University Press (1958)
4. Dughmi, S., Ghosh, A.: Truthful assignment without money. In: ACM Conference on Electronic Commerce, pp. 325–334 (2010)
5. Garg, N., Kavitha, T., Kumar, A., Mehlhorn, K., Mestre, J.: Assigning papers to referees. Algorithmica 58(1), 119–136 (2010)
6. Golden, B., Perny, P.: Infinite order Lorenz dominance for fair multiagent optimization. In: AAMAS, pp. 383–390 (2010)
7. Gross, O.: The bottleneck assignment problem. Technical Report P-1630, The Rand Corporation, Sta. Monica, CA (1959)
8. Koutsoupias, E.: Scheduling without payments. In: Persiano, G. (ed.) SAGT 2011. LNCS, vol. 6982, pp. 143–153. Springer, Heidelberg (2011)
9. Krysta, P., Ventre, C.: Combinatorial auctions with verification are tractable. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part II. LNCS, vol. 6347, pp. 39–50. Springer, Heidelberg (2010)
10. Lemaître, M., Verfaillie, G., Fargier, H., Lang, J., Bataille, N., Lachiver, J.M.: Equitable allocation of earth observing satellites resources. In: Proc of 5th ONERA-DLR Aerospace Symposium, ODAS 2003 (2003)
11. Lesca, J., Perny, P.: LP solvable models for multiagent fair allocation problems. In: ECAI, pp. 393–398 (2010)
12. Mestre, J.: Greedy in approximation algorithms. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 528–539. Springer, Heidelberg (2006)
13. Nisan, N., Ronen, A.: Algorithmic mechanism design (extended abstract). In: STOC, pp. 129–140 (1999)
14. Penna, P., Ventre, C.: Optimal collusion-resistant mechanisms with verification. In: ACM Conference on Electronic Commerce, pp. 147–156 (2009)
15. Schummer, J., Vohra, R.V.: Mechanism Design without Money. In: Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V. (eds.) Algorithmic Game Theory, pp. 243–266. Cambridge University Press (2007)
16. Vickrey, W.: Counterspeculation, Auctions and Competitive Sealed Tenders. Journal of Finance, 8–37 (1961)

Competitive Online Clique Clustering

Aleksander Fabijan[1], Bengt J. Nilsson[2], and Mia Persson[2]

[1] Faculty of Computer and Information Science, University of Ljubljana, Slovenia
Aleksander.Fabijan@mf.uni-lj.si

[2] Department of Computer Science, Malmö University, Sweden
{Bengt.Nilsson.TS,Mia.Persson}@mah.se

Abstract. Clique clustering is the problem of partitioning a graph into cliques so that some objective function is optimized. In online clustering, the input graph is given one vertex at a time, and any vertices that have previously been clustered together are not allowed to be separated. The objective here is to maintain a clustering the never deviates too far in the objective function compared to the optimal solution. We give a constant competitive upper bound for online clique clustering, where the objective function is to maximize the number of edges inside the clusters. We also give almost matching upper and lower bounds on the competitive ratio for online clique clustering, where we want to minimize the number of edges between clusters. In addition, we prove that the *greedy* method only gives linear competitive ratio for these problems.

1 Introduction

The correlation clustering problem and its different variants have been extensively studied over the past decades; see e.g. [2,5,6]. Several objective function are used in the literature, e.g., maximize the number of edges within the clusters plus the number of non-edges between clusters (MaxAgree), or minimize the number of non-edges inside the clusters plus the number of edges outside them (MinDisagree). In [2], Bansal et al. show that both the minimization (minimizing the number of disagreement edges) and the maximization (maximizing the number of agreement edges) versions are in fact NP-hard. However, from the point of view of approximation the maximization and minimization versions differ. In the case of maximizing agreements this problem actually admits a PTAS whereas in the case of minimizing disagreements it is APX-hard. Several efficient constant factor approximation algorithms are proposed when minimizing disagreements [2,5,6] and maximizing agreements [5].

Other measures require that the clusters are cliques, complete subgraphs of the original graph, in which case we can maximize the number of edges inside the cluster or minimize the number of edges outside the clusters. These measures give rise to the maximum and minimum edge clique partition problems (Max-ECP and Min-ECP for short) respectively; the computitional complexity and approximability of the aforementioned problems have attracted significant attention recently [7,9,11], and they have numerous applications within the areas of gene expression profiling and DNA clone classification [1,3,8,11].

In this paper, we consider the online variant of clique clustering where the vertices arrive one at a time, i.e. the input sequence is not known in advance. Specifically, upon an arrival of a new vertex, it is either clustered into an already existing cluster or forms a new singleton cluster. In addition, existing clusters can be merged together. The merge operation in an online setting is irreversible, once vertices are clustered together, they will remain so, and hence, a bad decision will have significant impact on the final solution. This online model was proposed by Charikar et al. [4] and has applications in information retrieval.

Our Results. We investigate the online Max-ECP and Min-ECP clustering for unweighted graphs and we provide upper and lower bounds for both these versions of clique clustering. Specifically, we consider the natural greedy strategy and prove that it is not constant competitive for Max-ECP clustering but has a competitive ratio that is at best inversely proportional to the number of vertices in the input. We prove that no deterministic strategy can have competitive ratio larger than 1/2. We further give a strategy for online Max-ECP clustering that yield constant competitive ratio. For Min-ECP clustering, we show a lower bound of $\Omega(n^{1-\epsilon})$, for any $\epsilon > 0$, for the competitive ratio of any deterministic strategy. The greedy strategy provides an almost matching upper bound since greedy belongs to the class of *maximal* strategies and these are shown to have competitive ratio $O(n)$. See Table 1 for a summary of our results.

Table 1. Summary of our results

Problem	*Lower Bound*	*Upper Bound*
Greedy Max-ECP	$2/(n-2)$	$1/(n-2)$
Max-ECP	$1/2$	0.032262
Greedy Min-ECP	$(n-2)/2$	$2n-3$
Min-ECP	$n^{1-\epsilon}/2$	

2 Preliminaries

We begin with some notation and basic definitions of the Max-ECP and Min-ECP clustering problems. They are defined on an input graph G, where $G = (V, E)$ with vertices V and edges E. Hence, we wish to find a partitioning of the vertices in V into clusters so that each cluster induces a clique in the corresponding subgraph of G. In addition, we want to optimize some objective function associated to the clustering. In the Max-ECP case, this is to maximize the total number of edges inside the clusters (agreements), whereas in the Min-ECP case, we want to minimize the number of edges outside the clusters (disagreements).

We will use the online models, motivated by information retrieval applications, proposed by [4], and by [10] for the online correlation clustering problem. We define *online* versions of Max-ECP and Min-ECP clustering in a similar way as [10]. Vertices (with their edges to previous vertices) arrive one at a time and must be clustered as they arrive. The only two operations allowed are: *singleton*(v), that creates a singleton cluster containing the single vertex v and *merge*(C_1, C_2), which merges two existing clusters into one, given that the resulting cluster induces a clique in the underlying graph. This means that once two vertices are clustered together, they can never be separated again.

Let C be a clustering consisting of clusters $c_1, c_2, \ldots, c_m$, where each cluster c_i forms a clique. The *profit* of c_i is $p(c_i) \overset{\text{def}}{=} \binom{|c_i|}{2}$ and the profit of C is

$$p(C) \overset{\text{def}}{=} \sum_{i=1}^{m} p(c_i) = \sum_{i=1}^{m} \binom{|c_i|}{2}.$$

We define the *cost* of C to be $|E| - p(C)$, where E is the set of edges in the underlying graph. Hence, in Max-ECP, we want to maximize the profit of the generated clustering and in Min-ECP we want to minimize the cost of the clustering.

It is common to measure the quality of an online strategy by its *competitive ratio*. This ratio is defined as the worst case ratio between the profit/cost of the online strategy and the profit/cost of an offline optimal strategy, one that knows the complete input sequence in advance. We will use the competitive ratio to measure the quality of our online strategies.

Note that in the online model, the strategy does not know when the last vertex arrives and as a consequence any competitive ratio needs to be kept for every vertex that arrives, from the first to the last.

We henceforth let OPT denote an offline optimal solution to the clustering problem we are currently considering. The context will normally suffice to specify which optimum solution is meant. We use OPT_{kn} to denote the offline optimum solution on vertices $v_{k+1}, \ldots, v_n$, where $k < n$ and these vertices are indexed in their order in the input sequence. We also use OPT_n to denote OPT_{0n}. Similarly, we use S_n to denote the solution of an online strategy on the n first vertices.

Note that we make no claims on the computational complexity of our strategies. In certain cases, our strategies use solutions to computationally intractable problems (such as clique partitioning problems in graphs) which may be considered impractical. However, our interest focuses on the relationship between the results of online strategies and those of offline optimal solutions, we believe that allowing the online strategy to solve computationally difficult tasks gives a fairer comparison between the two solutions.

3 Online Max-ECP Clustering

3.1 The Greedy Strategy for Online Max-ECP Clustering

The *greedy* strategy for Max-ECP clustering merges each input vertex with the largest current cluster that maintains the clique property. If no such merging is possible the vertex is placed in a singleton cluster. Greedy strategies are natural first attempts used to solve online problems and can be shown to behave well for certain of them. We show that the greedy strategy can be far from optimal for Max-ECP clustering.

Theorem 1. *The greedy strategy for Max-ECP clustering is no better than $2/(n-2)$-competitive.*

Proof. Consider an adversary that provides input to the strategy to make it behave as badly as possible. Our adversary gives greedy $n = 2k$ vertices in order from 1 to $2k$. Each odd numbered vertex is connected to its even successor, each odd numbered vertex is also connected to every other odd numbered vertex before it, and similarly, each even numbered vertex is also connected to every even numbered vertex before it; see Figure 1.

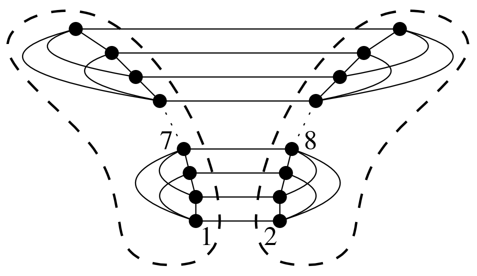


Fig. 1. Illustrating the proof of Theorem 1

The greedy strategy clusters the vertices as odd/even pairs, giving the clustering GDY_n having profit $p(GDY_n) = k$. An optimal strategy clusters the odd vertices in one clique of size k and the even vertices in another clique also of size k. The profit for the optimal solution is $p(OPT_n) = k(k-1)$. Hence, the worst case ratio between greedy and an optimum solution is at most $1/(k-1) = 1/(n/2-1)$ so the competive ratio is at most $2/(n-2)$. □

Next, we look at the upper bound for the greedy strategy.

Theorem 2. *The greedy strategy for Max-ECP clustering is $1/(n-2)$-competitive.*

Proof. Consider an edge e inside a cluster produced by the greedy strategy on n vertices. We introduce a weight function $w(e)$ as follows: if e also belongs to a cluster in OPT_n, then we set $w(e) \overset{\text{def}}{=} 1$. If e does not belong to any cluster in OPT_n, then the two endpoints v and v' belong to different clusters of OPT_n. Denote these two clusters by c_e and c'_e and we set $w(e) \overset{\text{def}}{=} |c_e| + |c'_e| - 2$, i.e., the number of edges in c_e and c'_e connected to v and v'. Note that not both c_e and c'_e can be singleton clusters, so $w(e) \geq 1$ in all cases.

Consider now the sum, $\sum_{e \in GDY_n} w(e)$, where we abuse notation slightly and let $e \in GDY_n$ denote that two end points of edge e lies in the same cluster of GDY_n, the greedy clustering on n vertices. The sum counts every edge in OPT_n at least once, since an edge lying in both GDY_n and OPT_n is counted once and an edge lying in GDY_n but not in OPT_n counts all the edges in OPT_n connected to the two endpoints. The fact that the sum counts all the edges in OPT_n follows since no two clusters in greedy can be merged to a single cluster. Hence,

$$p(OPT_n) \leq \sum_{e \in GDY_n} w(e) \leq \sum_{e \in GDY_n} |c_e| + |c'_e| - 2 \leq (|c_1| + |c_2| - 2) \cdot \sum_{e \in GDY_n} 1$$
$$\leq (|c_1| + |c_2| - 2) \cdot p(GDY_n) \leq (n-2) \cdot p(GDY_n),$$

where c_1 and c_2 denote the two largest clusters in OPT_n. □

3.2 A Lower Bound for Online Max-ECP Clustering

We present a lower bound for deterministic Max-ECP clustering.

Theorem 3. *Any deterministic strategy for Max-ECP clustering is at most* 1/2-*competitive.*

Proof. Again we use an adversarial argument and let the adversary provide $2k$ vertices, where every odd numbered vertex is connected to its subsequent even numbered vertex, v_1 to v_2, v_3 to v_4, etc. The game now continues in stages with the strategy constructing clusters followed by the adversary adding edges. In each stage the adversary looks at the clusters constructed; these are either singletons or pairs $\{v_{2i-1}, v_{2i}\}$. For each newly constructed pair cluster, the adversary adds two new vertices, v'_{2i-1} connected to v_{2i-1}, and, v'_{2i} connected to v_{2i}; see Figure 2. When the strategy fails to produce any new pair clusters in a stage, the adversary stops.

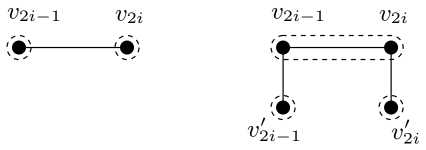


Fig. 2. Illustrating the proof of Theorem 3

Assume that the strategy at the end of the stages has constructed k' pair clusters, $k' \leq k$, thus giving a profit of k'. Note that the strategy can never produce the pairs $\{v_{2i-1}, v'_{2i-1}\}$ or $\{v_{2i}, v'_{2i}\}$ since these are revealed only if the pair $\{v_{2i-1}, v_{2i}\}$ is produced. The optimal solution in this case has profit $k + k'$ since this solution produces $2k'$ pair clusters $\{v_{2i-1}, v'_{2i-1}\}$ or $\{v_{2i}, v'_{2i}\}$, where the strategy produces $\{v_{2i-1}, v_{2i}\}$, in addition to $k - k'$ pairs $\{v_{2i-1}, v_{2i}\}$, where the strategy produces singleton clusters. Hence, the competitive ratio is

$$\frac{k'}{k + k'} \leq \frac{1}{2}, \qquad \text{for } 0 \leq k' \leq k,$$

giving the result. □

3.3 A Constant Competitive Strategy for Online Max-ECP Clustering

We present a new strategy for Max-ECP clustering and prove that it has constant competitive ratio. If the optimum solution OPT_n does not have any clusters of size larger than three, the strategy follows the greedy strategy. Otherwise, the strategy places arriving vertices in singleton clusters until the profit ratio between the current solution S'_n and the offline optimum solution OPT_n (of the n currently known vertices) goes below a threshold value c. When this happens, the strategy computes the *relative optimum* solution given the current clustering. The strategy is given in pseudocode below.

```
Strategy  Lazy
    /* Maintain clustering $S_n$ with profit $p(S_n)$ and let $c$ be a constant */
1      $n = 1$
2      while new vertex $v_n$ arrives do
2.1        $S'_n := S_{n-1} + singleton(v_n)$
2.2        Compute $OPT_n$
2.3        if the largest cluster in $OPT_n$ has size $\geq 4$ then
2.3.1          if $p(OPT_n) > c \cdot p(S'_n)$ then
2.3.1.1            Compute the relative optimum of $S'_n$, $\widehat{OPT}(S'_n)$
2.3.1.2            Construct $S_n$ from $\widehat{OPT}(S'_n)$ using only merge operations
               else
2.3.1.3            $S_n := S'_n$
               endif
           else
2.3.2          Construct $S_n$ using the Greedy strategy
           endif
2.4        Report $S_n$
2.5        $n := n + 1$
       endwhile
End  Lazy
```

Given a clustering S, the *relative optimum*, $\widehat{OPT}(S)$, is defined as follows: construct a graph G_S such that, for every cluster in S there is a vertex in G_S and two vertices in G_S are connected by an edge, if every pair of vertices in the two underlying clusters are connected. $\widehat{OPT}(S)$ is now the offline optimal clustering in G_S.

Given the current clustering, S'_n, the new clustering, S_n, is easily generated by constructing a cluster in S_n for each cluster in $\widehat{OPT}(S'_n)$ by merging the corresponding clusters in S'_n.

The following theorem follows directly from the construction of the strategy since the ratio between the profit of the optimal solution (given the current n vertices) and the profit of the online solution S_n never falls below the threshold c.

Theorem 4. *The Lazy strategy is $1/c$-competitive for online Max-ECP clustering.*

We will establish the value of c to be $c = (154 + 16\sqrt{61})/9$ in Lemma 3.

We give a relationship between the profits of the two clusterings OPT_{n-1} and OPT_n.

Lemma 1. *Let $c_{\max}$ be the largest cluster in OPT_{n-1}, having size k, then, for all $n > 2$ the profit $p(OPT_n) \leq p(OPT_{n-1}) \cdot (k+1)/(k-1)$.*

Proof. The maximum increase occurs if v_n, the arriving vertex, can be clustered together with the vertices in $c_{\max}$. The increase in profit in this cluster goes from $\binom{k}{2}$ to $\binom{k+1}{2}$. The maximum increase for the whole clustering occurs if $c_{\max}$ is the only non-singleton cluster in OPT_{n-1}, giving us a ratio of $\binom{k+1}{2}/\binom{k}{2} = (k+1)/(k-1)$. □

Let G be an undirected graph and let G_A and G_B be the two subgraphs induced by some partitioning of the vertices in G. Let C be a clustering on G and let A and B be the clusterings induced by C on G_A and G_B respectively.

Lemma 2. *If $p(A) > 0$ and $p(C)/p(A) = z > 1$, then $p(B)/p(C) \geq r(z)$ where $r(z)$ is*

$$r(z) = 1 - \frac{\sqrt{1+8z}-2}{z}.$$

Proof. Our proof is by induction on the number of clusters in C. We assume the clusters $c_1, \ldots, c_m$ in C are sorted on increasing number of vertices in a_i, where a_i is the cluster in A induced by the cluster c_i in C.

Similarly we denote by b_i the cluster in B induced by the cluster c_i in C.

We say that a cluster c_i is a *null cluster*, if the induced cluster a_i in A has $p(a_i) = 0$. This happens if a_i is either empty or a singleton set.

We first prove the base case, where we assume that C contains exactly one non-null cluster, i.e., $c_1, \ldots, c_{j-1}$ are clusters such that $p(a_i) = 0$, for $1 \leq i < j$ and c_j is the first cluster where $p(a_j) > 0$. Assume that $p(c_j)/p(a_j) = z''$ and that $|a_j| = l$, $|b_j| = l'$ and $|c_j| = l + l'$.

We prove the base case of the induction also using induction and assume for this base case that $j = 1$. In this case, $z = p(C)/p(A) = p(c_1)/p(a_1) = z''$ and we get by straightforward calculations that

$$\frac{p(B)}{p(C)} = \frac{p(b_1)}{p(c_1)} = 1 - \frac{\sqrt{1+4zl(l-1)}-l}{z(l-1)} \geq r(z),$$

since the expression before the inegueality is increasing in l, and therefore minimized when $l = 2$.

For the inductive case of the base case, we assume the result holds for $j-2 \geq 0$ null clusters and one non-null cluster and prove it for $j-1$ null clusters and one non-null cluster. Let $\{c_2, \ldots, c_j\}$ be denoted by C' and let A' and B' be the induced clusterings of C' in G_A and G_B. We set $p(C')/p(A') = z'$ and have when we add null cluster c_1 to the clustering that

$$z = \frac{p(C)}{p(A)} = \frac{p(C') + p(c_1)}{p(a_j)} = z' + \frac{p(c_1)}{p(a_j)},$$

giving us that $z' \leq z$ and

$$\frac{p(B)}{p(C)} = \frac{p(b_1) + p(B')}{p(C)} \geq \frac{p(c_1) - |c_1| + 1 + p(B')}{p(c_1) + p(C')}.$$

The inequality stems from the fact that a_1 can either be a singleton or an empty cluster, so b_1 either contains the same number of vertices as c_1 or one less.

By the induction hypothesis we have that $p(B') \geq r(z') \cdot p(C')$, and since $p(A) = p(A') = p(a_j) = p(c_1)/(z - z')$, $p(C') = z'p(a_j) = zp(a_j) - p(c_1)$, and $|c_1| = (1 + \sqrt{1 + 8(z - z')p(a_j)})/2$, this gives us that

$$\begin{aligned}\frac{p(B)}{p(C)} &\geq \frac{(z - z')p(a_j) + (1 - \sqrt{1 + 8(z - z')p(a_j)})/2 + z'r(z')p(a_j)}{zp(a_j)} \\ &= 1 + \frac{z'(r(z') - 1)}{z} + \frac{1 - \sqrt{1 + 8(z - z')p(a_j)}}{2zp(a_j)} \\ &\geq 1 + \frac{z'(r(z') - 1)}{z} + \frac{1 - \sqrt{1 + 16(z - z')}}{4z}.\end{aligned}$$

The last expression is a decreasing function of z' between 0 and z, so increasing z' to z yields $p(B)/p(C) \geq r(z)$. Hence the base case when C has zero or more null clusters and exactly one non-null cluster has been completed.

For the general induction step, assume the formula holds for $m - 1$ clusters, prove it for m clusters. We let $p(C)/p(A) = z$, $C' = \{c_1, \ldots, c_{m-1}\}$, $C = C' \cup \{c_m\}$, $p(C')/p(A') = z'$ and $p(c_m)/p(a_m) = z''$.

By the induction hypothesis we have that

$$\frac{p(B)}{p(C)} \geq \frac{r(z') \cdot p(C') + r(z'') \cdot p(c_n)}{p(C') + p(c_n)} = r(z')\frac{z'(z - z'')}{z(z' - z'')} + r(z'')\frac{z''(z' - z)}{z(z' - z'')}$$

The last expression decreases as z'' tends towards z, again giving us $p(B)/p(C) \geq r(z)$, thus proving our result. □

Lemma 3. *If, for a certain value of n, the selection in Step 2.3.1 yields true in the lazy strategy, then the profit $p(OPT_n) \leq a \cdot p(S_n)$ where $a < c$ is some constant.*

Proof. When the largest clusters in OPT_n has size at most three, we have from the proof of Theorem 2 that greedy has competitive ratio 1/4, and lazy will do at least as well in this case, since it follows the greedy strategy. So, we can assume that the largest cluster in OPT_n has size at least four. This also means that the size of the largest cluster in OPT_{n-1} is at least three.

We make a proof by induction on n, the number of steps in the algorithm. The base cases when $n = 1$, 2 and 3 follow immediately, since lazy (and greedy) computes optimal solutions in these cases, so $a \leq 4$ can be chosen as the constant, since the competitive ratio is $1/a \geq 1/4$.

Assume for the inductive case that Step 2.3.1 yields true at the n-th iteration and assume further that the previous time it happened it was in iteration k (or that the strategy followed greedy in this step). By the induction hypothesis we know that $p(OPT_k) \leq a \cdot p(S_k)$ for some constant $a < c$. Let OPT'_k be the clustering obtained from OPT_n induced by the vertices $v_1 \ldots v_k$. It is obvious that $p(OPT'_k) \leq p(OPT_k)$. Let E_{kn} be the set of edges between vertices inside clusters of OPT_n that have both endpoints among the vertices $v_{k+1} \ldots v_n$. Similarly, we define E'_{kn} to be the set of edges inside clusters that have one end point among the vertices $v_1 \ldots v_k$ and the other among $v_{k+1} \ldots v_n$. We now have that

$$p(OPT'_k) + |E'_{kn}| + |E_{kn}| = p(OPT_n).$$

Let S'_n be the clustering solution in iteration n just before the strategy reaches Step 2.3.1, i.e., when vertex v_n is put in a singleton cluster. This gives us, since $p(S'_n) = p(S_k)$,

$$p(OPT_n) > c \cdot p(S'_n) = c \cdot p(S_k) \geq \frac{c}{a} \cdot p(OPT'_k)$$

Since $p(OPT_n)/p(OPT'_k) \geq c/a$, by Lemma 2 the ratio $|E_{kn}|/p(OPT_n) \geq r(c/a)$.

Note that E_{kn} forms a clustering of vertices $v_{k+1}, \ldots, v_n$ that is independent of how vertices $v_1, \ldots, v_k$ are clustered. Therefore, when an new cluster S_n is recomputed in Step 2.3.1.1, it includes at least as many edges as both S_k and E_{kn} together. Furthermore, $p(S_{n-1}) = p(S_k)$ and $p(OPT_{n-1}) \leq c \cdot p(S_{n-1})$, since otherwise Step 2.3.1.1 would have been done already in the previous iteration. We have that

$$\begin{aligned} p(S_n) &\geq p(S_k) + p(OPT_{kn}) \geq p(S_k) + |E_{kn}| = p(S_{n-1}) + |E_{kn}| \\ &\geq \frac{p(OPT_{n-1})}{c} + |E_{kn}| \geq \frac{p(OPT_n)}{2c} + |E_{kn}| \\ &\geq \frac{p(OPT_n)}{2c} + r(c/a) \cdot p(OPT_n). \end{aligned}$$

The second to last inequality follows from Lemma 1, since the largest cluster in OPT_{n-1} must have size 3, and the last inequality was given above.

We must guarantee that

$$\frac{p(OPT_n)}{2c} + r(c/a) \cdot p(OPT_n) \geq \frac{p(OPT_n)}{a}$$

to prove the lemma, which is equivalent to finding constants $a \leq 4$ and c as small as possible so that $1/(2c) + r(c/a) \geq 1/a$. The inequality holds for $a = 4$ and in equality for $c = (154 + 16\sqrt{61})/9 \approx 30.9960$. □

From Theorem 4 it follows that the competitive ratio for the lazy strategy is $9/(154 + 16\sqrt{61}) \approx 0.032262$.

4 Online Min-ECP Clustering

4.1 A Lower Bound for Online Min-ECP Clustering

We present a lower bound for deterministic Min-ECP clustering.

Theorem 5. *Any deterministic strategy for Min-ECP clustering is no better than $n^{1-\epsilon}/2$-competitive, for every $\epsilon > 0$.*

Proof. An adversary provides the following vertices of an input graph in sequence. First, one $(k-1)$-clique, followed by one additional vertex connected to one of the previously given vertices, i.e., a lollipop graph; see Figure 3.

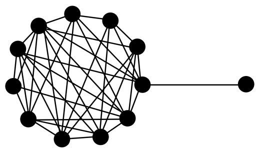


Fig. 3. A lollipop graph with a $(k-1)$-clique and a vertex connected by a single edge

We now consider different possibilities for clustering the k vertices. For convenience, we shall call an edge whose endpoints lie in two different clusters of a given clustering a *disagreement edge* (*disagreement* for short) for this clustering. First, let us assume that the strategy has clustered the input in such a way that the $(k-1)$-clique is not clustered as one cluster. Then this clustering has at least $k-2$ disagreements. An optimal clustering contains only one disagreement, between the $(k-1)$-clique and a singleton cluster containing the vertex outside the clique. Hence, the competitive ratio in this case is at least $(k-2)/1 = n-2$ since the number of vertices is $n = k$.

Assume next that the strategy has clustered the input as one $(k-1)$-clique and one singleton cluster. In this case, the adversary provides $k-1$ independent cliques of size m, where each of the vertices in an m-clique is connected to one particular vertex of the original $(k-1)$-clique; see Figure 4. No other edges exist in the input.

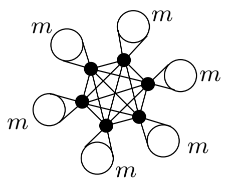


Fig. 4. Each of the $(k-1)$ vertices in the central clique is connected with m edges to an m-clique

The strategy can at best cluster the $k-1$ m-cliques as clusters, thus generating $m(k-1)$ disagreements. An optimal solution will, for m sufficently large, cluster the vertices in the original $(k-1)$-clique in each of the new cliques, generating a solution of $k-1$ $(m+1)$-cliques. This solution has $\binom{k-1}{2}$ disagreements. If we set $m = (k-2)^t/4$, where k is chosen so that m is an integer and t is some sufficiently large integer, then the competitive ratio becomes

$$\frac{(k-1)m}{\binom{k-1}{2}} = \frac{(k-2)^{t-1}}{2} \geq \frac{1}{2}\left(n^{1/(t+1)}\right)^{t-1} \geq \frac{n^{1-\frac{2}{t+1}}}{2} = \frac{n^{1-\epsilon}}{2},$$

for all $\epsilon > 0$, since the number of vertices in the input is $n = (k-1)m + k = (k-1)(k-2)^t/4 + k \leq (k-2)^{t+1}$, proving the lower bound. □

4.2 The Greedy Strategy for Online Min-ECP Clustering

In this section, we prove that the greedy strategy yields a competitive ratio of $n-2$, almost matching the lower bound provided in section 4.1. The greedy strategy was presented in Section 3.1.

Theorem 6. *The greedy strategy for Min-ECP clustering is no better than* $(n-2)/2$*-competitive.*

Proof. We let an adversary generate the same input sequence of $n = 2k$ vertex pairs as in the proof of Theorem 1. Greedy generates $2\binom{k}{2}$ disagreement edges whereas the optimum solution has k disagreement edges. The competitive ratio becomes

$$\frac{2\binom{k}{2}}{k} = k - 1 = \frac{n-2}{2}.$$

□

We say that a solution to Min-ECP clustering is *maximal*, if the solution cannot be improved by the merging of any clusters. A strategy for Min-ECP clustering is called *maximal*, if it always produces maximal solutions. Note that greedy belongs to this class of maximal strategies.

Theorem 7. *Any maximal online strategy for Min-ECP clustering problem is* $2n-3$*-competitive.*

Proof. Consider a disagreement edge e connecting vertices v and v' outside any cluster produced by the maximal strategy MAX_n on n vertices. We start by showing that either v or v' must have an adjacent edge that is a disagreement edge in OPT_n. We have two cases: if e is also a disagreement edge in OPT_n, there is a disagreement edge in OPT_n adjacent to v or v'.

Now, if e is not a disagreement edge in OPT_n, then one of v and v' connects to a vertex u, assume it is v, such that the edge $e' = (v, u)$ is a disagreement edge in OPT_n, i.e., there is a cluster in MAX_n containing v and u but not v' and there is a cluster in OPT_n containing v and v' but not u. The vertex u must exist, otherwise MAX_n would have clustered v and v' together, a contradiction.

In this way, we have proved that to each disagreement edge in MAX_n, there must be an adjacent disagreement edge in OPT_n.

Consider now a disagreement edge e in OPT_n. Potentially, all its adjacent edges can be disagreement edges for MAX_n, giving us in the worst case $2n - 4$ adjacent disagreement edges different from e and one where they coincide. Hence the worst case competitive ratio is $2n - 3$. □

From our observation that greedy belongs to the class of maximal strategies we have the following corollary.

Corollary. *Greedy is* $2n - 3$*-competitive.*

5 Conclusion

We have proved almost matching upper and lower bounds for clique clustering. Our main result is a constant competitive strategy for clique clustering when the cost measure is to maximize the total number of edges in the cliques. Our strategy does not consider the computational feasibility of clique clustering, which can be considered a drawback. However, we feel that since the lower bound adversarial arguments also allow this computation, our measure is fairer to the strategy. In addition, the computational problems required to be solved efficiently for our strategy are indeed efficiently solvable for large classes of graphs, such as chordal graphs, line graphs and circular-arc graphs [7].

References

1. Alvey, S., Borneman, J., Chrobak, M., Crowley, D., Figueroa, A., Hartin, R., Jiang, G., Scupham, A.T., Valinsky, L., Vedova, D., Yin, B.: Analysis of bacterial community composition by oligonucleotide fingerprinting of rRNA genes. Applied and Environmental Microbiology 68, 3243–3250 (2002)
2. Bansal, N., Blum, A., Chawla, S.: Correlation Clustering. Machine Learning 56(1-3), 89–113 (2004)
3. Ben-Dor, A., Shamir, R., Yakhini, Z.: Clustering Gene Expression Patterns. Journal of Computational Biology 6(3/4), 281–297 (1999)
4. Charikar, M., Chekuri, C., Feder, T., Motwani, R.: Incremental Clustering and Dynamic Information Retrieval. SIAM J. Comput. 33(6), 1417–1440 (2004)
5. Charikar, M., Guruswami, V., Wirth, A.: Clustering with Qualitative Information. In: Proc. 44th Annual Symposium on Foundations of Computer Science, FOCS 2003, pp. 524–533 (2003)
6. Demaine, E.D., Immorlica, N.: Correlation Clustering with Partial Information. In: Arora, S., Jansen, K., Rolim, J.D.P., Sahai, A. (eds.) RANDOM 2003 and APPROX 2003. LNCS, vol. 2764, pp. 1–13. Springer, Heidelberg (2003)
7. Dessmark, A., Jansson, J., Lingas, A., Lundell, E., Persson, M.: On the Approximability of Maximum and Minimum Edge Clique Partition Problems. Int. J. Found. Comput. Sci. 18(2), 217–226 (2007)
8. Figueroa, A., Borneman, J., Jiang, T.: Clustering binary fingerprint vectors with missing values for DNA array data analysis. Journal of Computational Biology 11(5), 887–901 (2004)

9. Figueroa, A., Goldstein, A., Jiang, T., Kurowski, M., Lingas, A., Persson, M.: Approximate clustering of incomplete fingerprints. J. Discrete Algorithms 6(1), 103–108 (2008)
10. Mathieu, C., Sankur, O., Schudy, W.: Online Correlation Clustering. In: Proc. 7th International Symposium on Theoretical Aspects of Computer Science (STACS 2010), pp. 573–584 (2010)
11. Shamir, R., Sharan, R., Tsur, D.: Cluster graph modification problems. In: Kučera, L. (ed.) WG 2002. LNCS, vol. 2573, pp. 379–390. Springer, Heidelberg (2002)

Optimal Network Decontamination with Threshold Immunity

Paola Flocchini[1], Fabrizio Luccio[2], Linda Pagli[2], and Nicola Santoro[3]

[1] Ottawa University
flocchini@site.uottawa.ca
[2] University of Pisa
{luccio,pagli}@di.unipi.it
[3] Carleton University
santoro@scs.carleton.ca

Abstract. We consider the *monotone connected graph search* problem, that is, the problem of decontaminating a network infected by a mobile virus, using as few mobile cleaning agents as possible and avoiding recontamination. After a cleaning agent has left a vertex v, this vertex will become recontaminated if m or more of its neighbours are infected, where $m \geq 1$ is a threshold parameter of the system. In the literature, the value of m has been limited to $m = 1$ or to the strict majority of the neighbours of the nodes.

In this paper, we consider this problem for any integer value $m \geq 1$. We direct our investigation to widely used interconnection networks, namely meshes, hypercubes, and trees. For each of these classes of networks, we establish matching upper and lower bounds on the number of agents necessary to decontaminate with threshold m. The upper bound proofs are constructive: we present optimal decontamination protocols; we also show that these protocols are optimal or near-optimal in terms of number of moves.

1 Introduction

The Framework. Parallel and distributed computing systems are designed around interconnection networks. As the size, the complexity, and the importance of a system increases, the presence of malicious threats cannot be avoided. Such threats may be brought by *intruders* that travel through the network and infect any visited site, as for example a virus. The focus of this paper is on counteracting such a threat by a team of mobile *cleaning agents* (or simply *agents*) that traverse the network decontaminating the visited sites.

The team of agents enter the network, viewed as a simple undirected graph, at a single vertex, called *homebase*. An agent located at a vertex v can move to any of the neighbouring nodes of v, decontaminating it with its presence; upon departure of the (last) agent, a vertex can become re-contaminated if a sufficient number of its neighbours are contaminated. The goal is to decontaminate the whole network using as small a team of cleaning agents as possible avoiding any recontamination. This problem, also known as *monotone connected graph search* and as *intruder capture*, has been extensively studied in the literature (e.g., see [2, 3, 5–9, 12, 14–19, 21, 22]).

The re-contamination process is regulated by a parameter m: if no agent is there, a decontaminated vertex v will become re-contaminated if m or more of its neighbours are infected. The number of agents necessary to decontaminate the entire network is clearly a function of m and of the basic parameters of the given network. In the literature, the problem has been extensively studied when $m = 1$, that is the presence of a single infected neighbour recontaminates a disinfected node with no agents. Under this assumption the problem has been investigated for a variety of network classes: *Trees* [2, 3, 6], *Hypercubes* [7], *Meshes* [9], *Pyramids* [21], *Chordal Rings* [8], *Tori* [8], *outerplanar graphs* [12], *chordal graphs* [19], *Sierpinski graphs* [18], *Star graphs* [15], *product graphs* [14], graphs with large clique number [22], while the study of *arbitrary graphs* has been started in [5, 13]. The only studies of decontamination with $m > 1$ are for *Toroidal Meshes* and *Trees* with strong majority threshold [17], and for *Toroidal Meshes* with arbitrary threshold [16].

Main Contributions. We investigate the decontamination problem for arbitrary $m \geq 1$. We focus on three common classes of interconnection networks: *Meshes*, *Hypercubes*, and *Trees*; for each network G in those classes, we study the number $\mathcal{A}(G, m)$ of agents necessary to decontaminate G with threshold m.

For each of these classes of networks, we establish matching upper and lower bounds. The upper bound proofs are constructive: we present optimal decontamination protocols. More precisely:

(i) We first consider k-dimensional *meshes* with $N = n_1 \times n_2 \times \cdots \times n_k$ vertices, where $k \geq 2$; w.l.g., let $2 \leq n_1 \leq n_2 \leq \cdots \leq n_k$. We prove that for each such mesh Z, $\mathcal{A}(Z, m) = 1$ for $m \geq k$ and $\mathcal{A}(Z, m) = n_1 \times n_2 \times ... \times n_{k-m}$ for $1 \leq m < k$. We show that these lower bounds are tight, by exhibiting a solution algorithm that uses these many agents.
(ii) We then consider k-dimensional *hypercubes* with $N = 2^k$ vertices. We show that for each such hypercube H, $\mathcal{A}(H, m) \leq 2^{k-m}$; we prove that this bound is tight exactly for $m = 1$ and the trivial cases $m = k$ and $m = k - 1$, and asymptotically tight (i.e., for $k \to \infty$) for the other values of m.
(iii) Finally we consider the family of *trees*. Unlike the case of meshes and hypercubes, the number of needed cleaning agents may be different for different trees of the same size. For every tree T and any value of $m \geq 1$ we determine the value $\mathcal{A}(T, m)$. We then prove that this number is also sufficient by presenting a simple decontamination protocol using precisely those many agents.

The algorithms for meshes and hypercube are established in a synchronous setting; this is not a limitation since, as shown in [7–9], with simple modifications to the algorithm structure, the use of one single extra agent enables to solve the problem also in asynchronous settings. The proposed algorithms for trees work immediately also in an asynchronous settings.

Although not the main concern of this paper, we also consider the minimum number of moves performed by an optimal-size team of agents. We prove that all our solution protocols are optimal or near-optimal, in order of magnitude, also with respect to this measure.

Due to space limitations, in the following the proofs are omitted.

2 Model and Basic Properties

The network is represented as a simple connected undirected graph G with N vertices. For a vertex v in G, let $d(v)$ denote its degree, and let $d(G)$ denote the maximum value of $d(v)$ among all the vertices of G.

Operating in G is a team of *agents*. The agents have distinct Ids, have their own local memory, can communicate with each other when at the same node, can move from node to neighbouring node, and execute the same protocol. Distinct Ids and face-to-face communication allow the agents to coordinate their activities and to assign different roles and tasks as and when required by the protocol.

Initially, all vertices are *infected* and the agents enter the network at a single vertex, called *homebase*. The presence of one or more agents at a vertex decontaminates that vertex making it *clean*. We say that at a given time a vertex is *gray* if infected, *black* if it is clean and it contains one or more agents, and *white* if it is clean but no agent is there. A white vertex is re-contaminated (i.e., it becomes gray) if m or more of its neighbours are gray; by definition, a black vertex does not become grey regardless of the color of its neighbours. Notice that, by definition, a white vertex v with $d(v) < m$ can never be re-contaminated. When an agent moves from u to v on edge (u, v), it protects u from possible contamination by v. A system so defined is said to have an immunity threshold m to recontamination, or simply to have *m-immunity*.

The task of the team of agents is to decontaminate G avoiding any recontamination. The goal is (1) to determine the *smallest team size* $\mathcal{A}(G, m)$ for which such a task can be achieved; and (2) to devise an *optimal monotone strategy*, that is a solution protocol that allows such a minimal team to decontaminate G without recontamination. A secondary goal is for the minimal team to be able to perform decontamination using as few moves as possible.

In terms of the number of agents sufficient to decontaminate a system with m-immunity, the following bounds hold for any network.

Property 1. Let G be a simple connected graph.
(i) $\mathcal{A}(G, m) = 1$ if $m \geq d(G)$.
(ii) $\mathcal{A}(G, m) \leq 2$ if $m = d(G) - 1$.

In a (step-)*synchronous* system the agents operate in synchronized *steps*: in each step all the agents communicate (with those at the same vertex), compute, and move (when required); these operations might not be instantaneous, but all operations started in step i are completed by step $i + 1$. In a *asynchronous* system there is no common notion of time or step, and every operation by each agent takes a finite but otherwise unpredictable amount of time.

The protocols we present are described from a global point of view; their operations are however fully distributed. Since the agents have distinct IDs, are aware of the topology, and initially they are all located at the same node, each agent will initially be assigned a specific role; this role allows the agent to locally determine the sequence of movement it has to perform according to the protocol. In the case of meshes and hypercubes, the protocols are described assuming a synchronous system; in such a system, each agent can locally compute also the timing of each of its moves. If the mesh/hypercube is asynchronous, the timing would given by an additional distinct agent whose task is to synchronize the operations. Due to space constraints, these details are not explicit in the following descriptions.

3 Decontaminating Meshes

A *k-dimensional mesh* Z (*k-mesh* for short) is a network of $N = n_1 \times n_2 \times \cdots \times n_k$ vertices, with $k \geq 2$ and $2 \leq n_1 \leq n_2 \leq \cdots \leq n_k$. Each vertex $v_{i_1,i_2,\ldots,i_k}$, $0 \leq i_j \leq n_j - 1$, is connected to the $2k$ vertices $v_{i_1-1,i_2,\ldots,i_k}$, $v_{i_1+1,i_2,\ldots,i_k}$, $v_{i_1,i_2-1,\ldots,i_k}$, $v_{i_1,i_2+1,\ldots,i_k}$, ..., $v_{i_1,i_2,\ldots,i_k-1}$, $v_{i_1,i_2,\ldots,i_k+1}$, whenever these indexes stay inside the closed intervals $[0, n_j - 1]$ (i.e. the border vertex $v_{i_1=0,i_2,\ldots,i_k}$ are not connected to $v_{i_1-1,i_2,\ldots,i_k}$, etc.).

3.1 Upper Bounds for Meshes

We present a protocol for decontaminating a synchronous k-mesh with m-immunity and analyze its complexity. The decontamination process consists of two phases. Phase 1 amounts to identifying an initial set C of vertices and have $|C|$ agents move into these vertices. In Phase 2, these agents move from C to clean the whole mesh in a synchronized way. The size of C depends on the values of m and k. Taking $v_{0,0,\ldots,0}$ as the homebase, we define the *initial set* C of a k-mesh to consist of: *(i)* the only vertex $v_{0,0,\ldots,0}$, for $m \geq k$; *(ii)* all the vertices with indices $i_k, i_{k-1}, \ldots, i_{k-m+1}$ equal to zero, for $1 \leq m < k$. In the latter case, we have $|C| = n_1 \times n_2 \times \ldots \times n_{k-m}$.

Algorithm MESH-ONE, shown in Figure 1, gives a general scheme of phase 1 for $m < k$. A key point is that each agent reaches its final destination through a shortest path from $v_{0,0,\ldots,0}$. The **move** operation of the last statement is done synchronously for all the agents involved. We have:

```
Algorithm MESH-ONE($k$,$m$)
let $C$ be the submesh of $Z$ composed of all the vertices with indices
        $i_k, i_{k-1}, \ldots, i_{k-m+1}$ equal to zero;
forany $v \in C$ choose a shortest path from $v_{0,0,\ldots,0}$ to $v$;
let $P$ be the set of such paths for all the vertices of $C$;
start with a set $\mathcal{A}$ of $|C|$ agents in $v_{0,0,\ldots,0}$;
while $\exists\, v \in C$ containing more than one agent:
    let $(v, w_1)$, ..., $(v, w_r)$ be the edges from $v$ included in a path in $P$;
    let $p_i$ be the number of paths in $P$ containing $(v, w_i)$;
    keep one agent in $v$ (its final destination);
    for $i = 1$ to $r$ move $p_i$ agents to $w_i$ through edge $(v, w_i)$.
```

Fig. 1. Decontamination phase 1 for a k-mesh Z, with $m < k$

Lemma 1. *For a k-mesh with $1 \leq m < k$,* MESH-ONE *is a monotone algorithm that places $n_1 \times n_2 \times ... \times n_{k-m}$ synchronous agents in the vertices of C in less than $\frac{1}{2}(k-m)\, n_{k-m}^{k-m+1}$ moves.*

Note that, by Lemma 1, the number of moves to put the agents in C is $O(N)$.

In phase 2 of the decontamination process, the agents move from the vertices of C until the whole k-mesh has been cleaned. This is attained by algorithm MESH-TWO, shown in Figure 2.

Formally note that a k-mesh Z can be built by connecting n_k (k-1)-meshes $Z_0, ..., Z_{n_k-1}$, called (k-1)-*submeshes* of Z, where Z_j is composed of all the vertices with index $i_k = j$. Each submesh Z_j, $1 \leq j \leq n_k - 2$, is *adjacent* to Z_{j-1}, Z_{j+1}, meaning that each vertex of Z_j is adjacent to a vertex of Z_{j-1} and to one of Z_{j+1} for decreasing and increasing values of the index i_k, respectively. The border submeshes Z_0, Z_{n_k-1} are only adjacent to Z_1, Z_{n_k-2}, respectively. Recursively a k-mesh contains $n_k \times n_{k-1} \times \cdots \times n_{k-j}$ disjoint $(k-j-1)$-submeshes, for $0 \leq j \leq k-1$. For $j = k-2$ each 1-submesh reduces to a chain of n_1 vertices. For $j = k-1$ each 0-submesh reduces to a single vertex. These $(k-j-1)$-submeshes will be denoted by $Z_{i_k, i_{k-1}, \ldots, i_{k-j}}$ with obvious meaning, where each index i_r spans from 0 to $n_r - 1$. Note that 0-submeshes have k indices with values equal to the ones of the corresponding vertex. The whole mesh Z has no indices. By definition we have that C is a submesh of Z. In fact $C = Z_{0,0,\ldots,0}$ where the number of indices is k for $m \geq k$ and m for $m < k$.

```
**Algorithm** MESH-TWO($k, m, s, Z_{i_k, i_{k-1}, \ldots, i_{k-s+1}}$)
**let** $Z' = Z_{i_k, i_{k-1}, \ldots, i_{k-s+1}}$;
**if** $s = m$
    (*i.e.* $Z'$ *is composed of* $n_{k-m+1}$ $(k-m)$-*submeshes one of which*
      *contains the agents*)
    **move** the agents through the adjacent $(k-m)$-submeshes of $Z'$, until
      $Z'$ has been cleaned;
    **let** $i_r$ be the index of $Z'$ with lowest value of $r$ such that $i_r < n_r - 1$;
    **if** such an $i_r$ exists
        **move** the agents in one step **to** $Z_{i_k, \ldots, i_r+1, \ldots, i_{k-s+1}}$
    (***else*** *the whole mesh* $Z$ *has been cleaned*);
**else for** $t = 0$ **to** $n_{k-s} - 1$ MESH-TWO($k, m, s+1, Z_{i_k, i_{k-1}, \ldots, i_{k-s+1}, t}$).
```

Fig. 2. Decontamination phase 2 for a k-mesh Z, with $m \leq k$

Algorithm MESH-TWO, whose initial call is MESH-TWO$(k, m, 1, Z_0)$ with the agents in C, works recursively on submeshes $Z_{i_k, i_{k-1}, \ldots, i_{k-s+1}}$ of different size, where s is the number of indices in the notation. The external call is on Z_0. A first remark is that for $m > k$ a single agent can decontaminate the mesh moving as for the case $m = k$, then the algorithm is defined for $1 \leq m \leq k$ and C is a submesh with $s = m$ indices in all cases.

Lemma 2. MESH-TWO *is a monotone decontamination algorithm with* $|C|$ *synchronous agents starting in* C*. For* $m = k$ *the single agent makes* $N - 1$ *moves. For* $1 \leq m < k$ *the* $n_1 \times n_2 \times ... \times n_{k-m}$ *agents make* $N - n_1 \times n_2 \times ... \times n_{k-m}$ *moves.*

Noting that for $m > k$ the mesh can be decontaminated by one agent that makes the same moves needed for $m = k$, and combining Lemmas 1 and 2, we immediately have:

Theorem 1. *A* k*-mesh can be decontaminated monotonically in* $M(m) = \Theta(N)$ *moves by* $A(m)$ *synchronous agents, where:*
$A(m) = 1$*, for* $m \geq k$*;*
$A(m) = n_1 \times n_2 \times ... \times n_{k-m}$*, for* $1 \leq m < k$*.*

Let us finally recall that, as mentioned in Section 1, a decontamination scheme for a general value of m has been proposed in [16] for k-meshes with toroidal closures, an essentially different and somehow simpler problem due to the absence of borders in the structure. With due transformations in the notation and the conventions adopted, the results of that paper can be summarized as: $A(m) = 2^m \times n_1 \times n_2 \times ... \times n_{k-m}$ for $1 \leq m \leq k-1$, and $A(m) = 2^{2k-m}$ for $k \leq m \leq 2k$. In all cases $M(m) = \Theta(N)$.

3.2 Lower Bounds and Optimality for Meshes

By definition, at any step of a monotone solution algorithm, all the clean vertices form a connected subgraph B that includes the homebase. B is called a *guarded block* and constitutes the key concept for a lower bound argument.

Let $\nu(B)$ and $\alpha(B)$ respectively denote the number of all the vertices of B, and the number of the vertices of B with at least m gray neighbours. Each of the latter vertices must contain an agent. For any given guarded block B, then, $\alpha(B)$ gives the minimum number of agents needed to protect B from re-contamination. In a decontamination process B varies at each step in a sequence $S = B_0$, B_1,...,$B_{M(m)-1}$, where B_0 contains the homebase only. Therefore $\mathcal{A}(Z, m) \geq max_i(\alpha(B_i))$ (see Figure 3).

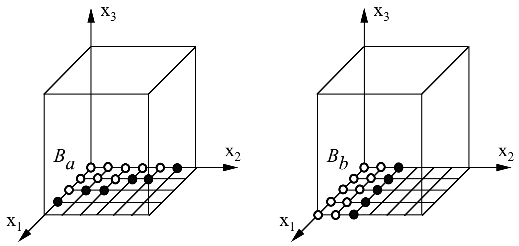


Fig. 3. Two guarded blocks B_a, B_b of 15 vertices for $k = 3$ and $m = 2$; $\alpha(B_a) = 6$ and $\alpha(B_b) = 5$, the latter being a minimum value for $\mathcal{A}(Z, m)$ in the plane $i_3 = 0$

Given that at least one agent is needed, and is sufficient to decontaminate a k-mesh Z with $m \geq k$, lower bounds on $\mathcal{A}(Z, m)$ will be studied for $1 \leq m < k$.

In the space of Z, we consider polytopes of order h (briefly h-topes), $0 \leq h \leq k$, with bounding hyperplanes "orthogonal" to the coordinate axes. In an h-tope vertices, edges, and faces of increasing order are called *f-faces*, with $f = 0, 1, 2, ..., h-1$ respectively. f-faces are in turn f-topes. Polytopes containing vertex $v_{0,0,...,0}$ are called *basic.* Note that in a basic f-face all the vertices have degree $k+f$, except the ones on the $(f'<f)$-subfaces that have degree $k+f'$.

Lemma 3. *Let the guarded blocks* $B_0, B_1, ...$ *grow initially in the subspace* $(x_1, x_2, ..., x_{k-m+1})$ *that is, for all the other indices* i_{k-m+2} *to* i_k *equal to zero. When a block* B_r *with* $\nu(B_r) \geq n_1 \times n_2 \times ... \times \lfloor n_{k-m+1}/2 \rfloor$ *is reached we have* $\alpha(B_r) \geq n_1 \times n_2 \times ... \times n_{k-m}$, *and there is a sequence* S *where* B_r *is a basic* $(k$-m+$1)$*-tope and the inequalities on* $\nu(B_r)$, $\alpha(B_r)$ *hold with the equal sign.*

This lemma immediately leads to the following lower bound theorem, that shows that the upper bound established in Theorem 1 is tight:

Theorem 2. *For a k-mesh* Z *with* $1 \leq m < k$ *we have* $\mathcal{A}(Z, m) \geq n_1 \times n_2 \times ... \times n_{k-m}$.

Once the number of agents has been minimized, a corresponding lower bound on the number of moves is immediately found. In fact the values given in Lemmas 1 and 2 have been computed by counting the number of moves of phases 1 and 2, that are both minimal for the chosen values of $\mathcal{A}(Z, m)$. We conclude that the detailed upper bounds on the number of moves given in the two lemmas are tight. In general we can state that $\mathcal{M}(m) = \Omega(N)$ thus matching the upper bound of Theorem 1 in order of magnitude.

4 Decontaminating Hypercubes

As well known, a *k-dimensional hypercube* H (or *k-cube* for short) is a network of $N = 2^k$ vertices coded with k-bit strings $s_0, ..., s_{N-1}$. Each vertex is denoted by its code. All vertices have degree k, in fact, the edges connect pairs of vertices (s_i, s_j) whose codes differ in exactly one bit. The restriction of H to vertices whose codes agree in exactly t bits is a $(k$-$t)$-cube, called a $(k$-$t)$-subcube of H.

4.1 Upper Bounds for Hypercubes

As for meshes, the decontamination process consists of phase 1 in which an initial set C of vertices is defined and $|C|$ agents are dispatched into these vertices; and phase 2 in which the agents move from C to clean the whole hypercube. The size of C depends on the values of m and k. Due to the total symmetry of the hypercube, any vertex can be chosen as the homebase. We take 00...0 as the homebase and let C be the $(k$-$m)$-subcube of H whose vertex codes start with m zeroes. So $|C| = 2^{k-m}$. Formally, letting $s_i[1, r]$ be the prefix of s_i from position 1 to position r, C is composed of all the vertices s_i with $s_i[1, m] = 00...0$.

Hypercube decontamination is carried out following the general scheme of algorithm CUBE, shown in Figure 4. At the end of phase 1, C contains 2^{k-m} agents, one per vertex, and is built moving the agents inside C only. Along the process, several agents are placed in the same vertex v at certain steps, one of which remains in v as its final destination while the others move away. In phase 2, the agents move in parallel waves cleaning all the (k-m)-subcubes of H. As we shall see, this phase requires some attention for visiting all the gray vertices exactly once.

Lemma 4. *For a k-cube with $m \leq k$, phase 1 of algorithm* CUBE *is monotone and places 2^{k-m} synchronous agents in the vertices of C in $(k-m) \cdot 2^{k-m-1}$ moves.*

In phase 2 the agents move from C to decontaminate all the (k-m)-subcubes of H. Due to the structure of the hypercube the vertices of any subcube H_1 are one to one adjacent to the vertices of other subcubes H_2 of the same dimensions. Assuming that each vertex of H_1 contains an agent, in one parallel step all these agents can be transferred onto H_2 for cleaning. The only problem is deciding the sequence of subcubes to be traversed in order to avoid sending agents to already decontaminated vertices. As indicated in algorithm CUBE - phase 2, a "Gray sequence" (e.g. a reflected binary sequence) of the prefixes $s[1, m]$ of the vertex codes solves the problem. We have:

Lemma 5. *Algorithm* CUBE - phase 2 *is a monotone decontamination algorithm using 2^{k-m} synchronous agents that start in C and make $N-2^{k-m}$ moves.*

Combining Lemmas 4 and 5 we immediately have:

Theorem 3. *A k-cube can be decontaminated monotonically with $A(m) = 2^{k-m}$ synchronous agents in $M(m) = N(1 + \frac{k-m-2}{2^{m+1}})$ moves.*

4.2 Lower Bounds and Optimality for Hypercubes

To some extent determining a lower bound on the number $\mathcal{A}(Z, m)$ of agents for decontaminating a k-cube is more complicated than for meshes. Here we study the problem for $1 \leq m \leq k-2$ as the values $A(k) = 1$ and $A(k\text{-}1) = 2$ obtained by the algorithm are obviously minimal.

Consider a particular guarded block B_r that contains 2^{k-m} white vertices. Note that one such a block must exist in any sequence $S = B_0, B_1, ...$ requiring a minimum number $L \leq 2^{k-m}$ of agents because the number of vertices in the blocks spans from 1 to N and the black vertices are limited to L. We have:

Theorem 4. *For a k-cube H with $1 \leq m \leq k-2$ we have $\mathcal{A}(m) \geq \frac{k-m+1}{k} 2^{k-m}$.*

As a consequence of Theorem 4, the upper bound 2^{k-m} for $\mathcal{A}(H, m)$ is strict for $m = 1$, and tends to that value for $m > 1$ if $k \to \infty$.

Theorem 5. *For a k-cube H with $1 \leq m \leq k-2$ we have $\mathcal{M}(H, m) \geq \Omega(N\, logN/log\, logN)$.*

In other words, the overall number of moves with algorithm CUBE in the worst case, i.e. for fixed m and increasing k, is within a $log\, logN$ factor from optimal.

```
**Algorithm** CUBE($k$,$m$)
**phase 1**
    **let** $C$ be the subcube of all the vertices $s$ with $s[1,m] = 00...0$;
    **start** with a set of $|C| = 2^{k-m}$ agents in vertex $00...0$;
    **if** $(m < k)$
        **for** $i = k$ **downto** $m + 1$
            **move** $2^{i-m-1}$ agents from each vertex $s$ with $s[1,i] = 00...00$
            **to** vertex $t$ with $t[1,i] = 00...01$.
**phase 2**
    **let** $s_0[1,m] = 00...0$ **and** $H_0 = C$;
    **choose** a Gray sequence $s_0, s_1, ...s_{2^m-1}$ of prefixes of length $m$;
    **let** $H_0, H_1, ..., H_{2^m-1}$ be the sequence of ($k$-$m$)-subcubes where all
        the vertices of $H_i$ have prefix $s_i$;
    **for** $(i = 1$ **to** $2^{m-1})$
        **move** all the agents from $H_{i-1}$ to the adjacent vertices of $H_i$.
```

Fig. 4. Decontamination of a k-cube H, with $1 \leq m \leq k$

5 Decontaminating Tree Networks

We now study the decontamination of tree networks. Observe that, in a tree T, the removal of an edge (u, v) in T uniquely identifies two rooted subtrees: the subtree $T[u \setminus v]$ rooted in u (and not containing v) and the $T[v \setminus u]$ rooted in v (and not containing u). In a tree T, a single agent performing a simple traversal starting from a leaf is sufficient to decontaminate T if $m \geq deg(T) - 1$. Hence in the following we will assume $m < deg(T) - 1$.

5.1 Lower Bounds for Trees

The minimum number of agents needed to decontaminate a tree T depends on the choice of the *homebase*, i.e., the node v of T from which the agents start. Let $\mathcal{A}(v, T, m)$ denote the minimum number of agents needed to decontaminate T when v is the homebase; then $\mathcal{A}(T, m) = \min_v\{\mathcal{A}(v, T, m)\}$ denotes the smallest number of agents needed to decontaminate T when the agents can choose the homebase.

Let $v_1, ..., v_k$ be the neighbours of v in T, where $k = deg(v)$. Without loss of generality, let $\mathcal{A}(v_i, T(v_i \setminus v), m) \geq \mathcal{A}(v_{i+1}, T(v_{i+1} \setminus v), m)$ for $1 \leq i < k$.

Theorem 6.
(i) If $k = 0$, *then* $\mathcal{A}(v, T, m) = 1$.
(ii) If $k > 0$, *then* $\mathcal{A}(v, T, m) \geq \mathcal{A}(v_1, T[v_1 \setminus v], m)$.
(iii) If $k > m$ *and* $\mathcal{A}(v_1, T[v_1 \setminus v], m = \mathcal{A}(v_{m+1}, T[v_{m+1} \setminus v], m)$, *then* $\mathcal{A}(v, T, m) \geq \mathcal{A}(v_1, T[v_1 \setminus v], m) + 1$.

Consider the function $\alpha(v, T, m)$ defined recursively as follows

$$\alpha(v,T,m) = \begin{cases} 1 & \text{if } k=0 \\ \alpha(v_1,T[v_1 \setminus v],m) & \text{if } 0<k\le m \\ \alpha(v_1,T[v_1 \setminus v],m) & \text{if } (k>m) \text{ and } (a_1 > a_{m+1}) \\ 1+\alpha(v_1,T[v_1 \setminus v],m) & \text{if } (k>m) \text{ and } (a_1 = a_{m+1}) \end{cases}$$

where $a_i = \alpha(v_i, T[v_i \setminus v], m)$.

For all pairs of neighbouring nodes u, v the values $\alpha(v,T,m), \alpha(u,T,m)$, $\alpha(v,T[v \setminus u],m)$ and $\alpha(u,T[u \setminus v)]m)$ can be computed by solving the recurrent relation; hence the value $\alpha(v,T,m)$ is uniquely determined for every v in T, and so is the value $\alpha(T,m) = \min_v\{\alpha(v,T,m)\}$. Note that, as in [3, 17], this computation can be performed efficiently using the "full saturation" technique [20]. The importance of the function α is that it is a lower bound on $\mathcal{A}$.

Theorem 7. $\mathcal{A}(v,T,m) \geq \alpha(v,T,m)$

We next consider a lower bound on the number of moves performed by a minimal team of agents. Let $\mathcal{M}(T,m)$ denote the minimum number of *moves* necessary to decontaminate T starting from v with $\mathcal{A}(T,m)$ agents and all agents returning to the homebase. By definition, $\mathcal{M}(T,m) = \min\{\mu(v,T,m) : \mathcal{A}(v,T,m) = \mathcal{A}(T,m)\}$ where $\mu(v,T,m)$ denote the minimum number of moves necessary to decontaminate T starting from v with $\mathcal{A}(v,T,m)$ agents, all returning to v. Again, let $v_1, ..., v_k$ be the neighbours of v in T, where $k = d(v)$.

Theorem 8.

$$\mu(v,T,m) = \begin{cases} \sum_{1\le j\le k}[\mu(v_j,T[v_j \setminus v],m) + 2\ \mathcal{A}(v_j,T[v_j \setminus v],m)] & \textit{if } k>0 \\ 0 & \textit{if } k=0 \end{cases}$$

5.2 Upper Bounds and Optimality for Trees

We first prove that the lower bound of Theorem 7 is tight. The proof is constructive. Consider protocol*Decontaminate* shown in Figure 5.

```
Algorithm DECONTAMINATE($T, v, m$)
move $\alpha(T, v)$ agents to $v$;
let $v_1, ..., v_k$ be the neighbours of $v$ in $T$, and wlg let
    $\alpha(v_i, T(v_i \setminus v), m) \geq \alpha(v_{i+1}, T(v_{i+1} \setminus v), m)$, $1 \leq i < k$.
for $j = k$ downto $j = 1$
    DECONTAMINATE($T(v_j \setminus v), v_j, m$).
    move all the agents at $v_j$ to $v$.
```

Fig. 5. Decontamination of a tree T starting from v, with $1 \leq m \leq k-1$

Theorem 9. *Algorithm* DECONTAMINATE(T, v, m) *monotonically decontaminates* T *starting from* v *using* $\alpha(v,T,m)$ *agents.*

From Theorems 7 and 9, it follows $\mathcal{A}(v,T,m) = \alpha(v,T,m)$.

```
Algorithm OptimalTreeDecontamination($T, m$)
choose as starting point a vertex $v$ such that
    $\alpha(v,T,m) = \mathcal{A}(T,m)$ and $\mu(v,T,m) = \mathcal{M}(T,m)$,
Decontaminate($T, v, m$).
```

Fig. 6. Optimal decontamination of a tree T, with $1 \leq m \leq k-1$

The protocol OptimalTreeDecontamination that uses precisely $\mathcal{A}(T,m)$ agents performing $M(T,m)$ moves is now straightforward (see Figure 6).

Theorem 10. *Protocol* OptimalTreeDecontamination *decontaminates monotonically any arbitrary tree T using $\mathcal{A}(T,m)$ agents performing $\mathcal{M}(T,m)$ moves.*

5.3 Other Bounds

The optimal protocol described in the previous section decontaminates any tree T with the minimal number of agents and an optimal number of moves. It assumes that the tree T is known. It is still possible to devise a simple and rather efficient protocol (shown in Figure 7) that decontaminates T requiring only knowledge of its diameter $h(T)$ and using precisely those many agents.

```
Algorithm UnknownTree
Place all $h$ agents at $v$ which becomes the root of $T$.
Starting from $v$ perform a depth-first traversal of $T$ with the following rules:
    (1) when going from a node $x$ to a new node $y$ bring all agents currently
        at $x$ but one (i.e., leave one agent at $x$);
    (2) when returning from a node $y$ bring all agents currently at $y$.
```

Fig. 7. Decontamination of a tree T of unknown topology

Theorem 11. *Protocol* UnknownTree *monotonically decontaminates a tree T of unknown topology using at most $h(T)$ agents.*

The of number of agents $h(T)$ of this simple protocol can be far off from $\mathcal{A}(T,m)$. However, for an infinite family of trees, its cost is no more than twice the cost of the optimal solution protocol that uses knowledge of T. Let $\mathcal{T}(k,h)$ denote the family of complete k-ary trees of diameter h; that is, the leaves have degree 1, all other nodes have degree k, and there is a node whose distance from all leaves is $h/2$.

Theorem 12. *For any $T \in \mathcal{T}(k,h)$, $v \in T$, $h = 2r \geq 4$, $k \geq m+2 \geq 3$ $\alpha(v,T,m) \geq r+1$.*

Theorem 12 extends the results of [2], established for $k = 3$ and $m = 1$. It states that any protocol decontaminating all trees of $\mathcal{T}(k,h)$ must use at least $h/2+1$ agents, implying the worst case optimality, within a constant factor of 2, of Protocol UnknownTree for that infinite class of trees.

References

1. Altshuler, Y., Dolev, S., Elovici, Y., Aharony, N.: TTLed random walks for collaborative monitoring. In: Proc. 2nd Int. Work. Network Sci. Comm. Net. (2010)
2. Barrière, L., Flocchini, P., Fomin, F.V., Fraignaud, P., Nisse, N., Santoro, N., Thilikos, D.M.: Connected graph searching. Inf. and Comp. 219, 1–16 (2012)
3. Barrière, L., Flocchini, P., Fraignaud, P., Santoro, N.: Capture of an intruder by mobile agents.Proceedings of the 14th Symposium on Parallel Algorithms and Architectures (SPAA), pp. 200–209 (2002)
4. Bienstock, D., Seymour, P.: Monotonicity in graph searching. Journal of Algorithms 12, 239–245 (1991)
5. Blin, L., Fraignaud, P., Nisse, N., Vial, S.: Distributed chasing of network intruders. Theoretical Computer Science 399(1-2), 12–37 (2008)
6. Dereniowski, D.: Connected searching of weighted trees. Theoretical Computer Science 412(41), 5700–5713 (2011)
7. Flocchini, P., Huang, M.J., Luccio, F.L.: Decontamination of hypercubes by mobile agents. Networks 52(3), 167–178 (2008)
8. Flocchini, P., Huang, M.J., Luccio, F.L.: Decontaminating chordal rings and tori using mobile agents. Int. J. on Found. of Comp. Sci. 18(3), 547–563 (2006)
9. Flocchini, P., Luccio, F.L., Song, L.X.: Size optimal strategies for capturing an intruder in mesh networks. In: Proceedings of the International Conference on Communications in Computing, pp. 200–206 (2005)
10. Flocchini, P., Mans, B., Santoro, N.: Tree decontamination with temporary immunity. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 330–341. Springer, Heidelberg (2008)
11. Fomin, F.V., Thilikos, D.M.: An annotated bibliography on guaranteed graph searching. Theoretical Computer Science 399(3), 236–245 (2008)
12. Fomin, F.V., Thilikos, D.M., Todineau, I.: Connected graph searching in outerplanar graphs. In: Proc. 7th Int. Conference on Graph Theory, (ICGT) (2005)
13. Ilcinkas, D., Nisse, N., Soguet, D.: The cost of monotonicity in distributed graph searching. Distributed Computing 22(2), 117–127 (2009)
14. Imani, N., Sarbazi-Azadb, H., Zomaya, A.Y.: Capturing an intruder in product networks. Journal of Parallel and Distributed Computing 67(9), 1018–1028 (2007)
15. Imani, N., Sarbazi-Azad, H., Zomaya, A.Y., Moinzadeh, P.: Detecting threats in star graphs. IEEE Trans. on Parallel and Distr. Systems 20(4), 474–483 (2009)
16. Luccio, F., Pagli, L.: A general approach to toroidal mesh decontamination with local immunity. In: Proc. 23rd Int. Par. and Distr. Processing Symp. (IPDPS) (2009)
17. Luccio, F., Pagli, L., Santoro, N.: Network decontamination in presence of local immunity. Int. J. of Foundation of Computer Science 18(3), 457–474 (2007)
18. Luccio, F.L.: Contiguous search problem in Sierpinski graphs. Theory of Computing Systems 44(2), 186–204 (2009)
19. Nisse, N.: Connected graph searching in chordal graphs. Discrete Applied Mathematics 157(12), 2603–2610 (2009)
20. Santoro, N.: Design and Analysis of Distributed Algorithms. John Wiley (2007)
21. Shareghi, P., Imani, N., Sarbazi-Azad, H.: Capturing an intruder in the pyramid. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) CSR 2006. LNCS, vol. 3967, pp. 580–590. Springer, Heidelberg (2006)
22. Yanga, B., Dyerb, D., Alspach, B.: Sweeping graphs with large clique number. Discrete Mathematics 309(18), 5770–5780 (2009)

Finding All Convex Cuts of a Plane Graph in Cubic Time

Roland Glantz and Henning Meyerhenke

Institute of Theoretical Informatics
Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
`{roland.glantz,meyerhenke}@kit.edu`

Abstract. In this paper we address the task of finding convex cuts of a graph. In addition to the theoretical value of drawing a connection between geometric and combinatorial objects, cuts with this or related properties can be beneficial in various applications, e. g., routing in road networks and mesh partitioning. It is known that the decision problem whether a general graph is k-convex is $\mathcal{NP}$-complete for fixed $k \geq 2$. However, we show that for plane graphs all convex cuts (i. e., $k = 2$) can be computed in polynomial time. To this end we first restrict our consideration to a subset of plane graphs for which the so-called alternating cuts can be embedded as plane curves such that the plane curves form an arrangement of pseudolines. For a graph G in this set we formulate a one-to-one correspondence between the plane curves and the convex cuts of a bipartite graph from which G can be recovered. Due to their local nature, alternating cuts cannot guide the search for convex cuts in more general graphs. Therefore we modify the concept of alternating cuts using the Djoković relation, which is of global nature and gives rise to cuts of bipartite graphs. We first present an algorithm that computes all convex cuts of a (not necessarily plane) bipartite graph $H' = (V, E)$ in $\mathcal{O}(|E|^3)$ time. Then we establish a connection between convex cuts of a graph H and the Djoković relation on a (bipartite) subdivision H' of H. Finally, we use this connection to compute all convex cuts of a plane graph in cubic time.

Keywords: Plane graphs, convex cuts, Djoković relation, partial cubes, bipartite graphs.

1 Introduction

A *convex cut* of a graph $G = (V, E)$ is a partition of V into V_1 and V_2 such that both subgraphs of G induced by V_1 and V_2 are convex. A convex subgraph of G, in turn, is a subgraph S of G such that for any pair of vertices v, w in S all shortest paths from v to w in G are fully contained in S. Following Artigas et al. [1], a convex k-partition in a graph is a partition of the vertex set into k convex sets. If G has a convex k-partition, then G is said to be *k-convex*. Deciding whether a (general) graph is k-convex is $\mathcal{NP}$-complete for fixed $k \geq 2$ [1]. Moreover, given a bipartite graph $G = (V, E)$ and an integer k, it is $\mathcal{NP}$-complete to decide whether the largest convex set $\emptyset \neq S \subset V$ has at least k vertices [8].

The fundamental notion of convexity in graphs can be used to draw a connection to continuous objects in a metric space. Note that there exists a different notion of convexity for plane graphs. A plane graph is called convex if all of its faces are convex polygons. This second notion is different and *not* object of our investigation. The notion of convexity in acyclic directed graphs, motivated by embedded processor technology, is also different [2]. There, a subgraph S is called convex if there is no *directed* path between any pair v, w in S that leaves S. In addition to being directed, these paths do not have to be *shortest* paths as in our case.

Graph partitions with particular properties are of high interest in many applications [3]. Sample applications that potentially benefit from convexity of a cut are parallel numerical simulations. For some linear solvers used in these simulations, the *shape* of the partitions, in particular short boundaries, small aspect ratios, but also connectedness and smooth boundaries, plays a significant role [12]. Convex subgraphs typically admit these properties. Another example is the preprocessing of road networks for shortest path queries by partitioning according to natural cuts [6]. The definition of a natural cut is not as strict as that of a convex cut, but they have a related motivation.

Both from a theoretical point of view and due to the practical importance, it is natural to ask whether the time complexity of finding convex cuts is polynomial for certain types of inputs. In this paper, we focus on plane graphs as they are important in a large number of applications. Their special structure gives rise to the hope that finding their convex cuts is fundamentally easier. To the best of our knowledge, there exists no polynomial-time algorithm yet for computing convex cuts of plane graphs.

1.1 Related Work

Artigas et al. [1] show that every connected chordal graph $G = (V, E)$ is k-convex, for $1 \leq k \leq |V|$. They also establish conditions on $|V|$ and p to decide if the pth power of a cycle is k-convex. Moreover, they present a linear-time algorithm that decides if a cograph is k-convex.

Our method for finding all convex cuts of a plane graph G is motivated by the work in Chepoi et al. [5] on the links between alternating and convex cuts of plane graphs. Plane graphs usually have alternating cuts that are not convex and convex cuts that are not alternating. Proposition 2 in [5] characterizes the plane graphs for which the alternating cuts coincide with the convex cuts in terms of a condition on the boundary of *any* alternating cut. In this paper we represent the alternating cuts as plane curves that we call embedded alternating paths (EAPs) – any EAP partitions G exactly like the alternating cut it represents. In contrast to [5], however, we focus on the intersections of the EAPs (i. e., alternating cuts).

If any pair of EAPs intersects at most once, we have a slight generalization of so-called *arrangements of pseudolines*. The latter arise in discrete geometry, computational geometry and in the theory of matroids [4]. Duals of arrangements of pseudolines are known to be partial cubes – a fact that has been applied to graphs in [9], for example. For basics on partial cubes we rely on the survey [13].

The following basic fact about partial cubes is crucial for our method for finding convex cuts: partial cubes are precisely the graphs that are bipartite and on which the Djoković relation [7] is transitive.

1.2 Paper Outline and Contribution

In Section 3 we first represent the alternating cuts of a plane graph $G = (V, E)$, as defined in [5], by EAPs. The main work here is on the proper embedding. We then study the case of G being *well-arranged*, as we call it, *i. e.,* the case in which the EAPs form an arrangement of pseudolines, or a slight generalization thereof. We show that the dual $G_{\mathcal{E}}$ of such an arrangement is a partial cube and reveal a one-to-one correspondence between the EAPs of G and the convex cuts of $G_{\mathcal{E}}$. Specifically, the edges of $G_{\mathcal{E}}$ intersected by any EAP form the cut-set of a convex cut of $G_{\mathcal{E}}$. Conversely, the cut-set of any convex cut of $G_{\mathcal{E}}$ is the set of edges intersected by a unique EAP of G.

The one-to-one correspondence between the EAPs of a well-arranged graph G and the convex cuts of bipartite $G_{\mathcal{E}}$ suggests that, for the case in which G is more general, it might be helpful to employ a bipartite graph that can be turned into G by taking a subgraph and contracting vertices of degree two. We choose the simplest way to generate such a graph G', *i. e.,* we subdivide any edge of G. Before we find all convex cuts of plane G using G', we show in Section 4 that all convex cuts of a bipartite graph $G' = (V', E')$ can be found in $\mathcal{O}(|E'|^3)$ time, even if G' is not plane. The fact that we can compute all convex cuts in bipartite graphs in polynomial time is no contradiction to the $\mathcal{NP}$-completeness of the decision problem whether the largest convex *set* in a bipartite graph has a certain size. In a convex cut both subgraphs have to be convex, while a convex set S makes no assumptions on $V \setminus S$. The key to finding all convex cuts of a plane graph is a theorem that holds for non-plane graphs, too. In particular, we characterize convex cuts of G in terms of two relations, one on G and one on G'. The latter is the Djoković relation.

The results of Section 4 are then used in Section 5 to find all convex cuts of a *plane* graph $G = (V, E)$. As in the case of well-arranged graphs, we iteratively proceed from an edge e to another edge opposite to a face bounded by e, and the number of such cuts is bounded by $2|E|$. This time, however, "opposite" is with respect to the Djoković relation on G'. Thus we arrive at a set of potentially convex cuts, which includes all convex cuts. It remains to check whether the potentially convex cuts are actually convex. This can be done in $\mathcal{O}(|V|^2)$ time per potentially convex cut, totalling up to $\mathcal{O}(|V|^3)$ for all such cuts.

2 Preliminaries

Our methods for finding all convex cuts apply to finite, undirected, unweighted, and connected graphs that are free of self-loops and whose vertices all have degree greater than one. The last two conditions are not essential: self-loops have no impact on the convex cuts, and any edge on a dead-end path toward a degree-one vertex gives rise to a trivial convex cut that does not interfere with the non-trivial convex cuts. We often associate a cut (V_1, V_2) with its *cut-set*, i. e., the set of edges running between V_1 and V_2.

If G is plane, we may identify V with a set of points in $\mathbb{R}^2$ and E with a set of plane curves that intersect only at their end points, which, in turn, make up V. For $e \in E$ with end points u, v ($u \neq v$) we sometimes write $e = \{u, v\}$ even when e is not necessarily determined by u and v due to parallel edges. F_∞ denotes the unbounded face of G and, if F is a face of G, we write $E(F)$ for the set of edges that bound F.

We denote the standard metric on G by $d_G(\cdot, \cdot)$. In this metric the distance between $u, v \in V$ amounts to the number of edges on a shortest path from u to v. A subgraph $S = (V_S, E_S)$ of a (not necessarily plane) graph H is an *isometric* subgraph of H if $d_S(u, v) = d_H(u, v)$ for all $u, v \in V_S$.

Following [7] and using the notation in [13], we set

$$W_{xy} = \{w \in V : d_G(w, x) < d_G(w, y)\} \quad \forall \{x, y\} \in E. \tag{1}$$

Let $e = \{x, y\}$ and $f = \{u, v\}$ be two edges of G. The Djoković relation θ on G's edges is defined as follows:

$$e \ \theta \ f \Leftrightarrow f \text{ has one end vertex in } W_{xy} \text{ and one in } W_{yx}. \tag{2}$$

The Djoković relation is reflexive, symmetric [15], but not necessarily transitive. The vertex set V of G is partitioned by W_{ab} and W_{ba} if and only if G is bipartite.

A *partial cube* $G_q = (V_q, E_q)$ is an isometric subgraph of a hypercube. For a survey on partial cubes see [13]. Partial cubes and θ are related in that a graph is a partial cube if and only if it is bipartite and θ is transitive [13]. Thus, θ is an equivalence relation on E_q, and the equivalence classes are cut-sets of G_q. Moreover, the cuts defined by these cut-sets are precisely the convex cuts of G_q [13]. If (V_q^1, V_q^2) is a convex cut, the (convex) subgraphs induced by V_q^1 and V_q^2 are called semicubes. If θ gives rise to k equivalence classes $E_q^1, \dots E_q^k$, and thus k pairs (S_a^i, S_b^i) of semicubes, where the ordering of the semicubes in the pair is arbitrary, one can derive a so-called Hamming labeling $b : V_q \mapsto \{0, 1\}^k$ by setting

$$b(v)_i = \begin{cases} 0 & \text{if } v \in S_a^i \\ 1 & \text{if } v \in S_b^i \end{cases} \tag{3}$$

In particular, $d_{G_q}(u, v)$ amounts to the Hamming distance between $b(u)$ and $b(v)$ for all $u, v \in V_q$. This is a consequence of the fact that the corners of a hypercube have such a labeling and that G_q is an isometric subgraph of a hypercube.

A *half-cube* H is a graph whose vertices correspond to the vertices in one part of a bipartite representation of a partial cube Q. Two vertices of H are adjacent in H if their distance in Q is two [5].

3 Partial Cubes from Embedding Alternating Paths

In Section 3.1 we define a multiset of (not yet embedded) alternating paths that are analogous to the alternating cuts defined in [5]. Section 3.2 is devoted to embedding the alternating paths into $\mathbb{R}^2$. In Section 3.3 we study the duals of these embeddings and show that they are partial cubes.

3.1 Alternating Paths

Intuitively, an EAP P runs through a face F of G such that the edges through which P enters and leaves F are opposite—or nearly opposite because, if $|E(F)|$ is odd, there is no opposite edge, and P has to make a slight turn to the left or to the right. The exact definitions leading up to (not yet embedded) alternating paths are as follows.

Definition 3.1 (Opposite edges, left, right, unique opposite edge). *Let $F \neq F_\infty$ be a face of G, and let $e, f \in E(F)$. Then e and f are called* opposite edges *of F if the lengths of the two paths induced by $E(F) \setminus \{e, f\}$ differ by at most one. If the two paths have different lengths, f is called the* left [right] opposite edge *of e if starting on e and running clockwise around F, the shorter [longer] path comes first. Otherwise, e and f are called* unique opposite edges.

Definition 3.2 (Alternating path graph $A(G) = (V_A, E_A)$). *The* alternating path graph $A(G) = (V_A, E_A)$ *of $G = (V, E)$ is the (non-plane) graph with $V_A = E$ and E_A consisting of all two-element subsets $\{e, f\}$ of E such that e and f are opposite edges of some face $F \neq F_\infty$.*

The alternating path graph defined above will provide the edges for the multiset of alternating paths defined next. We resort to a *multiset* for the sake of uniformity, i.e., to ensure that any edge of G is contained in exactly two alternating paths (see Figure 1a).

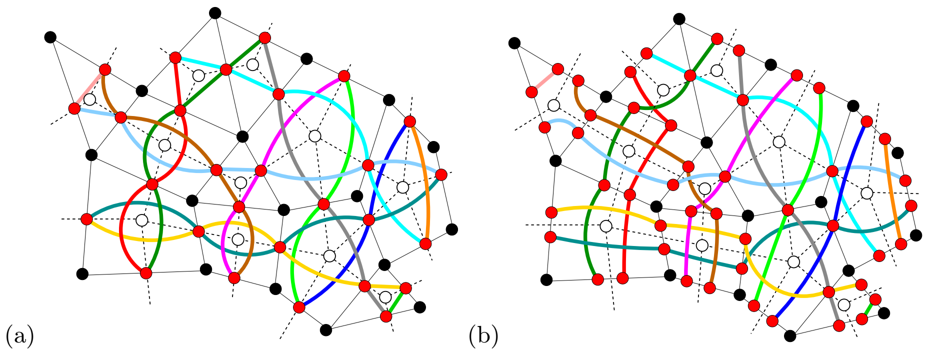


Fig. 1. Primal graph: Black vertices, thin solid edges. Dual graph: White vertices, dashed edges. (a) Multiset $\mathcal{P}(G)$ of alternating paths: Red vertices, thick solid lines. The paths in $\mathcal{P}(G)$ are colored. In this ad-hoc drawing all alternating paths that contain a vertex v_A (edge e of G) go through the same point on e, *i. e.,* where a red vertex was placed. (b) Collection $\mathcal{E}(G)$ of EAPs: Red vertices, thick solid colored lines.

Definition 3.3 ((Multiset $\mathcal{P}(G)$ of) Alternating paths in $A(G)$). *A maximal path $P = (v_A^1, e_A^1, v_A^2, \dots e_A^{n-1}, v_A^n)$ in $A(G) = (V_A, E_A)$ is called* alternating *if*

- *v_A^i and v_A^{i+1} are opposite for all $1 \leq i \leq n-1$ and*
- *if v_A^{i+1} is the left [right] opposite of v_A^i, and if j is the minimal index greater than i such that v_A^j and v_A^{j+1} are not unique opposites (and j exists at all), then v_A^{j+1} is the right [left] opposite of v_A^j.*

The multiset $\mathcal{P}(G)$ contains all alternating paths in $A(G)$: the multiplicity of P in $\mathcal{P}(G)$ is two if v_A^{i+1} is a unique opposite of v_A^i for all $1 \leq i \leq n-1$, and one otherwise.

3.2 Embedding of Alternating Paths

In this section we show that the alternating paths in $\mathcal{P}(G)$ can be embedded into $\mathbb{R}^2$ such that any edge $\{e, f\}$ of any path in $\mathcal{P}(G)$ turns into a non-self-intersecting plane curve with one end point on e and the other end point on f. Moreover, for any face F of G any pair of embedded paths intersects at most once in the filled polygon $\overline{F} = F \cup E(F)$. A path in $\mathcal{P}(G)$ with multiplicity $m \in \{1, 2\}$ gives rise to m embedded paths. Visually, we go from Figure 1a to Figure 1b.

In the following we set rules for embedding alternating paths locally, *i.e.*, into $\overline{F}$. Later on we will make sure that the locally embedded paths fit together at the face boundaries. We formulate the rules only for faces whose boundaries are *regular* polygons. We may do so because all filled polygons in $\mathbb{R}^2$ (with the same number of sides) are homeomorphic—a consequence of the Jordan-Schönflies theorem [14]. Thus, we may embed the alternating paths first into a regular polygon $\overline{F_r}$ and then map the embedded paths from $\overline{F_r}$ into $\overline{F}$ using a homeomorphism from $\overline{F_r}$ onto $\overline{F}$. The homeomorphism will not affect the *intersection pattern* of the local embedding, *i.e.*, information on which paths intersect in F or hit an edge in $E(F)$. The intersection pattern in $\overline{F_r}$ will depend not only on $\overline{F_r}$, but also on the cyclical order in which the EAPs from outside $\overline{F_r}$ hit $E(F)$. For examples see Figures 2b,c and Figure 3a.

Embedding Rules. We call a face F of G even [odd] if $|E(F)|$ is even [odd]. The rules for embedding alternating paths locally are:

1. The part of an EAP that runs through F_r is a straight line, and EAPs cannot coincide in F_r.
2. An EAP can hit $e \in E(F_r)$ only at e' s relative interior, *i.e.*, not at e's end vertices.
3. If two EAPs share a point p on an edge $e \in E(F_r)$, they must cross at p and not just touch.
4. Let $F_r \neq F_\infty$ be an even face of G, let e, f be unique opposite edges in $E(F_r)$, and let P_1, P_2 be the two non-embedded alternating paths in $\mathcal{P}(G)$ that contain the edge $\{e, f\}$ ($P_1 = P_2$ if and only if the multiplicity of P_1

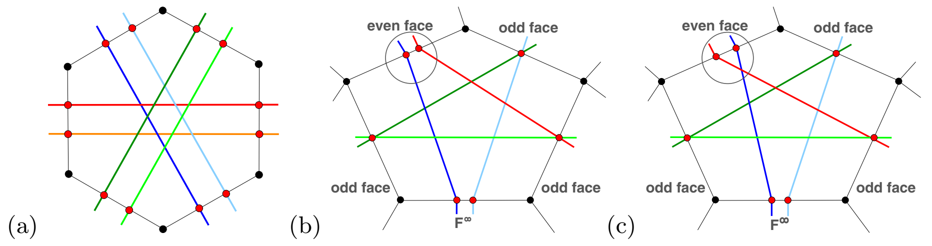


Fig. 2. (a) Intersection pattern of a filled hexagonal face (b,c) Two intersection patterns of a filled pentagonal face (see gray circle for difference)

is two). Then the parts of embedded P_1 and P_2 that run through $\overline{F_r}$ must form a pair of distinct parallel line segments (see Figure 2a).

5. Let $F_r \neq F_\infty$ be an odd face of G, let $e \in E(F_r)$, and let $P_1, P_2 \in \mathcal{P}(G)$ be the two paths that contain the vertex e. If e also bounds an even bounded face, embedded P_1, P_2 must hit e at two distinct points. If the other face is a bounded odd face, embedded P_1, P_2 must hit at a point on e. (see Figures 2b,c).
6. Let e be an edge of G that separates F_∞ from a bounded face, and let $P_1, P_2 \in \mathcal{P}(G)$ be the two paths that contain the vertex e. Then, embedded P_1, P_2 must hit e at two distinct points (see Figures 2b,c).

We now map the embedded paths from any $\overline{F_r}$ into $\overline{F}$ using a homeomorphism from $\overline{F_r}$ onto $\overline{F}$. The following is about tying the loose ends of the locally embedded paths, which all sit on edges of G, so as to arrive at a *global* embedding of the alternating paths (see Figures 3b,c). Let $e \in E(G)$, and let F, F' be the faces of G that are bounded by e. We have two locally embedded paths P_F^1, P_F^2 in $\overline{F}$ and two locally embedded paths $P_{F'}^1, P_{F'}^2$ in $\overline{F'}$ that all hit e. If F and F' are bounded odd faces, we bend the four paths such that they all hit the same point q on the interior of G (for details see below). Otherwise, let the end point of P_F^1, P_F^2, $P_{F'}^1$, and $P_{F'}^2$ be denoted by p_F^1, p_F^2, $p_{F'}^1$, and $p_{F'}^2$, respectively. Due to having used homeomorphisms to obtain the embedded paths in $\overline{F}$ and $\overline{F'}$, it holds that $p_F^1 \neq p_F^2$ and $p_{F'}^1 \neq p_{F'}^2$. Let $e = \{u, v\}$, and let q_1, q_2 be points on the interior of e such that q_1 is closer to u than q_2. Without loss of generality we may assume that p_F^1 is closer to u than p_F^2 and that $p_{F'}^1$ is closer to u than $p_{F'}^2$. We now bend p_F^1 and $p_{F'}^1$ [p_F^2 and $p_{F'}^2$] toward q_1 [q_2].

It remains to show that the bending operations above can be done such that the intersection patterns do not change in the interiors of F and F'. Recall that the paths are homeomorphic to straight line segments. Thus, there exists $\epsilon > 0$ such that all locally embedded paths in F and F' other than P_F^1, P_F^2, $P_{F'}^1$, and $P_{F'}^2$ keep a distance greater than ϵ to e. The bending, in turn, can be done such that it affects P_F^1, P_F^2, $P_{F'}^1$, and $P_{F'}^2$ only in an ϵ-neighborhood of e.

Notation 3.1. The collection of globally embedded alternating paths (EAPs) is denoted by $\mathcal{E}(G)$ (see again Figures 3b,c).

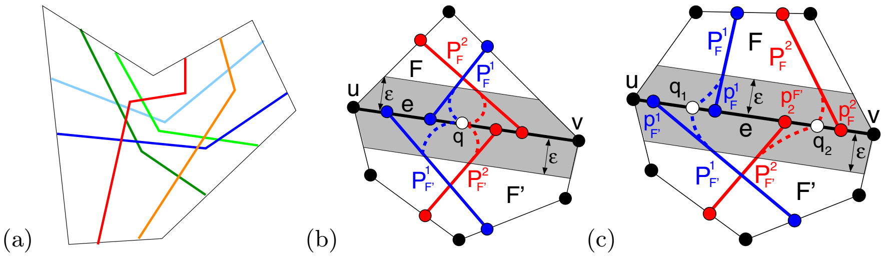


Fig. 3. (a) Hexagonal face with the same intersection pattern as in Figure 2a. (b,c) Bending of alternating paths is indicated by the dashed colored lines. For the notation see the text. (b) Bending toward a single point q on e. (c) Bending toward two points q_1, q_2 on e.

The EAPs in Figure 1b are special in that they form an arrangement in the following sense.

Definition 3.4 (Arrangement of alternating paths, well-arranged graph).
$\mathcal{E}(G)$ is called an arrangement of (embedded) alternating paths *if*

1. *none of the EAPs is a cycle,*
2. *none of the EAPs intersects itself, and*
3. *there exist no paths $P_1 \neq P_2 \in \mathcal{E}(G)$ such that $P_1 \cap P_2$ contains more than one point.*

We call a plane graph G well-arranged *if $\mathcal{E}(G)$ is an arrangement of alternating paths.*

The notion of an arrangement of alternating paths can be seen as a generalization of the notion of an arrangement of pseudolines [4]. The latter arrangements are known to have duals that are partial cubes [9].

3.3 Partial Cubes from Well-Arranged Graphs

The purpose of the following is to prepare for the definition of a graph $G_\mathcal{E}$ (see Definition 3.6), which will turn out to be a partial cube if G is well-arranged. In this case we are able to determine all convex cuts of $G_\mathcal{E}$.

Definition 3.5 (Domain $D(G)$ of G, facet of $\mathcal{E}(G)$, adjacent facets).
The domain $D(G)$ of G is the set of points covered by the vertices, edges and bounded faces of G. A facet of $\mathcal{E}(G)$ is a (bounded) connected component (in $\mathbb{R}^2$) of $D(G) \setminus (\bigcup_{e \in E(G)} e \cup \bigcup_{v \in V(G)} v)$. Two facets of $\mathcal{E}(G)$ are adjacent if their boundaries share more than one point.

In the following DEAP stands for Dual of Embedded Alternating Paths.

Definition 3.6 (DEAP graph $G_{\mathcal{E}}$ of G).
A DEAP graph $G_{\mathcal{E}}$ of G is a plane graph that we obtain from G by placing one vertex into each facet of $\mathcal{E}(G)$ and drawing edges between a pair (u, v) of these vertices if the facets containing u and v are adjacent in the sense of Definition 3.5. A vertex of $G_{\mathcal{E}}$ can also sit on the boundary of a face as long as it does not sit on an EAP from $\mathcal{E}(G)$ (for an example see the black vertex on the upper left in Figure 4a).

Since we are not interested in the exact location [course] of $G_{\mathcal{E}}$'s vertices [edges], there exists basically one DEAP graph of G. Due to the intersection pattern of the EAPs in G's bounded faces, as specified in Section 3.2 and illustrated in Figure 2, there are the following three kinds of vertices in $V(G_{\mathcal{E}})$:

Definition 3.7 (Primal, intermediate and star vertex of $G_{\mathcal{E}}$).

- ***Primal Vertices:*** *Vertices which represent a facet that contains a (unique) vertex v of G in its interior or on its boundary. As we do not care about the exact location of $G_{\mathcal{E}}$'s vertices, we may assume that the primal vertices of $G_{\mathcal{E}}$ are precisely the vertices of G.*
- ***Intermediate Vertices:*** *The neighbors of the primal vertices in $G_{\mathcal{E}}$.*
- ***Star Vertices:*** *The remaining vertices in $G_{\mathcal{E}}$.*

For an example of a DEAP graph see Figure 4, where the black, gray and white vertices correspond to the primal, intermediate, and star vertices, respectively.

Theorem 3.1. *The DEAP graph $G_{\mathcal{E}}$ of a well-arranged plane graph G is a partial cube.*

Proof. We denote the Hamming distance by $h(\cdot, \cdot)$. To show that $G_{\mathcal{E}} = (V_{\mathcal{E}}, E_{\mathcal{E}})$ is a partial cube, it suffices to specify a labeling $l : V_{\mathcal{E}} \mapsto \{0, 1\}^n$ for some $n \in \mathbb{N}$ such that $d_{G_{\mathcal{E}}}(u, v) = h(l(u), l(v))$ for all $u, v \in V_{\mathcal{E}}$ (see Section 2).

We set the length n of any binary vector $l(v)$ to the number of paths in $\mathcal{E}(G)$, and let the entries of $l(v)$ indicate v's position with respect to the paths in $\mathcal{E}(G)$. Specifically, we start by numbering the paths in $\mathcal{E}(G)$ from one to n, which yields the paths $P_1, \ldots, P_n$. For each $1 \leq i \leq n$ we then select one component of $D(G) \setminus P_i$. Then we set the ith entry of $l(v)$ to one if the face represented by v is in the selected component of $D(G) \setminus P_i$ (zero otherwise).

It remains to show that $d_{G_{\mathcal{E}}}(u, v) = h(l(u), l(v))$ for any pair $u \neq v \in V$. Since on any path of length k from u to v in $G_{\mathcal{E}}$ it holds that $h(l(u), l(v)) \leq k$, we have $d_{G_{\mathcal{E}}}(u, v) \geq h(l(u), l(v))$.

To see that $d_{G_{\mathcal{E}}}(u, v) = h(l(u), l(v))$, it suffices to show that u has a neighbor u' such that $h(l(u'), l(v)) < h(l(u), l(v))$ (because then there also exists u'' such that $h(l(u''), l(v)) < h(l(u'), l(v))$ and so on until v is reached in exactly $h(l(u), l(v))$ steps). The claim follows from the case distinction in the appendix of [10]. □

The Hamming labelling of $G_{\mathcal{E}}$ defined in the proof of Theorem 3.1 guarantees the following.

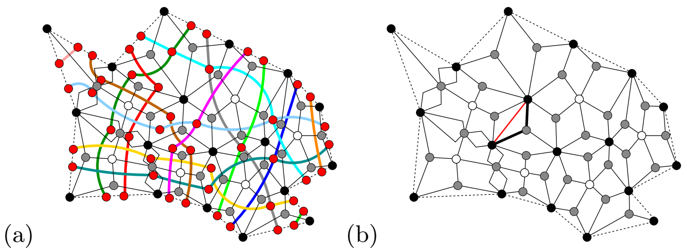


Fig. 4. DEAP graph $G_{\mathcal{E}}$ of the primal graph G shown in Figure 1a. (a) Collection $\mathcal{E}(G)$ of EAPs: Red vertices, thick solid colored lines. DEAP graph $G_{\mathcal{E}}$: Black, gray and white vertices, thin black solid lines. The black, gray and white vertices are the primal, intermediate and star vertices, respectively. The dashed polygonal line delimits $D(G)$. (b) $G_{\mathcal{E}}$ only. The red edge, however, is an edge of G. The path formed by the two bold black edges is an example of a path in $G_{\mathcal{E}}$ of length two that connects two primal vertices that are adjacent in G via an intermediate vertex in $G_{\mathcal{E}}$.

Corollary 3.1. *If G is well-arranged, there exists a one-to-one correspondence between the EAPs of G and the convex cuts of $G_{\mathcal{E}}$. Specifically, the edges intersected by any EAP of G form a cut-set of a convex cut of $G_{\mathcal{E}}$. Conversely, the cut-set of any convex cut of $G_{\mathcal{E}}$ is the set of edges intersected by a unique EAP of G.*

4 Convex Cuts of Not Necessarily Plane Graphs

In this section we assume that $H' = (V, E)$ is bipartite (but not necessarily plane). As mentioned in Section 2, any edge $e = \{a, b\}$ of H' gives rise to a cut of H' into W_{ab} and W_{ba}. The cut-set of this cut is $C_e = \{f \in E \mid e\ \theta\ f\} = \{f = \{u, v\} \in E \mid d_{H'}(a, u) = d_{H'}(b, v)\}$. In the following we characterize the cut-sets of the *convex* cuts of H'. This characterization is key to finding all convex cuts of a bipartite graph in $\mathcal{O}(|E|^3)$ time.

Lemma 4.1. *Let $H' = (V, E)$ be a bipartite graph, and let $e \in E$. Then C_e is the cut-set of a convex cut of H' if and only if $f\ \theta\ \hat{f}$ for all $f, \hat{f} \in C_e$.*

Proof. Let C_e be a non-convex cut of H', and let $e = \{a, b\}$. Then there exists a shortest path $P = \{v_1, \ldots v_n\}$ with both end vertices in, say W_{ab}, such that P has a vertex in W_{ba}. Let i be the smallest index such that $v_i \in W_{ba}$, and let j be the smallest index greater than i such that $v_j \in W_{ab}$. Then $f = \{v_{i-1}, v_i\}$ and $\hat{f} = \{v_j, v_{j+1}\}$ are two edges in C_e. Lemma 3.5 in [13] says that no pair of edges from a shortest path are related by θ, *i. e.,* $f\ \theta\ \hat{f}$ does not hold.

Conversely, let $f = \{u, v\}, \hat{f} = \{\hat{u}, \hat{v}\} \in C_e$ such that $f\ \theta\ \hat{f}$ does not hold. Without loss of generality assume $\hat{u} \in W_{uv}$, $\hat{v} \in W_{vu}$ and $d_{H'}(u, \hat{u}) < d_{H'}(v, \hat{v})$. Then $d_{H'}(v, \hat{v}) - d_{H'}(u, \hat{u}) \geq 2$. Indeed, $d_{H'}(v, \hat{v}) - d_{H'}(u, \hat{u})$ must be even because otherwise one of the two distances $d_{H'}(u, \hat{u})$ and $d_{H'}(v, \hat{v})$ is even and the other one is odd. Hence, due to H' being bipartite and the fact that (adjacent) u and v are in different parts of H', the vertices $\hat{u}$ and $\hat{v}$ are in the same part of H', a contradiction to $\hat{u}$ and $\hat{v}$ being connected by $\hat{f} \in C_e$. From $d_{H'}(v, \hat{v}) - d_{H'}(u, \hat{u}) \geq 2$ it follows that one can go from v to $\hat{v}$, which are both in W_{vu}, by first traversing f, then going from u to $\hat{u}$ within W_{uv}, and finally traversing $\hat{f}$. The length of this path is $\leq d_{H'}(v, \hat{v})$. Thus, C_e is not a cut-set of a convex cut of H'. □

Lemma 4.1 suggests to determine the convex cuts of H' as sketched in Algorithm 1.

Theorem 4.1. *Algorithm 1 computes all cut-sets of convex cuts of a bipartite graph H' using $\mathcal{O}(|E|^3)$ time and $\mathcal{O}(|E|)$ space.*

Proof. The correctness follows from Lemma 4.1 due to the symmetry of the Djoković relation. For each candidate cut-set, one needs to determine if all pairs of contained edges are Djoković related.

Regarding the running time, observe that for any edge e, the (not necessarily convex) cut-set C_e can be determined by using BFS to compute the distances of any vertex to the end vertices of e. Hence each C_e can be determined in time $\mathcal{O}(|E|)$. The inner for-loop has $\mathcal{O}(|E|))$ iterations. Each iteration has time complexity $\mathcal{O}(|E|))$ when using a linear-time set equality algorithm [11] (which requires linear space). Hence Algorithm 1 runs in $\mathcal{O}(|E|^3)$ time. Moreover, each cut-set is processed sequentially. Since no more than two cut-sets (with $\mathcal{O}(|E|)$ edges each) have to be stored at the same time, the space complexity of Algorithm 1 is $\mathcal{O}(|E|)$. □

A simple loop-parallelization over the edges in line 3 would lead to a parallel running time of $\mathcal{O}(|E|^2)$ with $\mathcal{O}(|E|)$ processors. If one is willing to spend more processors and a quadratic amount of memory, then even faster parallelizations are possible. Since they use standard PRAM results, we forego their description.

For the remainder of this section H denotes a graph (self-loops and degree-one vertices allowed). We subdivide each edge of H into two edges and thus get a bipartite graph H'. An edge e in H that is subdivided into edges e_1, e_2 of H' is called *parent* of e_1, e_2, and e_1, e_2 are called *children* of e. The Djoković relation on H' is denoted by θ'.

The next lemma characterizes the convex cuts of H in terms of the Djoković relation on H'. The lemma does, however, not imply that the convex cuts of H can be derived from those of H' in polynomial time.

Definition 4.1 (Relation τ). *Let $e = \{u_e, v_e\}$ and $f = \{u_f, v_f\}$ be edges of H. Then, $e \ \tau \ f$ if $d_H(u_e, u_f) = d_H(v_e, v_f) = d_H(u_e, v_f) = d_H(v_e, u_f)$.*

Lemma 4.2. *A cut of H with cut-set C is convex if and only if for all $e, f \in C$ either $e \ \tau \ f$ or (exclusive or) there exists a child e' of e and a child f' of f such that $e' \ \theta' \ f'$.*

Proof. Let $\{a', b'\}$ be child of $e = \{u_e, v_e\}$, let $\{c', d'\}$ be a child of $f = \{u_f, v_f\}$ and assume without loss of generality that $d_H(u_e, u_f) \leq d_{H'}(u_e, v_f)$. Then

Algorithm 1. Find all cut-sets of convex cuts of a bipartite graph H'

```
1: procedure EvaluateCutSets(bipartite graph $H'$)
   ▷ Computes the cut-set $C_{e^i}$ for each edge $e^i$ and stores in $isConvex[i]$ if $C_{e^i}$ is the cut-set of a convex cut
2:     Let $e^1, \dots e^m$ denote the edges of $H'$; initialize all $m$ entries of the array $isConvex$ as $true$
3:     for $i = 1, \dots, m$ do
4:         Determine $C_{e^i} = \{f^1, \dots, f^k\}$
5:         for all $f^j$ do
6:             Determine $C_{f^j}$
7:             if $C_{f^j} \neq C_{e^i}$ then
8:                 $isConvex[i] := false$
9:                 break
10:            end if
11:        end for
12:    end for
13: end procedure
```

$d_{H'}(a', c') = d_{H'}(b', d')$ if and only if either $e\ \tau\ f$ or (exclusive or) there exists a child e' of e and a child f' of f with $e'\ \theta'\ f'$. Indeed, let $d_{H'}(a', c') = d_{H'}(b', d')$. If $d_{H'}(a', c')$ is odd, i. e., the case shown in Figure 5, then $d_{H'}(u_e, u_f) = d_{H'}(v_e, v_f)$, and adding only one of the conditions $d_H(u_e, v_f) = d_H(u_e, u_f)$ and $d_H(u_f, v_e) = d_H(u_e, u_f)$ yields that only one child of e is θ'-related to only child of f. Adding the last condition results in $e\ \tau\ f$ and no θ'-relations of children. If $d_{H'}(a', c')$ is even, we cannot have $e\ \tau\ f$ and only one child of e is θ'-related to only child of f. Conversely, if $e\ \tau\ f$, then $d_{H'}(a', c')$ is odd, i. e., the case shown in Figure 5, and $d_{H'}(a', c') = d_{H'}(b', d')$ for at least one pair of children $\{a', b'\}$ and $\{c', d'\}$. If $e'\ \theta\ f'$, the end vertices of a', b' of e' and c', d' of f' fulfill $d_{H'}(a', c') = d_{H'}(b', d')$ by definition of θ'.

Let C be the cut-set of a *convex* cut that partitions V into V_1 and V_2, and let $e, f \in C$ (see Figure 5a). Thanks to the above it suffices to find a child $\{a', b'\}$ of e and a child $\{c', d'\}$ of f such that $d_{H'}(a', c') = d_{H'}(b', d')$. Let $e = \{u_e, v_e\}$ and $f = \{u_f, v_f\}$, and let w'_e $[w'_f]$ denote the vertex of H' that subdivides e $[f]$. Without loss of generality we assume $u_e, u_f \in V_1$ and $v_e, v_f \in V_2$. Since C is the cut-set of a convex cut, we know that $d_H(u_e, u_f)$ and $d_H(v_e, v_f)$ differ by at most one. If $d_H(v_e, v_f) = d_H(u_e, u_f)$, let $\{a', b'\} = \{u_e, w'_e\}$ and $\{c', d'\} = \{w'_f, v_f\}$ (this is the case illustrated in Figure 5a). Then, due to the degrees of w'_e and w'_f being two, $d_{H'}(a', c') = d_{H'}(u_e, w'_f) = d_{H'}(w'_e, v_f) = d_{H'}(b', d')$. If $d_H(u_e, u_f)$ and $d_H(v_e, v_f)$ differ by exactly one, we may assume without loss of generality that $d_H(v_e, v_f) = d_H(u_e, u_f) + 1$. Set $\{a', b'\} = \{w'_e, v_e\}$ and $\{c', d'\} = \{w'_f, v_f\}$. Then, due to the degrees of w'_e and w'_f being two, $d_{H'}(a', c') = d_{H'}(w'_e, w'_f) = d_{H'}(v_e, v_f) = d_{H'}(b', d')$.

Conversely, let C be the cut-set of a cut that partitions V into V_1 and V_2. We now assume that for all $e = \{u_e, v_e\}, f = \{u_f, v_f\} \in C$ that are not τ-related there exists a child e' of e and a child f' of f such that $e'\ \theta'\ f$. As above we

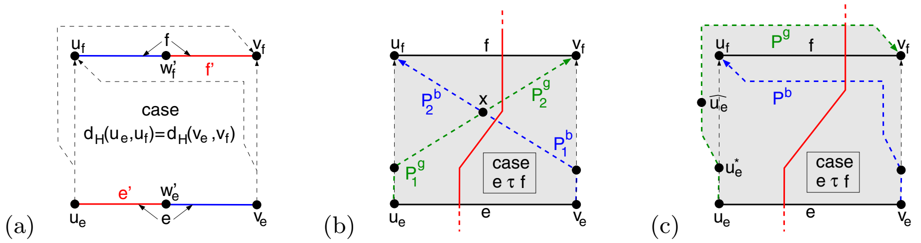


Fig. 5. (a) If there exists a shortest path from u_e to v_f [v_e to u_f] with length $d_H(u_e, u_f)$ (indicated by dashed lines), then the red [blue] children of e and f are not θ'-related, anymore. If both paths exist, no child of e is related to a child of f (and vice versa), and $e \; \tau \; f$. (b,c) Illustration to proof of Lemma 5.1. The red polygonal line indicates a convex cut of H, and the shaded region is R. In (b) we have $R = R'$.

assume without loss of generality that $u_e, u_f \in V_1$ and $v_e, v_f \in V_2$. There are four possibilities for the positions of e' and f' within e and f, only two of which need to be considered due to symmetry.

1. $e' = \{u_e, w'_e\}$ and $f' = \{u_f, w'_f\}$. In this case $d_{H'}(u_e, u_f) = d_{H'}(w'_e, w'_f) = d_{H'}(v_e, v_f) \pm 2$.
2. $e' = \{u_e, w'_e\}$ and $f' = \{w'_f, v_f\}$. Since the degrees of w'_e and w'_f are two, and since $e' \; \theta' \; f'$, any shortest path from u_e to w'_f runs via u_f, and any shortest path from w'_e to v_f runs via v_e. Hence, $d_{H'}(u_e, u_f) = d_{H'}(u_e, w'_f) - 1 = d_{H'}(w'_e, v_f) - 1 = d_{H'}(v_e, v_f)$.

Thus $d_H(u_e, u_f) = d_H(v_e, v_f) \pm 1$ for all $e = \{u_e, v_e\}, f = \{u_f, v_f\} \in C$. Hence, any shortest path with end vertices in V_1 [V_2] stays within V_1 [V_2], *i. e.,* C is the cut-set of a convex cut. □

5 Convex Cuts of Plane Graphs

Let $G = (V, E)$ denote a plane graph without self-loops and degree-one vertices. Analogous to Section 4, $G' = (V', E')$ denotes the (plane bipartite) graph that one obtains from G by placing a new vertex into the interior of each edge of G. Note that G and G' have the same faces. The method for finding all convex cuts of G in polynomial time that we present in this section is motivated by the following observations.

1. The cut-sets of a non-bipartite graph H can be characterized in terms of τ on H and θ' on H' (see Lemma 4.2).
2. Let C be the cut-set of a *plane* graph. Then one can brachiate through the edges of C by forming a cyclic sequence $(e_0, \ldots e_{|C|-1} = e_0)$ such that for any pair (e_i, e_{i+1}) of consecutive edges there exists a face F with $\{e_i, e_{i+1}\} \in E(F)$ (indices are modulo $|C|$). Such a cyclic sequence, in turn, is equivalent

to a cyclic sequence $(F_0, \ldots, F_{|C|-1})$, where the (unique) F_i are such that $\{e_i, e_{i+1}\} \in E(F_i)$. The cut-set C is non-cyclical if and only if $F_i = F_\infty$ for some i.

The idea for the following is to gather cut-sets of potentially convex cuts of G by brachiating through the edges of G while checking τ-relations on G and θ'-relations on G'. One of the problems is that the number of potentially convex cuts increases whenever one encounters an edge that is τ-related to one of the edges that have been gathered already. The following lemma, which is based on the scenarios depicted in Figures 5b,c, helps to solve the problem.

Lemma 5.1. *Let H be a plane graph, let $e = \{u_e, v_e\}$, $f = \{u_f, v_f\}$ be edges of H with $e\ \tau\ f$, and let $R \subset \mathbb{R}^2$ be the minimal region containing e, f, and the shortest paths from u_e to u_f, from v_e to v_f, from u_e to v_f and from u_f to v_e. If H_R denotes the subgraph of H whose vertices and edges are contained in R, then any convex cut of H_R with e and f in its cut-set can be extended to a convex cut of H in at most one way. In particular, a vertex $w \notin R$ belongs to the same part as v_f if and only if it is closer to v_f than to v_e.*

Proof. Without loss of generality we may assume that the convex cut of H_R, denoted by (V_R^u, V_R^v), is such that $u_e, u_f \in V_R^u$ and $v_e, v_f \in V_R^v$. We assume that (V^u, V^v) is a convex cut of H that extends (V_R^u, V_R^v), i. e., $V_R^u \subset V^u$ and $V_R^v \subset V^v$.

Let w be a vertex of H that is not in R. It suffices to show that (V_R^u, V_R^v) determines whether $w \in V^u$ or $w \in V^v$. Indeed, let R' denote the smallest region containing e, f, as well as the shortest paths from u_e to u_f and from v_e to v_f (see Figure 5b). As in Figure 5c, let P^g $[P^b]$ denote a shortest path from u_e to v_f $[v_e$ to $u_f]$. If P^g and P^b intersect, at a vertex denoted by x, the part before and after x is denoted by P_1^g and P_2^g $[P_1^b$ and $P_2^b]$ (see Figure 5b). Finally, the length of a path P is denoted by $|P|$. Up to symmetry it suffices to distinguish between the following cases.

1. P^g and P^b are fully contained in R' (see Figure 5b). In particular, P^g and P^b intersect at x. Assuming $|P_1^g| < |P_1^b|$ yields that $|P_2^g| > |P_2^b|$ (since $e\ \tau\ f$ and thus $|P_1^g| + |P_2^g| = |P_1^b| + |P_2^b|$). Hence, the concatenation of P_1^b and P_2^g is a path from u_e to u_f that is shorter than the one from v_e to v_f, a contradiction to $e\ \tau\ f$. Hence, $|P_1^g| = |P_1^b|$ and thus $|P_2^g| = |P_2^b|$. From $|P_1^g| + |P_2^b| = |P_1^g| + |P_2^g| = d_H(u_f, u_e)$ and $|P_1^b| + |P_2^g| = |P_1^b| + |P_2^b| = d_H(v_f, v_e)$ it follows that the concatenation of P_1^b and P_2^g, as well as the concatenation of P_1^g and P_2^b are shortest paths. One of the concatenations must be cut twice by (V_R^u, V_R^c), a contradiction to (V_R^u, V_R^c) being a convex cut of H_R. Hence, this case does not occur.
2. P^b is fully contained in R', but P^g is not (see Figure 5c). Let u_e^* be the last vertex on a shortest path from u_e to u_f that is also contained in P^g. Furthermore, let w be a vertex of H that is not contained in R, and let P be a shortest paths between w and u_f. We distinguish between the following cases.

(i) P enters R through a vertex u' of P^g that is located between u_e and u_e^*. Hence, there exists a shortest path P' from w to u_f that goes via u_e^*. Let P'' be the path that follows P' up to u_e^* and then follows P^g until it reaches v_f. If P'' is a shortest path, then $w \in V^u$. Otherwise there exists a shortest path from w to u_f via v_f, i. e., $w \in V^v$. In particular, w belongs to the same part as v_f if and only if w is closer to v_f than to v_e.

(ii) P enters R through a vertex $\widehat{u_e}$ on P^g that is located between u_e^* and the last vertex on P^g before the cut. Let P'' be the path that follows P up to $\widehat{u_e}$ and then follows P^g until it reaches v_f. If P'' is a shortest path, then $w \in V^u$. Otherwise there exists a shortest path from w to u_f via v_f, i. e., $w \in V^v$. In particular, w belongs to the same part as v_f if and only if w is closer to v_f than to v_e.

(iii) P enters R through any other vertex. Then the vertex of entry is in V^v, and since $v_e \in V^u$, it must hold that $w \in V^v$ and w is closer to v_f than to v_e.

3. Both P^b and P^g are fully contained in R'. Same as above, but using only cases (i) and (ii). □

Theorem 5.1. *All convex cuts of* $G = (V, E)$ *can be found using* $\mathcal{O}(|V|^3)$ *time and* $\mathcal{O}(|V|)$ *space.*

Proof. Below we assemble cut-sets of potentially convex cuts of G by starting at an edge of G. This method finds at least all convex cuts of G such that $e \; \tau \; f$ does not hold for all e, f in the cut-set. We will deal with the remaining cuts later.

1. Let $F_0 \neq F_\infty$ be a face of G' and let $e' = \{u'_e, v'_e\}$, $f' = \{u'_f, v'_f\}$, $g' = \{u'_g, v'_g\}$ be distinct edges in $E'(F_0)$. Then one cannot have $e' \theta' f'$ and $e' \; \theta' \; g'$ at the same time (see Figure 6a). Indeed, assuming the opposite, we may assume without loss of generality that $u'_f, u'_g \in W_{u'_e v'_e}$, $v'_f, v'_g \in W_{v'_e u'_e}$. Moreover, there exists a shortest path from u'_e to u'_g that is contained in $W_{u'_e v'_e}$ and a shortest path from v'_e to v'_f that is contained in $W_{v'_e u'_e}$. From G' being plane it follows, however, that the two paths intersect (in Figure 6a the two paths intersect at the vertex w), a contradiction to the fact that $W_{u'_e v'_e}$ and $W_{v'_e u'_e}$ are disjoint.
2. Let $F_0 \neq F_\infty$ be a face of G' and let $e_0 \in E(F_0)$ (see Figure 6b). Furthermore, let $e'_{0,l} = \{v_{0,l}, w'_0\}$ and $e'_{0,r} = \{w'_0, v_{0,r}\}$ be the two children of e_0. Here $v_{0,l}$ [$v_{0,r}$] is the left [right] vertex of e_0 when looking onto e_0 from the (unique) face $\neq F_0$ that bounds e_0, and w'_0 is the vertex in G' that subdivides e_0. As in Section 4, $C'_{e'} = \{f' \in E' \mid e' \; \theta' \; f'\}$ is a cut-set of G' for all $e' \in E'$. Due to item 1, $C'_{e'_{0,l}}$ [$C'_{e'_{0,r}}$] contains two edges of $E'(F')$, *i. e.*, $e'_{0,l}$ [$e'_{0,r}$] and an edge denoted by $e'_{1,l}$ [$e'_{1,r}$]. We denote the parent of $e'_{1,l}$ [$e'_{1,r}$] by $e_{1,l}$ [$e_{1,r}$].
3. If the two parents $e_{1,l}$ and $e_{1,r}$ are identical, Lemma 4.2 in conjunction with item 1 above yields that, if e_0 belongs to the cut-set of a convex cut, then $e_{1,l}$ must belong to the same cut-set.

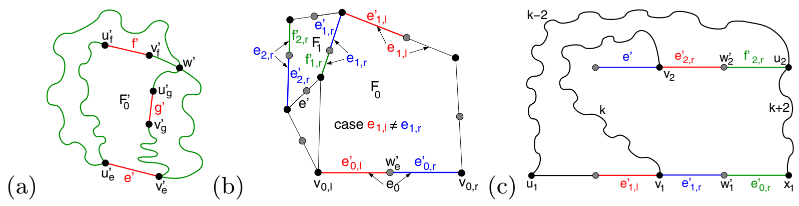


Fig. 6. Illustrations to proof of Theorem 5.1. (a) Face F'_0 of G' with distinct $e' = \{u'_e, v'_e\}$, $f' = \{u'_f, v'_f\}$, $g' = \{u'_g, v'_g\} \in E'(F'_0)$. The green curves indicate shortest paths. One cannot have $e'\theta' f'$ and $e'\theta' g'$ at the same time. (b) Only edges with a common color can form a cut-set of a convex cut. Cut-sets $\{e_0, e_{1,l}\}$ and $\{e_0, e_{1,r}, e_{2,r}\}$. (c) Illustration to item 4 in the proof of Theorem 5.1. The black curves indicate shortest paths and the labels on these curves indicate path lengths.

4. Assume that parents $e_{1,l}$ and $e_{1,r}$ are different. We now show that there can exist at most one convex cut whose cut-set contains e_0 and $e_{1,r}$. We also show how one can find this convex cut. An analogous result then also holds for the branch containing e_0 and $e_{1,r}$. Let F_1 denote the unique face with $e_{1,r} \in E(F_1)$ and $F_1 \neq F_0$ (see Figure 6b). Furthermore, let $f'_{1,r}$ denote the sibling of $e'_{1,r}$, i. e., the unique other edge whose parent is $e_{1,r}$. To simplify the following, we color the cut-sets $C'_{e'_{0,l}}$, $C'_{e'_{0,r}}$ and $C'_{f'_{1,r}}$ as in Figure 6b, i. e., red, blue and green, respectively. If we assume that there exists a convex cut of G with a cut-set containing e_0 and $e_{1,r}$, then Lemma 4.2 in conjunction with item 1 above yields that the cut-set contains exactly one edge in $E(F_1)$ other than $e_{1,r}$ and that this edge, denoted by $e_{2,r}$, either has a blue child or that the children's colors are red and green. In the first case all edges in the cut-set considered so far have a blue child. This is the case illustrated in Figure 6b. The second case does not occur. Indeed, consider the path P formed by the edges $e'_{1,l}$ (red), $e'_{1,r}$ (blue), and $f'_{1,r}$ (green) in this order. The second case implies that there exists a path P^* that consists of the three edges e' (blue), $e'_{2,r}$ (red), and $f'_{2,r}$ (green) in this order. With the notation in Figure 6c let k be the length of a shortest path between v_1 and v_2. Then, $f'_{1,r}\ \theta'\ f'_{2,r}$ implies that there exists a shortest path between x_1 and u_2 with length $k + 2$ (recall that the gray vertices have degree two). Furthermore, $e'_{1,r}\ \theta'\ e'_{2,r}$ implies that that there exists a shortest path between u_1 and u_2 with length $k - 2$. Hence, there exists a path from w'_1 to w'_2 via v_1, u_1 and u_2 that has length $k + 1$, a contradiction to $f'_{1,r}\ \theta'\ f'_{2,r}$.
5. An iterative application of the previous item yields that e_0 can be in the cut-set of at most two convex cuts, and that these potentially convex cuts can be determined in time $\mathcal{O}(|E|) = \mathcal{O}(|V|)$ and space $\mathcal{O}(|E|) = \mathcal{O}(|V|)$ once the cut-sets $C'_{e'_{0,l}}$, $C'_{e'_{0,r}}$ and $C'_{e'_{0,r}}$ have been determined which, in turn, can be done in time $\mathcal{O}(|V|)$ and space $\mathcal{O}(|V|)$. As in Theorem 4.1 it now follows that all convex cuts with τ-free cut-sets can be found in time $\mathcal{O}(|V|^3)$ and space $\mathcal{O}(|V|)$.

We now deal with convex cuts whose cut-sets contain τ-related edges. When we assemble cut-sets of potentially convex cuts, as above, and when we encounter an edge f with $e \ \tau \ f$, we extend the potential cut in the only possible way according to Lemma 5.1, i. e., a $w \notin R$ belongs to the same part as v_f if and only if it is closer to v_f than to v_e. This can be done in $\mathcal{O}(|V|)$ time by BFS starting at v_e and BFS starting at v_f. In particular, total time $\mathcal{O}(|V|)$ is not increased. □

6 Conclusions

We have presented an algorithm for computing all convex cuts of a plane graph in cubic time. To the best of our knowledge, it is the first polynomial-time algorithm for this task. On the way to this result, we first represented alternating cuts as plane curves (EAPs) and focussed on a subset of ℓ_1-graphs for which the EAPs basically form an arrangement of pseudolines. Thus we came across a one-to-one correspondence between the EAPs of a graph G and the convex cuts of a bipartite graph $\widetilde{G_{\mathcal{E}}}$, one half of which is G. A similar correspondence on general graphs, in conjunction with an algorithm that computes all convex cuts of a bipartite graph in cubic time, formed the basis for our algorithm to compute all convex cuts of a general plane graph in cubic time, too. Consequently, while the problem is $\mathcal{NP}$-hard for general graphs, we have shown that it becomes polynomial-time solvable for plane graphs. In future work we would like to investigate if the convexity test for potentially convex cuts can be accelerated asymptotically. It seems also worthwhile to apply the techniques based on the Djoković relation to other graph classes.

Acknowledgements. We thank Peter Sanders and Christian Schulz for helpful discussions on the topic.

References

1. Artigas, D., Dantas, S., Dourado, M., Szwarcfiter, J.: Partitioning a graph into convex sets. Discrete Mathematics 311(17), 1968–1977 (2011)
2. Balister, P., Gerke, S., Gutin, G., Johnstone, A., Reddington, J., Scott, E., Soleimanfallah, A., Yeo, A.: Algorithms for generating convex sets in acyclic digraphs. Journal of Discrete Algorithms 7(4), 509–518 (2009)
3. Bichot, C., Siarry, P.: Graph Partitioning. Wiley (2011)
4. Björner, A., Las Vergnas, M., Sturmfels, B., White, N., Ziegler, G.: Oriented Matroids, 2nd edn. Encyclopedia of Mathematics and its Applications, vol. 46. Cambridge University Press (1999)
5. Chepoi, S.: Clin d'œil on L_1-embeddable planar graphs. Discrete Applied Mathematics 80, 3–19 (1997)
6. Delling, D., Goldberg, A.V., Razenshteyn, I., Werneck, R.F.F.: Graph partitioning with natural cuts. In: Proc. 25th IEEE Intl. Parallel and Distributed Processing Symposium (IPDPS 2011), pp. 1135–1146 (2011)
7. Djoković, D.: Distance-Preserving Subgraphs of Hypercubes. Journal of Combinatorial Theory B 14, 263–267 (1973)

8. Dourado, M., Protti, F., Rautenbach, D., Szwarcfiter, J.: On the convexity number of graphs. Graphs and Combinatorics 28, 333–345 (2012)
9. Eppstein, D.: Cubic Partial Cubes from Simplicial Arrangements. The Electronic Journal of Combinatorics 13, #R79 (2006)
10. Glantz, R., Meyerhenke, H.: Finding all convex cuts of a plane graph in cubic time. Technical Report 2012,22, Karlsruhe Institute of Technology, Department of Informatics (2012)
11. Katriel, I.: On the algebraic complexity of set equality and inclusion. Information Processing Letters 92(4), 175–178 (2004)
12. Meyerhenke, H., Monien, B., Sauerwald, T.: A new diffusion-based multilevel algorithm for computing graph partitions. Journal of Parallel and Distributed Computing 69(9), 750–761 (2009); Best Paper Awards and Panel Summary: IPDPS 2008
13. Ovchinnikov, S.: Partial cubes: Structures, characterizations, and constructions. Discrete Mathematics 308, 5597–5621 (2008)
14. Thomassen, C.: The Jordan-Schönflies Theorem and the Classification of Surfaces. The American Mathematical Monthly 99, 116–130 (1992)
15. Wilkeit, E.: Isometric Embeddings in Hamming Graphs. Journal of Combinatorial Theory B 14, 179–197 (1990)

Shortest Paths with Bundles and Non-additive Weights Is Hard

Paul W. Goldberg* and Antony McCabe

Department of Computer Science, University of Liverpool, U.K.
{P.W.Goldberg,A.McCabe}@liverpool.ac.uk

Abstract. In a standard path auction, all of the edges in a graph are sold as separate entities, each edge having a single cost. We consider a generalisation in which a graph is partitioned and each subset of edges has a unique owner. We show that if the owner is allowed to apply a non-additive pricing structure then the winner determination problem becomes NP-hard (in contrast with the quadratic time algorithm for the standard additive pricing model). We show that this holds even if the owners have subsets of only 2 edges. For subadditive pricing (e.g. *volume discounts*), there is a trivial approximation ratio of the size of the largest subset. Where the size of the subsets is unbounded then we show that approximation to within a $\Omega(\log n)$ factor is hard. For the superadditive case we show that approximation with a factor of n^{ϵ} for any $\epsilon > 0$ is hard even when the subsets are of size at most 2.

1 Introduction

One of the most commonly studied types of *set-system* procurement auction is the path auction (e.g. [2,21,9,16]). In a typical path auction each seller, or *agent*, is represented by an edge in a graph and the *feasible sets* (the sets of sellers that are suitable to the buyer) are exactly those that contain a path between two specified vertices of the graph. This models a number of reasonable settings, such as routing over the internet or in a transport network.

In the standard setting, each of the edges is considered as if it were a separate entity and takes part in the auction as such. However, it seems reasonable to assume that multiple edges may actually be controlled by a single entity. In the examples of Internet routing or transport networks, it seems particularly likely that this assumption will hold. Therefore, we propose a model that incorporates the idea that some entity may control the pricing of a set of individual edges, and we allow them to price *bundles* of edges in a non-additive way. It is quite common, in economic and related literature, to study the concept of 'volume discounts' or 'bundling services' (see, e.g., [20,19]) — when the price charged for a collection of goods or services is lower than the sum of the individual prices. In many situations, producing large quantities of some commodity is more efficient

* Supported by EPSRC Grant EP/G069239/1 "Efficient Decentralised Approaches in Algorithmic Game Theory".

than producing smaller quantities and hence has lower unit cost. More directly, two reasonable motivations that may apply here are, firstly, that a path auction may actually be representative of something other than a network layout. It could be a model, for instance, to describe the components needed to produce some complicated commodity, such as an electronic device, where edges represent some component, and a path through the graph represents a collection of components that are sufficient to build the device. Alternatively, it is reasonable to assume, in general, that there may be some overhead to making each single purchase (such as the overhead of preparing bids, performing background and credit checks, or drawing up contracts). In this setting, the overhead may be substantial for selling a single item but could be much reduced for subsequent items. With this motivation it seems reasonable to allow a seller to alter their prices depending on the total number of edges they sell.

If we take volume discounts (or bundling) to an extreme level, we have a situation where a single price may be charged in order to purchase access to any or all of the edges owned by some entity. This can be particularly attractive in cases where there is surplus capacity on the network, such as at off-peak times, and it is the aim of the seller to create demand for their service that may otherwise not exist. A type of path auction where agents can own multiple edges in a graph was studied by Du et al. [7]. They show that allowing agents to manipulate which edges they declare ownership of may have disastrous effects on the payments that are made (more precisely, an exponential *frugality ratio* [16,18]). Hence we do not consider this, and simply assume that information on the ownership of edges is public knowledge.

Perhaps the most commonly seen auction mechanism is the Vickrey-Clarke-Groves (VCG) mechanism [22,5,13]. While this can be applied to any set-system auction, it requires that an optimal feasible set be chosen as 'winners'. However, this may not always be possible in polynomial time and this intractability is generally seen as extremely undesirable. There has also been attention in describing polynomial-time mechanisms, based on approximation algorithms, for a variety of settings [1,8,3] in which finding the optimal set (and hence running VCG) cannot be computed in polynomial time. Shortest path problems can be solved efficiently (in quadratic time [6]) and it is the aim of this paper to consider the complexity of finding an optimal solution when we apply bundling (or 'volume discounts'). Unfortunately, our results are largely negative. We find that even the most basic form of discount that we consider — a simple "buy at least x edges and get a discount of d" scheme — results in the winner-determination problem being NP-hard. This precludes running the VCG mechanism in polynomial time, which makes its use undesirable. We note that where the discounts are small relative to the costs then just ignoring discounts would be close to optimal. For the case of large discounts we show limits on the approximation ratio that may be achieved (subject to certain complexity theoretic assumptions).

Instead of having only negative 'discounts' we also consider positive values, which we call *volume supplements*. While superadditive pricing may not seem as readily justifiable as the subadditive case, it is worth studying for completeness.

We are able to show that the winner determination problem for superadditive pricing is also NP-hard, and it is also hard to find an approximate solution which is within a factor of n^ϵ of the optimal for any $\epsilon > 0$ where the supplement values are large in relation to the weights.

2 Preliminaries

Our model involves finding the shortest path between two specified vertices of a graph, taking into account the volume discounts that may be offered by the edge owners. In order to represent 'ownership' we partition the edges of a graph into disjoint sets, which we call *bundles*. Each bundle has a *discount vector*, which specifies a *discount value* depending on the number of edges in that bundle which are in the chosen path.

To compute a *discounted-weight* for a given path we add the weight of all of the edges in the path to the (negative) discount values for each of the bundles. Hence the aim of the problem is to discover the path between the specified vertices with the lowest discounted-weight, and we will see this is hard via a reduction from MINIMUM SET COVER, a well-known NP-hard problem [11].

Name SHORTEST PATH WITH DISCOUNTS
Instance A graph $G = (V, E)$, positive weight function w on edges, two distinct vertices s, t, a collection, $Z = \{Z_1, \ldots, Z_m\}$ of subsets (or *bundles*) of E ($Z_i \subseteq E$) such that $\forall e \in E$, e occurs in exactly one Z_i and a set of discount vectors $\mathcal{D} = \{\boldsymbol{d}^1, \ldots, \boldsymbol{d}^m\}$ (one for each Z_i) such that $d^i_j \leq 0$ for $j \in \{0, \ldots, |Z_i|\}$ (observe that, for ease of notation, this vector includes a discount value for zero edges).
Output The subset $P \subseteq E$ with minimum discounted-weight, given by $\sum_{e\in P} w(e) + \sum_{Z_i \in Z} d^i_{|Z_i \cap P|}$, that contains a path from s to t.

Name EXACT COVER BY 3-SETS
Instance A finite set X containing exactly $3n$ elements; a collection, C, of subsets of X each of which contains exactly 3 elements.
Output Does C contain an exact cover for X, i.e. a sub-collection of 3-element sets $D \subseteq C$ such that each element of X occurs in exactly one subset in D?

Name MINIMUM SET COVER
Instance A finite set X and a collection, C, of subsets of X.
Output What is the size of the minimum cover of X using C?

3 Complexity

We firstly show a simple reduction, from MINIMUM SET COVER, which shows hardness and also preserves approximation. Informally, we consider a multigraph on a line and show how the sets that 'cover' elements of a ground set (in a set cover) can be simulated with bundles of edges that cover gaps in the line which represents the ground set. The discounts are arranged such that the cost of every non-empty bundle of edges is 1, regardless of the specific number of edges used.

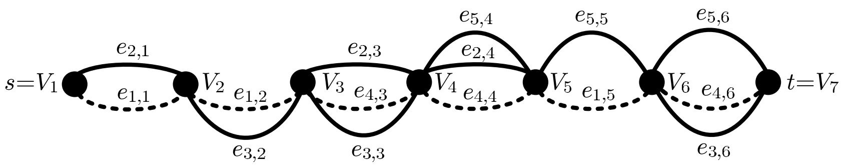
Lemma 1. MINIMUM SET COVER *is polynomial-time reducible to* SHORTEST PATH WITH DISCOUNTS.

Proof. Taking an instance I of MINIMUM SET COVER, let $m = |X|$ and let $n = |C|$. Build an instance $I' = (G, Z)$ of SHORTEST PATH WITH DISCOUNTS as follows. (We present G as a multigraph, although we will see that this assumption may be removed later on.) Let $n = |C|$ and let $m = |X|$. Create $m + 1$ vertices, labeled $V_1, \dots, V_{m+1}$, and let $s = V_1$ and $t = V_{m+1}$. For each $C_i \in C$, for all $j \in C_i$, add an edge $e_{i,j} = (V_j, V_{j+1})$ and add all of the $e_{i,j}$ edges to bundle Z_i. Let $w(e) = 1$ for all $e \in E$ be the weight function, and let the discount vectors be given by $\boldsymbol{d^i} = (0, 0, -1, -2, \dots, 1 - |Z_i|)$. Hence for a path P containing a non-zero number of edges from some bundle Z_i then $w(Z_i \cap P) + d^i_{|Z_I \cap P|} = 1$, so there is a contribution of exactly 1 to the discounted-weight for each included Z_i. Therefore, for any set of bundles $S \subseteq Z$ containing a path between s and t, the discounted-weight may be given by $|S|$.

An example of this reduction is shown in Fig. 1 and Fig. 2. Fig. 1 describes a MINIMUM SET COVER instance, for which the minimum cover is of size 2 (given by the set $D = \{C_1, C_4\}$). In the corresponding instance of SHORTEST PATH WITH DISCOUNTS in Fig. 2, the lowest discounted-weight path is of weight 2, consisting of the path $P = \{e_{1,1}, e_{1,2}, e_{1,5}, e_{4,3}, e_{4,4}, e_{4,6}\}$ (shown as dashed lines), and hence the bundles $S = \{Z_1, Z_4\}$.

$X = \{1,2,3,4,5,6\}$				
$C_1 = \{1,2,5\}$	$C_2 = \{1,3,4\}$	$C_3 = \{2,3,6\}$	$C_4 = \{3,4,6\}$	$C_5 = \{4,5,6\}$

Fig. 1. Example of MINIMUM SET COVER



$Z_1 = \{e_{1,1}, e_{1,2}, e_{1,5}\}$, $Z_2 = \{e_{2,1}, e_{2,3}, e_{2,4}\}$, $Z_3 = \{e_{3,2}, e_{3,3}, e_{3,6}\}$,
$Z_4 = \{e_{4,3}, e_{4,4}, e_{4,6}\}$, $Z_5 = \{e_{5,4}, e_{5,5}, e_{5,6}\}$

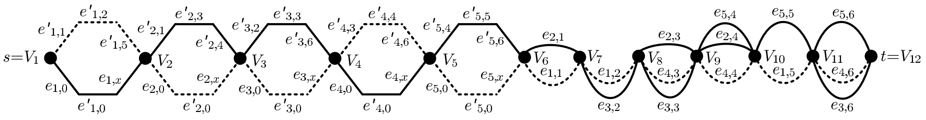
Fig. 2. SHORTEST PATH WITH DISCOUNTS construction from Fig. 1

Returning to the proof, a set S contains a path between s and t if and only if it contains an edge (V_j, V_{j+1}) for all $j \in \{1, \dots, m\}$. Hence, where S contains such a path, let $D \subseteq C$ be the set containing C_i if and only if $Z_i \in S$. Hence D contains a cover of X and $|D| \leq |S|$ — as an edge from (V_j, V_{j+1}) in some $Z_i \in S$ implies an element j in some $C_i \in D$. Similarly, where D contains a cover of X, then the corresponding set S contains a path from s to t and $|S| \leq |D|$. Following this translation, we have $|S| = |D|$ and the size of the minimum cover of X can be computed by finding a path P with lowest discounted weight (and hence the number of bundles in S which contain the edges in P). □

The reduction in Lemma 1 gives an instance of SHORTEST PATH WITH DISCOUNTS that is typically a multigraph. This can be amended to construct a simple graph, by replacing each edge with two edges in series (adding one new vertex shared by each pair of edges).

Lemma 1 shows that SHORTEST PATH WITH DISCOUNTS is NP-hard, in general, provided that the cardinality of the bundles is at least 3 (as a generalisation of EXACT COVER BY 3-SETS). As bundles of size 1 correspond to a polynomial-time computable shortest-path [6], we now consider the remaining case where the bundles are of size at most 2 — and show that this also results in an NP-hard problem, even when the discount values are arbitrarily small.

In Fig. 3 we see an example of the proposed reduction — Fig. 1 is encoded as an instance of SHORTEST PATH WITH DISCOUNTS; where the shortest discounted path is given by $P = \{e'_{1,1}, e'_{1,2}, e'_{1,5}, e_{2,0}, e'_{2,0}, e_{2,x}, e_{3,0}, e'_{3,0}, e_{3,x}, e'_{4,3}, e'_{4,4}, e'_{4,6}, e_{5,0}, e'_{5,0}, e_{5,x}, e_{1,1}, e_{1,2}, e_{4,3}, e_{4,4}, e_{1,5}, e_{4,6}\}$, which is marked as a dashed line. Observe that Fig. 1 is a 'yes' instance of EXACT COVER BY 3-SETS and a path of discounted-weight $21 + 9d$ exists only because just two 'upper' paths between V_1 and V_6 (containing $e'_{1,...}$ and $e'_{4,...}$) are sufficient to give a discount on all edges in a path from V_6 to t. A 'no' instance would require three or more upper paths, which must give a discounted-weight greater than $21 + 9d$.



$$Z = \{\{e_{1,1}, e'_{1,1}\}, \{e_{1,2}, e'_{1,2}\}, \{e_{1,5}, e'_{1,5}\}, \{e_{1,0}, e'_{1,0}\}, \{e_{1,x}\}, \{e_{2,1}, e'_{2,1}\}, \{e_{2,3}, e'_{2,3}\},$$
$$\{e_{2,4}, e'_{2,4}\}, \{e_{2,0}, e'_{2,0}\}, \{e_{2,x}\}, \{e_{3,2}, e'_{3,2}\}, \{e_{3,3}, e'_{3,3}\}, \{e_{3,6}, e'_{3,6}\}, \{e_{3,0}, e'_{3,0}\},$$
$$\{e_{3,x}\}, \{e_{4,3}, e'_{4,3}\}, \{e_{4,4}, e'_{4,4}\}, \{e_{4,6}, e'_{4,6}\}, \{e_{4,0}, e'_{4,0}\}, \{e_{4,x}\}, \{e_{5,4}, e'_{5,4}\},$$
$$\{e_{5,5}, e'_{5,5}\}, \{e_{5,6}, e'_{5,6}\}, \{e_{5,0}, e'_{5,0}\}, \{e_{5,x}\}\}$$

Fig. 3. SHORTEST PATH WITH DISCOUNTS translated from Fig. 1 as described in Theorem 1. The dashed line marks an optimal solution.

Theorem 1. SHORTEST PATH WITH DISCOUNTS *is NP-hard even when each bundle has size at most* 2 *and there is a single small discount value* $d < 0$.

Proof. Let $I = (X, C)$ be an instance of EXACT COVER BY 3-SETS, and build an instance $I' = (G, Z)$ of SHORTEST PATH WITH DISCOUNTS as follows. Let $n = |C|$ and let $m = |X|$. Create $n + m + 1$ new vertices, labeled $V_1, \ldots, V_{n+m+1}$, and let $s = V_1$ and $t = V_{n+m+1}$. Let $w(e) = 1$ for every edge $e \in G$.

For each of the subcollections $C_i \in C$, let the elements in C_i be given by $\{x, y, z\}$. Insert three edges in series $e'_{i,x}$,$e'_{i,y}$,$e'_{i,z}$ making an 'upper' path from V_i to V_{i+1} in G (adding unlabelled vertices, as needed). Next, insert three additional edges $e_{i,x} = (V_{n+x}, V_{n+x+1})$,$e_{i,y} = (V_{n+y}, V_{n+y+1})$,$e_{i,z} = (V_{n+z}, V_{n+z+1})$. Now create three bundles $Z_{i,x} = \{e'_{i,x}, e_{i,x}\}$, $Z_{i,y} = \{e'_{i,y}, e_{i,y}\}$, and $Z_{i,z} = \{e'_{i,z}, e_{i,z}\}$ and add these to Z.

For each $i \in \{1, \ldots, n\}$ create a second 'lower' path from V_i to V_{i+1} consisting of three new edges, in series, and add two new bundles to Z, one which contains two of these three new edges and another bundle containing the remaining edge. Let d be some small value, $-1 \leq d < 0$ for a discount parameter and let the discount vectors be fixed, $\boldsymbol{d}^j = (0, 0, d)$ for all $j \in Z$. (The purpose of the 'lower' paths are to create an alternative path from each V_i to V_{i+1} which has a discounted-weight of $3 + d$ — as two of the edges are in the same bundle, exactly one will be discounted.)

Taking $P \subseteq E$ to be the lowest discounted-weight path between s and t, let $S \subseteq Z$ be the minimal set of bundles that contain all of the edges in P. We can assume that all of the edges in P between V_{n+1} and V_{n+m+1} are 'discounted' edges (i.e. they are in some bundle with another edge that is also in P). This is without loss of generality, as for any path P containing an undiscounted edge e, there is another path P', having the same discounted-weight, that includes a discounted edge e' in place of e. To verify this, we know that edge e is in a bundle with another edge e'. As e is not discounted, we know that e' is not in the path. Hence, there is some 'lower' path chosen between V_1 and V_{n+1} that avoids e' with discounted-weight $3+d$. Create a path P' by including the 'upper' path that includes e', which increases the discounted-weight by $-d$ on this path, but also decreases the discounted-weight of edge e by $-d$, hence paths P and P' have the same discounted-weight.

Observe that any minimal path from s to t contains exactly $3n + m$ edges. Furthermore, we have seen that there is a minimum path P where the m edges from V_{n+1} to V_{n+m+1} are all discounted. Hence, we are interested in how many of the upper paths must be selected between V_1 and V_{n+1}, as these add to the weight of the solution when compared to the lower paths. Assume a collection D that contains a minimum cover of X; now consider a set of bundles S such that for every $C_i \in D$ all of $Z_{i,x}, Z_{i,y}, Z_{i,z}$ are present in S. Observe that S contains a path from V_{n+1} and V_{n+m+1}, or else D does not cover every element in X. Furthermore, S contains an upper path between two vertices in $V_i, \ldots, V_{i+1}$ only if D contains the corresponding set C_i. Let P be the path obtained from all the bundles in S, as well as 'lower' paths between any vertices in $\{V_1, \ldots, V_{n+1}\}$ that were not connected by S. We can then determine that the total discounted-weight of path P may be given by $3n+m+nd+md-|D|d$ (discounts are applied to m edges from V_{n+1} to V_{n+m+1}, and $n - |D|$ edges in the lower paths from V_1 to V_{n+1}). We can see that P is, in fact, a minimum discounted-weight path, as any lower weight path must have more lower paths chosen, yet still discounts all of the edges between V_{n+1} and V_{n+m+1}, which would imply a cover of X exists which is smaller than D, giving a contradiction.

We now claim that the path from s to t with lowest discounted-weight will have a weight of $3n + m + nd + md - (m/3)d$ if and only if C contains an exact cover for X, i.e. there is a sub-collection D of 3-element sets $D = (D_1, ..., D_n)$

such that each element of X occurs in exactly one subset in D. Observe that if C contains an exact cover, there is a set D such that $|D| = m/3$ and every element in X is contained in some C_i in D. Now consider the path P that is created, as described, by augmenting the set of bundles S (by adding lower paths, where needed). We have seen that the discounted-weight of P may be given by $3n+m+nd+md-|D|d$; hence where an exact cover exists then P has weight of $3n+m+nd+md-|m/3|d$. If no exact cover exists then the discounted-weight of P is at least $3n+m+nd+md-(|m/3|+1)d$.

Hence we have the decision problem "Is there a path in I' with discounted-weight of $3n+m+nd+md-(m/3)d$?" which has a 'yes' answer if and only if instance I contains an exact cover by 3-sets, which is NP-hard to compute. As computing a minimum discounted-weight path must answer this question (we have seen that no lower weight path can exist), then we see that computing the minimum discounted-weight path is also NP-hard. □

Thus we have seen NP-hardness in the case where one edge in a bundle is discounted, and one is not. If we assume a slightly more general scheme, in which a discount is only applied after x edges are purchased, it is a simple observation to extend this hardness result. In the construction given in Theorem 1, simply simulate every edge $e \in G$ by $x-1$ edges in series, which are contained in the same bundle. Hence, in this new graph, selecting any two of these new simulated edges will trigger the discount value.

It is also worth noting, again, that although this reduction was presented for simplicity as a multigraph, it would also work as a simple graph. All of the edges between V_{n+1} to V_{n+m+1} could be replaced by two edges, in series, having a new undiscounted edge added to the original edge. This would simply add a constant weight of m to every possible minimal path, and hence the complexity remains unchanged.

4 Inapproximability Results

If the discount values are small, relative to the weights, then simply ignoring the discount values and computing a shortest path would give a good polynomial-time approximation of the optimal solution. (Assuming a maximum weight of 1 and a discount value of d, then $1/(1+d)$ is an upper bound on the approximation ratio.) However, if we assume larger discount values, we can see inapproximability results, from Lemma 1. If we assume an unweighted case ($\forall e \in E, w(e) = 1$) and allow large discount values, then we have a scenario in which the sets of edges can be thought of as bundled together, where a single price applies to all non-empty subsets of a bundle.

We now discuss the implications of Lemma 1 in terms of the inapproximability of finding a minimum discounted-weight path, although this is primarily a short review of known set cover approximations.

4.1 Bounded Cardinality

Let k be an upper bound on the cardinality of the bundles; i.e. no single owner has more than k edges, and call this k-CARDINALITY SHORTEST PATH WITH DISCOUNTS. MINIMUM k-SET COVER is the variation in which the cardinality of all sets in C are bounded from above by a constant k; and hence it is a corollary of Lemma 1 that MINIMUM k-SET COVER may be computed exactly with k-CARDINALITY SHORTEST PATH WITH DISCOUNTS.

It is known that the MINIMUM k-SET COVER can be approximated to within a constant factor, but that no better than a constant factor is possible unless P=NP. (see, e.g., [10,14]). Fairly obviously, we can get a k-approximation with the 'bundling' discount scheme, by ignoring the discounts and simply computing the shortest path regardless. It is worth comparing this with the approximation ratios that are known for MINIMUM k-SET COVER. The best approximation ratio for MINIMUM k-SET COVER remains something of an open problem — the best known result for the weighted case is currently $H_k - \frac{k-1}{8k^9}$, where $H_k = \sum_{1,\dots k} \frac{1}{k}$ is the $k-$th harmonic number [14], and hence finding some better approximation ratio for k-CARDINALITY SHORTEST PATH WITH DISCOUNTS would likely be a difficult, but significant, result.

Theorem 2. *No polynomial-time algorithm with an approximation ratio of less than* $(1-\epsilon)\ln n$ *for any* $\epsilon > 0$ *exists for* SHORTEST PATH WITH DISCOUNTS *unless* $NP \subset D_{TIME}(n^{\log\log n})$.

Proof. In the proof of Lemma 1, we show that all possible solutions to a MINIMUM SET COVER problem exist with identical weight in an instance of SHORTEST PATH WITH DISCOUNTS and vice-versa, so it is approximation-preserving. Hence any approximation ratio for SHORTEST PATH WITH DISCOUNTS would give the same ratio for MINIMUM SET COVER. It was shown by Feige in [10], for MINIMUM SET COVER, that no approximation ratio of $(1-\epsilon)\ln n$ exists for any $\epsilon > 0$ under the assumption that $NP \subset D_{TIME}(n^{\log\log n})$, hence this also holds for SHORTEST PATH WITH DISCOUNTS. □

5 Superadditive Pricing

We now consider the alternative setting, where instead of a (negative) 'discount' value, we introduce a (positive) 'supplement'. We call this problem SHORTEST PATH WITH SUPPLEMENTS, which differs from SHORTEST PATH WITH DISCOUNTS in that each of the 'discount' vectors has only positive values $d^i_{|T|} \geq 0$ (we call these 'supplement' values).

We will see, firstly that even for small supplement values (any $d > 0$) that finding the optimal solution is NP-hard. If we have larger supplement values then we will also see strong inapproximability results. Hardness is shown with a reduction from EXACT COVER BY 3-SETS, as follows. The reduction is demonstrated with the example from Fig. 1, which is shown in Fig. 4. Observe that the shortest path has supplemented weight of 6 — which may be described by either $(1,2,5),(3,4,6)$ or $(3,4,6),(1,2,5)$.

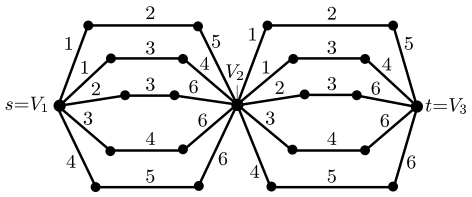


Fig. 4. Fig. 1 As a SHORTEST PATH WITH SUPPLEMENTS problem. $Z = \{1, 2, 3, 4, 5, 6\}$; edges are labeled with the 'owner'.

Theorem 3. SHORTEST PATH WITH SUPPLEMENTS *is NP-hard, even with bundles of size at most* 2.

Proof. In the first stage, we will ignore the restriction on the size of the bundles. Take the input X, C to an instance I of EXACT COVER BY 3-SETS. Let $m = |X|$, and build an instance I' of SHORTEST PATH WITH SUPPLEMENTS as follows. Let $Z' = X$ be the set of owners, and let $V' = \{V'_1, \ldots, V'_{m/3+1}\}$. For each triple $C_i \in C$ let the elements in C_i be given by $\{x, y, z\}$, and insert $m/3$ paths of length 3, one between each V'_j and V'_{j+1} for all $j \in \{1, \ldots, m/3\}$, and for each of these paths, let the owners of the edges be given by one each of $\{x, y, z\}$ (the ordering is unimportant). Let $m' = |E'|$, let there be a small supplement parameter, $1/m^2 > d > 0$, and let the supplement vectors be fixed, $\boldsymbol{d}^i = (0, 0, d, \ldots, d)$.

Observe that the shortest path from s to t contains exactly m edges. Also observe that, where a shortest path exists of weight m, this has exactly m edges and that no edges share the same owner. Where an exact cover by 3-sets exists, then there are exactly $m/3$ subsets of C that contain all of the elements in X. Hence there are $m/3$ paths (each of length 3) from V_1 to V_{m+1} which can be selected, such that no owner appears for more than one edge. Hence a shortest path of weight m exists in instance I' if and only if the instance I is a 'yes' instance of EXACT COVER BY 3-SETS, and SHORTEST PATH WITH SUPPLEMENTS is NP-hard.

Now, we can see that this still holds where the bundles are of at most size 2. For the graph $G' = V', E'$, build an input I'' with graph $G'' = V'', E''$. Replace every edge e' in E' by a path of length $m' - 1$, and label each of these edges $e''_{(e,j)}$ for all $j \in E' \setminus \{e'\}$. For every bundle Z_i in the input, consider every pair of edges $(u, v) \in Z_i$ and add a new bundle to Z'' containing the two edges labeled $e''_{(u,v)}$ and $e''_{(v,u)}$. Hence for any path $P' \subseteq E'$ of size t there is a corresponding path $P'' \subseteq E''$ (with the replaced edges) of size $(m' - 1)t$. Also observe that P'' contains two edges in the same bundle, $e''_{(u,v)}$ and $e''_{(v,u)}$ if and only if P' contained two edges (u, v) in the same bundle. Hence for every shortest path P' in I' with supplemented weight given by $|P| + xd$, the shortest path in I'' has supplemented weight $|P|(m' - 1) + xd$; hence the solution to instance I'' gives a solution to instance I', and thus instance I, and SHORTEST PATH WITH SUPPLEMENTS is NP-hard even with edge bundles of size at most 2. □

As in the subadditive case, it is easy to observe that where the supplement values are small then ignoring the discounts gives a good approximation ratio (assuming a maximum weight of 1 and a maximum supplement value of d, then d is an upper bound on the approximation ratio, as the optimal solution has a supplemented weight of at least the 'found' solution, and the supplemented weight of this found solution is bounded from above by a factor of d). However, if we allow supplement values to be as large as m, then we will see strong inapproximability results, from SHORTEST PATH WITH FORBIDDEN PAIRS.

Name SHORTEST PATH WITH FORBIDDEN PAIRS
Instance A weighted graph $G = (V, E, w)$ with two distinct vertices s, t and a collection, $F = \{F_1, \ldots, F_m\}$ of pairs of vertices.
Output The minimum weight path $P \subset E$ from s to t such that for every edge $e \in P$ adjacent to vertex x there is no other edge $e' \in P$ adjacent to vertex y where there is some $F_i \in F = \{x, y\}$.

It is known that no polynomial time approximation algorithm for SHORTEST PATH WITH FORBIDDEN PAIRS exists with an approximation ratio of n^ϵ for any $\epsilon > 0$, unless P=NP [15].

Theorem 4. *No polynomial-time algorithm with an approximation ratio of less than n^ϵ for any $\epsilon > 0$ exists for* SHORTEST PATH WITH SUPPLEMENTS *unless P=NP.*

Proof. Taking the input to a SHORTEST PATH WITH FORBIDDEN PAIRS, build an instance of SHORTEST PATH WITH SUPPLEMENTS with $G' = G$ (let $n = |V|$ and $m = |E|$). For each pair of forbidden vertices $\{x, y\}$ assume an edge $e \in E$ that is adjacent to one of these vertices (let it be x) and add a bundle Z_i containing this edge e and all edges that are adjacent to y. Assume an unweighted graph, i.e. $w(e) = 1$ for all $e \in E$, and that discount values are large; $d \geq n$; hence any SHORTEST PATH WITH SUPPLEMENTS is exactly a shortest path that contains no two edges in the same bundle (unless no such path exists, in which case the total weight is greater than n). Observe that any path through vertex x contains an edge adjacent to x, and hence if it also contains an edge adjacent to y then there is a bundle containing both edges and the supplemented weight is at least $n + 1$ (from the construction of bundles). Similarly, any path with supplemented weight of less than n contains no two edges in the same bundle, and hence does not visit the forbidden pair $\{x, y\}$.

Another instance I'' can be created with bundles of size at most 2 in the same way as in the proof of Theorem 3, by replacing each edge with m edges in series. Let the weight of each edge be $w''(e'') = 1/m$. Hence every SHORTEST PATH WITH SUPPLEMENTS (with weight of less than m) gives a path of the same size in I which avoids forbidden pairs. Let n'' be the number of vertices in I'', and observe that $n'' \leq n^3$. Any approximation algorithm for I'' that gives an approximation ratio of $\alpha = n''^\epsilon$ (with $\epsilon > 0$) for I'' would give an approximation ratio of $\alpha = n^{\epsilon/3}$ for SHORTEST PATH WITH FORBIDDEN PAIRS, which would imply P=NP. □

6 Conclusion and Open Problems

We gave a model for generalising shortest-path problems that appears to have relevance to economic theory. As 'volume discounts' appear in a wide range of literature, it seems an obvious goal to try and combine them with the well-studied path auction. However, the results we have seen are quite negative — even the very simplest discounting scheme, of "buy at least x edges and get a d discount" results in the winner determination problem becoming NP-hard, and hence the VCG mechanism may not be a practical solution.

Our only alternative mechanism, of ignoring the discounts, is obviously quite unsatisfactory — we have identified that allowing volume discounts is a desirable feature. We have also shown that better than $\Omega(\log(n))$ approximation ratios may not be possible, where the discount parameters may be large. This is a reasonable assumption in many settings, such as the supply of services when there is relatively little overhead, for example the use of transport or network infrastructures that have surplus capacity.

Considering the problem as k-CARDINALITY SHORTEST PATH WITH DISCOUNTS, we do not know how to improve on the approximation ratio of k, or if a ratio of k might be close to optimal. However, there is scope for the existence of an approximation algorithm with an improved approximation ratio. An approximation algorithm which does better than the naive approach of ignoring the discounts would likely be a fundamental contribution to a better auction mechanism. There could be economic benefits to being able to run a tractable procurement auction which takes advantage of these volume discounts, and this is an area that might benefit from future study.

The reductions of Theorem 1 and Theorem 3 show that finding an exact solution is NP-hard, even for series-parallel graphs. We may also be interested in discovering if there are certain other properties of the problem that are required for hardness results — for example, the distance between edges that are bundled together. However, it is easy to observe in Lemma 1 that the solutions to a single-pathed multigraph give exactly the solutions to a set-cover problem of the same size, and hence there exists a H_k-approximation for this class of graph. If the general case does not give rise to a better than k approximation, there may be interesting results in finding classes of graph that have a better approximation.

It is also worth noting that there is a difference in the approximation of the subadditive and superadditive cases. While the discount parameters may be reasonably limited by the weights of the edges, the supplement parameter does not have this limitation — and so we find that approximation may be harder in the superadditive case when the supplement parameter is large.

Differences in approximation can be particularly important in terms of auction mechanisms; the frugality ratio of an auction mechanism is often used to measure its performance in terms of payment. It was shown in [12] that the frugality ratio (of a truthful approximation mechanism) may be upper bounded a factor of the approximation ratio; hence a good approximation ratio is important in making an attractive auction mechanism. Mechanisms with a much better frugality ratio than VCG have been shown to exist for path auctions (e.g. [16,4,17]). It is a natural question to ask if similar approaches could improve frugality for this auction model. By combining those approaches with a good approximation algorithm, it may be possible to create a particularly desirable auction mechanism — that is both tractable and has good frugality in this setting.

References

1. Archer, A., Papadimitriou, C., Talwar, K., Tardos, E.: An approximate truthful mechanism for combinatorial auctions with single parameter agents. In: Procs. of 14th SODA, pp. 205–214 (2003)
2. Archer, A., Tardos, E.: Frugal path mechanisms. In: Procs. of 13th SODA, pp. 991–999 (2002)
3. Briest, P., Krysta, P., Vöcking, B.: Approximation techniques for utilitarian mechanism design. SIAM J. Comput. 40(6), 1587–1622 (2011)
4. Chen, N., Elkind, E., Gravin, N., Petrov, F.: Frugal mechanism design via spectral techniques. In: Procs. of 51st FOCS, pp. 755–764 (2010)
5. Clarke, E.H.: Multipart pricing of public goods. Public Choice 11(1) (1971)
6. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)
7. Du, Y., Sami, R., Shi, Y.: Path auctions with multiple edge ownership. Theoretical Computer Science 411(1), 293–300 (2010)
8. Elkind, E., Goldberg, L.A., Goldberg, P.W.: Frugality ratios and improved truthful mechanisms for vertex cover. In: Procs. of the 8th ACM-EC, pp. 336–345 (2007)
9. Elkind, E., Sahai, A., Steiglitz, K.: Frugality in path auctions. In: Procs. of 15th SODA, pp. 701–709 (2004)
10. Feige, U.: A threshold of ln n for approximating set cover. Journal of the ACM 45, 314–318 (1998)
11. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness (1979)
12. Goldberg, P.W., McCabe, A.: Commodity auctions and frugality ratios. In: Serna, M. (ed.) SAGT 2012. LNCS, vol. 7615, pp. 180–191. Springer, Heidelberg (2012)
13. Groves, T.: Incentives in teams. Econometrica 41(4), 617–631 (1973)
14. Hassin, R., Levin, A.: A better-than-greedy approximation algorithm for the minimum set cover problem. SIAM Journal on Computing 35(1), 189–200 (2005)
15. Kann, V.: Polynomially bounded minimization problems that are hard to approximate. Nord. J. Comput. 1(3), 317–331 (1994)
16. Karlin, A.R., Kempe, D., Tamir, T.: Beyond VCG: Frugality of truthful mechanisms. In: Procs. of 46th FOCS, pp. 615–626 (2005)
17. Kempe, D., Salek, M., Moore, C.: Frugal and truthful auctions for vertex covers, flows and cuts. In: Procs. of 51st FOCS, pp. 745–754 (2010)
18. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V.: Algorithmic Game Theory. Cambridge University Press (2007)
19. Phillips, R.: Pricing and Revenue Optimization. Stanford University Press (1995)
20. Stremersch, S., Tellis, G.J.: Strategic Bundling of Products and Prices: A New Synthesis for Marketing. Journal of Marketing 66(1), 55–72 (2000, 2002)
21. Talwar, K.: The price of truth: Frugality in truthful mechanisms. In: Alt, H., Habib, M. (eds.) STACS 2003. LNCS, vol. 2607, pp. 608–619. Springer, Heidelberg (2003)
22. Vickrey, W.: Counterspeculation, Auctions, and Competitive Sealed Tenders. The Journal of Finance 16(1), 8–37 (1961)

Cliques and Clubs*

Petr A. Golovach[1], Pinar Heggernes[1], Dieter Kratsch[2], and Arash Rafiey[1]

[1] Department of Informatics, University of Bergen, Norway
{petr.golovach,pinar.heggernes,arash.rafiey}@ii.uib.no
[2] LITA, Université de Lorraine - Metz, France
kratsch@univ-metz.fr

Abstract. Clubs are generalizations of cliques. For a positive integer s, an s-club in a graph G is a set of vertices that induces a subgraph of G of diameter at most s. The importance and fame of cliques are evident, whereas clubs provide more realistic models for practical applications. Computing an s-club of maximum cardinality is an NP-hard problem for every fixed $s \geq 1$, and this problem has attracted significant attention recently. We present new positive results for the problem on large and important graph classes. In particular we show that for input G and s, a maximum s-club in G can be computed in polynomial time when G is a chordal bipartite or a strongly chordal or a distance hereditary graph. On a superclass of these graphs, weakly chordal graphs, we obtain a polynomial-time algorithm when s is an odd integer, which is best possible as the problem is NP-hard on this class for even values of s. We complement these results by proving the NP-hardness of the problem for every fixed s on 4-chordal graphs, a superclass of weakly chordal graphs. Finally, if G is an AT-free graph, we prove that the problem can be solved in polynomial time when $s \geq 2$, which gives an interesting contrast to the fact that the problem is NP-hard for $s = 1$ on this graph class.

1 Introduction

Max Clique is one of the most fundamental problems in graph algorithms. Cliques model highly connected or correlated parts of networks and data sets, and consequently they find applications in numerous diverse fields. For many real problems, however, cliques present a too restrictive measure of connectivity (see e.g., [1,15,25,31]), and the notion of clubs were proposed to give more realistic models [3,26]. Given a graph $G = (V, E)$ on n vertices and an integer s between 1 and n, a vertex subset $S \subseteq V$ is an s-club if the subgraph of G induced by S has diameter at most s. Hence 1-clubs are exactly cliques, and every s-club is also an $(s + 1)$-club by definition. Notice the non-hereditary nature of s-clubs, which makes their behavior different from that of cliques for $s \geq 2$: although every subset of a clique is a clique, the same is not true for an s-club. In fact, deciding whether a given s-club is maximal, in the sense that no superset of it is an s-club, is NP-complete for every fixed $s \geq 2$ [27].

* This work is supported by the European Research Council and the Research Council of Norway.

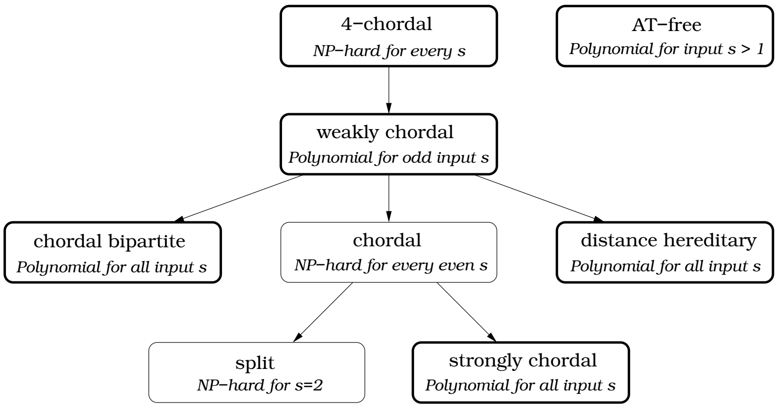


Fig. 1. The inclusion relationship among the mentioned graph classes, where each child is a subset of its parent. AT-free graphs are not related to the rest. Boxes with thicker frames contain the results proved in this paper.

Given a graph G and an integer s, the objective of the MAX s-CLUB problem is to compute an s-club of maximum cardinality. We are interested in the exact solution of this problem. Note that the problem becomes trivial if G has diameter at most s.

MAX s-CLUB is NP-hard for every fixed s, even on graphs of diameter $s + 1$ [7]. It remains NP-hard on bipartite graphs for every fixed $s \geq 3$, and on chordal graphs for every even fixed $s \geq 2$ [4]. On split graphs MAX 2-CLUB in NP-hard [4], whereas MAX s-CLUB has a trivial solution for all input $s \geq 3$. On general graphs, the problem is fixed-parameter tractable when parameterized by the solution size [13] for every fixed $s \geq 2$, or by the dual of the solution size [30] for every fixed s. Fixed-parameter tractability of MAX 2-CLUB has been studied also with respect to various other parameters [20]. Furthermore, MAX s-CLUB can be solved by an $O(1.62^n)$-time algorithm [13]. The problem can be solved in polynomial time on trees and interval graphs for all input values of s, and on graphs of bounded treewidth and graphs of bounded clique-width for every fixed s that is not a part of the input [29].

In this paper we show that MAX s-CLUB can be solved in polynomial time for all odd input values of s on weakly chordal graphs. For subclasses of weakly chordal graphs, we show that the problem can be solved in polynomial time for all input values of s on chordal bipartite graphs, strongly chordal graphs, and distance hereditary graphs. To complement these positive results, we show that on 4-chordal graphs, which form a superclass of weakly chordal graphs, the problem is NP-hard for every fixed s. In addition to these results, we show that the problem is solvable in polynomial time for all input $s \geq 2$ on AT-free graphs. Interestingly, MAX CLIQUE is NP-hard on this graph class. The inclusion relationship among the graph classes mentioned above is illustrated in Fig. 1, which also summarizes our results.

In this extended abstract, the proofs of Lemma 1, Theorem 6 and Lemma 4 are omitted due to space restrictions.

2 Definitions and First Observations

We refer to the textbook by Diestel [17] for any undefined graph terminology. We consider finite undirected graphs without loops or multiple edges. Such a graph $G = (V, E)$ is identified by its vertex set V and its edge set E. Throughout the paper, we let $n = |V|$ and $m = |E|$. The subgraph of G induced by $U \subseteq V$ is denoted by $G[U]$. For a vertex v, we denote by $N_G(v)$ the set of vertices that are adjacent to v in G. The *distance* $\text{dist}_G(u, v)$ between vertices u and v of G is the number of edges on a shortest path between them. The *diameter* $\text{diam}(G)$ of G is $\max\{\text{dist}_G(u, v) \mid u, v \in V\}$. The *complement* of G is the graph $\overline{G}$ with vertex set V, such that any two distinct vertices are adjacent in $\overline{G}$ if and only if they are not adjacent in G. For a positive integer k, the *k-th power* G^k of G is the graph with vertex set V, such that any two distinct vertices u, v are adjacent in G^k if and only if $\text{dist}_G(u, v) \leq k$. We say that P is a (u, v)-path if P is a path that joins u and v. The vertices of P different from u and v are the *inner* vertices of P. The *chordality* $ch(G)$ of a graph G is the length of the longest induced cycle in G; if G has no cycles, then $ch(G) = 0$. A set of pairwise adjacent vertices is a *clique*. A clique is *maximal* if no proper superset of it is a clique, and *maximum* if it has maximum size.

For a non-negative integer k, a graph G is *k-chordal* if $ch(G) \leq k$. A graph G is *weakly chordal* if both G and $\overline{G}$ are 4-chordal. A graph is *chordal bipartite* if it is both 4-chordal and bipartite. A graph is *chordal* if it is 3-chordal. A graph is a *split* graph if its vertex set can be partitioned in an independent set and a clique. A chord xy in a cycle C of even length is said to be *odd* if the distance in C between x and y is odd. A graph is *strongly chordal* if it is chordal and every cycle of even length at least 6 has an odd chord. A graph G is a *distance hereditary* if for any connected induced subgraph H of G, if u and v are in H, then $\text{dist}_G(u, v) = \text{dist}_H(u, v)$. An *asteroidal triple (AT)* is a set of three non-adjacent vertices such that between each pair of them there is a path that does not contain a neighbor of the third. A graph is *AT-free* if it contains no AT. Each of these graph classes can be recognized in polynomial (in most cases linear) time and they are closed under taking induced subgraphs [9,18]. See the monographs by Brandstädt et al. [9] and Golumbic [18] for more properties and characterizations of these classes and their inclusion relationships.

Let s be a positive integer. A set of vertices S in G is an *s-club* if $\text{diam}(G[S]) \leq s$. An s-club of maximum size is a *maximum* s-club. Given a graph G and a positive integer s, the Max s-Club problem is to compute a maximum s-club in G. Cliques are exactly 1-clubs, and hence Max 1-Club is equivalent to Max Clique.

Observation 1. *Let G be a graph and let s be a positive integer. If S is an s-club in G then S is a clique in G^s.*

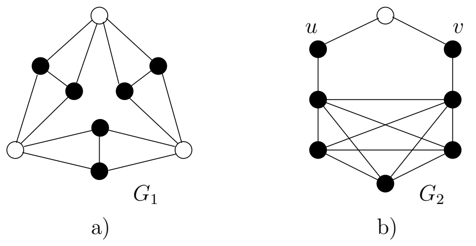


Fig. 2. Cliques in G^2 and 2-clubs

Although Observation 1 is easy to see, it is important to note that the backward implication does not hold in general: a (maximal) clique in G^s is not necessarily an s-club. To see this, let $s = 2$ and consider the graphs shown in Fig. 2 a) and b). The set of vertices S shown in black in Fig. 2 a) is the unique maximum clique of G_1^2, but clearly S is not a 2-club in G_1, as $G_1[S]$ is not even connected. The example in Fig. 2 b) shows that it does not help to require that $G[S]$ is connected: the set of black vertices S is a maximal clique in G_2^2, $G_2[S]$ is connected, but S is not a 2-club in G_2, because $\text{dist}_{G_2[S]}(u, v) = 3$. Observe also that a maximum s-club in G is not necessarily a maximal clique in G^s. Furthermore, the maximum size of a clique in G_1^2 is strictly greater than the maximum size of a 2-club in G_1. For the set of black vertices S in Fig. 2 b), $S \setminus \{u\}$ and $S \setminus \{v\}$ are maximum 2-clubs, whereas S is a maximal clique in G_2^2.

As we will show in Section 3, for some graph classes, maximal cliques in s-th powers are in fact s-clubs. For a positive integer s, we say that a graph class $\mathcal{G}$ has the *s-clique-power* property if for every graph $G \in \mathcal{G}$, every maximal clique in G^s is an s-club in G. Furthermore, we say that $\mathcal{G}$ has the *clique-power* property if every maximal clique in G^s is an s-club in G, for every positive integer s and every graph $G \in \mathcal{G}$. Due to Observation 1, we see that if G belongs to a graph class that has the clique-power property, then a vertex set S in G is a maximal s-club if and only if S is a maximal clique in G^s. As G^s can be computed in time $O(n^3)$ for any positive s, the following is immediate, and it will be the framework in which we obtain our results.

Proposition 1. *Let $\mathcal{G}$ be a graph class that has the clique-power property and let s be a positive integer.*

- *If* Max Clique *can be solved in time $O(f(n))$ on $\{G^s \mid G \in \mathcal{G}\}$, then* Max s-Club *can be solved in time $O(f(n) + n^3)$ on $\mathcal{G}$.*
- *If* Max Clique *is* NP*-hard on $\{G^s \mid G \in \mathcal{G}\}$, then* Max s-Club *is* NP*-hard on $\mathcal{G}$.*

3 Graph Classes That Have the Clique-Power Property

In this section we show that 4-chordal graphs and AT-free graphs have the clique-power property. We start with 4-chordal graphs, and we consider the cases $s = 2$ and $s \geq 3$ separately in the next two lemmas.

Lemma 1. *4-Chordal graphs have the 2-clique-power property.*

Lemma 2. *4-Chordal graphs have the s-clique-power property for every $s \geq 3$.*

Proof. To obtain a contradiction, assume that there is an integer $s \geq 3$ and a 4-chordal graph G such that G^s has a maximal clique S, but S is not an s-club in G. Let $u, v \in S$ be vertices at distance at least $s+1$ in $G[S]$. In particular, u, v are not adjacent in G. Since $u, v \in S$, $\text{dist}_G(u, v) \leq s$. Any shortest (u, v)-path in G has at least one vertex that is at distance at least $s+1$ from some vertex of S; otherwise all inner vertices of some (u, v)-path of length at most s would belong to maximal clique S in G^s, and u, v would be at distance at most s in $G[S]$. Consider a shortest (u, v)-path P in G that has the minimum number of vertices at distance at least $s+1$ from some vertex of S. Let x be an inner vertex of P at distance at least $s+1$ from a vertex $w \in S$ in G. Denote by r and t the vertices adjacent to x in P. Since P is a shortest path, r and t are not adjacent. For every vertex $h \in N_G(r) \cap N_G(t)$, let P_h be the path obtained from P by replacing subpath rxt with rht. Observe that by the choice of P, for every $h \in N_G(r) \cap N_G(t)$, P_h is a shortest (u, v)-path, and h is at distance at least $s+1$ from some vertex of S. For every vertex $h \in N_G(r) \cap N_G(t)$, let

$$U_h = \{g \in S \setminus \{u, v\} \mid \text{dist}_G(g, h) = s-1 \text{ and } \text{dist}_G(g, r) = \text{dist}_G(g, t) = s\}.$$

We may assume that $|U_x| = \max\{|U_h| \mid h \in N_G(r) \cap N_G(t)\}$, because otherwise we can replace rxt with rht in P. Notice that the set U_x might be empty.

Let Q_1 be a shortest (u, w)-path in G, and let Q_2 be a shortest (v, w)-path in G. Note that the length of each of these paths is at most s. Since $\text{dist}_G(x, w) > s$, x is not in Q_1 or Q_2, and x is not adjacent to w or any inner vertex of Q_1 or Q_2. Let X be the union of the vertices belonging to P, Q_1, and Q_2. Observe that $G[X]$ contains an induced cycle C that includes vertices x, r, t and edges xr, xt, because $G[X \setminus \{x\}]$ is connected by our construction. Since r, t are not adjacent and x is not adjacent to any vertex of $X \setminus \{r, t\}$, it follows that C has at least four vertices. Since G is 4-chordal, C has exactly four vertices. Let y be a vertex of C different from r, x, t. It follows that $\text{dist}_G(w, y) \leq s-1$, $\text{dist}_G(w, r) \leq s$, and $\text{dist}_G(w, t) \leq s$. As $\text{dist}_G(w, x) > s$, we conclude that $\text{dist}_G(w, y) = s-1$, and $\text{dist}_G(w, r) = \text{dist}_G(w, t) = s$.

Denote by Q a shortest (w, y)-path in G, and observe that Q has length $s-1$. Vertex y belongs to $N_G(r) \cap N_G(t)$. Notice that $w \in U_y$, and thus $U_y \neq \emptyset$. Since $|U_x| \geq |U_y|$ by our construction, and $w \notin U_x$, there is a vertex $z \in U_x \setminus U_y$. By the definition of U_x, $\text{dist}_G(z, r) = \text{dist}_G(z, t) = s$. Since $z \notin U_y$, $\text{dist}_G(z, y) \neq s-1$. The assumption that $\text{dist}_G(z, y) \leq s-2$ immediately implies that $\text{dist}_G(z, r) \leq s-1$, which gives a contradiction. Hence, $\text{dist}_G(z, y) \geq s$. Denote by R a shortest (z, x)-path in G, and note that R has length $s-1$ by the definition of U_x. Notice that r or t does not belong to Q or R. Furthermore, r or t is not adjacent to any vertex of Q or R, except y and x.

We claim that Q and R have no common vertex and there is no edge between a vertex of Q and a vertex of R. For contradiction, assume first that Q and R have a common vertex h. Let R' be the (h, x)-subpath of R, and let Q' be the (h, y)-subpath of Q. Denote by ℓ_1 the length of R' and by ℓ_2 the length of Q', and assume that $\ell_1 \leq \ell_2$. Then consider the following path from w to x: first take the (w, h)-subpath of Q and then from h to x the (h, x)-subpath of R. It follows that the length of this path is at most the length of Q, i.e., $s - 1$, which contradicts that $\mathrm{dist}_G(w, x) > s$. Hence if Q and R have a common vertex h then $\ell_1 > \ell_2$. Now consider the following path from z to y: first take the (z, h)-subpath of R and then the (h, y)-subpath of Q, which has length at most $s - 1$. This implies $\mathrm{dist}_G(z, y) \leq s - 1$, which contradicts our previous conclusion that $\mathrm{dist}_G(z, y) \geq s$. Consequently P and Q cannot have a common vertex. Suppose now that G has an edge $z'w'$ where z' is in R and w' is in Q. We choose $z'w'$ in such a way that the distance between x and z' in R is minimum. Recall that $xy \notin E$. If $z'y \in E$, then $\mathrm{dist}_G(z, y) \leq s - 1$. Hence, $z'y \notin E$. By the same arguments, $xw' \notin E$. Then the concatenation of xry, the (y, w')-subpath of Q, $w'z'$, and the (z', x)-subpath of R is an induced cycle on at least 5 vertices, contradicting that G is 4-chordal. We conclude that Q and R have neither common vertices nor adjacent vertices.

Since $z, w \in S$, we know that $\mathrm{dist}_G(z, w) \leq s$. Let F be a shortest (z, w)-path in G. We claim that vertices x, y, r do not belong to F, and neither x nor r is adjacent to any vertex of F. If x is in F, then the (z, x)-subpath of F has length at least the length of R, namely $s - 1$. But since $xw \notin E$, this contradicts that $\mathrm{dist}_G(z, w) \leq s$. Symmetrically, we observe that y is not in F either. If r is in F, since $\mathrm{dist}_G(z, r) = s$, then the (z, r)-subpath of F has length at least s, which contradicts either $\mathrm{dist}_G(z, w) \leq s$ or F is a shortest (z, w)-path. Now assume that x is adjacent to some vertex h of F. Since $\mathrm{dist}_G(w, x) \geq s + 1 \geq 4$, the (z, h)-subpath of F has length at most $s-3$, but then $\mathrm{dist}_G(z, x) \leq s-2$; again a contradiction. Let r be adjacent to a vertex h of F. Then because $\mathrm{dist}_G(r, w) = s$, the (w, h)-subpath of F has length at least $s-1$, but then $\mathrm{dist}_G(r, z) \leq 2 < s$, and we again obtain a contradiction. To complete the proof, it remains to notice that the union of the vertices of Q, R, and F, together with r, induces a subgraph of G with an induced cycle on at least 5 vertices, but this contradicts our assumption that G is 4-chordal. □

Theorem 1. *4-Chordal graphs have the clique-power property.*

Theorem 1 immediately follows from Lemmas 1 and 2. The example shown in Fig. 2 b) shows that this result is tight in the sense that 5-chordal graphs do not have the clique-power property.

Now we turn to AT-free graphs, and we show that they also have the clique-power property. In the following proof, we use additional terminology: Let u, v be vertices of G, and let P be a (u, v)-path in G. We say that P *sees* a vertex x of G if x belongs to P or x is adjacent to an inner vertex of P.

Theorem 2. *AT-free graphs have the clique-power property.*

Proof. To obtain a contradiction, assume that there is an integer $s \geq 2$ and an AT-free graph $G = (V, E)$, such that G^s has a maximal clique S, but S is not an s-club in G. Let $u, v \in S$ be vertices at distance at least $s + 1$ in $G[S]$. In particular, u and v are not adjacent in G.

Since $u, v \in S$, G has a (u, v)-path of length at most s. Let P be a (u, v)-path of length at most s in G that sees the maximum number of pairwise non-adjacent vertices of S. Let $U \subseteq S$ be a maximum size set of pairwise non-adjacent vertices of S containing u and v that are seen by P. The path P has an inner vertex x at distance at least $s+1$ from a vertex $w \in S$ in G. Otherwise, all inner vertices of P would be included in the maximal clique S in G^s, and u, v would be at distance at most s in $G[S]$. It follows that w is not adjacent to any vertex of U, because the distance between x and z in the subgraph of G induced by the vertices of P and U is at most $s - 1$ for any $z \in U$. Furthermore, by construction, P does not see w. Let u' be any vertex of U. Since $u', w \in S$, G has a (u', w)-path of length at most s. Let Q be a (u', w)-path of length at most s for any $u' \in U$ that sees the maximum number of vertices of U. By the choice of P, there is at least one vertex $r \in U$ such that Q does not see r. Let $W \subseteq U$ be the set of vertices of U seen by Q; hence $r \notin W$. Now there is a (r, w)-path R of length at most s in G, and by the choice of Q, there is at least one vertex $t \in W$ such that R does not see t. We claim that $\{r, t, w\}$ is an AT, contradicting that G is AT-free. Observe that $r, t \in U$, the vertices of P together with U induce a connected subgraph of G, and w is not adjacent to any vertex of P or U. Thus G has an (r, t)-path that does not contain a neighbor of w. Similarly, G has a (t, w)-path each of whose vertices belongs to Q or W, that does not contain a neighbor of r. Finally, R is an (r, w)-path that contains no neighbor of t. □

4 Algorithmic Consequences

In this section we obtain tractability and intractability results by combining the results of Section 3 with Proposition 1.

4.1 Polynomial Cases

Max s-Club has been studied on chordal graphs by Asahiro et al. [4]. Their results assume that chordal graphs have the clique-power property, but the property is neither stated nor proved in [4]. Balakrishnan and Paulraja [5,6] (see also [2]) showed that odd powers of chordal graphs are chordal. Consequently, Proposition 1 and Theorem 1 immediately imply that Max s-Club can be solved in polynomial time on chordal graphs. We can now generalize this result to weakly chordal graphs using Theorem 1, since weakly chordal graphs are 4-chordal. Brandstädt et al. [10] proved that odd powers of weakly chordal graphs are weakly chordal. Hayward et al. [21,22] showed that Max Clique can be solved in time $O(nm)$ on weakly chordal graphs. As a consequence of these results, Proposition 1, and Theorem 1, we obtain the following result.

Theorem 3. Max s-Club *can be solved in time $O(n^3)$ on weakly chordal graphs, for all positive odd input integers s.*

Recall that for every even integer s, Max s-Club is NP-hard on chordal graphs [4], and thus also on weakly chordal graphs and on 4-chordal graphs. For the strongly chordal subclass of chordal graphs we are able to show polynomial-time solvability for all values of s. Lubiw [24] showed that any power of a strongly chordal graph is strongly chordal. With this result, Proposition 1 and Theorem 1 immediately give the following.

Theorem 4. Max s-Club *can be solved in time $O(n^3)$ on strongly chordal graphs, for all positive input integers s.*

We move to distance hereditary graphs. Recall that they are 4-chordal, and consequently we can apply Theorem 1. Bandelt et al. [8] proved that even powers of distance hereditary graphs are chordal, and thus weakly chordal. It also follows from their results that odd powers of distance hereditary graphs are weakly chordal. Combining this with Proposition 1 and Theorem 1, we obtain the following result.

Theorem 5. Max s-Club *can be solved in time $O(n^3)$ on distance hereditary graphs, for all positive input integers s.*

Notice that if s is a fixed integer and not a part of the input, then the problem can be solved in linear time on distance hereditary graphs [29] because these graphs have clique-width at most 3 [19].

Chordal bipartite graphs form another subclass of 4-chordal graphs and of weakly chordal graphs. By Theorem 3, Max s-Club can be solved in polynomial time on chordal bipartite graphs, for odd values of s. For even values of s, s-th powers of chordal bipartite graphs are not necessary weakly chordal. We mention that they are not even perfect. A graph is *perfect* if neither the graph nor its complement contains an induced cycle of odd length [14]. Perfect graphs form a superclass of weakly chordal graphs, and Max Clique is solvable in polynomial time on them. Unfortunately, we cannot use this, due to the above. Still, we are able to solve Max s-Club on chordal bipartite graphs in polynomial time for even s using the following structural result that we find interesting also on its own.

Lemma 3. *Let $G = (X, Y, E)$ be a chordal bipartite graph and let s be any positive integer. Then $G^s[X]$ and $G^s[Y]$ are chordal graphs.*

Proof. By symmetry, it is sufficient to prove the lemma for $G^s[X]$. Since the lemma is trivially true for $s = 1$, let us assume that $s \geq 2$ for the rest of the proof. For contradiction suppose there is an induced cycle $C = x_0, x_1, \ldots, x_{k-1}, x_0$ of length at least 4 in $G^s[X]$. For ease of notation, let $x_k = x_0$, and read all indices modulo k throughout the proof. It follows that for every i between 0 and $k-1$, there is a shortest (x_i, x_{i+1})-path P_i of length at most s in G.

For every i, we first show that there is no edge $xy \in E$ with $x \in P_i$ and $y \in P_j$, for $j \notin \{i-1, i, i+1\}$. For contradiction suppose there is such an edge xy for some i. Without loss of generality we may assume that $k > j > i+1$. Since C is an induced cycle, $x_i x_j$ is not an edge of G^s and $x_{i+1} x_{j+1}$ is not an edge of G^s. Let P be the path obtained by the concatenation of the (x_i, x)-subpath of P_i and edge xy and the (y, x_j)-subpath of P_j. Similarly, let Q be the path obtained by the concatenation of the (x_{i+1}, x)-subpath P_i and edge xy and the (y, x_{j+1})-subpath of P_j. Observe that both P and Q have length at least $s+1$, because otherwise $x_i x_j$ or $x_{i+1}x_{j+1}$ would be an edge of G^s. Let ℓ_1 be the length of the (x_i, x)-subpath of P, and let ℓ_2 be the length of the (y, x_j)-subpath of P. Thus the length of P is $\ell_1 + \ell_2 + 1$, and consequently $\ell_1 + \ell_2 \geq s$. Observe that the length of Q is at most $2s - \ell_1 - \ell_2 + 1$, and since the length of Q is at least $s+1$, we have $2s - \ell_1 - \ell_2 \geq s$. Combining this inequality with $\ell_1 + \ell_2 \geq s$, we conclude that $\ell_1 + \ell_2 = s$. This implies that both P and Q have length exactly $s+1$. We can further conclude that both P_i and P_j have length exactly s. However, since $x_i, x_{i+1}, x_j, x_{j+1}$ are all in X, and G is a bipartite graph, there cannot be paths of length both s and $s+1$ between pairs of these vertices in G, which gives the desired contradiction.

The above also implies that P_i and P_j do not have common vertices for $j \notin \{i-1, i, i+1\}$, since this would imply an edge between a vertex of P_i and a vertex of P_j, under the above assumptions.

As a consequence of the above, if there is an edge in G between a vertex of P_i and a vertex of P_j, then we can assume that $j = i+1$. Observe that there is always an edge between every pair of consecutive paths P_i and P_{i+1}, and they might also share some vertices. For every i, we will call an edge $xy \in E$ with $x \in P_i$ and $y \in P_{i+1}$ a *long chord with respect to* x if there is no other vertex y' in the (y, x_{i+2})-subpath of P_{i+1} such that $xy' \in E$. Observe that x_i is not adjacent to any vertex of P_{i+1}, since C is an induced cycle in G^s and thus $x_i x_{i+2}$ is not an edge in G^s. We will now follow the long chords between consecutive paths, and construct an induced cycle C' in G as follows. Start with any vertex $x \in P_1$. Pick a vertex y in P_0 such that $yx \in E$ and the (y, x_1)-subpath of P_0 is longest. We traverse P_1 from x to x_2, and as soon as we come to a vertex that is adjacent to a vertex in P_2, we take the first long chord and go to P_2. For each i from 2 to $k-1$, we continue in this manner from P_i to P_{i+1}: once we are on P_i we continue on P_i towards x_{i+1} and we take the first long chord to P_{i+1}. At the end once we are in P_{k-1}, we take the first edge $y'y''$ such that $y' \in P_{k-1}$ and $y'' \in P_0$ and y'' is not an inner vertex of the (y, x_1)-subpath of P_0. If y' has other neighbors that are on the (y'', y)-subpath of P_0 then we take as y'' such a neighbor that is closest to y. We continue from y'' to y on P_0 and use the edge yx to close the cycle. Observe that, by the choices we made, the (y'', y)-subpath of P_0 is the only portion on P_0 that contributes to C', and no vertex on this subpath is adjacent to y' or x, except y'' that is adjacent to y', and y that is adjacent to x. All other edges of C' are long chords or portions of P_i that do not contain any neighbor of P_{i+1}, and hence C' is an induced cycle. Since there is no induced cycle of length more than 4 in G, and C' contains distinct vertices from each P_i for $0 \leq i \leq k-1$, we conclude that $k = 4$. Consequently, C' consists of y_0, y_1, y_2, y_3, such that $y_i \in P_i$ and $y_i y_{i+1} \in E$, for $i = 0, 1, 2, 3$. Let ℓ_i be the length of the (x_i, y_i)-subpath of P_i, and let ℓ_i' be the length of the (y_i, x_{i+1})-subpath of P_i, for $i = 0, 1, 2, 3$. Since C is an induced cycle in G^s, $x_i x_{i+2}$ is not an edge of G^s, and we can conclude that $\ell_i + \ell'_i \leq s$ and $\ell_i + \ell'_{i+1} \geq s$, for $i = 0, 1, 2, 3$. Adding up all four pairs of inequalities, we obtain that $\ell_i + \ell'_i = s$ and $\ell_i + \ell'_{i+1} = s$, for $i = 0, 1, 2, 3$. Consequently, we have a path between x_i and x_{i+2} of length $s + 1$ using the (x_i, y_i)-subpath of P_i, the edge $y_i y_{i+1}$, and the (y_{i+1}, x_{i+2})-subpath of P_{i+1}, whereas the length of P_i is s. Since $x_0, \ldots x_{k-1} \in X$ and G is bipartite, we cannot have paths of length both s and $s + 1$ between pairs of them. Therefore our initial assumption that $G^s[X]$ contained an induced cycle of length at least 4 is wrong, and $G^s[X]$ is chordal. □

The following theorem now follows from Lemma 3.

Theorem 6. MAX s-CLUB *can be solved in time $O(n^4)$ on chordal bipartite graphs, for all positive input integers s.*

Finally we move to AT-free graphs. MAX CLIQUE is NP-hard on AT-free graphs [28], and hence MAX 1-CLUB is NP-hard on AT-free graphs. Chang et al. [12] showed that for every $s \geq 2$ and every AT-free graph G, G^s is a cocomparability graph. Cocomparability graphs form a subclass of AT-free graphs. Fortunately MAX CLIQUE can be solved in polynomial time on cocomparability graphs [18]. This, combined with Proposition 1 and Theorem 2, gives the next result.

Theorem 7. MAX s-CLUB *can be solved in time $O(n^3)$ on AT-free graphs, for all positive input integers $s \geq 2$.*

4.2 Hardness on 4-Chordal Graphs

In Section 4.1 we proved that MAX s-CLUB can be solved in polynomial time on several subclasses of 4-chordal graphs. Here we complement these results by showing that the problem is NP-hard on 4-chordal graphs. By Proposition 1, it is sufficient to show that MAX CLIQUE is NP-hard on powers of 4-chordal graphs.

A *2-subdivision* of a graph is obtained by replacing every edge with a path of length three. A graph is a *2-subdivision* if it is a 2-subdivision of some graph. Given a graph G and an integer k, the decision problem CLIQUE asks whether G has a clique of size at least k, and the decision problem INDEPENDENT SET whether G has an independent set of size at least k. Clearly, X is a clique in G if and only if X is an independent set in $\overline{G}$. We use this duality to obtain the following lemma.

Lemma 4. CLIQUE *is* NP-*complete on* 4-*chordal graphs of diameter at most 2.*

Theorem 8. CLIQUE *is* NP-*complete on* $\{G^s \mid G$ *is* 4-*chordal*$\}$*, for every positive integer s.*

Proof. Asahiro et al. [4] proved that CLIQUE is NP-complete on even powers of chordal graphs, and consequently on even powers of 4-chordal graphs. For $s = 1$, the statement of Theorem 8 follows immediately from Lemma 4. Hence, it is sufficient to prove the theorem for odd $s > 1$. Let $s = 2r + 1$ for $r \geq 1$. We give a reduction from Clique on 4-chordal graphs of diameter at most 2, which is NP-complete by Lemma 4.

Let $G = (V, E)$ be a 4-chordal graph of diameter at most 2, which is input to Clique together with an integer k. Let $V = \{u_1, \ldots, u_n\}$, and let H be the graph obtained from G as follows: for each $i \in \{1, \ldots, n\}$, we add r new vertices $v_i^1, \ldots, v_i^r$ and r new edges $u_i v_i^1$, $v_i^1 v_i^2$, …, $v_i^{r-1} v_i^r$ to G. In other words, we attach a path $v_i^1, v_i^2; \ldots, v_i^r$ to every vertex u_i of G, via edge $u_i v_i^1$. Let us denote by U the set of vertices $\{u_1, \ldots, u_n\} \cup (\cup_{i=1}^{r-1}\{v_1^i, \ldots v_n^i\})$. Since $\mathrm{diam}(G) \leq 2$ and $s = 2r + 1$, we can observe the following:

- U is a clique in H^s,
- for every $i \in \{1, \ldots n\}$, v_i^r is adjacent to every vertex of U in H^s,
- for every pair $i, j \in \{1, \ldots, n\}$, v_i^r and v_j^r are adjacent in H^s is and only if u_i and u_j are adjacent in G.

Consequently, every maximal clique in H^s contains U as a subset. Furthermore, any set $\{u_{i_1}, \ldots, u_{i_k}\}$ of k vertices in G is a clique of G if and only if $\{v_{i_1}^r, \ldots, v_{i_k}^r\} \cup U$ is a clique in H^s. Since $|U| = rn$, we conclude that G has a clique of size at least k if and only if H^s has a clique of size at least $k + rn$, which completes the reduction. $\square$

Theorem 8 and Proposition 1 immediately give the following result.

Theorem 9. Max s-Club *is* NP*-hard on* 4*-chordal graphs, for every positive integer* s*.*

References

1. Abello, J., Pardalos, P., Resende, M.: On maximum clique problems in very large graphs. In: External memory algorithms and visualization. DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 50, pp. 119–130. AMS (1999)
2. Agnarsson, G., Greenlaw, R., Halldórsson, M.M.: On powers of chordal graphs and their colorings. In: Proceedings of SICCGTC 2000, vol. 144, pp. 41–65 (2000)
3. Alba, R.: A graph-theoretic definition of a sociometric clique. Journal of Mathematical Sociology 3, 113–126 (1973)
4. Asahiro, Y., Miyano, E., Samizo, K.: Approximating maximum diameter-bounded subgraphs. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 615–626. Springer, Heidelberg (2010)
5. Balakrishnan, R., Paulraja, P.: Correction to: "Graphs whose squares are chordal". Indian J. Pure Appl. Math. 12(8), 1062 (1981)
6. Balakrishnan, R., Paulraja, P.: Graphs whose squares are chordal. Indian J. Pure Appl. Math. 12(2), 193–194 (1981)
7. Balasundaram, B., Butenko, S., Trukhanov, S.: Novel approaches for analyzing biological networks. Journal of Combinatorial Optimization 10, 23–39 (2005)
8. Bandelt, H.J., Henkmann, A., Nicolai, F.: Powers of distance-hereditary graphs. Discrete Mathematics 145(1-3), 37–60 (1995)
9. Brandstädt, A., Le, V., Spinrad, J.: Graph Classes: A Survey. SIAM Monographs on Discrete Mathematics and Applications (1999)

10. Brandstädt, A., Dragan, F.F., Xiang, Y., Yan, C.: Generalized powers of graphs and their algorithmic use. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 423–434. Springer, Heidelberg (2006)
11. Chandran, L.S., Mathew, R.: Bipartite powers of k-chordal graphs. arXiv:1108.0277 [math.CO] (2012)
12. Chang, J.M., Ho, C.W., Ko, M.T.: Powers of asteroidal triple-free graphs with applications. Ars Comb. 67 (2003)
13. Chang, M.S., Hung, L.J., Lin, C.R., Su, P.C.: Finding large k-clubs in undirected graphs. In: Proceedings of the 28th Workshop on Combinatorial Mathematics and Computation Theory, pp. 1–10 (2011)
14. Chudnovsky, M., Robertson, N., Seymour, P., Thomas, R.: The strong perfect graph theorem. Ann. of Math (2) 164(1), 51–229 (2006)
15. Corneil, D., Perl, Y.: Clustering and domination in perfect graphs. Discrete Applied Mathematics 9, 27–39 (1984)
16. Dawande, M., Swaminathan, J., Keskinocak, P., Tayur, S.: On bipartite and multipartite clique problems. Journal of Algorithms 41, 388–403 (2001)
17. Diestel, R.: Graph theory, 4th edn. Graduate Texts in Mathematics, vol. 173. Springer, Heidelberg (2010)
18. Golumbic, M.C.: Algorithmic graph theory and perfect graphs, 2nd edn. Annals of Discrete Mathematics, vol. 57. Elsevier (2004)
19. Golumbic, M.C., Rotics, U.: On the clique-width of some perfect graph classes. Int. J. Found. Comput. Sci. 11(3), 423–443 (2000)
20. Hartung, S., Komusiewicz, C., Nichterlein, A.: Parameterized algorithmics and computational experiments for finding 2-clubs. In: Thilikos, D.M., Woeginger, G.J. (eds.) IPEC 2012. LNCS, vol. 7535, pp. 231–241. Springer, Heidelberg (2012)
21. Hayward, R., Hoàng, C.T., Maffray, F.: Optimizing weakly triangulated graphs. Graphs and Combinatorics 5(1), 339–349 (1989)
22. Hayward, R., Spinrad, J., Sritharan, R.: Weakly chordal graph algorithms via handles. In: Proceedings of SODA 2000, pp. 42–49 (2000)
23. Kloks, T., Kratsch, D.: Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph. Information Processing Letters 55 (1995)
24. Lubiw, A.: Γ-free matrices. Master thesis, Department of Combinatorics and Optimization, University of Waterloo (1982)
25. Luce, R.: Connectivity and generalized cliques in sociometric group structure. Psychometrika 15, 169–190 (1950)
26. Mokken, R.: Cliques, clubs and clans. Quality and Quantity 13, 161–173 (1979)
27. Pajouh, F.M., Balasundaram, B.: On inclusionwise maximal and maximum cardinality k-clubs in graphs. Discrete Optimization 9(2), 84–97 (2012)
28. Poljak, S.: A note on stable sets and colorings of graphs. Comment. Math. Univ. Carolinae 15, 307–309 (1974)
29. Schäfer, A.: Exact algorithms for s-club finding and related problems (2009), diplomarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena
30. Schäfer, A., Komusiewicz, C., Moser, H., Niedermeier, R.: Parameterized computational complexity of finding small-diameter subgraphs. Optimization Letters 6(5), 883–891 (2012)
31. Seidman, S.B., Foster, B.L.: A graph theoretic generalization of the clique concept. Journal of Mathematical Sociology 6, 139–154 (1978)
32. Tarjan, R.E., Yannakakis, M.: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. SIAM J. Comput. 13(3), 566–579 (1984)

List Coloring in the Absence of Two Subgraphs[⋆]

Petr A. Golovach[1] and Daniël Paulusma[2]

[1] Department of Informatics, University of Bergen, Norway
petr.golovach@ii.uib.no

[2] School of Engineering and Computing Sciences, Durham University
daniel.paulusma@durham.ac.uk

Abstract. A list assignment of a graph $G = (V, E)$ is a function $\mathcal{L}$ that assigns a list $L(u)$ of so-called admissible colors to each $u \in V$. The List Coloring problem is that of testing whether a given graph $G = (V, E)$ has a coloring c that respects a given list assignment $\mathcal{L}$, i.e., whether G has a mapping $c : V \to \{1, 2, \ldots\}$ such that (i) $c(u) \neq c(v)$ whenever $uv \in E$ and (ii) $c(u) \in L(u)$ for all $u \in V$. If a graph G has no induced subgraph isomorphic to some graph of a pair $\{H_1, H_2\}$, then G is called (H_1, H_2)-free. We completely characterize the complexity of List Coloring for (H_1, H_2)-free graphs.

1 Introduction

Graph coloring involves the labeling of the vertices of some given graph by integers called colors such that no two adjacent vertices receive the same color. The goal is to minimize the number of colors. Graph coloring is one of the most fundamental concepts in both structural and algorithmic graph theory and arises in a vast number of theoretical and practical applications. Many variants are known, and due to its hardness, the graph coloring problem has been well studied for special graph classes such as those defined by one or more forbidden induced subgraphs. We consider a more general version of graph coloring called list coloring and classify the complexity of this problem for graphs characterized by two forbidden induced subgraphs. Kratsch and Schweitzer [22] and Lozin [23] performed a similar study as ours for the problems graph isomorphism and dominating set, respectively. Before we summarize related coloring results and explain our new results, we first state the necessary terminology. For a more general overview of the area we refer to the surveys of Randerath and Schiermeyer [29] and Tuza [32], and to the book by Jensen and Toft [26].

1.1 Terminology

We only consider finite undirected graphs with no multiple edges and self-loops. A *coloring* of a graph $G = (V, E)$ is a mapping $c : V \to \{1, 2, \ldots\}$ such that $c(u) \neq c(v)$ whenever $uv \in E$. We call $c(u)$ the *color* of u. A *k-coloring* of G is

[⋆] This paper was supported by EPSRC (EP/G043434/1) and ERC (267959).

a coloring c of G with $1 \leq c(u) \leq k$ for all $u \in V$. The COLORING problem is that of testing whether a given graph admits a k-coloring for some given integer k. If k is *fixed*, i.e., not part of the input, then we denote the problem as k-COLORING. A *list assignment* of a graph $G = (V, E)$ is a function $\mathcal{L}$ that assigns a list $L(u)$ of so-called *admissible* colors to each $u \in V$. If $L(u) \subseteq \{1, \ldots, k\}$ for each $u \in V$, then $\mathcal{L}$ is also called a *k-list assignment*. We say that a coloring $c\colon\ V \to\ \{1, 2, \ldots\}$ *respects* $\mathcal{L}$ if $c(u) \in L(u)$ for all $u \in V$. The LIST COLORING problem is that of testing whether a given graph has a coloring that respects some given list assignment. For a fixed integer k, the LIST k-COLORING problem has as input a graph G with a k-list assignment $\mathcal{L}$ and asks whether G has a coloring that respects $\mathcal{L}$. The *size* of a list assignment $\mathcal{L}$ is the maximum list size $|L(u)|$ over all vertices $u \in V$. For a fixed integer ℓ, the ℓ-LIST COLORING problem has as input a graph G with a list assignment $\mathcal{L}$ of size at most ℓ and asks whether G has a coloring that respects $\mathcal{L}$. Note that k-COLORING can be viewed as a special case of LIST k-COLORING by choosing $L(u) = \{1, \ldots, k\}$ for all vertices u of the input graph, whereas LIST k-COLORING is readily seen to be a special case of k-LIST COLORING.

For a subset $S \subseteq V(G)$, we let $G[S]$ denote the *induced* subgraph of G, i.e., the graph with vertex set S and edge set $\{uv \in E(G) \mid u, v \in S\}$. For a graph F, we write $F \subseteq_i G$ to denote that F is an induced subgraph of G. Let G be a graph and $\{H_1, \ldots, H_p\}$ be a set of graphs. We say that G is *$(H_1, \ldots, H_p)$-free* if G has no induced subgraph isomorphic to a graph in $\{H_1, \ldots, H_p\}$; if $p = 1$, we may write H_1-free instead of (H_1)-free.

The *complement* of a graph $G = (V, E)$ denoted by $\overline{G}$ has vertex set V and an edge between two distinct vertices if and only if these vertices are not adjacent in G. The *union* of two graphs G and H is the graph with vertex set $V(G) \cup V(H)$ and edge set $E(G) \cup E(H)$. Note that G and H may share some vertices. If $V(G) \cap V(H) = \emptyset$, then we speak of the *disjoint union* of G and H denoted by $G + H$. We denote the disjoint union of r copies of G by rG. The graphs C_r, P_r and K_r denote the cycle, path, and complete graph on r vertices, respectively. The graph $K_{r,s}$ denotes the complete bipartite graph with partition classes of size r and s, respectively. The graph K_4^- denotes the *diamond*, which is the complete graph on four vertices minus an edge. The *line graph* of a graph G with edges $e_1, \ldots, e_p$ is the graph with vertices $u_1, \ldots, u_p$ such that there is an edge between any two vertices u_i and u_j if and only if e_i and e_j share an end-vertex in G.

1.2 Related Work

Kráľ et. al. [20] completely determined the computational complexity of COLORING for graph classes characterized by one forbidden induced subgraph. By combining a number of known results, Golovach, Paulusma and Song [13] obtained similar dichotomy results for the problems LIST COLORING and k-LIST COLORING, whereas the complexity classifications of the problems LIST k-COLORING and k-COLORING are still open (see, e.g., [14] for a survey).

Theorem 1. *Let H be a fixed graph. Then the following three statements hold:*

(i) Coloring *is polynomial-time solvable for H-free graphs if H is an induced subgraph of P_4 or of P_1+P_3; otherwise it is* NP*-complete for H-free graphs.*

(ii) List Coloring *is polynomial-time solvable for H-free graphs if H is an induced subgraph of P_3; otherwise it is* NP*-complete for H-free graphs.*

*(iii) For all $\ell \leq 2$, ℓ-*List Coloring *is polynomial-time solvable. For all $\ell \geq 3$, ℓ-*List Coloring *is polynomial-time solvable for H-free graphs if H is an induced subgraph of P_3; otherwise it is* NP*-complete for H-free graphs.*

When we forbid *two* induced subgraphs the situation becomes less clear for the Coloring problem, and only partial results are known. We summarize these results in the following theorem. Here, C_3^+ denotes the graph with vertices a, b, c, d and edges ab, ac, ad, bc, whereas F_5 denote the 5-vertex fan also called the gem, which is the graph with vertices a, b, c, d, e and edges $ab, bc, cd, ea, eb, ec, ed$.

Theorem 2. *Let H_1 and H_2 be two fixed graphs. Then the following holds:*

(i) Coloring *is* NP*-complete for (H_1, H_2)-free graphs if*

1. *$H_1 \supseteq_i C_r$ for some $r \geq 3$ and $H_2 \supseteq_i C_s$ for some $s \geq 3$*
2. *$H_1 \supseteq_i K_{1,3}$ and $H_2 \supseteq_i K_{1,3}$*
3. *H_1 and H_2 contain a spanning subgraph of $2P_2$ as an induced subgraph*
4. *$H_1 \supseteq_i C_3$ and $H_2 \supseteq_i K_{1,r}$ for some $r \geq 5$*
5. *$H_1 \supseteq_i C_3$ and $H_2 \supseteq_i P_{164}$*
6. *$H_1 \supseteq_i C_r$ for $r \geq 4$ and $H_2 \supseteq_i K_{1,3}$*
7. *$H_1 \supseteq_i C_r$ for $r \geq 5$ and H_2 contains a spanning subgraph of $2P_2$ as an induced subgraph*
8. *$H_1 \supseteq_i K_4$ or $H_1 \supseteq_i K_4^-$, and $H_2 \supseteq_i K_{1,3}$*
9. *$H_1 \supseteq_i C_r + P_1$ for $3 \leq r \leq 4$ or $H_1 \supseteq_i \overline{C_r}$ for $r \geq 6$, and H_2 contains a spanning subgraph of $2P_2$ as an induced subgraph.*

(ii) Coloring *is polynomial-time solvable for (H_1, H_2)-free graphs if*

1. *H_1 or H_2 is an induced subgraph of $P_1 + P_3$ or of P_4*
2. *$H_1 \subseteq_i C_3 + P_1$ and $H_2 \subseteq_i K_{1,3}$*
3. *$H_1 \subseteq_i C_3^+$ and $H_2 \neq K_{1,5}$ is a forest on at most six vertices*
4. *$H_1 \subseteq_i C_3^+$, and $H_2 \subseteq_i sP_2$ or $H_2 \subseteq_i sP_1 + P_5$ for $s \geq 1$*
5. *$H_1 = K_r$ for $r \geq 4$, and $H_2 \subseteq_i sP_2$ or $H_2 \subseteq_i sP_1 + P_5$ for $s \geq 1$*
6. *$H_1 \subseteq_i \overline{F_5}$, and $H_2 \subseteq_i P_1 + P_4$ or $H_2 \subseteq_i P_5$*
7. *$H_1 \subseteq_i \overline{P_5}$, and $H_2 \subseteq_i P_1 + P_4$ or $H_2 \subseteq_i 2P_2$.*

Proof. Král' et al. [20] proved Cases (i):1–4, 6–8. Golovach et al. [12] proved that 4-Coloring is NP-complete for (C_3, P_{164})-free graphs; this shows Case (i):5. Case (i):9 follows from the following result by Schindl [31]. For $1 \leq i \leq j \leq k$, let $S_{h,i,j}$ be the tree with only one vertex x of degree 3 that has exactly three leaves, which are of distance h, i and j to x, respectively. Let $A_{h,i,j}$ be the line graph of $S_{h,i,j}$. Then, for a finite set of graphs $\{H_1, \ldots, H_p\}$, Coloring is

NP-complete for $(H_1, \ldots, H_p)$-free graphs if the complement of each H_i has a connected component isomorphic neither to any graph $A_{i,j,k}$ nor to any path P_r.

Case (ii):1 follows from Theorem 1 (i). Because Coloring can be solved in polynomial time on graphs of bounded clique-width [19], and $(C_3 + P_1, K_{1,3})$-free graphs [2], $(F_5, P_1 + P_4)$-free graphs [4], (F_5, P_5)-free graphs [3] and $(\overline{P_5}, P_1 + P_4)$-free graphs [3] have bounded clique-width, Cases (ii):2,6–7 hold after observing in addition that $(\overline{P_5}, 2P_2)$-free graphs are b-perfect and Coloring is polynomial-time solvable on b-perfect graphs [16]. Gyárfás [15] showed that for all $r, t \geq 1$, (K_r, P_t)-free graphs can be colored with at most $(t-1)^{r-2}$ colors. Hence, Coloring is polynomial-time solvable on (K_r, F)-free graphs for some linear forest F if k-Coloring is polynomial-time solvable on F-free graphs for all $k \geq 1$. The latter is true for $F = sP_1 + P_5$ [7] and $F = sP_2$ (see e.g. [9]). This shows Case (ii):5, whereas we obtain Case (ii):4 by using the same arguments together with a result of Král' et al. [20], who showed that for any fixed graph H_2, Coloring is polynomial-time solvable on (C_3, H_2)-free graphs if and only if it is so for (C_3^+, H_2)-free graphs. Case (ii):3 is showed by combining the latter result with corresponding results from Dabrowski et al. [9] for (C_3, H_2)-free graphs obtained by combining a number of new results with some known results [5,6,24,27,28]. □

1.3 Our Contribution

We completely classify the complexity of List Coloring and ℓ-List Coloring for (H_1, H_2)-free graphs. For the latter problem we may assume that $\ell \geq 3$ due to Theorem 1 (iii).

Theorem 3. *Let H_1 and H_2 be two fixed graphs. Then* List Coloring *is polynomial-time solvable for (H_1, H_2)-free graphs in the following cases:*

1. $H_1 \subseteq_i P_3$ *or* $H_2 \subseteq_i P_3$
2. $H_1 \subseteq_i C_3$ *and* $H_2 \subseteq_i K_{1,3}$
3. $H_1 = K_r$ *for some* $r \geq 3$ *and* $H_2 = sP_1$ *for some* $s \geq 3$.

In all other cases, even 3-List Coloring *is* NP*-complete for (H_1, H_2)-free graphs.*

We note that the classification in Theorem 3 differs from the partial classification in Theorem 2. For instance, Coloring is polynomial-time solvable on $(C_3, K_{1,4})$-free graphs, whereas 3-List Coloring is NP-complete for this graph class. We prove Theorem 3 in Section 2, whereas Section 3 contains some concluding remarks. There, amongst others, we give a complete classification of the computational complexity of List Coloring and List 3-Coloring when a set of (not necessarily induced) subgraphs is forbidden.

2 The Classification

A graph G is a *split* graph if its vertices can be partitioned into a clique and an independent set; if every vertex in the independent set is adjacent to every vertex in the clique, then G is a *complete* split graph. The graph $K_n - M$ denotes a *complete* *graph minus a matching* which is obtained from a complete graph K_n after removing the edges of some matching M. Equivalently, a graph G is a complete graph minus a matching if and only if G is $(3P_1, P_1 + P_2)$-free [13]. The complement of a bipartite graph is called a *cobipartite* graph. Let G be a connected bipartite graph with partition classes A and B. Then we call $\overline{G}$ a *matching-separated* cobipartite graph if the edges of $\overline{G}$ that are between vertices from A and B form a matching in $\overline{G}$. The *girth* of a graph G is the length of a shortest induced cycle in G.

For showing the NP-complete cases in Theorem 3 we consider a number of special graph classes in the following three lemmas.

Lemma 1. 3-List Coloring *is* NP*-complete for:*

(i) complete bipartite graphs
(ii) complete split graphs
(iii) (non-disjoint) unions of two complete graphs
(iv) complete graphs minus a matching

Proof. The proof of Theorem 4.5 in the paper by Jansen and Scheffler [18] is to show that List Coloring is NP-complete on P_4-free graphs but in fact shows that 3-List Coloring is NP-complete for complete bipartite graphs. This shows (i). In the proof of Theorem 2 in the paper by Golovach and Heggernes [10] a different NP-hardness reduction is given for showing that 3-List Coloring is NP-complete for complete bipartite graphs. In this reduction a complete bipartite graph is constructed with a list assignment that has the following property: all the lists of admissible colors of the vertices for one bipartition class are mutually disjoint. Hence, by adding all possible edges between the vertices in this class, one proves that 3-List Coloring is NP-complete for complete split graphs. This shows (ii). Golovach et al. [13] showed (iii). The proof of Theorem 11 in the paper by Jansen [17] is to show that List Coloring is NP-complete for unions of two complete graphs that are not disjoint unions, but in fact shows that 3-List Coloring is NP-complete for these graphs. This shows (iv). □

Lemma 2. 3-List Coloring *is* NP*-complete for matching-separated cobipartite graphs.*

Proof. NP-membership is clear. To show NP-hardness we reduce from Satisfiability. It is known (see e.g. [8]) that this problem remains NP-complete even if each clause contains either 2 or 3 literals and each variable is used in at most 3 clauses. Consider an instance of Satisfiability with n variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$ that satisfies these two additional conditions. Let $\phi = C_1 \wedge \ldots \wedge C_m$. We construct a graph G with a list assignment $\mathcal{L}$ as follows (see Fig. 1).

- For each $i \in \{1, \ldots, n\}$, add six vertices $x_i^1, x_i^2, x_i^3, y_i^1, y_i^2, y_i^3$, introduce six new colors $i_1, i_2, i_3, i'_1, i'_2, i'_3$, assign lists of admissible colors $\{i_1, i'_1\}$, $\{i_2, i'_2\}$, $\{i_3, i'_3\}$ to x_i^1, x_i^2, x_i^3, respectively, and $\{i_1, i'_2\}, \{i_2, i'_3\}, \{i_3, i'_1\}$ to y_i^1, y_i^2, y_i^3, respectively.
- Add edges between all vertices x_i^h, y_i^h to obtain a clique with $6n$ vertices.
- For $j = 1, \ldots, m$, add four vertices u_j^1, u_j^2, u_j^3, w_j, introduce three new colors j_1, j_2, j_3, assign the list of admissible colors $\{j_1, j_2, j_3\}$ to w_j, and if C_j contains exactly two literals, then assign the list $\{j_3\}$ to u_j^3.

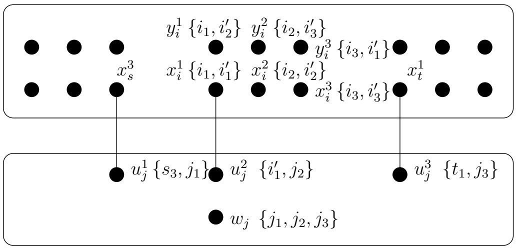


Fig. 1. An example of a graph G with a clause vertex $C_j = \overline{x}_s \vee x_i \vee \overline{x}_t$, where x_s, x_i, x_t occur for the third, first and first time in ϕ, respectively

- Add edges between all vertices u_j^h, w_j to obtain a clique with $4m$ vertices.
- For $j = 1, \dots, m$, consider the clause C_j and suppose that $C_j = z_1 \vee z_2$ or $C_j = z_1 \vee z_2 \vee z_3$. For $h = 1, 2, 3$ do as follows:
  - if $z_h = x_i$ is the p-th occurrence of the variable x_i in ϕ, then add the edge $u_j^h x_i^p$ and assign the list of colors $\{i_p', j_h\}$ to u_j^h;
  - if $z_h = \overline{x}_i$ is the p-th occurrence of the variable x_i in ϕ, then add the edge $u_j^h x_i^p$ and assign the list of colors $\{i_p, j_h\}$ to u_j^h.

  Notice that all the colors $i_1, i_2, i_3, i_1', i_2', i_3', j_1, j_2, j_3$ are distinct. From its construction, G is readily seen to be a matching-separated cobipartite graph.

We claim that ϕ has a satisfying truth assignment if and only if G has a coloring that respects $\mathcal{L}$. First suppose that ϕ has a satisfying truth assignment. For $i = 1, \dots, n$, we give the vertices x_i^1, x_i^2, x_i^3 colors i_1, i_2, i_3, respectively, and the vertices y_i^1, y_i^2, y_i^3 colors i_2', i_3', i_1' respectively, if $x_i = true$, and we give x_i^1, x_i^2, x_i^3 colors i_1', i_2', i_3', respectively, and y_i^1, y_i^2, y_i^3 colors i_1, i_2, i_3 respectively, if $x_i = false$. For $j = 1, \dots, m$, consider the clause C_j and suppose that $C_j = z_1 \vee z_2$ or $C_j = z_1 \vee z_2 \vee z_3$. Note that if C_j contains exactly two literals, then u_j^3 is colored by j_3. The clause C_j contains a literal $z_h = true$. Assume first that $z_h = x_i$ and that z_h is the p-th occurrence of the variable x_i in ϕ. Recall that u_j^h has list of admissible colors $\{i_p', j_h\}$ and that u_j^h is adjacent to x_i^p colored by i_p. Hence, we color u_j^h by i_p', w_j by j_h, and for $s \in \{1, 2, 3\} \setminus \{h\}$, we color u_j^s by j_s. Assume now that $z_h = \overline{x}_i$ and that z_h is the p-th occurrence of the variable x_i in ϕ. Symmetrically, we color u_j^h by i_p, w_j by j_h, and for $s \in \{1, 2, 3\} \setminus \{h\}$, we color u_j^s by j_s. We observe that for any distinct $i, i' \in \{1, \dots, n\}$, the lists of admissible colors of $x_i^1, x_i^2, x_i^3, y_i^1, y_i^2, y_i^3$ do not share any color with the lists of $x_{i'}^1, x_{i'}^2, x_{i'}^3, y_{i'}^1, y_{i'}^2, y_{i'}^3$. Also for any distinct $j, j' \in \{1, \dots, m\}$, the lists of colors of u_j^1, u_j^2, u_j^3, w_j do not share any color with he lists of $u_{j'}^1, u_{j'}^2, u_{j'}^3, w_{j'}$. Hence we obtained a coloring of G that respects $\mathcal{L}$.

Now suppose that c is a coloring of G that respects $\mathcal{L}$. We need the following claim that holds for all $1 \leq i \leq n$:

$$\text{either } c(x_i^1) = i_1,\ c(x_i^2) = i_2,\ c(x_i^3) = i_3 \text{ or } c(x_i^1) = i_1',\ c(x_i^2) = i_2',\ c(x_i^3) = i_3'.$$

In order to see this claim, first assume that $c(x_i^1) = i_1$. Then $c(y_i^1) = i_2'$, $c(x_i^2) = i_2$, $c(y_i^2) = i_3'$, and $c(x_i^3) = i_3$. Symmetrically, if $c(x_i^1) = i_1'$, then $c(y_i^3) = i_3$,

$c(x_i^3) = i'_3$, $c(y_i^2) = i_2$, and $c(x_i^2) = i'_2$. Hence, the claim holds, and we can do as follows. For $i = 1, \ldots, n$, we let $x_i = true$ if $c(x_i^1) = i_1$, $c(x_i^2) = i_2$, $c(x_i^3) = i_3$, and $x_i = false$ if $c(x_i^1) = i'_1$, $c(x_i^2) = i'_2$, $c(x_i^3) = i'_3$. We claim that this truth assignment satisfies ϕ. For $j \in \{1, \ldots, m\}$, consider the clause C_j and suppose that $C_j = z_1 \vee z_2$ or $C_j = z_1 \vee z_2 \vee z_3$. Recall that if C_j contains exactly two literals, then $c(u_j^3) = j_3$. We also observe that there is an index $h \in \{1, 2, 3\}$ such that $c(u_j^h) \neq j_h$ as otherwise it would be impossible to color w_j. Hence, if z_h is the p-th occurrence of the variable x_i in ϕ, then $c(u_j^h) = i'_p$ if $z_h = x_i$ and $c(u_j^h) = i_p$ if $z_h = \overline{x}_i$. If $c(u_j^h) = i'_p$, then $c(u_j^h) \neq c(x_i^p) = i_p$, and $x_i = true$. Otherwise, if $c(u_j^h) = i_p$, then $c(u_j^h) \neq c(x_i^p) = i'_p$, and $x_i = false$. In both cases C_j is satisfied. We therefore find that ϕ is satisfied. This completes the proof of Lemma 2. □

Lemma 3. List 3-Coloring *is* NP-*compete for graphs of maximum degree at most* 3 *with girth at least g, and in which any two vertices of degree* 3 *are of distance at least g from each other, for any fixed constant* $g \geq 3$.

Proof. NP-membership is clear. To show NP-hardness we reduce from a variant of Not-All-Equal Satisfiability with positive literals only. This problem is NP-complete [30] and defined as follows. Given a set $X = \{x_1, x_2, ..., x_n\}$ of logical variables, and a set $\mathcal{C} = \{C_1, C_2, ..., C_m\}$ of clauses over X in which all literals are positive, does there exist a truth assignment for X such that each clause contains at least one true literal and at least one false literal? The variant we consider takes as input an instance $(\mathcal{C}, X)$ of Not-All-Equal Satisfiability with positively literals only that has two additional properties. First, each C_i contains either two or three literals. Second, each literal occurs in at most three different clauses. One can prove that this variant is NP-complete by a reduction from the original problem via a well-known folklore trick (see e.g. [13]).

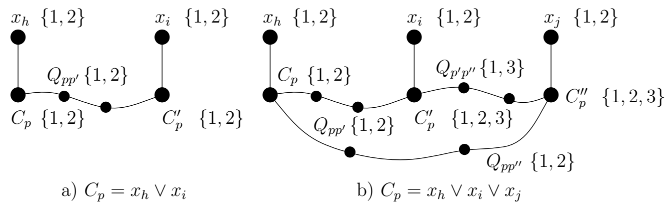


Fig. 2. The construction of G and $\mathcal{L}$ for $g = 3$

From an instance $(\mathcal{C}, X)$ as defined above, we construct a graph G and a list assignment $\mathcal{L}$ as follows. For each literal x_i we introduce a vertex that we denote by x_i as well. We define $L(x_i) = \{1, 2\}$. For each clause C_p with two literals, we fix an ordering of its literals, say x_h, x_i. We then introduce two vertices C_p, C'_p and add the edges $C_p x_h$ and $C'_p x_i$. We let C_p and C'_p be the end-vertices of a path $Q_{pp'}$ of odd length at least g, whose inner vertices are new vertices. We assign the list $\{1, 2\}$ to each vertex of $Q_{pp'}$. See Fig. 2 a). For each clause C_p with three literals, we fix an ordering of its literals, say x_h, x_i, x_j. We then introduce three vertices C_p, C_p', C_p'' and add edges C_px_h, $C_p'x_i$, $C_p''x_j$. We define $L(C_p) = \{1, 2\}$ and $L(C_p') = L(C_p'') = \{1, 2, 3\}$. We define paths $Q_{pp'}$, $Q_{pp''}$ and $Q_{p'p''}$, each with new inner vertices and of odd length at least g, that go from C_p to C_p', from C_p to C_p'', and from C_p' to C_p'', respectively. We assign the list $\{1, 2\}$ to each inner vertex of $Q_{pp'}$ and to each inner vertex of $Q_{pp''}$, whereas we assign the list $\{1, 3\}$ to each inner vertex of $Q_{p'p''}$. See Fig. 2 b). This completes our construction of G and $\mathcal{L}$. Because each clause contains at most three literals and each literal occurs in at most three clauses, G has maximum degree at most 3. By construction, G has girth at least g and any two vertices of degree 3 have distance at least g from each other. We claim that X has a truth assignment such that each clause contains at least one true literal and at least one false literal if and only if G has a coloring that respects $\mathcal{L}$.

First suppose that X has a truth assignment such that each clause contains at least one true literal and at least one false literal. We assign color 1 to every true literal and color 2 to every false literal. Suppose that C_p is a clause containing exactly two literals ordered as x_h, x_i Then, by our assumption, one of them is true and the other one is false. Suppose that x_h is true and x_i is false. Then we give C_p color 2 and C_p' color 1. Because the path $Q_{pp'}$ has odd length, we can alternate between the colors 1 and 2 for the inner vertices of $Q_{pp'}$. If x_h is false and x_i is true, we act in a similar way. Suppose that C_p is a clause containing three literals ordered as x_h, x_i, x_j. By assumption, at least one of the vertices x_h, x_i, x_j received color 1, and at least one of them received color 2. This leaves us with six possible cases. If x_h, x_i, x_j have colors $1, 1, 2$, then we give C_p, C_p', C_p'' colors 2,3,1, respectively. If x_h, x_i, x_j have colors $1, 2, 1$, then we give C_p, C_p', C_p'' colors 2,1,3, respectively. If x_h, x_i, x_j have colors $2, 1, 1$, then we give C_p, C_p', C_p'' colors 1,3,2, respectively. If x_h, x_i, x_j have colors $2, 2, 1$, then we give C_p, C_p', C_p'' colors 1,3,2, respectively. If x_h, x_i, x_j have colors $2, 1, 2$, then we give C_p, C_p', C_p'' colors 1,2,3, respectively. If x_h, x_i, x_j have colors $1, 2, 2$, then we give C_p, C_p', C_p'' colors 2,3,1, respectively. What is left to do is to color the inner vertices of the paths $Q_{pp'}$, $Q_{pp''}$, $Q_{p'p''}$. For the inner vertices of the first two paths we alternate between colors 1 and 2, whereas we alternate between colors 1 and 3 for the inner vertices of the last path. Because we ensured that in all six cases the vertices C_p, C_p' and C_p'' received distinct colors and the length of the paths is odd, we can do this. Hence, we obtained a coloring of G that respects $\mathcal{L}$.

Now suppose that G has a coloring that respects $\mathcal{L}$. Then every literal vertex has either color 1 or color 2. In the first case we make the corresponding literal true, and in the second case we make it false. We claim that in this way we obtained a truth assignment of X such that each clause contains at least one true literal and at least one false literal. In order to obtain a contradiction suppose that C_p is a clause, all literals of which are either true or false. First suppose that all its literals are true, i.e., they all received color 1. If C_p contains exactly two literals, then both C_p and C_p' received color 2, which is not possible. If C_p contains three literals, then C_p received color 2. Consequently, the colors of the inner vertices of the path $Q_{pp'}$ are forced. Because $Q_{pp'}$ has odd length, this means that the neighbor of C_p' that is on $Q_{pp'}$ received color 2. Then, because C_p' is adjacent to a literal vertex with color 1, we find that C'_p must have received color 3. However, following the same arguments, we now find that the three neighbors of $C_{p''}$ have colors 1,2,3, respectively. This is not possible. If all literals of C_p are false, we use the same arguments to obtain the same contradiction. Hence, such a clause C_p does not exist. This completes the proof of Lemma 3. □

Note that Lemmas 1 and 2 claim NP-completeness for 3-List Coloring on some special graph classes, whereas Lemma 3 claims this for List 3-Coloring, which is the more restricted version of List Coloring where only three distinct colors may be used in total as admissible colors in the lists of a list assignment.

We are now ready to prove Theorem 3.

Proof (of Theorem 3). We first show the polynomial-time solvable cases. Case 1 follows from Theorem 1 (ii). Any $(C_3, K_{1,3})$-free graph has maximum degree at most 2. Kratochvíl and Tuza [21] showed that List Coloring is polynomial-time solvable on graphs of maximum degree 2. This proves Case 2. By Ramsey's Theorem, every (K_r, sP_1)-free graph contains at most $\gamma(r, s)$ vertices for some constant $\gamma(r, s)$. Hence, we can decide in constant time whether such a graph has a coloring that respects some given list assignment. This proves Case 3.

Suppose that Cases 1–3 are not applicable. If both H_1 and H_2 contain a cycle, then NP-completeness of 3-List Coloring follows from Theorem 2 (i):1. Suppose that one of the graphs, say H_1, contains a cycle, whereas H_2 contains no cycle, i.e., is a forest.

First suppose that H_1 contains an induced C_r for some $r \geq 4$. Because H_2 is not an induced subgraph of P_3, we find that H_2 contains an induced $P_1 + P_2$ or an induced $3P_1$. If H_2 contains an induced $P_1 + P_2$, then every complete split graph is (H_1, H_2)-free. Hence NP-completeness of 3-List Coloring follows from Lemma 1 (ii). If H_2 contains an induced $3P_1$, then every union of two complete graphs is (H_1, H_2)-free. Hence NP-completeness of 3-List Coloring follows from Lemma 1 (iii).

Now suppose that H_1 contains no induced C_r for some $r \geq 4$, but suppose that it does contain C_3. If H_2 contains an induced $P_1 + P_2$, then every complete bipartite graph is (H_1, H_2)-free. Hence NP-completeness of 3-List Coloring follows from Lemma 1 (i). If H_2 contains an induced $K_{1,r}$ for some $r \geq 4$, then every graph of maximum degree at most 3 and of girth at least 4 is (H_1, H_2)-free. Hence, NP-completeness of 3-List Coloring follows from Lemma 3 after choosing $g = 4$. Suppose that H_2 contains neither an induced $P_1 + P_2$ nor an induced $K_{1,r}$ for some $r \geq 4$. Recall that H_2 is a forest that is not an induced subgraph of P_3. Then $H_2 = sP_1$ for some $s \geq 3$ or $H_2 = K_{1,3}$.

First suppose that $H_2 = sP_1$ for some $s \geq 3$. If H_1 is not a complete graph minus a matching, then every complete graph minus a matching is (H_1, H_2)-free. Hence NP-completeness of 3-List Coloring follows from Lemma 1 (iv). If H_1 is not a non-disjoint union of two complete graphs, then every non-disjoint union of two complete graphs is (H_1, H_2)-free. Hence NP-completeness of 3-List Coloring follows from Lemma 1 (iii). Now assume that H_1 is a complete graph minus a matching and also the non-disjoint union of two complete graphs. Then either H_1 is a complete graph or a complete graph minus an edge. However, H_1

is not a complete graph by assumption (as otherwise we would end up in Case 3 again). Hence H_1 is a complete graph minus an edge. Because H_1 contains C_3, this means that H_1 contains an induced K_4^-. However, then every matching-separated cobipartite graph is (H_1, H_2)-free. Hence NP-completeness of 3-List Coloring follows from Lemma 2.

Now suppose that $H_2 = K_{1,3}$. By repeating the arguments of the previous case, in which $H_2 = sP_1$ for some $s \geq 3$, we obtain NP-completeness of 3-List Coloring or find that H_1 is a complete graph or a complete graph minus an edge. If H_1 is a complete graph, then $H_1 \neq C_3$ by assumption (as otherwise we would end up in Case 2 again). This means that H_1 contains an induced K_4. If H_1 is a complete graph minus an edge, then H_1 contains an induced K_4^- as H_1 already contains the graph C_3. Hence, in both cases, every $(K_4, K_4^-, K_{1,3})$-free graph is (H_1, H_2)-free. Observation 3 in the paper of Král' et al. [20] tells us that Coloring is NP-complete for $(K_4, K_4^-, K_{1,3})$-free graphs. However, its proof shows in fact that 3-Coloring is NP-compete for this graph class. Hence, NP-completeness of 3-List Coloring follows.

Finally we consider the case when H_1 and H_2 contain no cycles, i.e., are both forests. Because neither of them is an induced subgraph of P_3, each of them contains an induced $3P_1$ or an induced $P_1 + P_2$. Recall that a graph is a complete graph minus a matching if and only if it is $(3P_1, P_2)$-free. Hence, any complete graph minus a matching is (H_1, H_2)-free. Then NP-completeness of 3-List Coloring follows from Lemma 1 (iv). This completes the proof of Theorem 3. □

3 Conclusion

We completely classified the complexity of List Coloring and ℓ-List Coloring for (H_1, H_2)-free graphs. The next step would be to classify these two problems for $\mathcal{H}$-free graphs, where $\mathcal{H}$ is an arbitrary finite set of graphs. However, even the case with three forbidden induced subgraphs is not clear. This is in stark contrast to the situation when we forbid subgraphs that may not necessarily be induced. For a set of graphs $\{H_1, \ldots, H_p\}$, we say that a graph G is *strongly* $(H_1, \ldots, H_p)$*-free* if G has no subgraph isomorphic to a graph in $\{H_1, \ldots, H_p\}$. For such graphs we can show the following result.

Theorem 4. *Let* $\{H_1, \ldots, H_p\}$ *be a finite set of graphs. Then* List Coloring *is polynomial-time solvable for strongly* $(H_1, \ldots, H_p)$*-free graphs if there exists a graph* H_i *that is a forest of maximum degree at most* 3*, every connected component of which has at most one vertex of degree* 3*. In all other cases, even* List 3-Coloring *is* NP*-complete for* $(H_1, \ldots, H_p)$*-free graphs.*

Proof. First suppose there exists a graph H_i that is a forest of maximum degree at most 3, in which every connected component contains at most one vertex of degree 3. Because H_i has maximum degree at most 3, every connected component of H_i is either a path or a subdivided claw. As such, H_i is not a subgraph of a graph G if and only if H is not a minor of G. In that case G has path-width at most $|V(H)| - 2$ [1]. Then the path-width, and hence, the treewidth of G is bounded, as H is fixed. Because List Coloring is polynomial-time solvable for graphs of bounded treewidth [18], we find that List Coloring is polynomial-time solvable for strongly H_i-free graphs, and consequently, for strongly $(H_1, \ldots, H_p)$-free graphs. Now suppose that we do not have such a graph H_i. Then every H_i contains either an induced cycle or is a forest with a vertex of degree at least 4 or is forest that contains a connected component with two vertices of degree 3. Then NP-completeness of List 3-Coloring follows from Lemma 3 after choosing the constant g sufficiently large. □

We note that a classification for Coloring and k-Coloring similar to the one in Theorem 4 for List Coloring and List 3-Coloring is not known even if only one (not necessarily induced) subgraph is forbidden; see Golovach et al. [11] for partial results in this direction.

Another interesting problem, which is still open, is the following. It is not difficult to see that k-Coloring is NP-complete for graphs of diameter d for all pairs (k, d) with $k \geq 3$ and $d \geq 2$ except when $(k, d) \in \{(3, 2), (3, 3)\}$. Recently, Mertzios and Spirakis [25] solved one of the two remaining cases by showing that 3-Coloring is NP-complete even for triangle-free graphs $G = (V, E)$ of diameter 3, radius 2 and minimum degree $\delta = \theta(|V|^{\epsilon})$ for every $0 \leq \epsilon \leq 1$. This immediately implies that List 3-Coloring is NP-complete for graphs of diameter 3. What is the computational complexity of List 3-Coloring for graphs of diameter 2?

References

1. Bienstock, D., Robertson, N., Seymour, P.D., Thomas, R.: Quickly excluding a forest. J. Comb. Theory, Ser. B 52, 274–283 (1991)
2. Brandstädt, A., Engelfriet, J., Le, H.-O., Lozin, V.V.: Clique-Width for 4-Vertex Forbidden Subgraphs. Theory Comput. Syst. 39, 561–590 (2006)
3. Brandstädt, A., Kratsch, D.: On the structure of (P_5, gem)-free graphs. Discrete Applied Mathematics 145, 155–166 (2005)
4. Brandstädt, A., Le, H.-O., Mosca, R.: Gem- and co-gem-free graphs have bounded clique-width. Internat. J. Found. Comput Sci. 15, 163–185 (2004)
5. Broersma, H.J., Golovach, P.A., Paulusma, D., Song, J.: Determining the chromatic number of triangle-free $2P_3$-free graphs in polynomial time. Theoretical Computer Science 423, 1–10 (2012)
6. Broersma, H.J., Golovach, P.A., Paulusma, D., Song, J.: Updating the complexity status of coloring graphs without a fixed induced linear forest. Theoretical Computer Science 414, 9–19 (2012)
7. Couturier, J.-F., Golovach, P.A., Kratsch, D., Paulusma, D.: List coloring in the absence of a linear forest. In: Kolman, P., Kratochvíl, J. (eds.) WG 2011. LNCS, vol. 6986, pp. 119–130. Springer, Heidelberg (2011)
8. Dahlhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., Yannakakis, M.: The complexity of multiterminal cuts. SIAM J. Comput. 23, 864–894 (1994)
9. Dabrowski, K., Lozin, V., Raman, R., Ries, B.: Colouring vertices of triangle-free graphs without forests. Discrete Mathematics 312, 1372–1385 (2012)
10. Golovach, P.A., Heggernes, P.: Choosability of P_5-free graphs. In: Královič, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 382–391. Springer, Heidelberg (2009)

11. Golovach, P.A., Paulusma, D., Ries, B.: Coloring Graphs Characterized by a Forbidden Subgraph. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 443–454. Springer, Heidelberg (2012)
12. Golovach, P.A., Paulusma, D., Song, J.: Coloring graphs without short cycles and long induced paths. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 193–204. Springer, Heidelberg (2011)
13. Golovach, P.A., Paulusma, D., Song, J.: Closing complexity gaps for coloring problems on H-free graphs. In: Chao, K.-M., Hsu, T.-s., Lee, D.-T. (eds.) ISAAC 2012. LNCS, vol. 7676, pp. 14–23. Springer, Heidelberg (2012)
14. Golovach, P.A., Paulusma, D., Song, J.: 4-Coloring H-free graphs when H is small. Discrete Applied Mathematics 161, 140–150 (2013)
15. Gyárfás, A.: Problems from the world surrounding perfect graphs. Zastosowania Matematyki Applicationes Mathematicae XIX (3-4), 413–441 (1987)
16. Hoàng, C.T., Maffray, F., Mechebbek, M.: A characterization of b-perfect graphs. Journal of Graph Theory 71, 95–122 (2012)
17. Jansen, K.: Complexity Results for the Optimum Cost Chromatic Partition Problem, Universität Trier, Mathematik/Informatik, Forschungsbericht 96–41 (1996)
18. Jansen, K., Scheffler, P.: Generalized coloring for tree-like graphs. Discrete Appl. Math. 75, 135–155 (1997)
19. Kobler, D., Rotics, U.: Edge dominating set and colorings on graphs with fixed clique-width. Discrete Applied Mathematics 126, 197–221 (2003)
20. Král', D., Kratochvíl, J., Tuza, Z., Woeginger, G.J.: Complexity of coloring graphs without forbidden induced subgraphs. In: Brandstädt, A., Le, V.B. (eds.) WG 2001. LNCS, vol. 2204, p. 254. Springer, Heidelberg (2001)
21. Kratochvíl, J., Tsuza, Z.: Algorithmic complexity of list colorings. Discrete Applied Mathematics 50, 297–302 (1994)
22. Kratsch, S., Schweitzer, P.: Graph isomorphism for graph classes characterized by two forbidden induced subgraphs. In: Golumbic, M.C., Stern, M., Levy, A., Morgenstern, G. (eds.) WG 2012. LNCS, vol. 7551, pp. 34–45. Springer, Heidelberg (2012)
23. Lozin, V.V.: A decidability result for the dominating set problem. Theoretical Computer Science 411, 4023–4027 (2010)
24. Maffray, F., Preissmann, M.: On the NP-completeness of the k-colorability problem for triangle-free graphs. Discrete Mathematics 162, 313–317 (1996)
25. Mertzios, G.B., Spirakis, P.G.: Algorithms and almost tight results for 3-colorability of small diameter graphs. In: van Emde Boas, P., Groen, F.C.A., Italiano, G.F., Nawrocki, J., Sack, H. (eds.) SOFSEM 2013. LNCS, vol. 7741, pp. 332–343. Springer, Heidelberg (2013)
26. Jensen, T.R., Toft, B.: Graph Coloring Problems. Wiley Interscience (1995)
27. Randerath, B.: 3-colorability and forbidden subgraphs. I., Characterizing pairs. Discrete Mathematics 276, 313–325 (2004)
28. Randerath, B., Schiermeyer, I.: A note on Brooks' theorem for triangle-free graphs. Australas. J. Combin. 26, 3–9 (2002)
29. Randerath, B., Schiermeyer, I.: Vertex colouring and forbidden subgraphs - a survey. Graphs Combin. 20, 1–40 (2004)
30. Schaefer, T.J.: The complexity of satisfiability problems. In: Proc. STOC 1978, pp. 216–226 (1978)
31. Schindl, D.: Some new hereditary classes where graph coloring remains NP-hard. Discrete Math. 295, 197–202 (2005)
32. Tuza, Z.: Graph colorings with local restrictions - a survey. Discuss. Math. Graph Theory 17, 161–228 (1997)

Query Complexity of Matroids

Raghav Kulkarni[1,⋆] and Miklos Santha[2,⋆⋆]

[1] Center for Quantum Technologies, Singapore
kulraghav@gmail.com

[2] LIAFA - University of Paris 7, France and Center for Quantum Technologies, National University of Singapore, Singapore
miklos.santha@liafa.jussieu.fr

Abstract. Let $\mathcal{M}$ be a bridgeless matroid on ground set $\{1, \ldots, n\}$ and $f_{\mathcal{M}} : \{0,1\}^n \to \{0,1\}$ be the indicator function of its independent sets. A folklore fact is that $f_{\mathcal{M}}$ is *evasive,* i.e., $D(f_{\mathcal{M}}) = n$ where $D(f)$ denotes the deterministic decision tree complexity of f. Here we prove query complexity lower bounds for $f_{\mathcal{M}}$ in three stronger query models: (a) $D_{\oplus}(f_{\mathcal{M}}) = \Omega(n)$, where $D_{\oplus}(f)$ denotes the parity decision tree complexity of f; (b) $R(f_{\mathcal{M}}) = \Omega(n/\log n)$, where $R(f)$ denotes the bounded error randomized decision tree complexity of f; and (c) $Q(f_{\mathcal{M}}) = \Omega(\sqrt{n})$, where $Q(f)$ denotes the bounded error quantum query complexity of f.

To prove (a) we propose a method to lower bound the *sparsity* of a Boolean function by upper bounding its partition size. Our method yields a new application of a somewhat surprising result of Gopalan et al. [11] that connects the sparsity to the *granularity* of the function.

As another application of our method, we confirm the Log-rank Conjecture for XOR functions [27], up to a poly-logarithmic factor, for a fairly large class of AC^0- XOR functions.

To prove (b) and (c) we relate the *ear decomposition* of matroids to the *critical* inputs of appropriate *tribe* functions and then use the existing randomized and quantum lower bounds for these functions.

Keywords: (parity, randomized, quantum) decision tree complexity, matroids, Fourier spectrum, read-once formulae, AC^0.

1 Introduction

1.1 Decision Tree Models

The decision tree or query model of computing is perhaps one of the simplest models of computation. Due to its fundamental nature, it has been extensively studied over last few decades; yet it remains far from being completely understood.

⋆ Supported by the French ANR Defis program under contract ANR-08-EMER-012 (QRAC project).

⋆⋆ This work was done when the first author was a post-doctoral fellow at: LIAFA - University of Paris 7 and LRI - University of Paris 11, France.

Fix a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$. A deterministic decision tree D_f for f takes $x = (x_1, \ldots, x_n)$ as an input and determines the value of $f(x_1, \ldots, x_n)$ using queries of the form " is $x_i = 1$? ". Let $C(D_f, x)$ denote the cost of the computation, that is the number of queries made by D_f on input x. The *deterministic decision tree complexity* of f is defined as $D(f) = \min_{D_f} \max_x C(D_f, x)$. A bounded error randomized decision tree R_f is a probability distribution over all deterministic decision trees such that for every input, the expected error of the algorithm is bounded by some fixed constant less than 1/2. The cost $C(R_f, x)$ is the highest possible number of queries made by R_f on x, and the *bounded error randomized decision tree complexity* of f is $R(f) = \min_{R_f} \max_x C(R_f, x)$. A bounded error quantum decision tree Q_f is a sequence of unitary operators, some of which depends on the input string. Broadly speaking, the cost $C(Q_f, x)$ is the number of unitary operators (quantum queries) which depend on x. The *bounded error quantum query complexity* of f is $Q(f) = \min_{Q_f} \max_x C(Q_f, x)$, where the minimum is taken over all quantum decision trees computing f. For a more precise definition we refer the reader to the excellent survey by Buhrman and de Wolf [8].

A natural theme in the study of decision trees is to understand and exploit the *structure* within f in order to prove strong lower bounds on its query complexity. A classic example is the study of non-trivial monotone graph properties. In the deterministic case it is known [23] that any such f of n vertex graphs has complexity $\Omega(n^2)$, and a famous conjecture [15] asserts that it is *evasive*, that is of maximal complexity, $D(f) = \binom{n}{2}$. In the randomized case the best lower bound (up to some polylogarithmic factor) is $\Omega(n^{4/3})$, and it is widely believed that in fact $R(f) = \Omega(n^2)$. In both models of computation, the structure that makes the complexity high is monotonicity and symmetry.

In this paper we study the decision tree complexity of another *structured* class, called *matroidal* Boolean functions, which arise from *matroids*. They form a subclass of monotone Boolean functions. These are the indicator functions of the independent sets of matroids. The matroidal Boolean functions inherit the rich combinatorial structure from matroids. Naturally, one may ask: what effect does this structure have on the decision tree complexity? It is a folklore fact that (modulo some degeneracies) such functions are *evasive*. Our main results in this paper are query complexity lower bounds for such functions in three stronger query models, namely: parity decision trees, bounded error randomized decision trees, and bounded error quantum decision trees. We give here a brief overview of the relatively less known model of *parity decision trees*.

A *parity decision tree* may query " is $\sum_{i \in S} x_i \equiv 1 \pmod 2$? " for an arbitrary subset $S \subseteq [n]$. We call such queries *parity queries*. For a parity decision tree P_f for f, let $C(P_f, x)$ denote the number of parity queries made by P_f on input x. The *parity decision tree complexity* of f is

$$D_\oplus(f) = \min_{P_f} \max_x C(P_f, x).$$

Note that $D_\oplus(f) \leq D(f)$ as " is $x_i = 1$? " can be treated as a parity query.

Parity decision trees were introduced by Kushilevitz and Mansour [18] in the context of learning Boolean functions by estimating their Fourier coefficients.

The *sparsity* of a Boolean function f, denoted by $||\widehat{f}||_0$, is the number of its non-zero Fourier coefficients. It turns out that the logarithm of the sparsity is a lower bound on $D_\oplus(f)$ [18,24,20]. Thus having a small depth parity decision tree implies only small number of Fourier coefficients to estimate.

Parity decision trees came into light recently in an entirely different context, namely in investigations of the *communication complexity* of XOR functions. Shi and Zhang [24] and Montanaro and Osborne [20] have observed that the deterministic communication complexity $DC(f^\oplus)$ of computing $f(x \oplus y)$, when x and y are distributed between the two parties, is upper bounded by $D_\oplus(f)$. They have also both conjectured that for some positive constant c, every Boolean function f satisfies $D_\oplus(f) = O((\log ||\widehat{f}||_0)^c)$. Settling this conjecture in affirmative would confirm the famous Log-rank Conjecture in the important special case of XOR functions. Montanaro and Osborne [20] showed that for a monotone Boolean function $D_\oplus(f) = O((\log ||\widehat{f}||_0)^2)$, and conjectured that actually $c = 1$.

1.2 Our Results and Techniques

In this paper $[n] := \{1, \ldots, n\}$. Let $\mathcal{M}$ be a matroid on ground set $[n]$ and $f_\mathcal{M}$ be the indicator function of the independent sets of $\mathcal{M}$. We refer the reader to Section 2 for relevant definitions. We describe now our lower bounds in the three computational model. We think that the most interesting case is the parity decision tree model since it brings together quite a few ideas.

Fourier Spectrum of Matroids Is Dense

Our main technical result is that the Fourier spectrum of *matroidal* Boolean functions is *dense.*

Theorem 1. *If $\mathcal{M}$ is a bridgeless matroid on ground set $[n]$ then*

$$\log ||\widehat{f_\mathcal{M}}||_0 = \Omega(n).$$

An immediate corollary of this result is the lower bound on the parity decision tree complexity.

Corollary 1. *If $\mathcal{M}$ is a bridgeless matroid on ground set $[n]$ then*

$$D_\oplus(f_\mathcal{M}) = \Omega(n).$$

Another corollary of the theorem is that $Q^*(f(x \oplus y))$, the quantum communication complexity of $f(x \oplus y)$ in the exact computation model with shared entanglement is maximal. Indeed, Buhrman and de Wolf [7] have shown that, up to a factor of 2, it is bounded from below by the logarithm of the rank of the communication matrix $f(x \oplus y)$. Since Shi and Zhang have proven [27] that the rank of the communication matrix is exactly $||\widehat{f}||_0$, the corollary indeed follows from Theorem 1.

Corollary 2. *If $\mathcal{M}$ is a bridgeless matroid then $Q^*(f_\mathcal{M}(x \oplus y)) = \Omega(n)$.*

To prove Theorem 1 we bring together various concepts and ideas from several not obviously related areas. The first part of our proof which relates partition size to Fourier spectrum is actually valid for any Boolean function. Our main ingredient is a relation (Proposition 3) stating that a small Euler characteristic implies that the sparsity of the function is high, that is the number of its non-zero Fourier coefficients is large. To prove this we use a recent result of Gopalan et. al. [11] (originated in the context of property testing) that crucially uses the Boolean-ness to connect the sparsity to the *granularity* - the smallest k such that all Fourier coefficients are multiple od $1/2^k$. Our second ingredient is to show (Lemma 2) that the Euler characteristic can be bounded by the partition size of the Boolean function. Finally to make this strategy work, we need to choose an appropriate restriction of the function so that the Euler characteristic of the restriction is non-zero.

When the rank of the matroid is small, the proof of Theorem 1 is in fact relatively easy. To conclude the proof when the rank is large we use a powerful theorem of Björner [4] which bounds the partition size of a matroidal Boolean function by the number of maximum independent sets.

In fact, the same method can be used to lower bound the sparsity of another large subclass of (not necessarily monotone) Boolean functions, namely the AC^0 functions. Hence for such functions parity queries can be simulated by ordinary ones only with a polynomial factor loss. The formal statement, analogous to Theorem 1 is the following:

Theorem 2. *If $f : \{0,1\}^n \to \{0,1\}$ has a circuit of depth d and size m then*

$$\log ||\widehat{f}||_0 = \Omega(\deg(f)/(\log m + d \log d)^{d-1}).$$

We would like to point out that the upper bound on the partition size for the class of AC^0 functions is highly non-trivial result(cf. [13]), whose proof relies crucially on the Switching Lemma.

Theorems 2 has an interesting corollary that the Log-rank conjecture holds for AC^0 XOR-functions. Indeed, as we have explained already, whenever $D_{\oplus}(f) = O((\log ||\widehat{f}||_0)^c)$, the Log-rank conjecture holds for $f^{\oplus}$. Obviously $D_{\oplus}(f) \leq D(f)$, and its is known [21] that $D(f) = \deg(f)^{O(1)}$. Therefore we have

Corollary 3. *Let M_f be the communication matrix of $f^{\oplus}$. If $f : \{0,1\}^n \to \{0,1\}$ is in AC^0 then*

$$DC(f^{\oplus}) \leq (\log \mathrm{rk}(M_f))^{O(1)}.$$

This means that in *exact* model [7] quantum and classical communication complexity of AC^0- XOR functions are polynomially related.

Randomized and Quantum Query Complexity

We obtain a nearly optimal lower bound on the randomized query complexity of matroids.

Theorem 3. *If $\mathcal{M}$ is a bridgeless matroid on ground set $[n]$ then*

$$R(f_{\mathcal{M}}) = \Omega(n/\log n).$$

It is widely conjectured that for every total Boolean function f, the relation $D(f) = O(Q(f)^2)$ holds (Conjecture 1 in [1]). Barnum and Saks (Theorem 2 in [1]) confirm this conjecture for AND-OR read-once formulae, and we are able to extend their result to read-once formulae over matroids.

Theorem 4. *If $f : \{0,1\}^n \to \{0,1\}$ is a read-once formula over matroids then*

$$Q(f) = \Omega(\sqrt{n}).$$

Our simple but crucial observation for proving lower bounds for randomized and quantum query complexity is that for any *matroidal* Boolean function f, one can associate, via the *ear decomposition* of matroids, a *tribe* function g such that f matches with g on all *critical* inputs. The lower bounds then follow from the partition bound for *tribe* functions obtained by Jain and Klauck [14] and the adversary bound for AND-OR read-once formulae by Barnum and Saks [1]. Our main contribution here is observing that certain lower bound methods for *tribe* functions generalize for the larger class of *matroidal* Boolean functions.

2 Preliminaries

2.1 Matroids and Matroidal Boolean Functions

Definition 1 (Matroid). *Let E be a finite set. A collection $\mathcal{M} \subseteq 2^E$ is called a* matroid *if it satisfies the following properties:*
(1) (non-emptiness) $\emptyset \in \mathcal{M}$;
(2) (hereditary property) *if $A \in \mathcal{M}$ and $B \subseteq A$ then $B \in \mathcal{M}$;*
(3) (augmentation property) *if $A, B \in \mathcal{M}$ and $|A| > |B|$ then there exists $x \in A \backslash B$ such that $x \cup B \in \mathcal{M}$.*

We call E the *ground set* of $\mathcal{M}$. The members of $\mathcal{M}$ are called *independent sets* of $\mathcal{M}$. If $A \notin \mathcal{M}$ then A is called *dependent* with respect to $\mathcal{M}$. A *circuit* in $\mathcal{M}$ is a minimal dependent set. For $A \subseteq E$, the *rank* of A with respect to $\mathcal{M}$ is defined as follows:

$$\mathrm{rk}(A, \mathcal{M}) := \max\{|B| \mid B \subseteq A \text{ and } B \in \mathcal{M}\}.$$

The *rank* or *dimension* of $\mathcal{M}$, denoted by $rk(\mathcal{M})$, is defined to be the rank of E with respect to $\mathcal{M}$.

A matroid $\mathcal{M}$ on ground set E can be identified with a Boolean function $f_{\mathcal{M}} : \{0,1\}^{|E|} \to \{0,1\}$ as follows: first identify $x \in \{0,1\}^{|E|}$ with a subset $S(x) := \{e \in E \mid x_e = 1\}$ of E; now let $f_{\mathcal{M}}(x) := 0 \iff S(x) \in \mathcal{M}$.

A function $f : \{0,1\}^n \to \{0,1\}$ is said to be *monotone increasing* if:

$$(\forall x, y \in \{0,1\}^n)(x \leq y \implies f(x) \leq f(y)),$$

where $x \leq y$ if for every $i \in [n] := \{1, \ldots, n\}$ we have $x_i \leq y_i$. The hereditary property of $\mathcal{M}$ translates to $f_{\mathcal{M}}$ being monotone.

We call a Boolean function f *matroidal* if there exists a matroid $\mathcal{M}$ such that $f \equiv f_{\mathcal{M}}$. Examples: AND, OR, MAJORITY, $\vee_{i=1}^{k} \wedge_{i=1}^{\ell} x_{ij}$.

An element $e \in E$ is called a *bridge* in $\mathcal{M}$ if e does not belong to any circuit of $\mathcal{M}$. If e is a bridge in $\mathcal{M}$ then the corresponding variable x_e of $f_{\mathcal{M}}$ is *irrelevant*, i.e., the function $f_{\mathcal{M}}$ does not depend on the value of x_e. Thus, for the purpose of query complexity, we can delete all the bridges and focus our attention on bridgeless matroids.

Ear Decomposition of Bridgeless Matroids

Let $\mathcal{M}$ be a matroid on ground set E. Let $T \subseteq E$. The *contraction* of $\mathcal{M}$ by T, denoted by $\mathcal{M}/T$, is a matroid on the ground set $E - T$ defined as follows:

$$\mathcal{M}/T := \{A \subseteq E - T \mid rk(A \cup T, \mathcal{M}) = |A| + rk(T, \mathcal{M})\}.$$

Definition 2 (Ear Decomposition [26]). *A sequence $(C_1, \ldots, C_k)$ of circuits of $\mathcal{M}$ is called an* ear decomposition *of $\mathcal{M}$ if:*
(1) $L_i := C_i - \bigcup_{j<i} C_j$ is non-empty and
(2) L_i is a circuit in $\mathcal{M}/\bigcup_{j<i} C_j$.

For $i = 1, \ldots, k$, the sets L_i are called *lobes*. An ear decomposition is *complete* if $\bigcup_{i=1}^{k} L_i = E$. Every bridgeless matroid admits a complete ear decomposition [10]. We identify complete ear decompositions with their lobe partition $E = \bigcup_{i=1}^{k} L_i$. For our randomized and quantum lower bounds we will crucially use the following proposition

Proposition 1. *Let $\mathcal{M}$ be a bridgeless matroid on ground set E and let $E = \bigcup_{i=1}^{k} L_i$ be a complete ear decomposition of $\mathcal{M}$. Let $e_1, \ldots, e_k \in E$ such that $e_i \in L_i$ and $L'_i := L_i - \{e_i\}$. Then $\bigcup_{i=1}^{k} L'_i$ is a maximum independent set of $\mathcal{M}$.*

2.2 Read-Once Formulae

Let $\mathcal{F}$ be a family of Boolean functions. A *read-once formula over $\mathcal{F}$* is a Boolean function represented by a rooted tree whose internal nodes are labeled by members of $\mathcal{F}$, and whose leaves are labelled by distinct variables. The inputs to each function are the outputs of its children.

If $\mathcal{F} = \{\wedge_n, \vee_n : n \in \mathbb{N}\}$ then we get the (unbounded fan-in) AND-OR read-once formulae. Given a complete ear decomposition $\bigcup_{i=1}^{k} L_i = E$ of a matroid, we associate to it the AND-OR read-once formula $g = \bigvee_{i=1}^{k} \bigwedge_{e \in L_i} x_e$. Such functions (OR's of AND's) are also called *tribe* functions.

Definition 3 (Critical Inputs of AND-OR Read-once Formulae). *An input is* critical *for an AND-OR read-once formula if for every AND gate at most one child evaluates to* 0 *and for every OR gate at most one child evaluates to* 1.

2.3 Fourier Spectrum of Boolean Functions

Every Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be uniquely represented by a real multilinear polynomial: $f(x_1,\ldots,x_n) = \sum_{S\subseteq[n]} \beta_S \prod_{i\in S} x_i$. Moreover, the coefficients β_S are integers. The *polynomial degree* of f is $deg(f) := max\{|S| \mid \beta_S \neq 0\}$. The *degree over* $\mathbb{F}_2$ of f is $deg_{\oplus}(f) := max\{|S| \mid \beta_S \neq 0 \mod 2\}$. The *Euler Characteristic* of f is $\chi(f) := \sum_{x\in\{0,1\}^n} (-1)^{|x|} f(x)$, where $|x|$ denotes the number of 1's in x. One can obtain the following expression for $\beta_{[n]}$ (cf. [2]):

$$\beta_{[n]} = \sum_{T\subseteq[n]} (-1)^{n-|T|} f(T) = (-1)^n \chi(f). \tag{1}$$

Fourier Spectrum

Let $f_{\pm} : \{-1,1\}^n \to \{-1,1\}$ be obtained from f as follows: $f_{\pm}(z_1,\ldots,z_n) := 1 - 2f(\frac{1-z_1}{2},\ldots,\frac{1-z_n}{2})$. Let $f_{\pm} : \{-1,1\}^n \to \{-1,1\}$ be represented by the following polynomial with real coefficients: $f_{\pm}(z_1,\ldots,z_n) = \sum_{S\subseteq[n]} \widehat{f}(S) \prod_{i\in S} z_i$. The above polynomial is unique and it is called the Fourier expansion of f. The $\widehat{f}(S)$ are called the Fourier coefficients of f. Note that:

$$\widehat{f}([n]) = \frac{(-1)^{n-1}\beta_{[n]}}{2^{n-1}} = \frac{\chi(f)}{2^{n-1}}. \tag{2}$$

The *sparsity* of a Boolean function f is $||\widehat{f}||_0 := |\{S \mid \widehat{f}(S) \neq 0\}|$. The *granularity* of a Boolean function is the smallest non-negative integer k such that each of its Fourier coefficients is an integer multiple of $1/2^k$.

3 Parity Decision Tree Complexity

In this section we prove Theorem 1. The following lemma which lower bounds the parity decision tree complexity by the sparsity is our starting point.

Lemma 1 (Shi and Zhang [27], Montanaro and Osborne [20]).

$$D_{\oplus}(f) = \Omega(\log ||\widehat{f}||_0).$$

The proof distinguishes two cases, according to the size of the rank of the matroid. In the first case, when the rank is small, the only property of matroidal Boolean functions we use is monotonicity. In the second case, when the rank is large,we proceed in two distinct steps as explained in the Introduction. Firstly we show that if the partition size of the function is small then its sparsity is high, a fact which is valid for any Boolean function. Secondly, in order to upper bound the partition size, we use *partitionability*, a strong topological property of matroids.

3.1 The Small Rank Case

A Boolean function f is said to be sensitive on i^{th} bit of input $x = (x_1, \ldots, x_n)$ if $f(x_1, \ldots, x_{i-1}, 1 - x_i, x_{i+1}, \ldots, x_n) \neq f(x)$. The sensitivity of f on input x, denoted by $s(f, x)$ is the number of sensitive bits of f on x. The sensitivity of a Boolean function f, denoted by $s(f)$ is $\max_x s(f, x)$.

Proposition 2. *If $\mathcal{M}$ is a matroid of rank r on ground set $[n]$ then*

$$\log ||\widehat{f_{\mathcal{M}}}||_0 \geq n - r.$$

Proof. It is easy to see that $s(f_{\mathcal{M}}) \geq n - r$ if $\mathcal{M}$ is a matroid of rank r on ground set $[n]$. In [3]) it is shown that for any Boolean function f we have $\log ||\widehat{f}||_0 \geq \deg_{\oplus}(f)$. In [20] it is proven that for monotone f we also have $\deg_{\oplus}(f) \geq s(f)$.

3.2 The Large Rank Case

Small Euler Characteristic Implies High Sparsity

Theorem 5 (Gopalan et. al., Theorem 12 in [11]). *If the sparsity of a Boolean function is s then its granularity is at most $\lfloor \log s \rfloor - 1$.*

Proposition 3. *If $f : \{0,1\}^n \to \{0,1\}$ such that $\chi(f) \neq 0$ then*

$$\log ||\widehat{f}||_0 = \Omega(n - \log |\chi(f)|).$$

Proof. If $\widehat{f}([n]) \neq 0$ then the granularity of f is $\Omega(\log(1/|\widehat{f}([n])|))$. From Equation 2 we know that $\widehat{f}([n]) = \chi(f)/2^{n-1}$. Together with Theorem 5 this gives the desired lower bound on the sparsity.

Euler Characteristic Is Upper Bounded by Partition Size

Definition 4 (Sub-cube Partition). *A Boolean sub-cube of the Boolean cube $\{0,1\}^n$ is an interval $[x, y] := \{z \mid x \leq z \leq y\}$, where $x, y \in \{0,1\}^n$. The sub-cube partition size of f, denoted by $P(f)$ is the smallest integer such that $f^{-1}(1)$ can be partitioned into $P(f)$ disjoint Boolean sub-cubes.*

Lemma 2. *For any Boolean function f, we have $|\chi(f)| \leq P(f)$.*

Proof. First note that no $x \in f^{-1}(0)$ contributes to $\chi(f)$. Let $\mathcal{C}$ be a sub-cube in the partition of $f^{-1}(1)$ into $P(f)$ parts. We can identify $\mathcal{C}$ with a partial Boolean assignment C that assigns 0 or 1 value to a subset $S_C \subseteq [n]$ variables. Note that this partial Boolean assignment certifies that the value of f is 1 on the entire $\mathcal{C}$, i.e., on any extension of C. If $|S_C| < n$ then:

$$|\{x \in \mathcal{C} \mid |x| \equiv 0 \pmod 2\}| = |\{x \in \mathcal{C} \mid |x| \equiv 1 \pmod 2\}|.$$

Therefore, the only $\mathcal{C}$'s that contributes to $\chi(f)$ have $|S_C| = n$ and hence $|\mathcal{C}| = 1$. In effect, such a $\mathcal{C}$ contributes ± 1 to $\chi(f)$.

Upper Bounding the Euler Characteristic of Matroids

Definition 5 (Partitionable Boolean Functions, cf. [16]). *A monotone decreasing Boolean function f is said to be partitionable if for every input $A \in f^{-1}(1)$ with maximal number of 1s, we can associate $\phi(A) \in f^{-1}(1)$ such that the $[\phi(A), A]$ partition $f^{-1}(1)$.*

Theorem 6 (Björner [4]). *If $\mathcal{M}$ is a matroid then $\neg f_{\mathcal{M}}$ is partitionable.*

Lemma 3. *If matroid $\mathcal{M}$ has N maximum independent sets then $|\chi(f_{\mathcal{M}})| \leq N$.*

Proof. From Theorem 6 we know that $\neg f_{\mathcal{M}}$ is partitionable. Thus for every maximum independent set A of $\mathcal{M}$ one can associate an independent set $\phi(A) \subseteq A$ such that $[\phi(A), A]$ form a partition of $\mathcal{M}$. Since each $[\phi(A), A]$ is a Boolean sub-cube, we get a sub-cube partition of $\neg f_{\mathcal{M}}$ with at most N parts. Now the lemma follows from Lemma 2 and from the fact that $|\chi(f)| = |\chi(\neg f)|$.

3.3 Putting Things Together

In order to use Proposition 3 we need to show that the Euler characteristic of bridgeless matroids is non-zero.

Proposition 4. *If $\mathcal{M}$ is a bridgeless matroid then we have $\chi(f_{\mathcal{M}}) \neq 0$.*

Proof. We prove by induction on the cardinality of the ground set of bridgeless matroids that $|\chi(\mathcal{M})| \neq 0$. For every matroid $\mathcal{M}$ on ground set E and for every $e \in E$, by definition $\mathcal{M} - \{e\}$ is the matroid whose ground set is $E \setminus \{e\}$ and whose independent sets are those of $\mathcal{M}$ not containing e. An element $e \in E$ is called *loop* if $\{e\}$ is a circuit in $\mathcal{M}$.

The inductive step distinguishes two cases. If there is a loop $e \in E$ then it is easy to check that $|\chi(f_{\mathcal{M}})| = |\chi(f_{\mathcal{M}-\{e\}})|$. If there is a non-loop element $e \in E$ then we denote by $\mathcal{C}_e$ the collection of the circuits of $\mathcal{M}$ that contain e. Kook shows that $|\chi(f_{\mathcal{M}})|$ satisfies the following recurrence (Theorem 1 in Kook [17]) :

$$|\chi(f_{\mathcal{M}})| = \sum_{C \in \mathcal{C}_e} |\chi(f_{\mathcal{M}/C})|.$$

Note that the operations contracting a cycle and deleting a loop both preserve the bridgelessness and reduce the cardinality of the ground set by at least one.

The only base case, under the assumption of bridgelessness, is a matroid on ground set $\{e\}$ where $\{e\}$ is a circuit. It is easy to see that $\chi \neq 0$ in this case.

We can now give the proof of Theorem 1.

Proof. Let r be the rank of $\mathcal{M}$ and N be the number of maximum independent sets of $\mathcal{M}$. If $n - r \geq \frac{2n}{3}$ then the lower bound follows from Proposition 2. If $n - r < \frac{2n}{3}$ then:

$$|\chi(f)| \leq N \leq \binom{n}{r} = \binom{n}{n-r} \leq 2^{H(1/3)n},$$

where $|\chi(f)| \leq N$ follows from Lemma 3, and $N \leq \binom{n}{r}$ follows from the fact that every maximal independent set of a matroid has the same cardinality. The last inequality uses the assumption: $n - r < \frac{2}{3}n$. The H there denotes the binary entropy function: $H(\epsilon) = -\epsilon \log \epsilon - (1-\epsilon)\log(1-\epsilon)$. Since $H(1/3) < 1$, the theorem follows from Proposition 3 and Proposition 4.

Remark A. Since intersection of two intervals is again an interval, the partition size of the intersection of two matroid can be upper bounded when the rank of either of the matroid is small. Hence our proof goes through for the indicator functions of intersection of two matroids.

Remark B. The tribe function $\bigvee_{i=1}^{\sqrt{n}} \bigwedge_{j=1}^{\sqrt{n}} x_{ij}$ shows that Theorem 1 does not hold by replacing $\log ||\widehat{f}||_0$ with $s(f)$. We do not know if it holds with $\deg_{\oplus}(f)$.

4 Randomized Query Complexity

Let $\mathcal{M}$ be a bridgeless matroid on ground set $[n]$ with a complete ear decomposition $[n] = \cup_{i=1}^{r} L_i$. First we do some preprocessing. For $0 \leq t \leq \log n$, let

$$E_t := \bigcup_{i : 2^t \leq |L_i| < 2^{t+1}} L_i.$$

Choose an index t_0 such that $|E_{t_0}| \geq n/\log n$. Let f' be a restriction of $f_{\mathcal{M}}$ obtained by fixing the variables outside E_{t_0} as follows: For each $L_i \not\subseteq E_{t_0}$, fix some $e_i \in L_i$ and set $x_{e_i} = 0$, and for $e \in L_i - \{e_i\}$ set $x_e = 1$. Furthermore for each $L_i \subseteq E_{t_0}$, fix arbitrarily all but 2^{t_0} variables in L_i and set their values to 1.

We re-label the indices so that $L_1, \ldots, L_k \subseteq E_{t_0}$ and $L_{k+1}, \ldots, L_r \not\subseteq E_{t_0}$. This allows us to index the variables of f' by x_{ij} for $i \in [k]$ and $j \in [\ell]$, where $\ell = 2^{t_0}$ and x_{ij} is the j^{th} among the ℓ unrestricted variables in L_i. Thus f' is a function on $k \times \ell$ variables where and $k \times \ell \geq n/(2\log n)$.

$$g := \bigvee_{i=1}^{k} \bigwedge_{j=1}^{\ell} x_{ij}.$$

Lemma 4. *If f is a monotone increasing Boolean function on $k \times \ell$ variables that matches with g on all the critical inputs then*

$$R(f) = \Omega(k \times \ell).$$

Jain and Klauck prove the above Lemma for the case $k = \ell$ (Theorem 4 in [14]). An adaptation of their proof (Appendix ??) gives the general case. From Proposition 1 we have:

Lemma 5. *The function f' matches with g on all critical inputs.*

Theorem 3 is an immediate consequence of Lemma 4 and Lemma 5.

5 Quantum Query Complexity

Let $\mathcal{M}$ be a bridgeless matroid on ground set $[n]$ with a complete ear decomposition $[n] = \cup_{i=1}^{k} L_i$, and let g be the tribe function associated with it.

Lemma 6 (Barnum and Saks, Theorem 2 in [1]). *If f is a Boolean function on n variables that matches with g on all the critical inputs then: $Q(f) = \Omega(\sqrt{n})$.*

From Proposition 1 we have:

Lemma 7. *The function $f_{\mathcal{M}}$ matches with g on all critical inputs.*

Theorem 7. *If $\mathcal{M}$ is a bridgeless matroid on ground set $[n]$ then:*

$$Q(f_{\mathcal{M}}) = \Omega(\sqrt{n}).$$

Theorem 4 is an extension of the above theorem to read-once formulae over the family of matroidal Boolean function. Its proof is deferred to Appendix ??.

An Upper Bound

The following theorem follows along the lines of Theorem 11 in Childs and Kothari [9] (Appendix ??).

Theorem 8. *If $\mathcal{M}$ is a matroid of rank r on ground set $[n]$ then*

$$Q(f_{\mathcal{M}}) = O(\sqrt{rn}).$$

References

1. Barnum, H., Saks, M.E.: A lower bound on the quantum query complexity of read-once functions. J. Comput. Syst. Sci. 69(2), 244–258 (2004)
2. Beigel, R.: The Polynomial Method in Circuit Complexity. In: Structure in Complexity Theory Conference, pp. 82–95 (1993)
3. Bernasconi, A., Codenotti, B.: Spectral Analysis of Boolean Functions as a Graph Eigenvalue Problem. IEEE Trans. Computers 48(3), 345–351 (1999)
4. Björner, A.: Matroid Applications (ch. 7.). In: Homology and Shellability of Matroids and Geometric Lattices, pp. 226–283 (1992)
5. Björner, A., Lovász, L., Yao, A.C.-C.: Linear Decision Trees: Volume Estimates and Topological Bounds. In: STOC 1992, pp. 170–177 (1992)
6. Brassard, G., Hoyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. In: Quantum Computation and Information. Contemporary Mathematics, vol. 305, pp. 53–74. AMS (2002)
7. Buhrman, H., de Wolf, R.: Communication Complexity Lower Bounds by Polynomials. In: IEEE Conference on Computational Complexity, pp. 120–130 (2001)
8. Buhrman, H., de Wolf, R.: Complexity measures and decision tree complexity: a survey. Theor. Comput. Sci. 288(1), 21–43 (2002)
9. Childs, A.M., Kothari, R.: Quantum query complexity of minor-closed graph properties. In: STACS 2011, pp. 661–672 (2011)

10. Coullard, C.R., Hellerstein, L.: Independence and Port Oracles for Matroids, with an Application to Computational Learning Theory. Combinatorica 16(2), 189–208 (1996)
11. Gopalan, P., O'Donnell, R., Servedio, R.A., Shpilka, A., Wimmer, K.: Testing Fourier Dimensionality and Sparsity. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 500–512. Springer, Heidelberg (2009)
12. Lov, K.,, G.: A Fast Quantum Mechanical Algorithm for Database Search. In: STOC 1996, pp. 212–219 (1996)
13. Impagliazzo, R., Matthews, W., Paturi, R.: A satisfiability algorithm for AC0. In: SODA 2012, pp. 961–972 (2012)
14. Jain, R., Klauck, H.: The Partition Bound for Classical Communication Complexity and Query Complexity. In: IEEE Conference on Computational Complexity, pp. 247–258 (2010)
15. Kahn, J., Saks, M.E., Sturtevant, D.: A topological approach to evasiveness. Combinatorica 4(4), 297–306 (1984)
16. Kleinschmidt, P., Onn, S.: Signable Posets and Partitionable Simplicial Complexes. Discrete and Computational Geometry 15(4), 443–466 (1996)
17. Kook, W.: A new formula for an evaluation of the Tutte polynomial of a matroid. Discrete Mathematics 300(1-3), 235–238 (2005)
18. Kushilevitz, E., Mansour, Y.: Learning Decision Trees Using the Fourier Spectrum. SIAM J. Comput. 22(6), 1331–1348 (1993)
19. Leonardos, N., Saks, M.: Lower Bounds on the Randomized Communication Complexity of Read-Once Functions. Computational Complexity 19(2), 153–181 (2010)
20. Montanaro, A., Osborne, T.: On the communication complexity of XOR functions. CoRR abs/0909.3392 (2009)
21. Nisan, N., Szegedy, M.: On the Degree of Boolean Functions as Real Polynomials. Computational Complexity 4, 301–313 (1994)
22. Nisan, N., Wigderson, A.: On Rank vs. Communication Complexity. Combinatorica 15(4), 557–565 (1995)
23. Rivest, R.L., Vuillemin, J.: On Recognizing Graph Properties from Adjacency Matrices. Theor. Comput. Sci. 3(3), 371–384 (1976)
24. Shi, Y., Zhang, Z.: Communication Complexities of XOR functions CoRR abs/0808.1762 (2008)
25. Simon, D.R.: On the Power of Quantum Computation. SIAM J. Comput. 26(5), 1474–1483 (1997)
26. Szegedy, B., Szegedy, C.: Symplectic Spaces And Ear-Decomposition Of Matroids. Combinatorica 26(3), 353–377 (2006)
27. Zhang, Z., Shi, Y.: On the parity complexity measures of Boolean functions. Theor. Comput. Sci. 411(26-28), 2612–2618 (2010)

A New Dynamic Graph Structure for Large-Scale Transportation Networks*

Georgia Mali[1,2], Panagiotis Michail[1,2],
Andreas Paraskevopoulos[1,2], and Christos Zaroliagis[1,2]

[1] Computer Techn. Institute & Press “Diophantus”, Patras University Campus, GR
[2] Dept of Computer Eng. & Informatics, University of Patras, 26504 Patras, Greece
{mali,michai,paraskevop,zaro}@ceid.upatras.gr

Abstract. We present a new dynamic graph structure specifically suited for large-scale transportation networks that provides simultaneously three unique features: compactness, agility and dynamicity. We demonstrate its practicality and superiority by conducting an experimental study for shortest route planning in large-scale European and US road networks with a few dozen millions of nodes and edges. Our approach is the first one that concerns the dynamic maintenance of a large-scale graph with ordered elements using a contiguous memory part, and which allows an arbitrary online reordering of its elements.

1 Introduction

In recent years we observe a tremendous amount of research for efficient route planning in road and other public transportation networks, witnessing extremely fast algorithms that answer point-to-point shortest path queries in a few msecs (in certain cases even less) in road networks with a few dozen millions of nodes and edges after a certain preprocessing phase; see e.g., [3,6,10,12]. These algorithms, known as *speed-up techniques*, are clever extensions/variations of the classical Dijkstra's algorithm. Speed-up techniques employ not only heuristics to improve the query search space, but also optimizations in the way they are implemented. The graph structures used (mostly variations of the adjacency list representation) are not only *compact*, in the sense that they store nodes and edges in adjacent memory addresses, but also support arbitrary offline *reordering* of nodes and edges to increase reference locality. The latter, known as *internal node reordering*, effectively improves node locality by offline arranging nodes within memory, thus improving cache efficiency and query running times.

These graph structures are very efficient when the graph remains static but may suffer badly when updates occur, since an update must shift a great amount of elements in memory in order to keep compactness and locality. Updates either can occur explicitly (reflecting, for instance, changes in a road network varying

* This work was supported by the EU FP7/2007-2013 (DG CONNECT.H5-Smart Cities & Sustainability), under grant agreement no. 288094 (project eCOMPASS). Work done while the last author was visiting the Karlsruhe Institute of Technology.

from traffic jams and planned constructions to unforseen disruptions, etc), or may constitute an inherent part of an algorithm. The latter is evident in many state-of-the-art approaches, e.g., [3,6,10,12], which perform a mandatory preprocessing phase that introduces many "shortcuts" to the graph that in turn involve repetitively deleting and inserting edges, often intermixed with limited-scope executions of Dijkstra's algorithm. Thus, it is essential that any graph structure used in such cases can support efficient insertions and deletions of nodes and edges (dynamic graph structure). Hence, for efficient routing in large-scale networks, a graph structure is required supporting the following features.

1. *Compactness*: ability to efficiently access adjacent nodes or edges, a requirement of all speed-up techniques based on Dijkstra's algorithm.
2. *Agility*: ability to change and reconfigure its internal layout in order to improve the locality of the elements, according to a given algorithm.
3. *Dynamicity*: ability to efficiently insert or delete nodes and edges.

An obvious choice is an adjacency list representation, implemented with linked lists of adjacent nodes, because of its simplicity and dynamic nature. Even though it is inherently dynamic, in a way that it supports insertions and deletions of nodes and edges in $O(1)$ time, it provides no guarantee on the actual layout of the graph in memory (handled by the system's memory manager). Therefore, it does have dynamicity but it has neither compactness nor agility.

A very interesting variant of the adjacency list, extensively used in several speed-up techniques (see e.g., [3]), is the *forward star* graph representation [1,2], which stores the adjacency list in an array, acting as a dedicated memory space for the graph. The nodes and edges can be laid out in memory in a way that is optimal for the respective algorithms, occupying consecutive memory addresses which can then be scanned with maximum efficiency. This is very fast when considering a static graph, but when an update is needed, the time for inserting or deleting elements is prohibitive because large shifts of elements must take place. Thus, a forward star representation offers compactness and agility, and therefore ultra fast query times, but does not offer dynamicity. For this reason, a dynamic version was developed [14] that offers constant time insertions and deletions of edges, at the expense of slower query times and limited agility. The main idea is to move a node's adjacent edges to the end of the edge array in order to insert new edges adjacent to the node.

Motivated by the efficiency of the (static and dynamic) forward star representation, we present a new graph structure for directed graphs, called *Packed-Memory Graph*, which supports all the aforementioned features. In particular:

– Scanning of adjacent nodes or edges is optimal (up to a constant factor) in terms of time and memory transfers, and therefore comparable to the maximum efficiency of the static forward star representation (compactness).
– Nodes and edges can be reordered *online* within allocated memory in order to increase any algorithm's locality of reference, and therefore efficiency. Any speed-up technique can give its desired node ordering as input to our graph structure (agility).

- Inserting or deleting edges and nodes compares favourably with the performance of the adjacency list representation implemented as a linked list and the dynamic forward star representation, and therefore it is fast enough for all practical applications (dynamicity).

To assess the practicality of our new graph structure, we conducted a series of experiments on shortest path routing algorithms on large-scale European and US road networks, implementing the Bidirectional and the A^* variants of Dijkstra's algorithm, as well as all the ALT-based algorithms in [11]. Our goal was to merely show the performance gain of using our structure compared to the adjacency list or the dynamic forward star representation on shortest path routing algorithms, rather than beating the running times of the best speed-up techniques.

Our experiments showed that our graph structure outperforms the adjacency list and dynamic forward star graph structures in query time (frequent operations), while being a little slower in update time (rare operations). Hence, our graph structure is expected to be more efficient in practical applications with intermixed operations of queries and updates, a fact verified by our experiments. Note that our graph structure is neither a speed-up technique, nor a dynamic algorithm. It can, however, increase the efficiency of any speed-up technique or dynamic algorithm implemented on top of it. As our experiments show, this can be beneficial for the mandatory preprocessing phase of such techniques.

To the best of our knowledge, our approach is the first one concerning the dynamic maintenance of a large-scale graph with ordered elements (i) using a *contiguous* part of memory, and (ii) allowing an arbitrary online reordering of its elements *without* violating its contiguity.

Our graph structure builds upon the cache-oblivious packed-memory array [4]. The main idea is to keep the graph's elements intermixed with empty elements so that insertions can be easily accommodated, while ensuring that deletions do not leave large chunks of memory empty. This is achieved by monitoring and reconfiguring the density of elements within certain parts of memory *without* destroying their relative order. Exposition details and missing proofs due to space limitations, in the rest of the paper, can be found in the full version [13].

2 Preliminaries

Let $G = (V, E)$ be a directed graph with node set V ($n = |V|$), edge set E ($m = |E|$), and edge weight function $wt : E \to \mathbb{R}_0^+$.

Graph Representations. There are multiple data structures for graph representations and their use depends heavily on the characteristics of the input graph and the performance requirements of each specific application. We assume the reader is familiar with the *adjacency list representation.* A variant of the adjacency list is the *forward star representation* [1,2], in which the node list is implemented as an array, and all adjacency lists are appended to a single edge array sorted by their source node. Unique non-overlapping *adjacency segments* of the edge array contain the adjacency list of each node. Each node points to the segment in the edge array containing its adjacent edges. The additional information attached to nodes or edges is stored in the same way as in the adjacency list. The main drawback of the forward star is that in order to insert an edge at a certain adjacency segment, all edges after the segment must be shifted to the right. Clearly, this is an $O(m)$ operation. For this reason, a dynamic version [14] was developed where each adjacency segment has a size equal to a power of 2, containing the edges and some empty cells at the end. When inserting an edge, if there are empty cells in the respective segment, the new edge is inserted in one of them. Otherwise, the whole segment is moved to the end of the edge array, and its size is doubled. Deletions are handled in a virtual manner, just emptying the respective cells rather than deallocating reserved memory.

Packed-Memory Array. A packed-memory array [4] maintains N ordered elements in an array of size $P = cN$, where $c > 1$ is a constant. The cells of the array either contain an element x or are considered empty. Hence, the array contains N ordered elements and $(c-1)N$ empty cells called *holes*. The goal of a packed-memory array is to provide a mechanism to keep the holes in the array uniformly distributed, in order to support efficient insertions, deletions and scans of (consecutive) elements. This is accomplished by keeping intervals within the array such that a constant fraction of each interval contains holes. When an interval of the array becomes too full or too empty, breaching its so-called *density thresholds*, its elements are spread out evenly within a larger interval by keeping their relative order. This process is called a *rebalance* of the (larger) interval. Note that during a rebalance an element may be moved to a different cell within an interval. We shall refer to this as the *move* of an element to another cell. The density thresholds and the rebalance ranges are monitored by a complete binary tree on top of the array. More details can be found in [13].

3 The Packed-Memory Graph (PMG)

Structure. Our graph structure consists of three packed-memory arrays, one for the nodes and two for the edges of the graph (viewed as either outgoing or incoming) with pointers associating them. The two edge arrays are copies of each other, with the edges sorted as outgoing or incoming in each case. Therefore, the description and analysis in the following will consider only the outgoing edge array. The structure and analysis is identical for the incoming edge array. A graphical representation of our new graph structure is shown in Fig. 1.

Let $P_n = 2^k$, where k is such that $2^{k-1} < n \leq 2^k$. The nodes are stored in a packed-memory array of size P_n with *node density* $d_n = \frac{n}{P_n}$. Therefore, the packed-memory node array has size $P_n = c_n n$ where $c_n = 1/d_n$. Each node is stored in a separate cell of the packed-memory node array along with any information associated with it. The nodes are stored with a specific arbitrary order $u_0, u_1, ..., u_{n-2}, u_{n-1}$ which is called *internal node ordering* of the graph. This ordering may have a great impact on the performance of the algorithms implemented on top of our new graph structure.

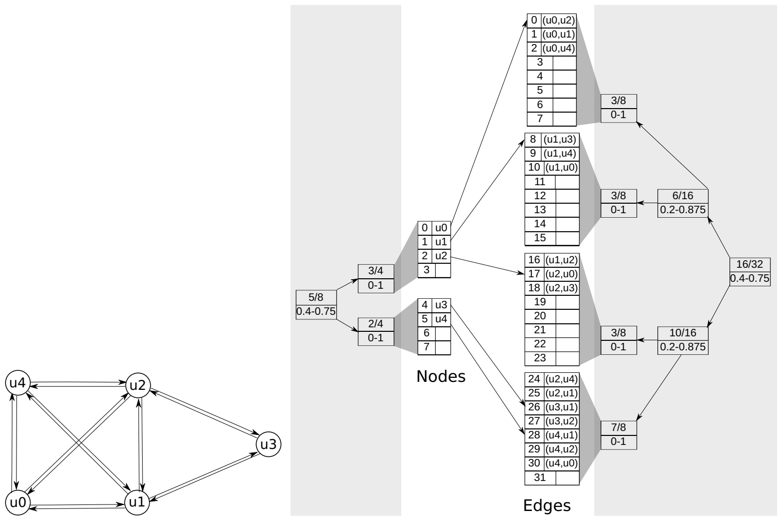


Fig. 1. Packed-memory Graph representation for the graph on the left

Let $P_m = 2^l$, where l is such that $2^{l-1} < m \leq 2^l$. The edges are also stored in a packed-memory array of size P_m with *edge density* $d_m = \frac{m}{P_m}$. Therefore, the packed-memory edge array has size $P_m = c_m m$ where $c_m = 1/d_m$. Each edge is stored in a separate cell of the packed-memory edge array along with any information associated with it, such as the edge weight. The edges are laid out in a specific order, which is defined by their source node. More specifically, we define a partition $C = \{E_{u_0}, E_{u_1}, ..., E_{u_{n-2}}, E_{u_{n-1}}\}$ of the edges of the graph according to their source nodes, where $E_{u_i} = \{e \in E | source(e) = u_i\}$, $E_{u_i} \cap E_{u_j} = \emptyset$, $\forall i, j$, $i \neq j$, and $E_{u_0} \cup E_{u_1} \cup ... \cup E_{u_{n-2}} \cup E_{u_{n-1}} = E$. The sets E_{u_i}, $0 \leq i < n$, are then stored consecutively in a unique range of cells of the packed-memory edge array in the same order as the one dictated by the internal node ordering in the packed-memory node array. This range is denoted by R_{u_i} and its length is $O(|E_{u_i}|)$ due to the properties of the packed-memory edge array. Every node u_i stores a pointer to the start and to the end of R_{u_i} in the edge array (for clarity, end pointers are not shown in Fig. 1). The end of R_{u_i} is at the same location as the start of $R_{u_{i+1}}$, since the outgoing edge sets have the same ordering as the nodes. If a node u_i has no outgoing edges, both of its pointers point to the start of $R_{u_{i+1}}$. Hence, given a node u_i, *determining* R_{u_i} takes $O(1)$ time.

Operations. Our new graph structure supports the following operations, whose asymptotic bounds are given in Table 1 and their proofs can be found in [13].

Scanning Edges. In order to scan the outgoing edges of a node u, the range, R_u, including them is determined by the pointers stored in the node. Then this range is sequentially scanned returning every outgoing edge of u.

Table 1. Space, time and memory transfer complexities (the latter in the cache-oblivious model [9]). B: cache block size; Δ: maximum node degree (typically $O(1)$ in large-scale transportation networks).

	Adjacency List	**Dynamic Forward Star**	**Packed-memory Graph**
Space	$O(m+n)$	$O(m+n)$	$O(m+n)$
Time			
Scanning S edges	$O(S)$	$O(S)$	$O(S)$
Inserting/Deleting an edge	$O(1)$	$O(1)$	$O(\log^2 m)$
Inserting a node	$O(1)$	$O(1)$	$O(\Delta \log^2 n)$
Deleting a node u (with adjacent edges)	$O(\Delta)$	$O(n+\Delta)$	$O(\Delta \log^2 m + \Delta \log^2 n)$
Internal relocation of a node u (with adj. edges)	not supported	not supported	$O(\Delta \log^2 m + \Delta \log^2 n)$
Memory Transfers			
Scanning S edges	$O(S)$	$O(1+S/B)$	$O(1+S/B)$
Inserting/Deleting an edge	$O(1)$	$O(1)$	$O(1+\frac{\log^2 m}{B})$
Inserting a node	$O(1)$	$O(1)$	$O(1+\frac{\Delta \log^2 n}{B})$
Deleting a node u (with adjacent edges)	$O(\Delta)$	$O(1+\frac{n+\Delta}{B})$	$O(1+\frac{\Delta \log^2 m + \Delta \log^2 n}{B})$
Internal relocation of a node u (with adj. edges)	not supported	not supported	$O(1+\frac{\Delta \log^2 m + \Delta \log^2 n}{B})$

Inserting Nodes. In order to insert a node u_i between two existing nodes u_j, u_{j+1}, we identify the actual cell that should contain u_i and execute an insert operation in the packed-memory node array. Clearly, this insertion changes the internal node ordering. The insert operation may result in a rebalance of some interval of the packed-memory node array, and some nodes being moved into different cells. For each node that is moved, its edges are updated with the new position of the node. The outgoing edge pointers of the newly created node u_i (which has not yet any adjacent edges) point to the start of the range $R_{u_{j+1}}$.

Deleting Nodes. In order to delete a node u_i between two existing nodes u_{i-1}, u_{i+1}, we first have to delete all of its outgoing edges, a process that is described in the next paragraph. Then, we identify the actual node array cell that should be cleared and execute a delete operation in the packed-memory node array. The delete operation may also result in a rebalance as before, so, for each node that is moved, its edges are updated with the new position of the node.

Inserting or Deleting Edges. When a new edge (u_i, u_j) is inserted (deleted), we proceed as follows. First, node u_i and its outgoing edge range R_{u_i} are identified. Then, the cell to insert to (delete from) within this range is selected and an insert (delete) operation in this cell of the packed-memory edge array is executed. This insert (delete) operation may cause a rebalance in an interval of the packed-memory edge array, causing some edges to be moved to different cells. As a result, the ranges of other nodes are changed too. When a range R_{u_k} changes, the non-zero ranges R_{u_x} and R_{u_y}, $x < k < y$, adjacent to it change too.

Note that x may not be equal to $k-1$ and y may not be equal to $k+1$, since there may be ranges with zero length adjacent to R_{u_k}. In order for R_{u_x}, R_{u_y} and the pointers towards them to be updated, the next and previous nodes of u_k with outgoing edges need to be identified. Let the maximum time required to identify these nodes be denoted by T_{up}. In [13] we describe how to implement this operation efficiently and explain that for all practical purposes $T_{up} = O(1)$.

Internal Node Reordering. Due to its design, our graph structure has the ability to internally reorder its nodes, which can be an attractive property. It does so, by removing an element from its original position and reinserting it to arbitrary new position. We call this operation an *internal relocation* of an element. In fact, a relocation is nothing more than a deletion and reinsertion of an element, two operations that have efficient running times and memory accesses.

Comparison with Other Dynamic Graph Structures. An adjacency list (ADJ) representation supports optimal insertions/deletions of nodes and the scanning of the edges is fast enough to be used in practice. However, since there is no guarantee for the memory allocation scheme, the nodes and edges are most probably scattered in memory, resulting in many cache misses and less efficiency during scan operations, especially for large-scale networks. Finally, an adjacency list representation provides (inherently) no support for any (re-)ordering of the nodes and edges in arbitrary relative positions in memory.

A dynamic forward star (DynFS) representation succeeds in storing the adjacent edges of each node in consecutive memory locations. Hence, the least amount of blocks is transferred into the cache memory during a scan operation of a node's adjacent edges. Moreover, it can efficiently insert or delete new edges in constant time. When inserting an edge adjacent to a node u, if there is space in the adjacency segment of u, it gets directly inserted there. Otherwise, the adjacency segment is moved to the end of the edge array, and its size is doubled. Therefore, it is clear that edge insertions can be executed in $O(1)$ amortized time and memory transfers. The edge deletion scheme consists of just emptying the respective memory cells, without any sophisticated rearranging operations taking place. Clearly, this also takes constant time. However, due to the particular update scheme, a specific adjacency segment ordering cannot be guaranteed. Moving an adjacency segment to the end of the edge array clearly destroys any locality of references between edges with different endpoints, resulting in slower query operation. Since the nodes are stored in an array, inserting a new node u at the end of the array is performed in constant time, while deleting u must shift all elements after u and therefore takes $O(n)$ time in addition to the time needed to delete all edges adjacent to u. Finally, the dynamic forward star as it is designed cannot support internal relocation of a node u (with adjacent edges). Simply inserting some new adjacent edges causes u to be moved again to the end of the array, rendering the previous relocation attempt futile.

A packed-memory graph (PMG) representation, due to its memory management scheme, achieves great locality of references at the expense of a small update time overhead (but fast enough to be used in practice). No update operation can implicitly alter the relative order of nodes and edges, therefore relative elements are always stored in memory as close as possible. Finally, the elements can be efficiently reordered to favour the memory accesses of any algorithm.

4 Experiments

To assess the practicality of our new graph structure, we conducted a series of experiments on shortest path routing algorithms on real world large-scale transportation networks (European and US road networks) in static and dynamic scenarios. For the former, we use the query performance of the static forward star (FS) structure as a reference point, since FS stores all edges adjacent to a node in consecutive memory locations. For the latter, we considered mixed sequences of point-to-point shortest path queries and updates.

All experiments were conducted on an Intel(R) Core(TM) i5-2500K CPU @ 3.30GHz with a cache size of 6144Kb and 8Gb of RAM. Our implementations were compiled by GCC version 4.4.3 with optimization level 3. The road networks for our experiments were acquired from [7,8] and consist of the road networks of Italy, Germany, Western US and Central US. Edge weights represent travel distances. Experiments with travel times exhibited similar relative performance.

4.1 Algorithms

We implemented the full set of shortest path algorithms considered in [11]. These algorithms are based on the well known Dijkstra's algorithm, and its Bidirectional and A^* variants. Given a source node s and a target node t, the A^* (or goal-directed) variant modifies the priority of a node according to a monotone heuristic function $h_t : V \to \mathbb{R}$ which gives a lower bound estimate $h_t(u)$ for the cost of a path from a node u to t. An example for h_t is the Euclidean distance between two nodes. The A^* algorithm can also be used in a bidirectional manner, using a symmetric heuristic function h_s.

The key contribution in [11] is a highly effective heuristic function for the A^* algorithm using the triangle inequality theorem and precomputed distances to a few important nodes (*landmarks*), resulting in the so-called ALT algorithm. During a query, ALT computes the lower bounds in constant time using the precomputed shortest distances to landmarks with the triangle inequality. The efficiency of ALT depends on the initial selection of landmarks. In [11], two approaches are used as the quitting criteria of the bidirectional A^* algorithm. In the *symmetric approach* the search stops when one of the searches is about to settle a node v for which $d(s,v)+h_t(v)$ or $d(v,t)+h_s(v)$ is larger than the shortest path distance found so far. In the *consistent approach*, the heuristic functions H_s and H_t are used instead of h_s and h_t, such that $H_s(u) + H_t(u) = c$, where c is a constant. These are defined as either *average heuristic functions* $H_t(u) = -H_s(u) = \frac{h_t(v)-h_s(v)}{2}$ or *max heuristic functions* $H_t(v) = \max\{h_t(v), h_s(t) - h_s(v) + b\}$ and $H_s(v) = \min\{h_s(v), h_t(s) - h_t(v) + b\}$, where b is a constant.

In summary, our performance evaluation consists of the following algorithms:

- D: Dijkstra's algorithm.
- B: Bidirectional variant of Dijkstra's algorithm.
- AE: A^* search with Euclidean lower bounds.
- BEM: Bidirectional AE with the max heuristic function.
- BEA: Bidirectional AE with the average heuristic function.
- AL: Regular ALT algorithm.
- BLS: Symmetric bidirectional ALT.
- BLM: Consistent bidirectional ALT with the max heuristic function.
- BLA: Consistent bidirectional ALT with the average heuristic function.

Each of these algorithms was implemented once, supporting interchangeable graph representations in order to minimize the impact of factors other than the performance of the underlying graph structures. All structures (ADJ, DynFS, PMG) were implemented under an abstraction layer, having as target to keep only the essential core parts different, such as the accessing of a node or an edge. Thus, the only factor differentiating the experiments is the efficiency of accessing and reading, as well as inserting or deleting nodes and edges in each structure. Clearly, the abstraction layer yields a small performance penalty, however, it provides the opportunity to compare all graph structures in a fair, uniform manner.

4.2 Results

Static Performance. We start by analyzing and comparing the real-time performance of the aforementioned algorithms on a static scenario, i.e., consisting only of shortest path queries. Following [11], we used two performance indicators: (a) the shortest path computation time (machine-dependent), and (b) the *efficiency* defined as the number of nodes on the shortest path divided by the number of the settled nodes by the algorithm (machine-independent).

In each experiment, we considered 10000 shortest path queries. For each query, the source s and the destination t were selected uniformly at random among all nodes. In our experiments with the ALT-based algorithms (AL, BLS, BLM, BLA) we used at most 16 landmarks, as it is the case in [11]. The query performance can be increased by adding more well-chosen landmarks.

Our experimental results on the road network of Central US are shown in Table 2 (results on the other road networks are reported in [13]). For each algorithm and each graph structure, we report the average running times in ms. We also report the space consumption for each graph structure. As expected, our experiments confirm the ones in [11] regarding the relative performance of the evaluated algorithms, with BLA being the best (see [13] for an explanation).

The major outcome of our experimental study is that there is a clear speedup of PMG over ADJ and DynFS in all selected algorithms. This is due to the fact that, at the expense of a small space overhead, the packed-memory graph achieves greater locality of references, less cache misses, and hence, better performance during query operations. In our experiments, PMG is roughly 25% faster than ADJ and 10% faster than DynFS. In addition, its performance is very close to the optimal performance of the static forward star (FS), taking into

Table 2. Running times (ms) in the road network of Central US; Eff ≡ Efficieny

Central US	$n =$ 14,081,816 $m =$ 34,292,496					
	ADJ (3.054Gb)	**DynFS** (3.269Gb)	**PMG** (3.297Gb)	**FS** (3.054Gb)	Eff. (%)	Eff. (%) [11]
D	4,474.65	3,418.06	2,864.33	2,766.28	0.07	0.09
B	3,012.38	2,303.91	2,048.01	1,963.67	0.11	0.14
AE	2,055.29	1,522.24	1,414.35	1,370.49	0.17	0.10
BEM	1,881.84	1,448.47	1,274.99	1,236.90	0.19	–
BEA	1,853.45	1,429.97	1,237.39	1,208.97	0.20	0.14
AL	223.26	173.62	161.93	151.34	3.75	1.87
BLS	257.47	205.35	176.84	172.46	3.67	2.02
BLM	211.50	172.11	161.27	157.41	7.43	3.27
BLA	146.51	116.88	108.401	104.88	10.02	3.87

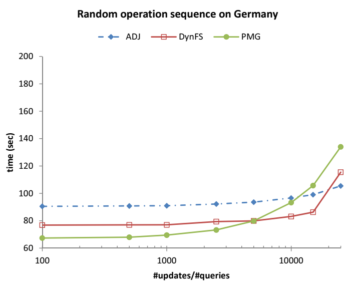
consideration that PMG is a dynamic structure. It is roughly 3% slower in the larger graph of Central US (denser PMG). Note that the denser the PMG is, the smaller size it has, the more it resembles an FS and the more it matches its query performance. Hence, if we take FS as a reference point for query performance in static scenarios, then PMG is the dynamic structure closest to it.

The space overhead of PMG is such because we chose its node and edge densities in a way that the sizes of the node and edge arrays are equal to the next power of 2 of the space needed. Note that the node and edge densities can be fine-tuned according to our expectation of future updates and the particular application. However, this is not our prime concern in this experimental study.

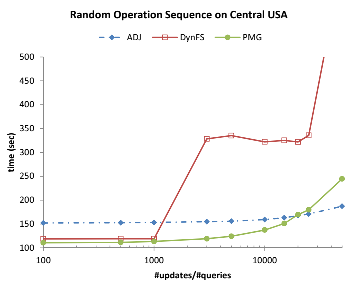
Dynamic Performance. We report results on random and realistic sequences of operations using the three structures ADJ, DynFS, PMG. Further results with dynamic operations and internal node relocations are reported in [13].

Random Sequence of Operations. We have compared the performance of the graph structures on sequences of random, uniformly distributed, mixed operations as a typical dynamic scenario. These sequences contain either random shortest path queries (BLA) or random updates (edge insertions and deletions). All sequences contain the same amount of shortest path queries (1000 queries) with varying order of magnitude of updates in the range $[10^5, 5 \times 10^7]$. To have the same basis of comparison, the same sequence of shortest path queries is used in all experiments. The update operations are chosen at random between edge insertions and edge deletions. When inserting an edge, we do not consider this edge during the shortest path queries, since we do not want insertions to have any effect on them. In a deletion operation, we select at random a previously inserted edge to remove. We remove no original edges, since altering the original graph structure between shortest path queries would yield non-comparable results.

The experimental results are reported in Figure 2, where the horizontal axis (log-scale) represents the ratio of the number of updates and the number of



(a)



(b)

Fig. 2. Running times on mixed sequences of operations consisting of 1000 queries and updates of varying length in $[10^5, 5 \times 10^7]$

queries, while the vertical axis represents the total time for executing the sequence of operations. The experiments verify our previous findings. While the running times of the queries dominate the running times of the updates, PMG maintains a speed-up over ADJ and DynFS. In the road network of Germany (resp. Central US), the number of updates should be at least 5000 (resp. 20000) times more than the number of (BLA) queries in order for PMG to be inferior to DynFS, and at least 10000 more to be inferior to ADJ. DynFS manages to have good running times in Germany, which is smaller in size, and hence more memory space is available for allocation. However, the running times of DynFS are hindered by the blow-up in its size when it operates on a larger graph, which is apparent with the Central US road network.

Realistic Sequence of Operations. In order to compare the graph structures in a typical practical scenario, we have measured running times for the preprocessing stage of Contraction Hierarchies(CH) [10]. The CH preprocessing stage iteratively removes nodes and their adjacent edges from the graph, and shortcuts their neighbouring nodes if the removed edges represent shortest paths (this is examined by performing consecutive shortest path queries). We have recorded the sequence of operations (shortest path queries, edge insertions, node and edge deletions) executed by the preprocessing routine of the CH code [5], and given it as input to our three data structures. This is a well suited realistic example, since it concerns one of the most successful techniques for route planning requiring a vast amount of shortest path queries and insertions of edges. Note that CH treats deletions of nodes and edges virtually, hence they are not included in our sequence of operations. We used the aggressive variant [5] of the CH code, since it provides the most efficient hierarchy. The results are shown in Table 3.

On all three graph structures, insertions can be processed so fast that have a minimum effect on the total running time of this sequence, even though they dominate (by a factor of 6) the shortest path queries. On the other hand, shortest

Table 3. Total time for processing 3,615,855 shortest path queries and 21,342,855 edge insertions, generated during CH preprocessing on Germany

Germany	Total Time (sec)
ADJ	32,194.4
DynFS	19,938.2
PMG	17,087.7

path queries are the slowest operations on this sequence, and since our graph structure has the best performance on them, it can easily outperform the two other graph structures. Our experiments also show that the use of PMG could improve the particular stage of the CH preprocessing by at least 14%.

Acknowledgements. We would like to thank Daniel Delling for many fruitful and motivating discussions, and Kostas Tsichlas for introducing us to the cache-oblivious data structures.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms and Applications. Prentice Hall, Englewood Cliffs (1993)
2. ARRIVAL Deliverable D3.6. Improved Algorithms for Robust and Online Timetabling and for Timetable Information Updating. ARRIVAL Project (March 2009), `http://arrival.cti.gr/uploads/3rd_year/ARRIVAL-Del-D3.6.pdf`
3. Bauer, R., Delling, D.: SHARC: Fast and robust unidirectional routing. ACM Journal of Experimental Algorithmics 14 (2009)
4. Bender, M.A., Demaine, E., Farach-Colton, M.: Cache-Oblivious B-Trees. SIAM Journal on Computing 35(2), 341–358 (2005)
5. Contraction Hierarchies source code, `http://algo2.iti.kit.edu/routeplanning.php`
6. Delling, D., Goldberg, A.V., Nowatzyk, A., Werneck, R.F.: PHAST: Hardware-Accelerated Shortest Path Trees. In: IPDPS 2011. IEEE (2011)
7. 9th DIMACS Implementation Challenge – Shortest Paths, `http://www.dis.uniroma1.it/challenge9/index.shtml`
8. 10th DIMACS Implementation Challenge – Graph Partitioning and Graph Clustering, `http://www.cc.gatech.edu/dimacs10/`
9. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: Proc. 40th IEEE FOCS 1999, pp. 285–297 (1999)
10. Geisberger, R., Sanders, P., Schultes, D., Delling, D.: Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 319–333. Springer, Heidelberg (2008)
11. Goldberg, A.V., Harrelson, C.: Computing the Shortest Path: A^* Search Meets Graph Theory. In: Proc. SODA, pp. 156–165 (2005)
12. Goldberg, A.V., Kaplan, H., Werneck, R.F.: Reach for A^*: Shortest Path Algorithms with Preprocessing. DIMACS 74, 93–139 (2009)
13. Mali, G., Michail, P., Paraskevopoulos, A., Zaroliagis, C.: A New Dynamic Graph Structure for Large-Scale Transportation Networks, Technical Report eCOMPASS-TR-005, eCOMPASS Project (October 2012), `http://www.ecompass-project.eu/?q=node/135`
14. Schultes, D.: Route Planning in Road Networks. PhD Dissertation, University of Karlsruhe (2008)

Capacitated Rank-Maximal Matchings

Katarzyna Paluch

Institute of Computer Science, University of Wrocław, Poland

Abstract. We consider capacitated rank-maximal matchings. Rank-maximal matchings have been considered before and are defined as follows. We are given a bipartite graph $G = (\mathcal{A} \cup \mathcal{P}, \mathcal{E})$, in which $\mathcal{A}$ denotes applicants, $\mathcal{P}$ posts and edges have ranks – an edge (a,p) has rank i if p belongs to (one of) a's ith choices. A matching M is called *rank-maximal* if the largest possible number of applicants is matched in M to their first choice posts and subject to this condition the largest number of appplicants is matched to their second choice posts and so on. We give a combinatorial algorithm for the capacitated version of the rank-maximal matching problem, in which each applicant or post v has capacity $b(v)$. The algorithm runs in $O(\min(B, C\sqrt{B})m)$ time, where C is the maximal rank of an edge in an optimal solution and $B = \min(\sum_{a\in\mathcal{A}} b(a), \sum_{p\in\mathcal{P}} b(p))$ and n, m denote the number of vertices/edges respectively. (B depends on the graph, however it never exceeds m.) The previously known algorithm [11] for this problem has a worse running time of $O(Cnm \log(n^2/m) \log n)$ and is not combinatorial –it is based on a weakly polynomial algorithm of Gabow and Tarjan using scaling. To construct the algorithm we use the generalized Gallai-Edmonds decomposition theorem, which we prove in a convenient form for our purposes. As a by-product we obtain a faster (by a factor of $O(\sqrt{n})$) algorithm for the Capacitated House Allocation with Ties problem.

1 Introduction

In this paper we consider capacitated rank-maximal matchings. The *Rank-Maximal Matchings problem* is defined as follows. We are given a bipartite graph $G = (\mathcal{A} \cup \mathcal{P}, \mathcal{E})$, in which $\mathcal{A}$ denotes applicants, $\mathcal{P}$ posts and edges have ranks – an edge (a,p) has rank i if p belongs to (one of) a's ith choices. Our task is to compute a rank-maximal matching of G, where a *matching* is a set of edges, no two of each share a vertex and a matching M is called ***rank-maximal*** if the largest possible number of applicants is matched in M to their first choice posts and subject to this condition the largest number of appplicants is matched to their second choice posts and so on. In the *Capacitated Rank-Maximal Matchings problem* each applicant a can have many - up to $b(a)$ posts and each post p can be occupied by many - up to $b(p)$ applicants. In other words we are also given a function $b : V = \mathcal{A} \cup \mathcal{P} \to N$ and we want to find a b-matching M. $M \subseteq \mathcal{E}$ is a ***b-matching*** of G if each vertex v has at most $b(v)$ edges of M incident with it. Let r denote the highest rank of an edge in graph G. For each b-matching M

we define its ***signature*** as an r-tuple $(x_1, x_2, \ldots, x_r)$ such that x_i denotes the number of edges having rank i in M. We define an r-ordering on r-tuples: We say that $(x_1, x_2, \ldots, x_r) >_r (y_1, y_2, \ldots, y_r)$ if there exists j $(0 \leq j < r)$ such that for each $k \leq j$ we have $x_k = y_k$ and $x_{j+1} > y_{j+1}$. Using the notion of a signature we will say that b-matching M is rank-maximal if it has the largest signature under r-ordering among all b-matchings of G.

Previous Results. Rank-maximal matchings were first considered by Irving in [6], who devised an algorithm running in $O(n^2c^3)$ time for the version in which there are no ties and c denotes the upper bound on the length of every preference list. In [7] a combinatorial algorithm is given that runs in time $O(\min(n, C\sqrt{n})m)$, where n and m denote the number of vertices and edges in the graph and C the maximal rank of an edge in an optimal solution. Notice that C can be of order m. (In [7] the running time of the algorithm is said to be $O(\min(n+C, C\sqrt{n})m)$, but in fact it is $O(\min(n, C\sqrt{n})m)$.) In [12] Michail solves the rank-maximal matchings problem by using maximum weight matchings and obtains an $O(\min(n+C, C\sqrt{n})m)$ algorithm. An algorithm for the Capacitated Rank-Maximal matchings problem was given by Mehlhorn and Michail in [11]. It runs in $O(Cnm\log(n^2/m)\log n)$ time and is based on a weakly polynomial algorithm of Gabow and Tarjan using scaling.

New Results. We give a combinatorial algorithm for computing rank-maximal b-matchings that runs in $O(\min(B, C\sqrt{B})m)$ time, where C is the maximal rank of an edge in an optimal solution and $B = \min\{\sum_{a\in\mathcal{A}} b(a), \sum_{p\in\mathcal{P}} b(p)\}$ and m denotes the number of edges in the graph. This algorithm is similar in spirit to that in [7]. However, the extension is not straightforward. In particular we have to generalize the properties of the Gallai-Edmonds decomposition to b-matchings. For computing maximum cardinality b-matchings we use the fast algorithm of Gabow ([5]). Let us notice that for the one-to-many version of the problem (i.e. each applicant has capacity one and each post has an arbitrary capacity) we obtain the same running time as for the one-to-one version. (Observe that in the one-to-many version $B = O(n)$.) One-to-many version of the problem has applications in allocating students to schools for example. As a by-product we obtain a faster by a factor of $O(\sqrt{n})$ algorithm for the Capacitated House Allocation with Ties problem presented by Manlove and Sng in [10] and we sketch an algorithm for the capacitated version of the Bounded Unpopularity Matchings problem considered in [4]. (We are able to do so because popular matchings and bounded unpopularity matchings can be thought of as variations of rank-maximal matchings.)

Related Work. Matchings with ranked edges have been long known in the economics literature, see [14] for example. Various other types of matchings with one-sided preferences have also been studied, for example popular, bounded unpopularity, Pareto-optimal (see [1],[2],[3], [4], [9], [10], [13] for some of them).

2 Preliminaries and the Gallai-Edmonds Decomposition Theorem

In this section we recall the Gallai-Edmonds Decomposition theorem for b-matchings which is presented in Chapter 10 of [8]. We prove it in a somewhat different version.

Let $G = (V, E)$ be a simple, undirected, loopless graph (not necessarily bipartite), b – a function $V \to N$, B – an edge subset of E and M – a b-matching of G.

An edge belonging to B will be referred to as a ***B-edge*** and an edge not belonging to B as a ***non-B-edge***. We call a vertex v ***unsaturated*** with respect to a b-matching M if v has less than $b(v)$ M-edges incident with it and we call v ***saturated*** (wrt M) if v has $b(v)$ M-edges incident with it. ***An alternating (with respect to B) path*** P is any sequence of vertices $(v_1, v_2, \ldots, v_k)$ such that edges on P are alternately B-edges and non-B-edges and no edge occurs on P more than once and $v_1 \neq v_k$. ***An alternating (with respect to B) cycle*** C has the same definition as an alternating wrt B path except that $v_1 = v_k$ and additionally $(v_{k-1}, v_k) \in B$ iff $(v_1, v_2) \notin B$. Let us note that both an alternating cycle and path can go through a given vertex more than once. We will sometimes treat an alternating path or cycle as an edge set and sometimes as a sequence of edges. An alternating (wrt a b-matching M) path is called ***augmenting (wrt M)*** if it begins and ends with a non-M-edge and if it begins and ends with an unsaturated (in M) vertex.

For any given two vertex sets $C, D \subseteq V$, an edge connecting a vertex in C with a vertex in D will be called a ***CD edge*** and a vertex in C will be called a ***C vertex***. For any edge set E', $deg_{E'}(v)$ denotes the number of E'-edges incident with v. For two edge sets $B_1, B_2 \subseteq E$, the symmetric difference $B_1 \oplus B_2$ denotes $(B_1 \setminus B_2) \cup (B_2 \setminus B_1)$.

The Gallai-Edmonds decomposition theorem for b-matchings is easier to formulate if we use the notion of b-edgesets and optimal b-edgesets defined as follows.

Definition 1. *Any $B \subseteq E$ is called a **b-edgeset**. Let $\delta(B) = \sum_{v \in V} |deg_B(v) - b(v)|$. We will say that B is an **optimal b-edgeset** if there exists no $B' \subseteq E$ such that $\delta(B') < \delta(B)$.*

If B is an optimal b-edgeset, then it may happen that $deg_B(v) > b(v)$. Hence not every optimal b-edgeset is a b-matching. However it is easy to see that every maximum b-matching is an optimal b-edgeset.

For any b-edgeset B we will say that a vertex v is ***deficient (in B)*** if $deg_B(v) < b(v)$ and ***overloaded (in B)*** if $deg_B(v) > b(v)$. We partition vertices V of G into four sets E, O, I, U as follows. Vertex v is in E iff there exists an optimal b-edgeset such that v is deficient in it and in no optimal b-edgeset is v overloaded. Vertex v is in O iff there exists an optimal b-edgeset such that v is overloaded in it and in no optimal b-edgeset is v deficient. Vertex v is in I iff there exists an optimal b-edgeset in which v is deficient and an optimal b-edgeset in which v is overloaded. Vertex v is in U if it is not an E, O or I vertex.

First we show that the above partition of the set of vertices can be computed efficiently. For a given b-matching M and vertex v we will say that path P is ***good for v (with respect to M)*** if *(i)* P is alternating wrt M, *(ii)* P begins at an unsaturated in M vertex and ends on v and *(iii)* the beginning edge of P incident with the unsaturated vertex (if P contains at least one edge) is a non-M-edge.

Theorem 1. *Let M denote any maximum cardinality b-matching. Then*

- *vertex v is an I vertex iff there exists an even-length good path for v and there exists an odd-length good path for v,*
- *vertex v is an E vertex iff v is not an I vertex and there exists an even-length good path for v,*
- *vertex v is an O vertex iff v is not an I vertex and there exists an odd-length good path for v,*
- *vertex v is a U vertex iff there does not exist a good path for v,*

where all considered good paths are wrt M.
If graph G is bipartite then it does not contain an I vertex.

Proof. It suffices to show that there exists an even-length (corr. odd-length) good path for v wrt M iff v is deficient (resp. overloaded) in some optimal b-edgeset.

First suppose that there exists an even-length (corr. odd-length) good path P for v wrt M. Assume further that P starts at x and ends at v. Let B denote $M \oplus P$. For every vertex $w \notin \{x, v\}$ we have $deg_B(w) = deg_M(w)$. For vertices x, v we have $deg_B(x) = deg_M(x) + 1$ and $deg_B(v) = deg_M(v) - 1$ (In the case of an odd-length P we have correspondingly $deg_B(x) = deg_M(x) + 1$ and $deg_B(v) = deg_M(v) + 1$.) Therefore $\delta(B) = \delta(M)$ and thus B is an optimal b-edgeset. In the case of an odd-length P, we can notice that it cannot happen that v is deficient in M, because then P would be augmenting wrt M, contradicting the fact that M is a maximum size b-matching. Thus v is saturated in M. Additionally B is such that v is deficient (corr. overloaded) in it.

Conversely suppose now that v is deficient (resp. overloaded) in some optimal b-edgeset B. $M \oplus B$ can be partitioned into a set S of edge-disjoint alternating paths and cycles (see [8] for example). Let us partition $M \oplus B$ in a maximal way i.e. so that S does not contain two alternating paths P_1, P_2 such that $P_1 \cup P_2$ is an alternating path or cycle. Suppose that $S = \{P_1, P_2, \ldots, P_{k_1}, C_1, C_2, \ldots, C_{k_2}\}$, where each P_i denotes an alt. path and each C_i an alt. cycle. Let $M_1 = M \oplus P_1$ and for each i, $2 \leq i \leq k_1$ let $M_i = M_{i-1} \oplus P_i$. Then we can notice that for each vertex w we have either $deg_M(w) < deg_{M_1}(w) \leq deg_{M_2}(w) \leq \ldots \leq deg_{M_{k_1}}(w) = deg_B(w)$ or $deg_M(w) \geq deg_{M_1}(w) \geq deg_{M_2}(w) \geq \ldots \geq deg_{M_{k_1}}(w) = deg_B(w)$. It is because $M \oplus B$ has been partitioned in a maximal way and thus for each vertex w we have that all ending or beginning edges of paths $P_1, \ldots, P_{k_1}$ incident with w are solely B-edges (and thus non-M-edges) or solely non-B-edges (and thus M-edges).

Case 1: v is deficient in B. If v is deficient in M, we are done (as there exists a zero-length good path for v.) Hence let us assume that v is not deficient in

M. Since v is not deficient in M, in S there exists a path P beginning at some vertex w and ending on v with an M-edge.

Claim. P is good for v.

Proof. Since v is saturated in M and P ends with an M-edge at v we have $b(v) = deg_M(v) > deg_{M \oplus P} \geq deg_B(v)$. Therefore $deg_B(v) < deg_{B \oplus P}(v) \leq b(v)$. Suppose that P is not good for v.

If P begins with an M-edge, then we have $b(w) \geq deg_M(w) > deg_{M \oplus P}(w) \geq deg_B(w)$. Thus $deg_B(w) < deg_{B \oplus P}(w) \leq b(w)$. This however would mean that $\delta(B) = \delta(B \oplus P) + 2$ and thus that B is not an optimal b-edgeset. A contradiction.

If w is not deficient in M and P begins with a non-M-edge, then we have $b(w) = deg_M(w) < deg_{M \oplus P}(w) \leq deg_B(w)$. (If w is not deficient in M, then it is saturated in M.) Therefore $deg_B(w) > deg_{B \oplus P}(w) \geq b(w)$ and again we obtain that $\delta(B) = \delta(B \oplus P) + 2$ and thus that B is not an optimal b-edgeset. □

Since P is good for v and ends with an M-edge, P is even-length.

Case 2: v is overloaded in B. v is not overloaded in M (M does not contain overloaded vertices). Since v is not overloaded in M, in S there exists an alternating path P beginning at some vertex w and ending on v with a non-M-edge.

Claim. P is good for v.

Proof. Since v is overloaded in B, we have $deg_B(v) > deg_{B \oplus P}(v) \geq b(v)$. Further on the proof proceeds as in Claim 2. □

Since P is good for v and ends with a non-M-edge, P is odd-length.

Finally, let us suppose that graph G is bipartite and there exists a vertex v in G that is an I vertex. Therefore there exist good paths P_1, P_2 for v wrt M such that one is odd-length, say P_1, and the other – P_2, is even-length. Suppose that P_1 begins at an unsaturated vertex w_1 and P_2 at an unsaturated vertex w_2. We can observe that neither $v = w_1$ nor $v = w_2$, because it would mean that P_1 is an augmenting path, which would contradict M being maximum. Let v' be the first common vertex of P_1 and P_2 when we start going along P_1 and P_2 from w_1 and w_2 respectively. Let P'_i $(i = 1, 2)$ denote the subpath of P_i from w_i till the first occurence of v' on P_i. We can notice that it cannot happen that $|P'_1| + |P'_2|$ is odd because then $P'_1 \cup P'_2$ would be an augmenting path. Hence $|P'_1| + |P'_2|$ is even and thus $|P_1 \setminus P'_1| + |P_2 \setminus P'_2|$ is odd. Let P denote the multi edge set $(P_1 \setminus P'_1) \cup (P_2 \setminus P'_2)$ i.e. P contains two copies of edge e if it occurs in both paths. Then for every vertex u, we have that $deg_P(u)$ is even. Let now P' denote $P \setminus ((P_1 \setminus P'_1) \cap (P_2 \setminus P'_2))$. We obtain that $|P'|$ is odd and for every vertex u, $deg_{P'}(u)$ is even. Therefore P' can be partitioned into edge-disjoint (simple) cycles, one of which at least will have to be of odd-length, a contradiction. □

Next we give some properties of vertices and edges that can be derived from the Gallai-Edmonds decomposition theorem.

Lemma 1. *Let M be a maximum b-matching and E, O, I, U vertex sets defined as above. Then*

– *every O vertex and every U vertex is saturated in M,*
– *every EE edge and every EU edge is an M-edge,*
– *no OO edge nor OU edge is an M-edge,*
– *the cardinality of M equals $\sum_{v \in O} b(v) + |EE \text{ edges}| + |EU \text{ edges}| + |(UU \text{ edges}) \cap M| + |(II \text{ edges}) \cap M| + |(EI \text{ edges}) \cap M|$.*

Proof. Every O and U vertex is saturated in M by definition.

Consider an E vertex v. By definition and Theorem 1, we have that there exists a good even-length path P from an unsaturated vertex, say u, to v and there does not exist a good odd-length path from an unsaturated vertex to v. Hence P ends on v with an M-edge and all occurences of v on P are at an even-length distance from u. Therefore we can assume that v occurs on P only once - at the end; if it occurs more than once, we take a subpath of P from u till the first occurence of v. Suppose next we have a non-M-edge $e = (v, w)$. Then $P \cup e$ is an odd-length good path for w. Therefore w is either an O vertex or an I vertex. This way we obtain that every EE edge and every EU edge is an M-edge.

Consider an O vertex v. By definition and Theorem 1, there exists a good odd-length path P from an unsaturated vertex to v and there does not exist a good even-length path for v. Hence P ends on v with a non-M-edge and as before we can assume that v occurs on P only once – at the end. Suppose we have an M-edge $e = (v, w)$. Then $P \cup e$ is an even-length good path for w. Therefore w is either an E vertex or an I vertex, which means that no OO edge nor OU edge is an M-edge.

Edge sets: (1) edges incident with a vertex in O, (2) EE edges, (3) EU edges, (4) UU edges, (5) II edges, (6) EI are pairwise disjoint, therefore the cardinality of M equals $\sum_{v \in O} b(v) + |EE \text{ edges}| + |EU \text{ edges}| + |(UU \text{ edges}) \cap M| + |(II \text{ edges}) \cap M| + |(EI \text{ edges}) \cap M|$. □

3 Algorithm for Rank-Maximal *b*-Matchings

In this section we present a combinatorial algorithm for the Capacitated Rank-Maximal Matchings problem.

Let G_i denote graph $(V, \mathcal{E}_1 \cup \mathcal{E}_2 \cup \dots \mathcal{E}_i)$, where $\mathcal{E}_j$ denotes edges of rank j.

Algorithm RMBM given below runs in phases. In phase i only edges of rank at most i are present and the aim of phase i is to compute a rank-maximal b-matching of G_i. In phase 1 this task is fairly easy. A rank-maximal b-matching M_1 of G_1 is just a maximum b-matching of G_1 and we have several algorithms at hand for computing it. We choose the one by Gabow [5]. In phase 2 when we want to compute a rank-maximal b-matching M_2 of G_2, we would like to start from M_1 and augment it somehow so that we obtain M_2. We cannot however simply add edges of rank 2 and augment M_1 as in the process the number of rank 1 edges might decrease i.e. we might end up with M_2 having a smaller number of rank 1 edges than M_1. Here Theorem 1 and Lemma 1 come in handy as thanks to them we will know which vertices must be saturated in each rank-maximal

b-matching of G_1, which edges must be present in each rank-maximal b-matching of G_1 and which edges cannot occur in M_1. If some vertex v is saturated in each rank-maximal matching of G_1, then there is no point in adding edges of rank higher than 1 incident with v. If throughout the algorithm vertex v will have only rank 1 edges incident with it, then the number of rank 1 edges incident with v will stay the same, because augmenting a given b-matching N along augmenting paths never decreases the number of N-edges incident with any vertex. By appropriate deletions of certain edges and moving the other ones to the sets S_i (set S_i will contain edges that are present in every rank-maximal b-matching of G_i), we are able to reduce the task of computing a rank-maximal b-matching of G_i to computing a maximum b_i-matching of G'_i, where b_i and G'_i are correspondingly an appropriately modified function b and graph G_i.

```
***Algorithm RMBM (short for rank-maximal $b$-matching)***

***Input:*** Graph $G = (\mathcal{A} \cup \mathcal{P}, \mathcal{E})$, a partition of $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2 \cup \ldots \cup \mathcal{E}_r$, a
function $b : \mathcal{A} \cup \mathcal{P} \to N$
***Output:*** a rank-maximal $b$-matching of $G$.

Let $G'_1 = G_1, M_0 = \emptyset, b_1 = b, S_0 = \emptyset$.
For $i = 1$ to $r$ do the following steps
    if $\mathcal{E}_i \neq \emptyset$, then
    1. Determine a max. $b_i$-matching $M_i$ in $G'_i$ by augmenting $M_{i-1} \setminus S_{i-1}$.
    2. Partition the vertices of $\mathcal{A} \cup \mathcal{P}$ into three disjoint sets: $E_i$, $O_i$, and
       $U_i$. Vertex $v$ belongs to $E_i(O_i)$ iff there exists a good even-length
       (odd-length) path for $v$ in $G'_i$ wrt $M_i$. Vertex $v$ belongs to $U_i$ iff there
       does not exist a good path for $v$ in $G'_i$ wrt $M_i$.
    3. Delete all edges incident with a vertex in $O_i \cup U_i$ from $\mathcal{E}_j, \forall j > i$.
       $O_i \cup U_i$ are the vertices that are saturated in every maximum $b_i$-
       matching of $G'_i$. Delete all edges in $G'_i$ connecting two vertices in
       $O_i$ or a vertex in $O_i$ with a vertex in $U_i$. These are the edges that
       are not used by any maximum $b_i$-matching of $G'_i$. Let $T_i = \{(v, w)$ :
       $(v, w)$ is an $E_iE_i$ or $E_iU_i$ edge of $G'_i\}$. $T_i$ contains the edges used by
       every maximum $b_i$-matching of $G'_i$. Let $S_i = S_{i-1} \cup T_i$. Remove $T_i$
       from $G'_i$. For every $v \in \mathcal{A} \cup \mathcal{P}$ let $b_{i+1}(v)$ denote $b_i(v) - deg_{T_i}(v)$. Add
       the edges in $\mathcal{E}_{i+1}$ to $G'_i$. Call the resulting graph $G'_{i+1}$.
    otherwise
        Let $b_{i+1} = b_i,\ M_i = M_{i-1},\ S_i = S_{i-1}$. Add the edges in $\mathcal{E}_{i+1}$ to
        $G'_i$. Call the resulting graph $G'_{i+1}$.
Output $M_r \cup S_{r-1}$.
```

We will show that the following invariants hold:

- every rank-maximal b-matching in G_i has all its edges in $G'_i \cup S_{i-1}$
- $M_i \cup S_{i-1}$ is a rank-maximal b-matching in G_i.

3.1 Correctness

We start with the following technical fact:

Fact 1. *For each $i, 1 \leq i \leq r$, $deg_{S_{i-1}}(v) = \sum_{j=1}^{i-1} deg_{T_j}(v) = b(v) - b_i(v)$.*

Next we give the following lemma, which proves that the edges that are deleted during phase $i+1$ do not belong to any rank-maximal b-matching of G_{i+1} and that the edges from S_i all belong to any rank-maximal b-matching of G_{i+1} provided that we keep the invariants until the end of phase i.

Lemma 2. *Suppose that every rank-maximal b-matching of G_i is of the form $M_i \cup S_{i-1}$, where M_i is a maximum b_i-matching of G'_i and S_{i-1} is as described in Algorithm RMBM. Then every rank-maximal b-matching of G_{i+1} is contained in $G'_{i+1} \cup S_i$ and the whole S_i is contained in every rank-maximal b-matching of G_{i+1}.*

Proof. We need to show that the edges that we removed in the $(i+1)$th phase of the algorithm do not belong to any rank-maximal b-matching of G_{i+1} and all the edges of S_i belong to every rank-maximal b-matching of G_{i+1}.

Let N_{i+1} be any rank-maximal b-matching of G_{i+1}. Then its signature is $(s_1, s_2, ..., s_i, s_{i+1})$. $N_i = N_{i+1} \cap \mathcal{E}_{\leq i}$ is a b-matching with signature $(s_1, s_2, ..., s_i)$ and is therefore, a rank-maximal b-matching of G_i. So, by the assumption, N_i is of the form $M_i \cup S_{i-1}$, where M_i is a b_i-maximum matching in G'_i. By Lemma 1, M_i does not use any edge of G_i connecting two vertices in O_i or a vertex in O_i with a vertex in U_i. Therefore N_{i+1} does not use them either. Also all E_iE_i and E_iU_i edges belong to M_i and hence they also belong to N_{i+1}. O_i and U_i-vertices are saturated in a b_i-matching M_i and thus they are also saturated in a b-matching N_i because by Fact 1 $deg_{S_{i-1}}(v) = b(v) - b_i(v)$. Therefore in N_{i+1} all O_i and U_i-vertices have only edges of $\mathcal{E}_{\leq i}$ incident with them and edges of $\mathcal{E}_{>i}$ incident with a vertex in $O_i \cup U_i$ can be safely deleted from G'_i.

So, N_{i+1} is contained in $G'_{i+1} \cup S_i$ and all the edges of $S_i = S_{i-1} \cup T_i$ belong to N_{i+1}. □

By deleting appropriate edges and moving others to sets S_i we can ensure that the number of edges of each smaller rank is preserved throughout the algorithm:

Lemma 3. *For every i, j such that $j > i$, the number of edges of rank at most i is the same in $M_i \cup S_{i-1}$ and $M_j \cup S_{j-1}$.*

Proof. Since M_j is obtained from M_i by successive augmentations, for every vertex v we have that the number of $(M_i \cup S_{i-1})$ edges incident with v is not less than the number of $(M_j \cup S_{j-1})$ edges incident with v. Hence, all vertices in U_i and O_i are saturated in b-matching $M_j \cup S_{j-1}$.

Since G'_j has

– no edges of rank greater than i incident with vertices in O_i and U_i,
– no E_iE_i or E_iU_i edges of rank at most i, as they are contained in S_i and thus in S_{j-1},
– no O_iU_i or O_iO_i edges of rank at most i,

$M_j \cup S_{j-1}$ has at least as many edges of rank at most i as $M_i \cup S_{i-1}$. $M_j \cup S_{j-1}$ cannot have more edges of rank at most i than $M_i \cup S_{i-1}$ because all the edges of rank $\leq i$ in $G'_j \cup S_{j-1}$ belong to $G'_i \cup S_{i-1}$, $S_{i-1} \subseteq S_j$, M_i is a maximum b_i-matching of G'_i and for each $v \in V$, $b_i(v) = b(v) - deg_{S_{i-1}}(v)$. □

Now we prove the correctness of Algorithm RMBM.

Theorem 2. *For every i, the following statements hold:*
(i) *Every rank-maximal b-matching in G_i is of the form $M_i \cup S_{i-1}$, where M_i is a maximum b_i-matching in G'_i;*
(ii) *$M_i \cup S_{i-1}$ is a rank-maximal b-matching in G_i.*

Proof. We prove this by induction on i. Since all edges in G_1 have the same rank, a rank-maximal b-matching in G_1 is the same as a maximum b-matching. Since M_1 is a maximum b-matching in G_1 and $S_0 = \emptyset$ and $b_1 = b$, both statements hold for $i = 1$.

Let us now assume that the statements are true for i. We will prove they are also true for $i+1$. Since $M_i \cup S_{i-1}$ is a rank-maximal b-matching in G_i, its signature is $(s_1, \ldots, s_i)$. Suppose the signature of $M_{i+1} \cup S_i$ is $(r_1, \ldots, r_i, r_{i+1})$. By Lemma 3, we know that for every k between 1 and i, $\sum_{j=1}^{k} s_i = \sum_{j=1}^{k} r_i$.

It follows that the signature of $M_{i+1} \cup S_i$ is $(s_1, \ldots, s_i, r_{i+1})$ for some $r_{i+1} \leq s_{i+1}$.

By the induction hypothesis, every rank-maximal b-matching of G_i is of the form $M'_i \cup S_{i-1}$, where M'_i is a maximum b_i-matching of G'_i. Hence, by Lemma 2, any rank-maximal b-matching of G_{i+1} is contained in $G'_{i+1} \cup S_i$. Thus there is a matching of cardinality $s_1 + \ldots + s_i + s_{i+1}$ in $G'_{i+1} \cup S_i$. Since M_{i+1} is a maximum b_{i+1}-matching in G'_{i+1} and $deg_{S_i}(v) = b(v) - b_{i+1}$, the cardinality is at least $s_1 + \ldots s_i + s_{i+1}$. Thus $r_{i+1} = s_{i+1}$.

We get that $M_{i+1} \cup S_i$ is a rank-maximal matching in G_{i+1}. We must also show that every rank-maximal matching of G_{i+1} is of the form $M'_{i+1} \cup S_i$, where M'_{i+1} is a maximum b_{i+1}-matching in G'_{i+1}. Let N_{i+1} be any rank-maximal matching of G_{i+1}. By the induction hypothesis and Lemma 2, we know that N_{i+1} is contained in $G'_{i+1} \cup S_i$ and the whole S_i is contained in N_{i+1}. Let $t = |S_i|$. $N_{i+1} \setminus S_i$ is a b_{i+1}-matching of G'_{i+1} and has cardinality $s_1 + s_2 + \ldots + s_{i+1} - t$, which is equal to the cardinality of M'_{i+1}, which is a maximum b_{i+1}-matching of G'_{i+1}. Hence, $N_{i+1} \setminus S_i$ is also a maximum b_{i+1}-matching of G'_{i+1}. This completes the proof of the theorem. □

3.2 The Running Time of the Algorithm

Theorem 3. *Algorithm RMBM runs in $O(\min(B, C\sqrt{B})m)$ time, where C is the maximal rank of an edge in an optimal solution and*

$B = \min(\sum_{a \in \mathcal{A}} b(a), \sum_{p \in \mathcal{P}} b(p))$ and m denotes the number of edges in the graph.

Proof. During phase $i+1$ of Algorithm RMBM we first augment the b_{i+1}-matching $M_i \setminus S_i$ to the maximum b_{i+1}-matching M_{i+1}. Using the algorithm of Gabow [5], this takes time $O(\min(\sqrt{B}, |M_{i+1}| - |M_i| + 1) \cdot m)$. Next we compute the partition of the vertex set and then delete and move edges. This takes time $O(m)$. The total number of phases is r. We will show that the number of phases in which we augment the b_{i+1}-matching is at most C.

Let us notice that when we add edges in $\mathcal{E}_{i+1}$ to G'_i and $\mathcal{E}_i$ is a non-empty set, then we add only E_iE_i edges which means that in phase $i+1$ there will be at least one augmentation. It is so because of the following. Let (a,p) be an E_iE_i edge. Then in G'_i in phase i before some edges are deleted or moved to set S_i, there exist good even-length paths P_1 and P_2 for a and p. Let us notice that no edge of either P_1 or P_2 is deleted from G'_i in phase i, because P_1 and P_2 consist only of E_iO_i edges. Hence P_1 and P_2 are still present in G'_{i+1}. Because G'_{i+1} is bipartite, P_1 and P_2 are vertex-disjoint. Therefore $P_1 \cup (a,p) \cup P_2$ is an augmenting path wrt $(M_i \setminus S_i)$.

If $\mathcal{E}_{i+1}$ is empty, then phase $i+1$ takes constant time. Thus every phase in which $\mathcal{E}_{i+1}$ is nonempty, adds at least one augmentation and we finish when there are no more edges to add. Hence the last phase is phase C. □

4 Extensions

The *Capacitated House Allocation problem with Ties* (CHAT) is defined in [10] as follows. We are given a bipartite graph $G = (A \cup H, E)$, where $A = \{a_1, a_2, \ldots, a_{n_1}\}$ is the set of agents, $H = \{h_1, h_2, \ldots, h_{n_2}\}$ is the set of houses and the edges have ranks. An edge (a,h) has rank i if h belongs to (one of) a's ith choices. Also each house has a capacity $b(h)$ which indicates the number of agents that can be matched to it. Each agent a has capacity $b(a) = 1$. Given two b-matchings M, M', we say that an agent a *prefers* M to M' if either (i) a is unmatched in M' and matched in M or (ii) a is matched in M and M' and prefers the house he is matched with in M to the house he is matched with in M'. We say that a b-matching M is *popular* if there does not exist a b-matching M' such that the number of agents who prefer M' to M is greater than the number of agents who prefer M to M'. The goal of the problem is to establish whether a given instance admits a popular b-matching and find one if it exists.

In the algorithm presented in [10] an *EOU* labeling needs to be computed. To that end the authors build a graph $C(G_1)$ (where $G_1 = (A \cup H, E_1)$), which contains $b(h)$ clones of each house. As a result graph $C(G_1)$ has $O(n_1 m)$ edges and computing *EOU* labeling takes $O(n_1 m)$ time (m denotes $|E|$). Instead one can use the results from Theorem 1 and Lemma 1 and compute the labeling in $O(m)$ time. The modified algorithm runs in $O(\sqrt{B}m)$ time while the original one from [10] runs in $O((\sqrt{B} + n_1)m)$ time, where $B = \min(n_1, \sum_{h \in H} b(h))$.

The *Bounded Unpopularity Matchings problem* is considered in [4].

We can give an algorithm for the capacitated version of this problem as well as for the Capacitated House Allocation problem with Ties by slightly changing Algorithm RMBM.

Assume that we are given a graph $G = (\mathcal{A} \cup \mathcal{P}, \mathcal{E})$, a partition of $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2 \cup \ldots \cup \mathcal{E}_r$ and a function $b : \mathcal{A} \cup \mathcal{P} \to N$ such that $b(a) = 1$ for each $a \in \mathcal{A}$. Then as in [4] and [10] we add a dummy new vertex p_a to $\mathcal{P}$ for each agent a and an edge (a, p_a) to $\mathcal{E}_{r+1}$ and also put $b(p_a) = 1$ (i.e. p_a is a last resort post or house for a and edge (a, p_a) has rank $r + 1$). We will call the modified algorithm Algorithm BUBM (short for bounded unpopularity b-matchings). Algorithm BUBM is defined as Algorithm RMBM in which each $\mathcal{E}_i$ is replaced by $\tilde{\mathcal{E}}_i$ everywhere except for the input.

We define $\tilde{\mathcal{E}}_i$ as follows. $\tilde{\mathcal{E}}_1 = \mathcal{E}_1$. For $i > 1$ edge $(a, p) \in \tilde{\mathcal{E}}_i$ iff

- $(a, p) \in G$
- (a, p) is not deleted in phase j of Algorithm BUBM, where $j < i$,
- (a, p) does not belong to $\tilde{\mathcal{E}}_j$ for any $j < i$,
- there does not exist edge (a, p') having higher rank than (a, p) and satisfying the above conditions.

Proposition 1. *If graph G admits a popular b-matching then Algorithm BUBM outputs it after at most two phases. Otherwise Algorithm BUBM computes a bounded unpopularity b-matching having analogous properties as the one in [4].*

References

1. Abraham, D.J., Cechlárová, K., Manlove, D.F., Mehlhorn, K.: Pareto Optimality in House Allocation Problems. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 3–15. Springer, Heidelberg (2004)
2. Abraham, D.J., Chen, N., Kumar, V., Mirrokni, V.S.: Assignment Problems in Rental Markets. In: Spirakis, P.G., Mavronicolas, M., Kontogiannis, S.C. (eds.) WINE 2006. LNCS, vol. 4286, pp. 198–213. Springer, Heidelberg (2006)
3. Abraham, D.J., Irving, R.W., Kavitha, T., Mehlhorn, K.: Popular Matchings. SIAM J. Comput. 37(4), 1030–1045 (2007)
4. Huang, C.-C., Kavitha, T., Michail, D., Nasre, M.: Bounded Unpopularity Matchings. Algorithmica 61(3), 738–757 (2011)
5. Gabow, H.N.: An Efficient Reduction Technique for Degree-Constrained Subgraph and Bidirected Network Flow Problems STOC, pp. 448–456 (1983)
6. Irving, R.W.: Greedy matchings. Technical report TR-2003-136, University of Glasgow (April 2003)
7. Irving, R.W., Kavitha, T., Mehlhorn, K., Michail, D., Paluch, K.E.: Rank-maximal matchings. ACM Transactions on Algorithms 2(4), 602–610 (2006)
8. Lovasz, L., Plummer, M.D.: Matching Theory. Ann. Discrete Math., vol. 29. North-Holland, Amsterdam (1986)
9. Mahdian, M.: Random popular matchings. In: ACM Conference on Electronic Commerce, pp. 238–242 (2006)
10. Manlove, D.F., Sng, C.T.S.: Popular Matchings in the Capacitated House Allocation Problem. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 492–503. Springer, Heidelberg (2006)

11. Mehlhorn, K., Michail, D.: Network Problems with Non-Polynomial Weights and Applications (2005) (manuscript)
12. Michail, D.: Reducing rank-maximal to maximum weight matching. Theor. Comput. Sci. 389(1-2), 125–132 (2007)
13. Mestre, J.: Weighted Popular Matchings. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 715–726. Springer, Heidelberg (2006)
14. Roth, A.E., Postlewaite, A.: Weak versus strong domination in a market with indivisible goods. J. Math. Econom. 4, 536–546 (1977)

A Faster CREW PRAM Algorithm for Computing Cartesian Trees[⋆]

Chung Keung Poon and Hao Yuan

Department of Computer Science,
City University of Hong Kong, Kowloon, Hong Kong
{csckpoon,haoyuan}@cityu.edu.hk

Abstract. Cartesian tree is a fundamental data structure with many applications in the areas of data structures and string processing. In this paper, we study the construction of a Cartesian tree on a parallel computation model. We present a CREW PRAM algorithm that runs in $O(\log n)$ parallel time and has linear work and space. This improves upon the best previous result of Blelloch and Shun which takes $O(\log^2 n)$ time and linear work/space.

1 Introduction

The Cartesian tree of an array was defined by Vuillemin [9] as follows: Given an array $A[1 \dots n]$, the Cartesian tree of the array is a binary tree $C = (V, E)$ where each node $v \in V$ has two attributes Pos(v) and Val(v), and there is a one-to-one correspondence between a node v and an entry of A. Each node v corresponds to the Pos(v)-th entry in the array and Val(v) = A[Pos(v)]. A Cartesian tree C has the dual properties of being a binary search tree w.r.t. the Pos attributes and a min-heap w.r.t. the Val attributes. That is,

– for any node v, Pos(v) is bigger than the Pos attributes of any descendant in the left subtree of v and smaller than the Pos attributes of any descendant in the right subtree of v.
– for any node v, Val(v) is smaller than or equal to the Val attribute of any its descendant.

Cartesian trees have been studied intensively because of their importance in many data structural problems, where its most notable application is in the Range Minimum Query (RMQ) problem on an array. The Cartesian tree completely captures the information necessary to determine the solution for any range minimum query on an input array. It was shown that any range minimum query can be reduced to a Nearest Common Ancestor (NCA) query on the corresponding Cartesian tree of the array in constant time [4]. See the recent work of Yuan and Atallah [10] for more detailed discussion about RMQ. See also

⋆ This work is supported by the Research Grants Council of Hong Kong under grant 9041688 (CityU 124411).

the work of Demaine et al. [3], which presented a cache-oblivious algorithm to construct Cartesian trees. Demaine et al. [3] also showed that Cartesian trees can be defined on trees rather than arrays, to solve the bottleneck edge query problem on trees and undirected graphs.

Cartesian trees are also useful in the computation of suffix tree, a fundamental data structure in string processing, see [5]. Given a suffix array and the longest common prefix (LCP) information for pairs of suffixes that correspond to adjacent entries in the suffix array, one can apply an algorithm for computing a Cartesian tree to generate the suffix tree. See [6,2].

Gabow et al. [4] showed that the Cartesian tree of an array can be constructed in linear time by a simple stack-based sequential algorithm. Regarding parallel algorithms, Berkman et al. [1] presented a parallel algorithm that can generate a Cartesian tree for arrays with values from a fixed size domain. Iliopoulos and Rytter [6] gave an algorithm for general arrays but it requires $O(n \log n)$ work. Recently, Blelloch and Shun [2] designed two CREW PRAM algorithms that use $O(n)$ work and space. One algorithm has running time $O(\min\{h \log n, n\})$ where h is the height of the Cartesian tree. The other algorithm has $O(\log^2 n)$ time.

In this paper, we give a CREW PRAM algorithm that has an improved running time of $O(\log n)$ while keeping the work and space linear. In the next section, we introduce our notations and some preliminaries. After reviewing the $O(\log^2 n)$ time, $O(n)$ work and space algorithm of Blelloch and Shun [2] in section 3, we present our algorithm that has $O(\log n)$ time but non-optimal work and space in section 4. Then we improve it to have $O(n)$ work and space in section 5. Finally, we conclude the paper with a few open problems.

2 Preliminaries

Our input is an array $A[1 \dots n]$. Without loss of generality, we assume that all the values in the input array A are distinct. The Cartesian tree $C = (V, E)$ for array $A[1 \dots n]$ has node set $V = \{v_1, v_2, \dots, v_n\}$ and each v_i corresponds to the array entry $A[i]$. That is, $\mathrm{Pos}(v_i) = i$ and $\mathrm{Val}(v_i) = A[i]$. For convenience of presentation, we denote by $\mathrm{Parent}(u)$ the parent of node u in C. In actual implementation, we can represent the Cartesian tree structure by a pointer array $\mathrm{Parent}[1 \dots n]$ so that $\mathrm{Parent}[i] = j$ if $\mathrm{Parent}(v_i) = v_j$ and $\mathrm{Parent}[i] = 0$ if v_i is the root of the Cartesian tree. Such pointer array completely specifies the structure of the Cartesian tree.

Given a Cartesian tree C, we define its *left-spine* (resp. *right-spine*) as the sequence of nodes on the path from the root to the leftmost (resp. rightmost) node of C. Note that the nodes on a spine are sorted according to the Val attributes.

A Cartesian tree can be constructed by the following divide-and-conquer approach, which is used in, for example, [2] and [6]: *Split the input array into two halves, recursively construct a Cartesian tree for each half, and finally merge them into one Cartesian tree.* As the two recursive calls are independent, this framework can be readily parallelized. Clearly, how to perform the Cartesian tree merging efficiently is the key to designing an efficient algorithm for the original (Cartesian tree construction) problem.

In the rest of this paper, we will discuss several recursive algorithms following the above approach and refer to their recursion trees. We use symbols like x, y and z to refer to nodes of a recursion tree while reserving symbols like u, u_i, v and w for nodes of a Cartesian tree. Each node x in a recursion tree corresponds to a subarray of A in an obvious way. We denote by C_x the Cartesian tree of the subarray that corresponds to node x. Furthermore, we denote by L_x and R_x the left- and right-spine of C_x respectively. The symbol $|L_x|$ represents the number of nodes in L_x, etc.

In the sequel, we refer to the $O(\log^2 n)$ time, $O(n)$ work and space CREW PRAM algorithm of Blelloch and Shun [2] as the `Blelloch-Shun` algorithm.

3 The Blelloch-Shun Algorithm

In this section, we review the merging of Cartesian trees in the `Blelloch-Shun` algorithm. Consider an arbitrary node x in the recursion tree and let y and z be its left and right child. Suppose the Cartesian trees C_y and C_z have been constructed and the next task is to merge C_y and C_z to form C_x. Blelloch and Shun [2] showed that this can be done by "merging" the right-spine, R_y, of C_y with the left-spine, L_z,of C_z. Recall that R_y and L_z are two lists of nodes sorted according to their Val attributes, i.e., for any node u in R_y, $\text{Val}(u) > \text{Val}(\text{Parent}(u))$ and similarly for nodes in L_z. We merge them into one sorted list by updating their parent pointers appropriately.

Blelloch and Shun [2] showed that such merging, which only updates the Parent pointers of nodes in R_y and L_z, suffices to combine the two Cartesian trees into one. First, the min-heap property is maintained since we will only set the parent pointer of a node u to some node v with a smaller value, i.e., $\text{Val}(v) < \text{Val}(u)$. Second, consider a node u in R_y. All its descendants (including itself) have Pos attributes larger than any node in L_z. If the parent of u is changed to some node v in L_z, u becomes the left child of v and the descendants of the left subtree of v have Pos attributes smaller than $\text{Pos}(v)$. Furthermore, node v still has at most two children because the original left child of v must now change its parent to u or some node below u in R_y. Thus, the resultant tree is still a binary search tree w.r.t. the Pos attributes. Similar argument holds when a node in L_z changes its parent to some nodes in R_y.

3.1 More Details

Let v_y and v_z be the root of C_y and C_z respectively. Without loss of generality, assume that $\text{Val}(v_y) > \text{Val}(v_z)$. Let u be the lowest node (i.e., the node farthest away from the root v_z) in L_z such that $\text{Val}(v_y) > \text{Val}(u)$. Let L_z^b be the part of L_z below u and let L_z^a be the part above and including u.

Algorithm 1. Cartesian Tree Merging in the `Blelloch-Shun` Algorithm

```
1 Search for the lowest node $u$ in $L_z$ such that $\mathrm{Val}(v_y) > \mathrm{Val}(u)$. Split $L_z$ into $L_z^a$
  and $L_z^b$ such that $L_z^a$ is the list of nodes above (and including) $u$ in $L_z$ and $L_z^b$
  the list of nodes below $u$ in $L_z$.
2 Merge $R_y$ and $L_z^b$ into one list sorted according to the Val attributes by
  changing the Parent pointers.
3 Set $R_x = R_z$ and $L_x = join(L_z^a, L_y)$.
```

To merge R_y and L_z, it suffices to merge R_y and L_z^b. (The part L_z^a will always lie above the merged list of R_y and L_z^b.) More importantly, Blelloch and Shun observed that after merging, only the node v_y (among those nodes in R_y and L_z^b) will be present in a spine of C_x. They exploited this observation to acheive $O(n)$ work and space. In particular, a node not present in a spine of C_x will never be involved in future spine merging at higher levels of recursion. Thus, if m_x denotes the number of nodes in R_y and L_z^b that will not be present in future spine mergings, then $m_x = |R_y| + |L_z^b| - 1$ and $\sum_x m_x = O(n)$ where the summation is over all node x in the recursion tree. Using the merging algorithm of Shiloach and Vishkin [8] (see also [7]), merging of R_y and L_z^b will take $O(\log n)$ time and $O(|R_y| + |L_z^b|) = O(m_x)$ work. Hence the total work for spine merging over all nodes in the recursion tree is $O(\sum_x m_x) = O(n)$.

To allow the recursion at the parent of node x to carry out the same Cartesian tree merging strategy, we need to prepare L_x and R_x of C_x as well. The right-spine R_x is simply R_z while the left-spine L_x can be formed by joining (concatenating) the spines L_y and L_z^a. More precisely, denote by $join(L_1, L_2)$ the list composed of elements in L_1 followed by those in L_2, in the order they appeared in L_1 and L_2. Then we set L_x to $join(L_z^a, L_y)$. The pseudocode for the Cartesian tree merging is shown in Algorithm 1.

To facilitate the operations on the spines, each spine is implemented by a binary search tree using the Val attributes as search keys. The searching of u in L_z in Step (1) translates to a standard search in binary search tree. Splitting L_z into L_z^a and L_z^b is accomplished by a corresponding split operation on the binary search tree for L_z. The join operation, $join(L_z^a, L_y)$, in Step (3) is done by merging the two binary trees corresponding to L_y^a and L_z. In addition, the Parent pointer of the first node in L_z is set to point to the last node in L_y^a.

It is easy to see that each Cartesian tree merging takes only $O(\log n)$ time. Since there are $\log n$ levels of recursion, the total running time is $O(\log^2 n)$. Regarding the work, the manipulation on the spines in Step (1) and (3) requires at most $O(\log|C_x|)$ work where $|C_x|$ is the number of nodes in the Cartesian tree C_x, i.e, the size of the subarray corresponding to node x. Hence if $W(n)$ represents the total work for Step (1) and (3) over all nodes in the recursion tree, then we have $W(n) = 2W(n/2) + O(\log n)$. Solving for $W(n)$, we have that $W(n) = O(n)$. We have also seen that the merging of R_y and L_z^b in Step (2) takes $O(n)$ work in total. It follows that the total work is $O(n)$.

4 A Faster Algorithm with Superlinear Work

We first describe a simpler CREW PRAM algorithm that requires $O(\log n)$ time, and $O(n \log n)$ work and space. The algorithm has the same divide-and-conquer structure as the `Blelloch-Shun` algorithm. That is, it splits the input into two equal halves, recursively constructs a Cartesian tree for each half and finally merge them into one Cartesian tree. The major difference is in the Cartesian tree merging process.

The main idea of our improvement over previous algorithms is to pull part of the Cartesian tree merging process out of the recursion, and assign processors to finish the merging jobs along with the recursive process. More specifically, Step (2) of Algorithm 1 could be buffered by a scheduler to run without affecting the executions of other parts of Algorithm 1. The scheduler will finish all the buffered jobs in $O(\log n)$ time and $O(n)$ work. In addition, Step (1) and (3) of Algorithm 1 also needs to be finished very fast in order to achieve the claimed $O(\log n)$ running time.

4.1 Details of the Algorithm

We first describe the main data structure employed in our algorithm. While the spines are implemented by binary search trees in the `Blelloch-Shun` algorithm, they will be implemented by sorted arrays in our preliminary algorithm. More precisely, for each recursion level ℓ, we will have an array $\text{Spine}_\ell[1 \dots n]$ to store the left-spine and right-spine of each node in level ℓ of the recursion tree. So, each entry of the array stores a node u of the Cartesian tree and the nodes of a spine are stored contiguously in sorted order according to their Val attributes. Clearly, the total storage for all the Spine arrays is $O(n \log n)$.

Now we explain the algorithmic steps. Consider an arbitrary node x in the recursion tree and assume the notations defined in the previous section. For Step (1), we deploy one processor per node in L_z to locate node u in $O(1)$ time and $O(|L_z|)$ work.

In Step (2), we merge R_y and L_z^b, which are sorted lists according to the Val atttributes, into one sorted list using the same merging algorithm of Shiloach and Vishkin [8]. At first sight, it seems that using this $O(\log n)$ time algorithm will result in an overall running time of $O(\log^2 n)$. However, as discussed in section 3, only v_y will be present in future spine merging processes. This allows Blelloch and Shun to obtain $O(n)$ work algorithms. Here, we make use of this observation differently. We observe that the position of v_y in the left-spine, L_x, of C_x can be determined independent of the merging of R_y and L_z^b. Therefore, this spine merging process does not affect (or block) the execution of Step (3) at node x. Once Step (1) and (3) are completed, the recursion can return to the parent node of x so that the Cartesian tree merging process there can be started, whether Step (2) at node x has been completed or not.

In Step (3), our algorithm needs to prepare L_x and R_x in a sorted array so that Step (1) of the higher recursion level can be carried out in the same manner as described above. The left-spine L_x is obtained by joining L_y and L_z^a into one contiguous sorted array and setting the Parent pointer of the first node of L_y to the last node of L_z^a. To do the concatenation, suppose the nodes y and z are at recursion level $\ell + 1$ while node x is at recursion level ℓ. Then we assign a processor to each entry of L_y and L_z^a (which are stored in an allocated region in the array $\text{Spine}_{\ell+1}[1..n]$) and copy the values into the appropriate region in $\text{Spine}_{\ell}[1..n]$. As for the right-spine R_x, it is just the same as R_z and we just need to copy it from its allocated region in $\text{Spine}_{\ell}[1..n]$ to the appropriate region in $\text{Spine}_{\ell-1}[1..n]$.

4.2 Complexity Analysis

We first analyze the work. In Step (1), searching for node u requires $O(|L_z|)$ work. In Step (3), joining L_y and L_z^a and copying R_y take $O(|L_y| + |L_z^a| + |R_y|)$ work. Note that $O(|L_z| + |L_y| + |L_z^a| + |R_y|) \leq O(|C_x|)$. Since the collection of C_x's for all nodes x in the same recursion level induces a partition of the n elements of the input array A, the sum of all the work in the same recursion level is $O(n)$. Hence, total work over all recursion levels for Step (1) and (3) is $O(n \log n)$. The independent spine mergings in Step (2) of all the recursion nodes require a total of $O(n)$ work using Shiloach and Vishkin's algorithm [8]. Therefore, the whole algorithm requires $O(n \log n)$ work.

Regarding the time complexity, there are $\log n$ levels of recursion, each taking $O(1)$ time for Step (1) and (3). The recursion at a node x can return to its parent node after Step (1) and (3) are finished without waiting for Step (2). Therefore, Step (1) and (3) for all the nodes in the recursion tree can be finished in $O(\log n)$ time. The finish time for all the spine mergings in Step (2) is also bounded by $O(\log n)$ because the worst finish time of a merge is at most $O(\log n) + O(\log n)$, where the first $O(\log n)$ is the time to start the spine merging (whose worst case happens when the merge is at the root recursion), and the second $O(\log n)$ is the time to do the merging with Shiloach and Vishkin's algorithm. Hence the overall parallel time is $O(\log n)$ for the whole algorithm.

Theorem 1. *Given an input array of size n, its Cartesian tree can be constructed in $O(\log n)$ time with $O(n \log n)$ work and space on a CREW PRAM.*

5 Our Linear Work and Space Algorithm

To reduce the work and space of our algorithm to $O(n)$, we use a bucketing technique. Our algorithm will have two phases. In the first phase, we build a Cartesian tree for each consecutive $\log n$ elements in the input array A. So, there are $n/\log n$ Cartesian trees, each with $\log n$ nodes (assuming n is a multiple of $\log n$ and $\log n$ is an integer for simplicity). This can be done in $O(\log n)$ time with $O(n/\log n)$ processors, each running a sequential algorithm. Hence this phase takes $O(n)$ work. For convenience, we call these trees *micro-Cartesian trees.*

In the second phase, the remaining problem is to merge these $n/\log n$ Cartesian trees into one Cartesian tree. As discussed before, this amounts to merging the left and right-spines of these trees appropriately. Our approach is to represent each spine of a micro-Cartesian tree by a *super-node* and adapt our previous algorithm to merge the Cartesian trees by merging spines that are made up of super-nodes.

More precisely, we define a *super-node* v to have, besides the Val and Pos attributes, a List attribute so that $\text{List}(v)$ is the sequence of nodes represented by v. The nodes in $\text{List}(v)$ are sorted in increasing order of their Val attributes. Furthermore, $\text{Val}(v)$ and $\text{Pos}(v)$ are defined as the Val attribute and Pos attribute of the first node in $\text{List}(v)$. Hence $\text{Val}(v)$ is the minimum Val attribute of all nodes in $\text{List}(v)$. We refer to spines made up of super-nodes as *super-spines.*

Initially the spine represented by a super-node is the left- or right-spine of a micro-Cartesian tree. We will see that as the computation proceeds, the spine represented by a super-node may be split but spines in different super-nodes will never be joined into one. Therefore, $\text{List}(v)$ contains at most $O(\log n)$ nodes at any time.

Let $\tilde{A}$ be an array of size $2n/\log n$, storing the initial $2n/\log n$ super-nodes. (Each micro-Cartesian tree contributes two super-nodes, one for its left-spine, one for its right-spine.) For the implementation details, we can assume that these spines are stored contiguously in sorted order (according to their Val attributes) in an array B of size $O(n)$. The List attribute of each super-node in $\tilde{A}$ just needs to have a pointer to the corresponding spine in B.

Now consider a divide-and-conquer approach to construct one Cartesian tree from the $n/\log n$ micro-Cartesian trees represented by the array $\tilde{A}$. Let x be an arbitrary node at level ℓ in the recursion tree and let y and z be its children at level $\ell+1$. Assume that the super-spines of the Cartesian trees C_y and C_z have been prepared in array $\text{Spine}_{\ell+1}$ so that each super-spine is a sequence of super-nodes sorted according to their Val attributes and each super-node represents a sequence of $O(\log n)$ (Cartesian tree) nodes. We denote these super-spines as $\tilde{L}_y$, $\tilde{R}_y$, $\tilde{L}_z$ and $\tilde{R}_z$, to distinguish them from those spines in the previous section, which are sequences of Cartesian tree nodes (not super-nodes). We will construct a left super-spine, $\tilde{L}_x$, of C_x, also as a sequence of super-nodes, stored contiguously and sorted in their Val attributes in Spine_ℓ and likewise for $\tilde{R}_x$. Moreover, we will maintain that each super-node represents at most $O(\log n)$ Cartesian tree nodes.

Let v_y and v_z be the root of C_y and C_z respectively. Again, assume that $\text{Val}(v_y) > \text{Val}(v_z)$ without loss of generality. The required modifications to Algorithm 1 and complexity analysis are explained as follows:

Step (1) We search for the position of v_y in $\tilde{L}_z$, i.e., we locate the lowest super-node u in $\tilde{L}_z$ such that $\text{Val}(v_y) > \text{Val}(u)$. This is done, as before, by deploying one processor per super-node in $\tilde{L}_z$. This requires $O(1)$ time and $O(|\tilde{L}_z|)$ work. Using similar analysis as before, the total work for this task over all nodes x in the recursion tree is $O((n/\log n)\log n) = O(n)$.

Next, we search for $\text{Val}(v_y)$ in $\text{List}(u)$. This can be done in $O(1)$ time using $O(|\text{List}(u)|) = O(\log n)$ processors in parallel. Hence the total work for this task over all nodes x in the recursion tree is $O((n/\log n)\log n) = O(n)$.

We then split the Cartesian tree nodes in $\mathrm{List}(u)$ into two lists, one for those with Val attributes less than $\mathrm{Val}(v_y)$ and one for the other nodes. The former list is still represented by the super-node u while the latter will be represented by a new super-node u'. We let $\tilde{L}_z^a$ be the part of $\tilde{L}_z$ above (and including) u and let $\tilde{L}_z^b$ be the sequence of super-nodes started with u' and followed by the part of $\tilde{L}_z$ below u.

Step (2) We merge the spines represented by $\tilde{R}_y$ and $\tilde{L}_z^b$ into one list by modifying the Parent pointers of the Cartesian tree nodes in B. Imagine the sequence of Cartesian tree nodes formed by concatenating $\mathrm{List}(v)$ for each super-node v in $\tilde{R}_y$, in the order v appeared in $\tilde{R}_y$. That sequence is sorted in the Val attributes. The same holds for $\tilde{L}_z^b$ as well. Therefore, we can perform the merging of sorted lists using the algorithm of Shiloach and Vishkin (with a slight adaptation so that the algorithm accesses the elements to be merged by a pointer reference).

The merging algorithm of Vishkin and Shiloach takes $O(\log n)$ time while the work is proportional to the total number of (Cartesian tree) nodes merged. Since the merging in this step over all nodes in the merging tree works on disjoint sets of Cartesian tree nodes, the total work is $O(n)$. Moreover, the merging can be done independent of each other. Thus, this step for all nodes x in the recursion tree only adds an additive term of $O(\log n)$ in the parallel time complexity.

Step (3) Finally, we need to join $\tilde{L}_y$ and $\tilde{L}_z^a$ to form $\tilde{L}_x$ while $\tilde{R}_x$ is simply the same as $\tilde{R}_z$. The latter task is accomplished by copying the super-nodes of $\tilde{R}_z$ in $\mathrm{Spine}_{\ell+1}$ to appropriate places in Spine_{ℓ}. Similarly, the former task can be done by copying the super-nodes of $\tilde{L}_y$ and $\tilde{L}_z^a$ stored in $\mathrm{Spine}_{\ell+1}$ to appropriate places in Spine_{ℓ}. Furthermore, the Parent pointer of the topmost node in $\tilde{L}_y$, i.e., v_y, should be set to point to the last node in $\tilde{L}_z^a$.

Note that the number of super-nodes in $\tilde{L}_x$ and $\tilde{R}_x$ can be one larger than the total number of super-nodes in $\tilde{L}_y$, $\tilde{R}_y$, $\tilde{L}_z$ and $\tilde{R}_z$ due to the splitting of $\mathrm{List}(u)$ mentioned in Step 1. There are $n/\log n$ leaf nodes in the recursion tree, $n/(2\log n)$ nodes in the level above the leaf level, $n/(4\log n)$ nodes in the next higher level, etc. Therefore, the number of new super-nodes for each level is at most $n/\log n$. So, doubling the size of Spine_{ℓ} for each level ℓ above the leaf level would provide sufficient storage to accomodate the original and extra super-nodes. It follows that the total storage required is $O((n/\log n)\log n) = O(n)$. Furthermore, this step can be done in $O(1)$ time and $O(|\tilde{L}_y| + |\tilde{L}_z^a| + |\tilde{R}_z|)$ work. It follows that the total work for this step over all nodes in the recursion tree is $O((n/\log n)\cdot\log n) = O(n)$.

It is clear from the above discussion that the total work and storage for all steps over all nodes in the recursion tree is $O(n)$ and the parallel time is $O(\log n)$.

Theorem 2. *Given an array of size n, its Cartesian tree can be constructed in $O(\log n)$ time with $O(n)$ work and space on a CREW PRAM.*

6 Conclusion

In this work, we presented an $O(\log n)$-time CREW PRAM algorithm for constructing Cartesian trees using linear work and space. This result improves the previous best known parallel algorithms for Cartesian trees. One open question is whether the running time could be further reduced to $O(\log\log n)$. If not, could a tight lower bound be proved?

The work of Demaine et al. [3] showed that Cartesian trees can be defined on trees rather than arrays, to solve the bottleneck edge query problem on trees and undirected graphs. To the best of our knowledge, no parallel algorithm has been studied for this type of Cartesian trees. So it would be interesting to see if our technique can be applied to this type of Cartesian trees to get a non-trivial parallel algorithm.

Acknowledgment. The authors would like to thank the anonymous referees for their constructive comments.

References

1. Berkman, O., Schieber, B., Vishkin, U.: Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values. Journal of Algorithms 14, 371–380 (1993)
2. Blelloch, G.E., Shun, J.: A simple parallel cartesian tree algorithm and its application to suffix tree construction. In: ALENEX 2011, pp. 48–58 (2011)
3. Demaine, E.D., Landau, G.M., Weimann, O.: On Cartesian trees and range minimum queries. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 341–353. Springer, Heidelberg (2009)
4. Gabow, H., Bentley, J.L., Tarjan, R.E.: Scaling and related techniques for goemetry problems. In: Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, pp. 135–143. ACM (1984)
5. Gusfield, D.: Algorithms on strings, trees and sequences. Cambridge University Press (1997)
6. Iliopoulos, C., Rytter, W.: On parallel transformations of suffix arrays into suffix trees. In: 15th Australasian Workshop on Combinatorial Algorithms (2004)
7. Jaja, J.: An introduction to parallel algorithms. Addison-Wesley Professional (1992)
8. Shiloach, Y., Vishkin, U.: Finding the maximum, merging and sorting in a parallel computation model. Journal of Algorithms 2(1), 88–102 (1981)
9. Vuillemin, J.: A unifying look at data structures. Communications of the ACM 23(4), 229–239 (1980)
10. Yuan, H., Atallah, M.J.: Data structures for range minimum queries in multidimensional arrays. In: Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 150–160 (2010)

Advice Complexity of the Online Coloring Problem[⋆]

Sebastian Seibert[1], Andreas Sprock[2], and Walter Unger[1]

[1] Lehrstuhl für Informatik I, RWTH Aachen, Germany
{seibert,quax}@cs.rwth-aachen.de
[2] Department of Computer Science, ETH Zurich, Switzerland
andreas.sprock@inf.ethz.ch

Abstract. We study online algorithms with advice for the problem of coloring graphs which come as input vertex by vertex. We consider the class of all 3-colorable graphs and its sub-classes of chordal and maximal outerplanar graphs, respectively.

We show that, in the case of the first two classes, for coloring optimally, essentially $\log_2 3$ advice bits per vertex (bpv) are necessary and sufficient. In the case of maximal outerplanar graphs, we show a lower bound of 1.0424 bpv and an upper bound of 1.2932 bpv.

Finally, we develop algorithms for 4-coloring in these graph classes. The algorithm for 3-colorable chordal and outerplanar graphs uses 0.9865 bpv, and in case of general 3-colorable graphs, we obtain an algorithm using < 1.1583 bpv.

1 Introduction

An online algorithm deals with the natural situation that the input arrives piecemeal. In contrast to the offline case, the algorithm must compute a part of the solution for the already given piece of input at every time step. Once a part of the solution is computed, it must not be changed. The standard way to measure the quality of an online algorithm is the *competitive analysis*. Here, the quality of the solution given by the online algorithm is compared to the quality of the best possible solutions computable offline, i.e., after knowing the whole input. This concept was introduced in [19], for a more detailed introduction we refer to the standard literature, e.g., [4, 12].

Measuring the quality of an online algorithm by comparing its output to an optimal offline solution gives a universally applicable yardstick. However, it has the disadvantage that, for many problems, the output of the best possible online algorithms will still be far from the offline solutions.

In order to better understand and quantify this gap, the idea of online algorithms with advice was introduced by [6] and has been further investigated, e.g., in [2, 3, 7, 13]. Here, one asks how much additional information (advice) an online algorithm needs to close this gap, respectively, which progress can be made with limited advice.

[⋆] The research is partially funded by the SNF grant 200021–141089.

Formally, one introduces an advisor (an oracle that knows the whole input) who provides an unlimited advice bit string to the online algorithm. For any online algorithm with advice, the *advice complexity* measures the number of bits read by the online algorithm. The *advice complexity* of a problem is the advice complexity of the best online algorithm (achieving a given competitive ratio).

Coloring vertices of a graph such that no two adjacent vertices get the same color is a very well known and intensively studied problem. For an online version, the most obvious input order is the following which will be studied here. In every step, a new vertex gets revealed, together with all edges to the previously revealed vertices. Now, the newly revealed vertex has to be colored before the next one is revealed.

It turns out that online coloring is hard and no constant competitive ratio is possible [17]. For the class of k-colorable graphs on n vertices, it has been proven that any online coloring algorithm needs $\Omega\left((\log n/(4k))^{k-1}\right)$ colors in the worst case [20]. For an overview of classical online coloring see [15, 16].

A first study of online path coloring with advice was done in [8]. Further studies of online coloring of bipartite graphs was done in [1]. In this paper, we study online coloring with advice on the class of all 3-colorable graphs, and on its sub-classes of 3-colorable chordal and outerplanar graphs, respectively. We want to know how much advice is necessary, respectively sufficient, in order to color these graphs optimally. Also, we investigate how much advice can be saved if we allow a fourth color. The results mentioned above imply that using only a constant number of colors is a big improvement over coloring without advice.

For a lower bound, we show that at least $1.0424 \cdot n$ advice bits are necessary to color a maximal outerplanar graph with n vertices optimally. For 3-colorable chordal graphs (and thus for general 3-colorable graphs), we get a lower bound of $(\log_2 3 - \varepsilon) \cdot n$ bits (for arbitrarily small ε).

On the other hand, we describe an algorithm to color general 3-colorable graphs optimally using $1.5863 \cdot n$ bits, and we can color maximal outerplanar graphs optimally using $1.2932 \cdot n$ bits (where n again is the number of vertices). Note that 3-coloring is known to be $\mathcal{NP}$-hard [9], but, due to the definition of our model, there is no computational restriction for the advisor.

Table 1. Overview of the results on the number of bits per vertex for online coloring

number of bits per vertex	lower bounds	upper bounds	
	3-coloring	3-coloring	4-coloring
3-colorable graphs	$\log_2 3 - \varepsilon$[1]	1.5863	1.1583
3-colorable chordal graphs	$\log_2 3 - \varepsilon$	1.5863	0.9865
maximal outerplanar graphs	1.0424	1.2932	0.9865

Moreover, we analyze the advice needed for coloring 3-colorable graphs with a competitive ratio of 4/3. In other words, we want to color 3-colorable graphs with four colors. Note that the offline version of this problem is also known to be

[1] $\log_2 3 \geq 1.5849$.

$\mathcal{NP}$-hard [10,14]. Here, we show how to obtain a 4-coloring for any 3-colorable graph with 1.1583 advice bits per vertex.

Additionally, we develop an algorithm to color 3-colorable chordal graphs with four colors by using less than one bit (0.9865) of advice per vertex. An overview is shown in Table 1.

In Section 2, we fix our notation, and we show how the string of advice bits can efficiently be used for one-out-of-three decisions. Section 3 is dedicated to show our main results. Due to space restrictions, mainly we have restricted ourselves to describing the ideas. For the formal proofs and algorithms, see also [18]. We conclude in Section 4.

2 Preliminaries

We use the following notation for online algorithms analogous to [3], for coloring graphs, and for the problem at hand, respectively.

Definition 1. *[3] Consider an input sequence $I = (x_1, \ldots, x_n)$ for some minimization problem U. An **online algorithm** $\mathtt{A}$ computes the output sequence $\mathtt{A}(I) = (y_1, \ldots, y_n)$, where $y_i = f(x_1, \ldots, x_i)$ for some function f. The **cost** of the solution is given by $\mathrm{cost}(\mathtt{A}(I))$. An algorithm $\mathtt{A}$ is **c-competitive**, for some $c \geq 1$, if there exists a constant α such that, for every input sequence I, $\mathrm{cost}(\mathtt{A}(I)) \leq c \cdot \mathrm{cost}(\mathtt{Opt}(I)) + \alpha$, where $\mathtt{Opt}$ is an optimal offline solution for the problem. If $\alpha = 0$, then $\mathtt{A}$ is called **strictly c-competitive**. Finally, $\mathtt{A}$ is **optimal** if it is strictly 1-competitive.*

A **chord** of a cycle $H = x_1, x_2, \ldots, x_k$ of length k is an edge of two vertices $\{x_i, x_j\}$, that is no edge of the cycle $\{x_i, x_j\} \notin E(H)$. A cycle is chordless if it contains no chords. Additionally, we deal with the graph classes of chordal and maximal outerplanar graphs. For this, we give a formal definition of it.

Definition 2 (Hajnal, Surányi [11], Brandstädt et al. [5]). *A graph G is **chordal** if each cycle in G of length at least 4 has at least one chord.*

Definition 3 (Planar graph). *A graph $G = (V, E)$ is called **planar**, if it can be drawn in a plane, such that no two edges intersect. Such a drawing of a planar graph in the plane is called an **embedding**. A planar graph $G = (V, E)$ is **maximal** if for every nonadjacent pair of vertices v_i, v_j the graph $G' = (V, E \cup \{v_i, v_j\})$ is not planar.*

Definition 4 (outerplanar graph). *A graph $G = (V, E)$ is called **outerplanar**, if it has an planar embedding in plane, such that no vertex is totally surrounded by edges. A outerplanar graph is **maximal** if there are no two nonadjacent vertices v_i, v_j such that the graph $G' = (V, E \cup \{v_i, v_j\})$ is still outerplanar.*

Next, we fix the notation for coloring graphs.

Definition 5. *Given a graph $G = (V, E)$, a coloring function c is a function that maps every vertex $v_i \in V$ to one color from $\{1, \ldots, k\}$, for some $k \in \mathbb{N}$, such that $c(v_i) \neq c(v_j)$, for all v_i, v_j with $\{v_i, v_j\} \in E(G)$. We denote the the minimal number of colors necessary for a coloring of G with $\chi(G)$.*

For any vertex $v \in V$, we denote by $N(v) = \{w \in V \mid \{v, w\} \in E\}$ the set of neighbor vertices of v in G. If G is directed, E contains ordered pairs, and the set of predecessors of v is $Pred(v) = \{w \in V \mid (w, v) \in E\}$.

Definition 6. *The **Online Coloring Problem with Advice, in Vertex-Revealing Mode (OColA$_V$)** is the following online problem: The input is an unweighted, undirected graph $G = (V, E)$ with $|V(G)| = n$ and an order $\prec$ of revealing on the set of vertices. The goal is to find a minimum-cost coloring function $c : V \to \{1, \ldots, n\}$ for the vertices in G, where the cost of a coloring is the number of used colors.*

In each time step i, the next vertex $v_i \in V$ (in the order $\prec$) is revealed, together with all edges $\{\{v_i, v_j\} \mid j < i\}$, and the online algorithm has to decide which color $c(v_i)$ the vertex v_i gets. To this end, it can read a certain number of advice bits.

For every instance $I = (G, \prec)$, where $G = (V, E)$, we get a directed graph $G^{\prec} = (V, E')$ by giving a direction on every edge $e \in E$ depending on the order of revealing the vertices. Every edge $e = \{v_i, v_j\} \in E$ is directed from v_i to v_j, i.e., $(v_i, v_j) \in E'$, iff v_i is revealed before v_j. Additionally, for a subset of vertices $V_x \subset V(G)$, we denote by $G_{V_x} = G \mid_{V_x}$ the subgraph of G induced by V_x. For developing algorithms to color a 3-colorable graph optimally, we need a method to read a one-out-of-three decision from a Boolean advice string. For this, we use the following lemma.

Lemma 1. *Reading several one-out-of-three decisions from a bit string costs $46/29 < 1.5863 \approx \log_2 3$ bits on average.*

Proof. When the first one-out-of-three decision is necessary, 46 bits get read from the advice string. By these 46 bits and the corresponding 2^{46} different possible bit combinations, 29 three-way decisions[2] can be encoded, because $2^{46} \geq 3^{29}$. With this, for the first three-way decision, the algorithm gets the results of the next 28 three-way decisions at the same time and keeps them in its memory. This leads to an average of the information needed for a three-way decision of $46/29 < 1.5863 \approx \log_2 3 \approx 1.58496$ bits. In general, if n one-out-of-three decisions have to be done, this costs at most $1.5863(n - 1) + 46 = 1.5863n + d$ bits, where $d < 45$. □

3 Algorithms and Lower Bounds

In this section, we first turn to the lower bounds. We will show that more than one bit of advice per vertex is necessary to color a maximal outerplanar graph optimally, i.e., by three colors.

[2] Note, there are better but larger values for x and y to make the factor closer to one.

Theorem 1. *For any $k \in \mathbb{N}$, there exists a maximal outerplanar graph G_k on $n = 4k + 9$ vertices and an ordering $\prec$ on the vertices of G_k such that every deterministic online algorithm for OColA$_V$ on $(G, \prec)$ needs at least $\frac{1}{2} \cdot \left(\log_2 3 + \frac{1}{2}\right) \cdot n - 12 > 1.0424 \cdot n - 12$ advice bits to generate an optimal coloring.*

For the proof see [18]. Additionally, we want to show that $\log_2 3$ bits per vertex are necessary for coloring any 3-colorable graph online optimally. For this, we give the following theorem.

Theorem 2. *For any $k \in \mathbb{N}$, there exists a 3-colorable chordal graph G on $n = k + 3$ vertices and an ordering $\prec$ such that every deterministic online algorithm for OColA$_V$ on $(G, \prec)$ needs at least* $\log_2 3 \cdot (k - 1) - 1 = \log_2 3 \cdot (n - 4) - 1$ *advice bits to be optimal.*

Now we are ready to investigate several algorithms for coloring graphs online with given advice. Let G be a graph with an order $\prec$, and let $G^{\prec}$ be the corresponding directed graph. Depending on the direction of the edges in $G^{\prec}$, we define the function $p : V(G) \to \{1, 2, 3\}$ for all vertices in $G^{\prec}$, where $p(v) = i$ describes which position v has in a triangle as follows.

Every vertex v_x, that was revealed as isolated, i.e., has outgoing edges only, is the first vertex of any triangle it belongs to. Such a vertex gets the label 1, that is $p(v_x) = 1$.

Every vertex v_y that is connected to one or more already revealed vertices, but not closing a triangle, has $p(v_y) = 2$. (In any triangle, there is at most one ingoing edge).

Finally, every vertex v_z which closes one or more triangles (has two ingoing edges in one triangle) gets $p(v_z) = 3$.

That way, we partition the vertices of a given input instance $G^{\prec}$ into three classes $V_i = \{v \in V(G) \mid p(v) = i\}$, for $i \in \{1, 2, 3\}$. For an example showing the different types of vertices, see Figure 1.

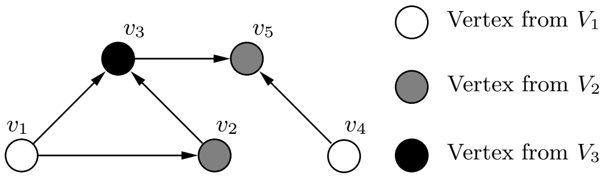


Fig. 1. Vertices of different types

In every step of the coloring algorithm, when a new vertex v occurs, there exists a set of colors by which v may be colored. We denote, for every vertex v, the set of allowed colors by $C_v = \{1, \ldots, 3\} \setminus \{c(w) \mid w \in N(v)\}$. Note that, since we use an ordered set of colors, we may speak of a 'smallest' color in C_v.

We start with Algorithm 1, which colors an arbitrary 3-colorable graph G online with 3 colors, where G is revealed according to an order $\prec$. For this, we need 1.5863 advice bits per vertex on average.

The idea of the first algorithm is quite simple. For every isolated vertex $v \in V_1$, the algorithm asks for the optimal color (one out of three). For every vertex $w \in V_2$ connected to an already colored vertex, the algorithm asks for the correct color from the the remaining colors C_w (at most one out of two), and every vertex $x \in V_3$ gets colored by the only remaining color. This leads to the following lemma.

Lemma 2. *Let G be a graph with $\chi(G) = 3$, and let $G^{\prec}$ be an input instance for the OColA$_V$. Algorithm 1 colors G optimally with at most $(n-3)$ one-out-of-three decisions and at most one one-out-of-two decision. With this, Algorithm 1 uses less than $1.5863 \cdot (n-3) + 1 + 45 = 1.5863 \cdot n + d$ advice bits*[3]*, where $d < 42$.*

Proof. Let $G^{\prec}$ be an input instance of a graph G with $\chi(G) = 3$ and $|V(G)| = n$. In the worst case, $G^{\prec}$ contains $n-2$ vertices in V_1. Otherwise, G does not contain a cycle, and it is a forest and thus two-colorable. Algorithm 1 does not use information for the first vertex, because here the coloring can be arbitrary.

For the second revealed vertex, even if it is revealed as isolated, only one bit of advice is necessary for knowing whether it gets the same color as the first vertex or a different one. For all further vertices, except the last one, a one-out-of-three decision might be necessary. Summing up, Algorithm 1 needs at most $(n-3)$ one-out-of-three and one one-out-of two decisions. We know from Lemma 1 that a one-out-of-three decision needs less than 1.5863 bits in the average. This leads to less than $1.5863 \cdot (n-3) + 1 + 45 = 1.5863 \cdot n + d$ bits at all, where $d < 42$. □

Now, we observe that, since vertices in V_1 have outgoing edges only, no two of them can be connected.

Observation 1. *Let $G^{\prec}$ be the directed graph resulting from G and the order $\prec$ of revealing. Then V_1 is an independent set in $G^{\prec}$, respectively in G.*

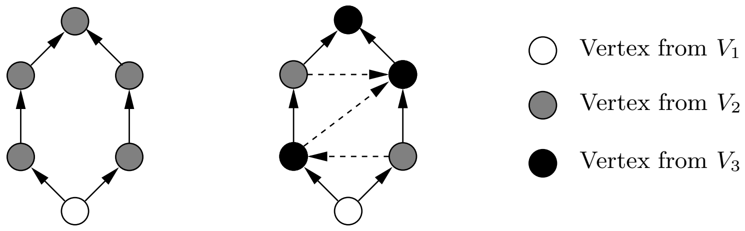
This leads us to the following lemma, which holds for general chordal graphs.

Lemma 3. *Let G be a chordal graph, $(G, \prec)$ be an input instance for OColA$_V$, and let $G^{\prec}$ be the corresponding directed graph. For the set $A = V_1 \cup V_2$, the subgraph $G^{\prec}_A$ is a forest.*

Proof. If $G^{\prec}_A$ is not a forest, it contains at least one cycle. This cycle has to be extended by edges to triangles because G is chordal. Hence, such a cycle contains at least three vertices from A which form a triangle. The vertex v_l from this triangle, that is revealed last, has two incoming edges in $G^{\prec}_A$ and is consequently an element of V_3 (see Figure 2(a),(b)), which is a contradiction to the assumption. □

In the following, we analyze OColA$_V$ on the class of outerplanar graphs. Therefore, we need some observations and lemmata for this class of graphs.

[3] The constant 45 is a result of the method to encode the one-out-of-three decisions in the advice.



(a) A hypothetical cycle in $G^{\prec}$

(b) Types of the vertices when all edges are included

Fig. 2. Example of a cycle in a chordal graph

Lemma 4. *Let G be an outerplanar graph on n vertices and let $G^{\prec}$ be an input instance for OColA$_V$. For the set V_1 of vertices that are revealed as isolated, $|V_1| \leq 1/2 \cdot n$.*

Proof. Let G be an outerplanar graph, hence $\chi(G) = 3$, and let $G^{\prec}$ be a corresponding input instance. This implies that all vertices in G lie on an outer cycle. We know from Observation 1 that all vertices of type V_1 are independent in G. This implies that, between two vertices $v, w \in V_1$, there has to be at least one vertex $x \in V_2$ to connect them. This yields $|V_2| \geq |V_1| - 1$. Additionally, at least at the end one vertex $y \in V_3$ is necessary to close the cycle, otherwise G would be two-colorable. □

Using Lemma 4, we can analyze Algorithm 1 with the following result.

Lemma 5. *Let G be a maximal outerplanar graph with $V(G) = n$, and let $G^{\prec}$ be a corresponding input instance for the OColA$_V$. Then Algorithm 1 colors G optimally, using less than $1.29315 \cdot n + 45$ advice bits.*

Proof. Let G be a maximal outerplanar graph with $V(G) = n$, and let $G^{\prec}$ be a corresponding input instance for the OColA$_V$. Let V_1, V_2, and V_3 be the corresponding sets of vertices. According to Lemma 4, $|V_1| \leq |V_2| + |V_3|$. This leads, by using also Lemma 3, to the following inequalities: $\frac{|V_1|}{n} \leq 0.5$ and $\frac{|V_1|}{n} + \frac{|V_2|}{n} < 1$. For the number of advice bits per vertex A_{BpV} for Algorithm 1, we have

$$A_{BpV} \leq \frac{|V_1|}{n} \cdot 1.5863 + \frac{|V_2|}{n}. \tag{1}$$

We maximize the right-hand side of (1) by setting $\frac{|V_1|}{n} = 0.5$. This implies $\frac{|V_2|}{n} < 0.5$, and thus $A_{BpV} < 0.5 \cdot 1.5863 + 0.5 = 1.2931$.

For the upper bound on the number of advice bits used by Algorithm 1, this means: $A_b < 1.29315 \cdot n + d$ where $d \leq 45$ is the number of bits needed to encode the last ≤ 28 one-out-of-three decisions. □

In addition to the results for an optimal coloring, we now use the alternative online Algorithm 2, which colors an arbitrary 3-colorable graph G with 4 colors.

The idea is to color all vertices of V_1, which are revealed as isolated, with an additional color 4 and to ask for every revealed vertex from V_2 and V_3 for advice according to an optimal coloring of G using the colors $\{1, 2, 3\}$.

Following this strategy, advice is only necessary for vertices of V_2 (1.5863 bits) and for vertices of V_3 (1 bit). So this strategy is efficient for instances with a high number of isolated vertices.

This leads us to Algorithm 3, which combines the strategies of Algorithm 1 and Algorithm 2. With it, we can color all 3-colorable graphs with at most four colors. To know what to do, the algorithm reads at the beginning the first bit of the advice tape and, depending on this bit, it decides which of the two strategies it follows.

The following lemma shows that Algorithm 3 colors G optimally if, for the vertices of $G^{\prec}$, $|V_1|/n \cdot 1.5863 + |V_3|/n \leq 1.15822$. Otherwise it colors G with four colors. In both cases, it needs at most $1.1582196 \cdot n + d$ advice bits.

Lemma 6. *Let G be a graph with $V(G) = n$ and $\chi(G) = 3$, and let $G^{\prec}$ be a corresponding input instance for the $OColA_V$. There exists an advice tape with which Algorithm 3 colors G with four colors, using at most* $1.1582196 \cdot n + 45$ *bits.*

Proof. There exists a 4-coloring for G, where all vertices revealed as isolated have the same color, because $\chi(G) = 3$ and the vertices from set V_1 are independent (see Observation 1). In such a coloring, the algorithm needs a one-out-of-three decision for every vertex from V_2 and a one-out-of-two decision for every vertex from V_3, because it is already connected to at least one already colored vertex from V_2.

Now, we compute the maximum of advice bits used, by combining both algorithms. Algorithm 1 uses $\leq |V_1| \cdot 1.5863 + |V_2|$ bits, and Algorithm 2 uses $\leq |V_2| \cdot 1.5863 + |V_3|$ many bits. This leads to a maximal number of advice bits per vertex at $\frac{|V_1|}{n} = 0.26986$, $\frac{|V_2|}{n} = 0.73014$ and thus to at most 1.1582196 bits per vertex. This leads to an upper bound of $1.1582196 \cdot n + d$ bits overall. $\square$

For giving the idea of the next algorithm, we analyze, for a chordal graph G, the graph G' which is obtained from G by edge contraction.

Lemma 7. *Let G be a chordal graph with $\chi(G) = c$. For every graph G', that is obtained by contracting an edge of G, G' is chordal and $\chi(G') \leq c$.*

Proof. Assume that G is a chordal graph with $\chi(G) = c$ and G' is obtained by contracting the edge $\{a, b\}$ in G. Assume that G' contains a vertex-induced cycle of length > 3 containing x, where x is the vertex contracted from edge $\{a, b\}$. Let be $x, v, w, \cdots, z, x$ this cycle, then either $a, v, w, \cdots, z, a$ is also a cycle of length > 3 in G or $a, v, w, \cdots, z, b, a$ is a cycle of length > 4 in G. Both alternatives are a contradiction to our assumption. It follows that G' is chordal as well.

Now we show that $\chi(G') \leq \chi(G)$. Assume $\chi(G') > \chi(G)$. We have that x is in a clique $C = \{x, v_1, v_2, \cdots, v_c\}$ of size $> \chi(G)$. But the clique size of $\{a, v_1, v_2, \cdots, v_c\}$ and $\{b, v_1, v_2, \cdots, v_c\}$ is at most $\chi(G)$. Thus there exists i, j with $\{a, v_i\} \notin E(G)$ and $\{b, v_j\} \notin E(G)$. If $i = j$, then C would not be a clique

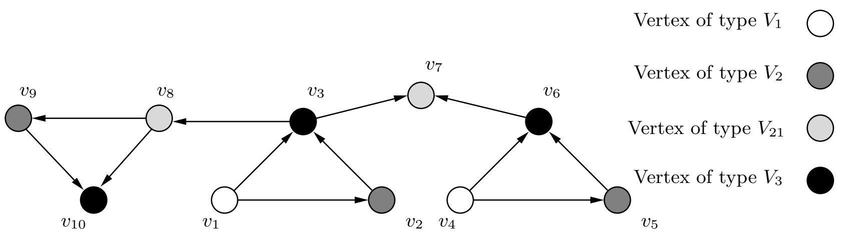


Fig. 3. Vertices of types V_{21}

of size $> \chi(G)$. Thus a, v_i, v_j, b is a vertex-induced cycle of length 4 in G, which is a contradiction. □

Now, we present Algorithm 4 for coloring 3-colorable chordal graphs with 4 colors. For this, we separate the vertices $V(G)$ into two sets $A := V_1 \cup V_2$ and $B := V_3$. We know that, for every chordal graph G, the graph G_A restricted to the vertices in A is a forest (see Lemma 3).

The idea is to color each tree of G_A by a pair of colors $(i, 4)$, for some $i \in \{1, 2, 3\}$. The remaining vertices in V_3 will be colored using only colors $\{1, 2, 3\}$. We will show later that such a coloring always exists.

Before we can describe Algorithm 4, we have to move a few vertices inside A. It might happen that a vertex v from V_2 is revealed as the first one of a tree in G_A. This can occur when all its predecessors in $G^{\prec}$ are in V_3 (see v_7, v_8 in Figure 3). However, in this case, we note that v cannot have a neighbor in V_1. Such a neighbor w would be revealed after v, and consequently the edge orientation would be (v, w), an ingoing edge for w, thus $w \notin V_1$. Therefore, we can define $V_{21} = \{v \in V_2 | \mathit{Pred}(v) \subseteq V_3\}$, and move V_{21} to V_1, more precisely $V_1' = V_1 \cup V_{21}$ and $V_2' = V_2 \setminus V_{21}$, while still preserving independence of V_1'.

Remark 1. V_1' is an independent set in G, respectively in $G^{\prec}$.

Looking again at Algorithm 2, we observe that all it needs from V_1 is that it is an independent set since all vertices from V_1, and only those, are colored by color 4. Consequently, the algorithm works the same when using V_1' instead of V_1. Let us call this variant Algorithm 2′.

We are now ready to describe new Algorithm 4. Here, V_1' contains exactly those vertices from A which are revealed without a predecessor from A, while, for all vertices in V_2', such a predecessor exists. Consequently, Algorithm 4 asks, for every vertex $x \in V_1$, for two pieces of advice. First, it wants to know which pair of colors $(i, 4)$ will be used to color the tree x belongs to. Secondly, it asks which of the two colors x gets itself.

There are three possible pairs of colors. This leads to a combination of a one-out-of-three and a one-out-of-two decision. Thus, at most 2.5863 bits are needed for every vertex from V_1.

With this information, obviously the algorithm is able to color all vertices $x \in V_1'$. Also, all vertices from V_2' can be colored because, at the moment a vertex v from V_2' is revealed, it has a predecessor w in A, and, for w, the color pair of the tree both belong to is known, as well as the color w gets. Hence, v is colored by the other color from that pair without further advice. Inside the trees of G_A, such a coloring is clearly possible, but we still have to show later that this way a correct coloring of the whole graph is constructed.

Finally, for every vertex $z \in V_3$, which closes one or more triangles, the algorithm asks for a one-out-of-two decision, because such vertices have to be connected to at least two already colored and connected vertices (x, y), with different colors and so, in the worst case, two possible colors remain for z ($|C_z| \leq 2$). The new algorithm needs at most $|V_1'| \cdot 2.5863 + |V_3| + \mathit{const}$ many bits.

To prove that such a coloring exists for every 3-colorable chordal graph, we give a further algorithm, which describes how an oracle can find the related coloring and with this the right advice tape. Again, we use $A = V_1 \cup V_2$ and $B = V_3$. We build the graph G' by subsuming every connected component of G_A in one vertex. When G is a 3-colorable chordal graph, the graph G' is 3-colorable as well (see Lemma 7). Thus, we use a 3-coloring c' of G'.

The 4-coloring c for G can be derived from c' in the following way. For a vertex $v' \in V(G')$, we distinguish two cases. If v' was constructed by an edge contraction, we color the contracted tree in G with the colors $\{c'(v'), 4\}$, and if v' corresponds directly to a vertex $v \in V(G)$ we define $c(v) := c(v')$. With this procedure, we get a coloring which satisfies the needed properties for Algorithm 4. With the corresponding advice tape, Algorithm 4 needs $\frac{|V_1|}{n} \cdot 2.5863 + \frac{|V_3|}{n}$ bits of advice per vertex. This leads us to the following lemma.

Lemma 8. *Let $G = (V, E)$ be a 3-colorable chordal graph and let $G^{\prec}$ be the corresponding input instance for OColA$_V$. There exists a coloring $c : V(G) \to \{1, 2, 3, 4\}$ and with this an advice tape such that Algorithm 4 can color $G^{\prec}$ with the coloring function c.*

For proving the claim, we give the algorithm to find such a coloring and prove that, for every 3-colorable chordal graph, such a coloring will be found.

Proof. Let $G^{\prec}$ be the given input instance. From the order of the vertices, we can build three sets of vertices V_1, V_2 and V_3. We build from this two sets $A = V_1 \cup V_2$ and $B = V_3$. We know that the graph G_A is a forest (see Lemma 3). Each tree of G_A is two-colorable. In the following, we will color each tree of G_A with two colors, where one of the two colors will be 4 for all trees. To find the corresponding color for each tree, we have to determine a coloring, which fits with a coloring for the vertices in B.

For finding such a coloring, we build the graph G' from G by contraction of edges between vertices of A. This leads us to a surjective function $m : V(G) \to V(G')$. If $T_i \subset A$ is a tree in G_A, then for all $t_j, t_k \in V(T_i), j \neq k$, we have $m(t_j) = m(t_k)$.

We know that G' is 3-colorable and chordal as well as G (see Lemma 7), so we can find an optimal coloring c' for G' with three colors, because G is a 3-colorable chordal graph.

We extend the coloring function c' to c for G by coloring all vertices $v \in B \subset V(G)$ which are also vertices in G' with $c(v) := c'(v)$. For all vertices

$w_i \in A \subset V(G)$ which are represented by one vertex $x \in V(G')$, with $m(w_i) = x$ we color the corresponding tree alternatingly in $(c'(x), 4)$.

It is obvious that the coloring c needs at most 4 colors because c' was a coloring with 3 colors and the color 4 is used additionally. The new coloring is proper for G because no two vertices with color 4 are connected to each other, and each tree T_i in G_A is colored properly with the two colors $(c'(x), 4)$. If the vertex x represents a tree T_x in G_A, it follows that all vertices in G which are connected to the tree T_x represented by x are connected to x in G'. If c' is a proper coloring for G', it follows that $c'(y) \neq c'(x)$, for all vertices $y \in N_{G'}(x)$. If now the tree which is represented by x in G' is colored only with $c(x)$ and 4, it follows that c is a proper coloring. □

Putting everything together, we can combine the previous algorithms into a final one, Algorithm 5, formally stated in [18]. This algorithm uses the first two advice bits to decide which of the Algorithms 1, 2', 4 it shall use. Consequently, it always makes use of the best possible advice-per- vertex ratio among those three. This results in the following analysis.

Theorem 3. *Let $G = (V, E)$ be a 3-colorable chordal graph with $|V(G)| = n$ and let $G^{\prec}$ be the corresponding input instance for OColA$_V$. Algorithm 5 colors $G^{\prec}$ with 4 colors using at most* $0,9865 \cdot n + 47$ *advice bits.*

Proof. We have seen before that there exists an advice tape for any of the three Algorithms 1, 2', 4. Now, we show that, in any case, there is one of the three strategies that colors G using $0,9865 \cdot n + 47$ advice bits. Let $z_1 = \frac{|V_1|}{n}$, $z_1' = \frac{|V_1'|}{n}$, $z_2 = \frac{|V_2|}{n}$, $z_2' = \frac{|V_2'|}{n}$ and $z_3 = \frac{|V_3|}{n}$.

For the needed advice bits A_1 for Algorithm 1, A_2 for Algorithm 2', and A_4 for Algorithm 4, we have (with $z_1 + z_2 = z_1' + z_2'$, $z_1 \leq z_1'$)

$$\begin{aligned} A_1 &\leq 1.5863 \cdot z_1 + z_2 \leq 1.5863 \cdot z_1' + z_2' \\ A_2 &\leq 1.5863 \cdot z_2' + z_3 \\ A_4 &\leq 2.5863 \cdot z_1' + z_3 \end{aligned}$$

Additionally, $z_1' + z_2' + z_3 = 1$. The corresponding convex space has its maximum at $z_1' = 0.30667$, $z_2' = 0.5$, and $z_3 = 0.19333$, and there it needs $A_g = 0.98647 \cdot n$ bits. The additive constant 47 consists of the two bits read at the beginning and the usual 45 bits that can remain in each of the sub-algorithms from the block of bits read for one-out-of-three decisions. □

4 Conclusion

We introduced first research results for online coloring algorithms with advice for 3-colorable graphs. For planar, chordal and general 3-colorable graphs we presented nearly matching lower and upper bounds on the number of advice bits for the 3-coloring. We also gave 4-coloring online algorithms with advice for those graph classes (see Table 1). It remains to extend the lower bounds to the 4-coloring. The extension to other graph classes, k-coloring, and general coloring is also very interesting.

References

1. Bianchi, M.P., Böckenhauer, H.-J., Hromkovič, J., Keller, L.: Online coloring of bipartite graphs with and without advice. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 519–530. Springer, Heidelberg (2012)
2. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R.: On the advice complexity of the k-server problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 207–218. Springer, Heidelberg (2011)
3. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R., Mömke, T.: On the advice complexity of online problems. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 331–340. Springer, Heidelberg (2009)
4. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press (1998)
5. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. Society for Industrial and Applied Mathematics (1999)
6. Dobrev, S., Královič, R., Pardubská, D.: How much information about the future is needed? In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 247–258. Springer, Heidelberg (2008)
7. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 427–438. Springer, Heidelberg (2009)
8. Forišek, M., Keller, L., Steinová, M.: Advice complexity of online coloring for paths. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 228–239. Springer, Heidelberg (2012)
9. Garey, M.R., Johnson, D.S.: Computers and Intractability. W. H. Freeman and Co. (1979)
10. Guruswami, V., Khanna, S.: On the hardness of 4-coloring a 3-colorable graph. In: Proceedings. 15th Annual IEEE Conference on Computational Complexity, pp. 188–197 (2000)
11. Hajnal, A., Surányi, J.: Über die Auflösung von Graphen in vollständige Teilgraphen. In: Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae Sectio Mathematicae, vol. 1, pp. 113–121 (1958)
12. Hromkovič, J.: Design and Analysis of Randomized Algorithms: Introduction to Design Paradigms (Texts in Theoretical Computer Science. An EATCS Series). Springer, Heidelberg (2005)
13. Hromkovič, J., Královič, R., Královič, R.: Information complexity of online problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010)
14. Khanna, S., Linial, N., Safra, S.: On the hardness of approximating the chromatic number. In: Proc. of ISTCS 1993, pp. 250–260 (1993)
15. Kierstead, H.: Recursive and on-line graph coloring. In: Ershov, Y.L., Goncharov, S., Nerode, A., Remmel, J., Marek, V. (eds.) Handbook of Recursive Mathematics Volume 2: Recursive Algebra, Analysis and Combinatorics. Studies in Logic and the Foundations of Mathematics, vol. 139, pp. 1233–1269. Elsevier (1998)
16. Kierstead, H., Trotter, W.: On-line graph coloring. In: McGeoch, L.A., Sleator, D.D. (eds.) On-line Algorithms. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 7, pp. 85–92. AMS—DIMACS—ACM (1992)

17. Lovász, L., Saks, M.E., Trotter, W.T.: An on-line graph coloring algorithm with sublinear performance ratio. Discrete Mathematics 75(1-3), 319–325 (1989)
18. Seibert, S., Sprock, A., Unger, W.: Advice complexity of the online vertex coloring problem. Technical Report 765, ETH Zürich (2012)
19. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM 28(2), 202–208 (1985)
20. Vishwanathan, S.: Randomized online graph coloring. Journal of Algorithms 13(4), 657–669 (1992)

Sparse Linear Complementarity Problems

Hanna Sumita[1], Naonori Kakimura[2], and Kazuhisa Makino[1]

[1] Department of Mathematical Informatics,
University of Tokyo, Tokyo 113-8656, Japan
[2] College of Arts and Sciences,
University of Tokyo, Tokyo 153-8902, Japan
{Hanna_Sumita,makino}@mist.i.u-tokyo.ac.jp,
kakimura@global.c.u-tokyo.ac.jp

Abstract. In this paper, we study the sparse linear complementarity problem, denoted by k-LCP: the coefficient matrix has at most k nonzero entries per row. It is known that 1-LCP is solvable in linear time, while 3-LCP is strongly NP-hard. We show that 2-LCP is strongly NP-hard, while it can be solved in $\mathrm{O}(n^3 \log n)$ time if it is sign-balanced, i.e., each row has at most one positive and one negative entries, where n is the number of constraints. Our second result matches with the currently best known complexity bound for the corresponding sparse linear feasibility problem. In addition, we show that an integer variant of sign-balanced 2-LCP is weakly NP-hard and pseudo-polynomially solvable, and the generalized 1-LCP is strongly NP-hard.

1 Introduction

Given a matrix $M \in \mathbb{R}^{n\times n}$ and a vector $q \in \mathbb{R}^n$, the *linear complementarity problem* (LCP) is to find vectors $w, z \in \mathbb{R}^n$ such that

$$w - Mz = q, \quad w, z \geq 0, \quad w^\top z = 0. \tag{1}$$

We denote a problem instance of LCP with M, q by LCP(M, q). We say that n is the *order* of LCP(M, q), where we note that the size of LCP(M, q) is $\mathrm{O}(n^2)$. The LCP, introduced by Cottle [10], Cottle and Dantzig [11], and Lemke [24], is one of the most widely studied mathematical programming problems, which, for example, contains linear and convex quadratic programming problems. Deciding whether LCP(M, q) has a solution for an arbitrary matrix M is NP-complete [7]. However, there are several classes of matrices M for which the associated LCP can be solved in polynomial time: for instance, positive semidefinite matrices [22], and *Z-matrices* (all off-diagonal entries are nonpositive) [3,14,25]. It is also known that M is a P-matrix, in which principal minors are all positive, if and only if LCP(M, q) has a unique solution for every q [27]. For details of theory of LCPs, see the books of Cottle, Pang, and Stone [13] and Murty [26].

In this paper, we focus on LCP with sparse coefficient matrix M. We denote by k-LCP the LCP whose coefficient matrix has at most k nonzero entries per row. For example, 2-LCP can have the following matrices:

$$M_1 = \begin{pmatrix} 0 & -1 & 3 \\ 0 & 1 & 1 \\ -2 & 0 & 0 \end{pmatrix}, \; M_2 = \begin{pmatrix} 2 & -1 & 0 \\ 0 & 3 & 0 \\ 5 & 0 & -4 \end{pmatrix}.$$

Remark that the general LCP can be reduced to 3-LCP by introducing new variables.

Sparse LCP appears in the context of game theory. For example, mean payoff games can be formulated as 3-LCP [2]. Moreover, bimatrix games, which can be formulated as LCP, has been investigated in terms of sparsity in algorithmic game theory. A bimatrix game is *k-sparse* if each column and row in both payoff matrices of the game have at most k nonzero entries [6,8,16,18].

Sparsity has also been attracting attention for the feasibility problem of systems of linear inequalities. A system of linear inequalities, i.e., a system of the form $Ax \leq b$, where $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$, can be reformulated as a system of linear inequalities where each inequality involves at most three variables. If each inequality involves at most two variables, it is called a *TVPI system*[1]. A TVPI system can be naturally represented as a graph which has a vertex for each variable and an edge for each inequality, where an edge connects the vertices corresponding to the variables involved by the inequality. Shostak [29] proved that feasibility of a TVPI system can be decided by following paths and cycles in such a graph. This idea was used to design the first polynomial-time algorithm [1]. Cohen and Megiddo [9] and Hochbaum and Naor [20] proposed improved algorithms which run in $\mathrm{O}(mn^2(\log m + \log^2 n))$ time and $\mathrm{O}(mn^2 \log m)$ time, respectively, where m and n denote the number of constraints and variables, respectively. Any TVPI system can further be transformed to a *sign-balanced* TVPI system, where the two nonzero coefficients in each inequality have opposite signs. A sign-balanced TVPI system is also called a *monotone* TVPI system.

We say that 2-LCP is *sign-balanced* if the coefficient matrix M has at most one positive and negative entries per row. The matrix M_2 above is such an example. We note that sign-balanced TVPI systems with nonnegativity constraints can be formulated as sign-balanced 2-LCP.

The first main result of this paper is to present a polynomial-time combinatorial algorithm for sign-balanced 2-LCP.

Theorem 1. *Sign-balanced 2-LCP of order n can be solved in $\mathrm{O}(n^3 \log n)$ time.*

We remark that the complexity of Theorem 1 matches with the currently best known bound, due to Hochbaum and Naor [20], for the feasibility problem of sign-balanced TVPI systems with nonnegativity constraints. This implies that in order to improve the complexity of Theorem 1, we need to have a faster algorithm for the feasibility problem of sign-balanced TVPI systems with nonnegativity constraints.

[1] TVPI stands for "two variables per inequality."

It should also be noted that Theorem 1 is not obtained from the results for the other well-known subclasses of LCP that focus on the sign pattern of M, such as *Z-LCP* (i.e., the coefficient matrix is restricted to be a Z-matrix) and sign-solvable LCP introduced by Kakimura [21].

On the other hand, it turns out that 2-LCP seems to be intractable.

Theorem 2. *2-LCP is NP-hard in the strong sense.*

Since 1-LCP can easily be solved in linear time, Theorems 1 and 2 completely reveal computational complexity of the LCP in terms of sparsity. Note that 3-LCP is clearly NP-hard, since LCP can be reduced to 3-LCP in polynomial time. The first row in Table 1 summarizes our results for LCP.

Toward proving Theorem 1, we first design a simple combinatorial algorithm for sign-balanced 2-LCP. For a given instance of sign-balanced 2-LCP(M, q), consider the TVPI system S obtained by dropping the complementarity condition in (1). The algorithm computes the least element of S to find one of given constraints that needs to be satisfied with equality. By repeating this at most $n + 1$ times, we can find in polynomial time a solution of the instance or conclude that it is infeasible. To reduce the running time, we exploit deep results for sign-balanced TVPI systems. Cohen and Megiddo [9] presented an efficient procedure to decide whether a given feasible TVPI system is still feasible by adding new upper and lower bounds for each variable. We apply the procedure to S, which is not necessarily feasible, to find a constraint that needs to be satisfied with equality. Note that the obtained result by applying the Cohen–Megiddo's procedure might be wrong, since we might apply it to an infeasible system S. Thus after finishing all the iterations, we check if the obtained result is correct or not.

The LCP is said to be *unit* if the coefficient matrix M is restricted to belong to $\{0, \pm 1\}^{n \times n}$.

Theorem 3. *Unit sign-balanced 2-LCP of order n can be solved in* $\mathrm{O}(n^2 \log n)$ *time.*

The result is based on the framework of the simple algorithm for sign-balanced 2-LCP, in which we compute the least element by reduction to the shortest-path problem.

In addition, we discuss an integer variant of LCP. Given a matrix $M \in \mathbb{Z}^{n \times n}$, a vector $q \in \mathbb{Z}^n$, and a positive integer d, the *integer LCP* is the problem to find two integer vectors w, z satisfying (1) with $z \in \{0, 1, \ldots, d-1\}^n$. Integer LCP was first considered by Du Val [17] and Chandrasekaran [4] in the context of least element theory. Chandrasekaran, Kabadi and Sridhar [5] and Cunningham and Geelen [15] independently proposed sufficient conditions on a matrix M such that for every q, LCP(M, q) has an integer solution.

In this paper, we obtain the following result on integer sparse LCP. See also Table 1.

Theorem 4. *Integer sign-balanced 2-LCP is weakly NP-hard, and can be solved in pseudo-polynomial time.*

The weak NP-hardness follows from the fact that finding an integer solution to a sign-balanced TVPI system is weakly NP-hard [23]. The algorithm in Theorem 4 has a similar framework to the algorithm in Theorem 1. We here need to find the least element of integer solutions in the sign-balanced TVPI systems obtained from a given LCP instance, which can be done in pseudo-polynomial time [19,20]. Note that the proof of Theorem 2 immediately implies that integer 2-LCP is NP-hard in the strong sense.

Finally, we investigate a generalization of LCPs in terms of sparsity. The *generalized* LCP (GLCP), which was introduced by Cottle and Dantzig [12], is a generalization of LCP from a square coefficient matrix to a vertical rectangular one.

Theorem 5. 1-*GLCP* (*i.e., the GLCP whose coefficient matrix that has at most one nonzero entry per row*) *is NP-hard in the strong sense.*

Table 1. Computational complexity of k-LCPs

k	1	sign-balanced 2	$2 \leq$
LCP	$\mathrm{O}(n)$	$\mathbf{O(n^3 \log n)}$	**NP-hard**
integer LCP	$\mathrm{O}(n)$	**pseudo-polynomial**	**NP-hard**
GLCP		**NP-hard**	

This paper is organized as follows. Section 2 describes existing results of sign-balanced TVPI systems. Section 3 proposes a simple polynomial-time algorithm for sign-balanced 2-LCP. Section 4 improves the algorithm in Section 3 using the Cohen–Megiddo's procedure. Section 5 shows the NP-hardness of 2-LCP. Due to the space limitation, the proofs of Theorem 3–5 are omitted. They may be found in the full version of the paper [30].

2 Sign-Balanced TVPI Systems

Let F be the feasible region of a sign-balanced TVPI system. It is well known that if $x, y \in F$ then the *meet* $z = x \wedge y$ is contained in F, where $(x \wedge y)_i = \min(x_i, y_i)$. Indeed, for each inequality $a_j x_j + a_k x_k \geq b_i$ $(a_j > 0,\ a_k < 0)$, we may assume that $z_j = y_j$, and we have $a_j z_j + a_k z_k \geq a_j y_j + a_k y_k \geq b_i$, which implies $z \in F$. If F is bounded below, then there is a vector $u \in F$ such that for any $z \in F$, we have $z \geq u$. Such a vector is called the *least element* of F. Moreover, $F \cap \mathbb{Z}^n$ also has these properties.

The remaining of this section is organized as follows. In Section 2.1, we present Shostak's characterization of infeasibility of a TVPI system by a graph. The characterization is used by Cohen and Megiddo to design a combinatorial algorithm for solving TVPI systems. To solve the problem, their algorithm decides $\mathrm{O}(n(\log^2 n + \log m))$ times whether a given feasible TVPI system is still feasible by adding a bound for one variable, where m and n are the number of constraints and variables, respectively. Cohen and Megiddo presented a procedure

(Algorithm 2.18 in [9]) which solves a more general decision problem based on Shostak's characterization, which runs in $O(n^2)$ time. We describe the procedure in Section 2.2, which will be used in Section 4.

2.1 Characterization by a Graph

Shostak [29] introduced a representation of a TVPI system by a graph and gave a characterization of infeasibility of the system in terms of the graph. The characterization is a generalization of a negative cycle in the shortest-path problem.

Let S be a TVPI system over variables $x_1, \ldots, x_n$. Shostak [29] represented S as an undirected graph $G = (V, E)$ as follows. For each variable x_i, the graph G has the vertex v_i. Moreover, G has an additional vertex v_0. For each inequality $ax_j + bx_k \le c$ $(a, b \neq 0)$, the graph G has the edge $\{v_j, v_k\}$. For each single-variable inequality $x_i \ge \alpha$ (or $x_i \le \beta$), the graph G has the edge $\{v_i, v_0\}$. For notational convenience, we introduce a new variable x_0 corresponding to v_0, and regard $x_i \ge \alpha$ (resp., $x_i \le \beta$) as $x_i + bx_0 \ge \alpha$ (resp., $x_i + bx_0 \le \beta$) with $b = 0$.

Let $P = (e_1, \ldots, e_l)$ be a path in G, where $e_i = \{v_{p_i}, v_{p_{i+1}}\}$ represents an inequality $a_i x_{p_i} + b_i x_{p_{i+1}} \le c_i$ for $i = 1, \ldots, l$. If b_i and a_{i+1} have opposite signs for $i = 1, \ldots, l-1$, that is, one is positive and the other is negative, then P is said to be *admissible*. Note that the reverse of an admissible path is also admissible, and v_0 cannot be an intermediate vertex of an admissible path. An admissible path P *induces* a new inequality $a_P x_{p_1} + b_P x_{p_{l+1}} \le c_P$ by eliminating common variables $x_{p_2}, \ldots, x_{p_l}$. For example, two inequalities $a_i x_{p_i} + b_i x_{p_{i+1}} \le c_i$ and $a_{i+1} x_{p_{i+1}} + b_{i+1} x_{p_{i+2}} \le c_{i+1}$ imply $a_i |a_{i+1}| x_{p_i} + b_{i+1} |b_i| x_{p_{i+2}} \le c_i |a_{i+1}| + c_{i+1} |b_i|$. Any feasible solution to S satisfies all new inequalities induced by admissible paths in G.

A path is called a *loop* if the initial and last vertices are identical. An admissible loop L with initial vertex v_{p_1} induces a single-variable inequality $(a_L + b_L) x_{p_1} \le c_L$. Note that if $v_{p_1} = v_0$, then $a_L = b_L = 0$ holds. We define the *extended graph* $\bar{G}$ of G by adding for each simple admissible loop L in G with initial vertex v_i $(v_i \neq v_0)$, a new edge which represents the single-variable inequality induced by L. If $\bar{G}$ has an admissible loop L that induces a new inequality $(a_L + b_L) x_i \le c_L$ such that $a_L + b_L = 0$ and $c_L < 0$, then the loop L is called *infeasible*, in the sense that there is no vector satisfying the new inequality. Shostak showed infeasibility of S is equivalent to existence of a simple infeasible loop in $\bar{G}$.

Theorem 6 ([29]). *A TVPI system S is feasible if and only if the extended graph $\bar{G}$ has no simple infeasible loop.*

2.2 Cohen–Megiddo's Procedure

In this subsection, we present a procedure of Cohen and Megiddo, which corresponds to Algorithm 2.18 in [9].

Let S be a feasible TVPI system, which may contain a single-variable linear inequality. By Theorem 6, $\bar{G}$ has no simple infeasible loop. Let T be a set of single-variable inequalities, and G_T be the graph associated with $S \cup T$. Theorem 6 implies that infeasibility of $S \cup T$ is equivalent to existence of a simple infeasible loop L in the extended graph $\bar{G}_T$ of G_T. Since $\bar{G}$ has no simple infeasible loop, L contains the vertex v_0, and at least one of the two edges incident to v_0 is an edge of T. Let $T' \subseteq T$ be the set of single-variable inequalities corresponding to the one or two edges. Then $|T'| \leq 2$ and $S \cup T'$ is infeasible by definition.

Given a feasible TVPI system S and a set T of single-variable inequalities, the Cohen–Megiddo's procedure decides whether $S \cup T$ is feasible or not, by detecting a simple infeasible loop in $\bar{G}_T$ if exists. By above discussion, the procedure can be equivalently written as follows:

Cohen–Megiddo's procedure

Input: a feasible TVPI system S and a set T of single-variable linear inequalities.

Output: find a nonempty set $T' \subseteq T$ such that $|T'| \leq 2$ and $S \cup T'$ is infeasible, or return that $S \cup T$ is feasible.

In particular, when S is a feasible sign-balanced TVPI system, the output T' of the Cohen–Megiddo's procedure has at most one upper and lower bounds. This is implicitly shown in [9], but we give a proof for correctness.

Lemma 1. *Let S be a feasible sign-balanced TVPI system. Let T be a set of single-variable linear inequalities such that $S \cup T$ is infeasible. Then the output $T' \subseteq T$ of the Cohen–Megiddo's procedure contains at most one upper and lower bounds.*

Proof. Let L be a simple infeasible loop with initial vertex v_0 and e_1, e_2 be edges of L incident to v_0. Let P be the path obtained by $L \setminus \{e_1, e_2\}$. Since S is sign-balanced, the inequality induced by P has one positive and negative coefficients. Since L is admissible, this means that either e_1 or e_2 represents an upper bound and the other represents a lower bound. At least one of e_1 and e_2 is due to T, and hence the statement holds. $\square$

Cohen and Megiddo achieved the following running time.

Theorem 7 ([9]). *Let S be a feasible TVPI system with m inequalities and n variables. The Cohen–Megiddo's procedure terminates in* $\mathrm{O}(mn)$ *time.*

3 Simple Algorithm for Sign-Balanced 2-LCP

In this section, we present an $\mathrm{O}(n^4 \log n)$ time algorithm for sign-balanced 2-LCP. The main idea of our algorithm is reduction of the problem to a sign-balanced TVPI system.

LCP(M, q) can be regarded as a problem to find a vector z satisfying $z^\top(Mz + q) = 0$ in $F := \{z \mid Mz + q \geq 0,\ z \geq 0\}$. Once we obtain such a vector z, the pair (w, z), where $w = Mz + q$, is a solution to LCP(M, q). We denote $SOL(M, q) := \{z \mid Mz + q \geq 0,\ z \geq 0,\ z^\top(Mz + q) = 0\}$.

If $F = \emptyset$, then LCP(M, q) has no solution. Suppose that $F \neq \emptyset$. Since F is the feasible region of a sign-balanced TVPI system bounded below, F has the least element u. If u satisfies $u^\top(Mu + q) = 0$, then u is clearly a solution to LCP(M, q). Otherwise, i.e., if u does not satisfy $u^\top(Mu + q) = 0$, then there is an index $i \in [n] := \{1, \ldots, n\}$ such that $u_i > 0$ and $(Mu + q)_i > 0$. This implies that any $z \in SOL(M, q)$ satisfies $z_i \geq u_i > 0$, and hence z satisfies $(Mz + q)_i = 0$. Thus $SOL(M, q) \subseteq (F \cap \{z \mid (Mz + q)_i \leq 0\})$, which means that we can restrict F with a constraint $(Mz + q)_i \leq 0$, that is, replace F by $F \cap \{z \mid (Mz + q)_i \leq 0\}$. Since the inequality $(Mz + q)_i \leq 0$ has at most one positive and negative coefficients, F is still the feasible region of a sign-balanced TVPI system. Moreover, any $z \in F$ satisfies $z_i(Mz + q)_i = 0$.

We repeat the procedure mentioned above until the least element of F satisfies $z^\top(Mz + q) = 0$ or F turns out to be empty, i.e., LCP(M, q) is infeasible. Consequently, sign-balanced 2-LCP(M, q) is solved. Note that the number of repetition is at most $n + 1$.

The algorithm is summarized as follows.

Algorithm 1.
```
Step 1. $F := \{z \mid Mz + q \geq 0, z \geq 0\}$.
Step 2. For $j = 0, \ldots, n$
    Find the least element $u$ of $F$. If $u$ does not exist, then return that LCP$(M, q)$ is infeasible.
    If $u$ satisfies $u^\top(Mu+q) = 0$, then return $u$. If $u$ does not satisfy $u^\top(Mu+q) = 0$, then find an index $i$ such that $u_i > 0$ and $(Mu + q)_i > 0$, update $F \leftarrow \{z \in F \mid (Mz + q)_i \leq 0\}$, and go to the next iteration.
```

It remains to discuss how to find the least element of F at Step 2. The least element of F is obtained by solving the linear programming problem $\min\{\mathbf{1}^\top z \mid z \in F\}$, where $\mathbf{1}$ is the vector whose elements are all one. Since Algorithm 1 requires to find the least element at most $n + 1$ times, Algorithm 1 can find a solution to LCP(M, q) in polynomial time if exists.

The least element can be found more efficiently in a combinatorial way. Hochbaum and Naor [20] noted that their algorithm for TVPI systems can compute the least element of a sign-balanced TVPI system. Their algorithm runs in $\mathrm{O}(n^3 \log n)$ time, where n is the order of a given LCP instance, and hence the running time of Algorithm 1 reduces to $\mathrm{O}(n^4 \log n)$ time in total.

Theorem 8. *Algorithm 1 solves sign-balanced 2-LCP of order n in $\mathrm{O}(n^4 \log n)$ time.*

For example, consider LCP(M, q), where

$$M = \begin{pmatrix} -1 & 1 \\ -2 & 1 \end{pmatrix}, \quad q = \begin{pmatrix} -2 \\ 3 \end{pmatrix}.$$

The least element of F is $u = (0\ 2)^\top$, which does not satisfy $z^\top(Mz + q) = 0$ since $Mu + q = (0\ 5)^\top$. In this case, we have $u_2 > 0$ and $(Mu + q)_2 > 0$. We update F to $F \cap \{z \mid -2z_1 + z_2 + 3 \leq 0\}$. Then the least element shifts to $u' = (5\ 7)^\top$. Since $Mu' + q = 0$, we have $u' \in SOL(M, q)$.

4 Improved Algorithm for Sign-Balanced 2-LCP

Recall that the main step of Algorithm 1 is finding the least element of the feasible region of a sign-balanced TVPI system S to detect an index i such that $z_i > 0$ for any solution z of a given LCP(M, q). In our improved algorithm, we directly detect such an index i without finding the least element.

For that purpose, we execute the Cohen–Megiddo's procedure described in Section 2.2 by setting T to be a set of single-variable inequalities in the form of $z_i \leq 0$. However, the procedure is guaranteed to run correctly only for feasible TVPI systems, while S is not necessarily feasible in our algorithm. In fact, the procedure may return that $S \cup T$ is "feasible" when S is infeasible. For example, let S be the TVPI system consisting of the following eight constraints:

$$\begin{array}{lll} e_1 : z_1 - 2z_2 \leq -1, & e_2 : 2z_2 - z_3 \leq 3, & e_3 : z_3 - z_1 \leq -3, \\ e_4 : -z_1 + 2z_2 \leq 1, & e_5 : -2z_2 + z_3 \leq -3, & \\ e_6 : z_1 \geq 0, & e_7 : z_2 \geq 0, & e_8 : z_3 \geq 0, \end{array}$$

and $T = \{e_9 : z_3 \leq 0\}$. Let G be the graph associated with S as described in Section 2.1, whose edges are $e_1, \ldots, e_8$. The simple admissible loop (e_1, e_2, e_3) induces $0 \leq -1$, which implies infeasibility of S by Theorem 6. However, the extended graph $\bar{G}_T$ of $S \cup T$, which coincides with the graph G_T associated with $S \cup T$, has only two simple admissible loops with initial vertex v_0, namely, (e_6, e_1, e_2, e_9) and (e_7, e_2, e_9). Since neither of the two loops is infeasible, the Cohen–Megiddo's procedure decides that $S \cup T$ is "feasible," which is a contradiction.

Nevertheless, if the Cohen–Megiddo's procedure finds a nonempty subset T' of size at most two, then the system $S \cup T'$ is known to be infeasible without regard to feasibility of S. Moreover, we will show in Lemma 2 below that T' has the form $\{z_i \leq 0\}$, which corresponds to an index i such that $z_i > 0$ for any solution z to LCP(M, q). Then, in a similar way to Algorithm 1, we can add a new constraint $(Mz + q)_i \leq 0$ to S by complementarity, and repeat this until the Cohen–Megiddo's procedure returns that $S \cup T$ is "feasible." During and at the end of the repetition, we do not require that the sign-balanced TVPI system S is feasible. Instead, we need to solve a sign-balanced TVPI system at the last step in order to verify the feasibility of S.

A formal description of our algorithm is given as follows. For a set $I \subseteq [n]$, let $\bar{I} := [n] \setminus I$. We denote by z_I a subvector of z which consists of entries with coordinates in $I \subseteq [n]$.

Algorithm 2.

Step 1. $I := [n]$, $F := \{z \mid Mz + q \geq 0, z \geq 0\}$.

Step 2. While $I \neq \emptyset$, do Step 3.

Step 3. Let S be the sign-balanced TVPI system $Mz + q \geq 0, z \geq 0, (Mz + q)_{\bar{I}} \leq 0$ and $T = \{z_i \leq 0 \mid i \in I\}$.
Execute the Cohen–Megiddo's procedure with inputs S and T. If the procedure returns $T' = \{z_i \leq 0\}$ (for some $i \in I$), then update $I \leftarrow I \setminus \{i\}$, $F \leftarrow F \cap \{z \mid (Mz + q)_i \leq 0\}$ and go to the next iteration. Otherwise, go to Step 4.

Step 4. Find a feasible vector of $F \cap \{z \mid z_I \leq 0\}$, that is, solve

$$z_I = 0,\ z_{\bar{I}} \geq 0,\ (Mz+q)_I \geq 0,\ (Mz+q)_{\bar{I}} = 0. \qquad (2)$$

If a feasible vector z^* exists, then return z^*. Otherwise, return that LCP(M, q) is infeasible.

For correctness of Algorithm 2, we show the following lemma. Note that throughout Algorithm 2, F remains to be the feasible region of a sign-balanced TVPI system.

Lemma 2. *Let LCP(M, q) be a sign-balanced 2-LCP instance. LCP(M, q) has a solution if and only if the sign-balanced TVPI system* (2) *is feasible.*

Proof. The if-part is easy to see. It suffices to show the only-if-part. Suppose that LCP(M, q) has a solution. At Step 1, F is not empty because $SOL(M, q) \subseteq F$.

We will show that throughout the execution of Step 3, it holds that

1. any $z \in SOL(M, q)$ satisfies $z_i > 0$ for all $i \in \bar{I}$, and
2. $SOL(M, q) \subseteq F$.

These claims hold at the beginning, that is, when $I = [n]$. Suppose that the claims hold for some $I \subseteq [n]$. We may assume that the Cohen–Megiddo's procedure returns a nonempty subset $T' \subseteq T$. Since T contains only upper bounds, T' contains only one upper bound, that is, $z_i \leq 0$ for some $i \in I$, by nonemptiness of F and Lemma 1. Since $S \cup \{z_i \leq 0\}$ is infeasible, any $z' \in F$ satisfies $z'_i > 0$, and hence any $z \in SOL(M, q)$ satisfies $(Mz + q)_i = 0$ by $SOL(M, q) \subseteq F$. This implies that $SOL(M, q) \subseteq (F \cap \{z \mid (Mz + q)_i \leq 0\})$. Thus the claims hold.

Therefore, S always has a solution during Step 3 by the second claim. Hence, when we go to Step 4, $S \cup \{z_i \leq 0 \mid i \in I\}$ is feasible, because the Cohen–Megiddo's procedure works correctly. Thus (2) has a solution. □

We discuss the time complexity of Algorithm 2. The following theorem implies Theorem 1.

Theorem 9. *Algorithm 2 solves sign-balanced 2-LCP of order n in* $\mathrm{O}(n^3 \log n)$ *time.*

Proof. The number of repetitions in Step 3 is at most n since $|I|$ decreases by one at each repetition. The execution time of each repetition is $\mathrm{O}(n^2)$ time by Theorem 7. Therefore, Algorithm 2 takes $\mathrm{O}(n^3)$ time to go through Step 3. At Step 4, since the TVPI system (2) has at most $3n$ inequalities, it is solvable in $\mathrm{O}(n^3 \log n)$ time by the algorithm of Hochbaum and Naor [20]. This concludes the proof. □

Remark 1. The running time of Algorithm 2 can be written as $\mathrm{O}(n^3 + T_{\mathrm{LI}}(n, n))$ time, where $T_{\mathrm{LI}}(m, n)$ denotes the time complexity for solving a TVPI system with m constraints and n variables. In other words, Algorithm 2 reduces sign-balanced 2-LCP to sign-balanced TVPI system in $\mathrm{O}(n^3)$ time.

Remark 2. A sign-balanced TVPI system $Ax \leq b, x \geq 0$ where $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^n$ can be formulated as a sign-balanced 2-LCP instance with

$$M = \begin{array}{c|cc} & m & n \\ m & 0 & -A \\ n & 0 & 0 \end{array}, \quad q = \begin{pmatrix} b \\ 0 \end{pmatrix}.$$

This implies that sign-balanced 2-LCP cannot be solved faster than sign-balanced TVPI system with nonnegativity constraints, whose current best running time is $\mathrm{O}(n^3 \log n)$ [20] when $m = \mathrm{O}(n)$. Theorem 9 shows that Algorithm 2 achieves the same running time.

5 NP-Hardness for 2-LCP

In this section, we prove Theorem 2, which says that 2-LCP is NP-hard. This is contrast to the fact that a TVPI system can be solved in polynomial time even if the system is not sign-balanced. The NP-hardness can be proved by reduction of monotone one-in-three 3SAT to 2-LCP.

Given a monotone 3CNF formula $\psi = \bigwedge_{j=1}^{m}(x_{j_1} \vee x_{j_2} \vee x_{j_3})$ with n literals, the *monotone one-in-three 3SAT* is a problem to decide whether there exists an assignment to $(x_1, \ldots, x_n)$ so that for each clause, exactly one literal is true. The monotone one-in-three 3SAT is introduced and proved to be NP-complete by Schaefer [28].

Proof of Theorem 2. Let $\psi = \bigwedge_{j=1}^{m}(x_{j_1} \vee x_{j_2} \vee x_{j_3})$ be a monotone one-in-three 3SAT instance with n literals. We construct an instance of 2-LCP of order $n+9m$ from ψ as follows: for each literal $i = 1, \ldots, n$, define

$$w_i + z_i = 1. \tag{3}$$

Moreover, for each clause $j = 1, \ldots, m$, letting $p_j = n + 9(j-1)$, set

$$w_{p_j+1} + z_{j_2} + z_{j_3} = 1,\ z_{j_1} + w_{p_j+2} + z_{j_3} = 1,\ z_{j_1} + z_{j_2} + w_{p_j+3} = 1, \tag{4}$$

and in addition, set

$$\begin{aligned} w_{p_j+4} - z_{p_j+1} - z_{j_1} &= -1, & w_{p_j+5} + z_{p_j+1} + z_{j_1} &= 1, \\ w_{p_j+6} - z_{p_j+2} - z_{j_2} &= -1, & w_{p_j+7} + z_{p_j+2} + z_{j_2} &= 1, \\ w_{p_j+8} - z_{p_j+3} - z_{j_3} &= -1, & w_{p_j+9} + z_{p_j+3} + z_{j_3} &= 1. \end{aligned} \tag{5}$$

Consider the instance of 2-LCP consisting of the above constraints (3), (4) and (5). Note that (5) is equivalent to

$$z_{p_j+1} + z_{j_1} = 1,\ z_{p_j+2} + z_{j_2} = 1,\ z_{p_j+3} + z_{j_3} = 1, \tag{6}$$

since $w_{p_j+\ell} \geq 0$ for $\ell = 4, 5, \ldots, 9$.

We denote by M and q the coefficient matrix and the constant vector of the above instance of 2-LCP. We will show that $\mathrm{LCP}(M, q)$ has a solution if and only if the monotone one-in-three 3SAT instance ψ is a true instance.

First assume that $\mathrm{LCP}(M, q)$ has a solution (w, z). By (3), for any $i = 1, \ldots, n$, it holds that $(w_i, z_i) = (0, 1)$ or $(w_i, z_i) = (1, 0)$. Assign each literal x_i true if $z_i = 1$ and false otherwise.

We will claim that x is a truth assignment for ψ, that is, each clause has exactly one true literal. Indeed, for each clause j, if $z_{j_1} = 0$ then $w_{p_j+1} = 0$ by (6) and the complementarity, and hence exactly one of z_{j_2} and z_{j_3} is equal to one by the first equation in (4). If $z_{j_1} = 1$ then $z_{j_2} = z_{j_3} = 0$ by the second and third equations in (4). Thus each clause has exactly one true literal.

Conversely, assume that ψ is a true instance. Let $x = (x_1, \dots, x_n)$ be a truth assignment of ψ. Define $z \in \mathbb{R}^{n+9m}$ as follows: For $i = 1, \dots, n$, set $z_i = 1$ if x_i is true, and $z_i = 0$ if x_i is false. For $j = 1, \dots, m$, set $z_{p_j+\ell} = 1 - z_{j_\ell}$ for $\ell = 1, 2, 3$ and $z_{p_j+\ell} = 0$ for $\ell = 4, \dots, 9$. Define $w = Mz + q$. Then the pair (w, z) satisfies (3), (4) and (5), and $w, z \geq 0$ holds.

We claim that the pair (w, z) is a solution of LCP(M, q). To prove this, it remains to show that the pair (w, z) satisfies $w^\top z = 0$. For $i = 1, \dots, n$, it clearly holds that $w_i z_i = 0$ by (3). Let $j \in \{1, \dots, m\}$. Since the clause j has exactly one true literal, we may suppose by symmetry that $z_{j_1} = 1$ and $z_{j_2} = z_{j_3} = 0$. By (4), it holds that $w_{p_j+1} = 1$ and $w_{p_j+2} = w_{p_j+3} = 0$. On the other hand, we have $z_{p_j+1} = 0$ and $z_{p_j+2} = z_{p_j+3} = 1$ by (6), which means that $w_{p_j+\ell} z_{p_j+\ell} = 0$ for $\ell = 1, 2, 3$. For $\ell = 4, \dots, 9$, we have $w_{p_j+\ell} z_{p_j+\ell} = 0$ since $z_{p_j+\ell} = 0$. Thus the complementarity condition is satisfied.

Therefore, LCP(M, q) has a solution if and only if ψ is a true instance, and thus the statement holds. □

Acknowledgement. This work is partially supported by a Grant-in-Aid for Scientific Research from the Ministry of Education, Culture, Sports, Science and Technology of Japan, by the Global COE "The Research and Training Center for New Development in Mathematics," and by JST, ERATO, Kawarabayashi Large Graph Project.

References

1. Aspvall, B., Shiloach, Y.: A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality. SIAM Journal on Computing 9, 827–845 (1980)
2. Björklund, H., Svensson, O., Vorobyov, S.: Linear complementarity algorithms for mean payoff games. Technical Report 2005-05, DIMACS: Center for Discrete Mathematics and Theoretical Computer Science (2005)
3. Chandrasekaran, R.: A special case of the complementary pivot problem. Opsearch 7, 263–268 (1970)
4. Chandrasekaran, R.: Integer programming problems for which a simple rounding type algorithm works. Combinatorial Optimization 8, 101–106 (1984)
5. Chandrasekaran, R., Kabadi, S.N., Sridhar, R.: Integer solution for linear complementarity problem. Mathematics of Operations Research 23, 390–402 (1998)
6. Chen, X., Deng, X., Teng, S.-H.: Sparse games are hard. In: Spirakis, P.G., Mavronicolas, M., Kontogiannis, S.C. (eds.) WINE 2006. LNCS, vol. 4286, pp. 262–273. Springer, Heidelberg (2006)
7. Chung, S.J.: NP-completeness of the linear complementarity problem. Journal of Optimization Theory and Applications 60, 393–399 (1989)
8. Codenotti, B., Leoncini, M., Resta, G.: Efficient computation of nash equilibria for very sparse win-lose bimatrix games. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 232–243. Springer, Heidelberg (2006)

9. Cohen, E., Megiddo, N.: Improved algorithms for linear inequalities with two variables per inequality. SIAM Journal on Computing 23, 1313–1347 (1994)
10. Cottle, R.W.: The principal pivoting method of quadratic programming. In: Dantzig, G.B., Veinott, A.F. (eds.) Mathematics of Decision Sciences, Part 1, pp. 142–162. American Mathematical Society, Providence R. I. (1968)
11. Cottle, R.W., Dantzig, G.B.: Complementary pivot theory of mathematical programming. Linear Algebra and Its Applications 1, 103–125 (1968)
12. Cottle, R.W., Dantzig, G.B.: A generalization of the linear complementarity problem. Journal on Combinatorial Theory 8, 79–90 (1970)
13. Cottle, R.W., Pang, J.S., Stone, R.E.: The Linear Complementarity Problem. Academic Press, Boston (1992)
14. Cottle, R.W., Veinott, A.F.: Polyhedral sets having a least element. Mathematical Programming 3, 238–249 (1972)
15. Cunningham, W.H., Geelen, J.F.: Integral solutions of linear complementarity problems. Mathematics of Operations Research 23, 61–68 (1998)
16. Daskalakis, C., Papadimitriou, C.H.: On oblivious PTAS's for Nash equilibrium. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, pp. 75–84 (2009)
17. Du Val, P.: The unloading problem for plane curves. American Journal of Mathematics 62, 307–311 (1940)
18. Hermelin, D., Huang, C., Kratsch, S., Wahlström, M.: Parameterized two-player Nash equilibrium. Algorithmica, 1–15 (2012)
19. Hochbaum, D.S., Megiddo, N., Naor, J., Tamir, A.: Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality. Mathematical Programming 62, 69–83 (1993)
20. Hochbaum, D.S., Naor, J.: Simple and fast algorithms for linear and integer programs with two variables per inequality. SIAM Journal on Computing 23, 1179–1192 (1994)
21. Kakimura, N.: Sign-solvable linear complementarity problems. Linear Algebra and Its Applications 429, 606–616 (2008)
22. Kojima, M., Megiddo, N., Noma, T., Yoshise, A.: A Unified Approach to Interior Point Algorithms for Linear Complementarity Problems. LNCS, vol. 538. Springer, Heidelberg (1991)
23. Lagarias, J.C.: The computational complexity of simultaneous Diophantine approximation problems. SIAM Journal on Computing 14, 196–209 (1985)
24. Lemke, C.E.: Bimatrix equilibrium points and mathematical programming. Management Science 11, 681–689 (1965)
25. Mangasarian, O.L.: Linear complementarity problems solvable by a single linear program. Mathematical Programming 10, 263–270 (1976)
26. Murty, K.G.: Linear Complementarity, Linear and Nonlinear Programming. Internet Edition (1997)
27. Samelson, H., Thrall, R.M., Wesler, O.: A partition theorem for Euclidean n-space. Proceedings of the American Mathematical Society 9, 805–807 (1958)
28. Schaefer, T.J.: The complexity of satisfiability problems. In: Proceedings of the 10th Annual ACM Symposium on Theory of Computing, pp. 216–226 (1978)
29. Shostak, R.: Deciding linear inequalities by computing loop residues. Journal of the ACM 28, 769–779 (1981)
30. Sumita, H., Kakimura, N., Makino, K.: Sparse linear complementarity problems. METR 2013-02, Department of Mathematical Informatics, University of Tokyo (2013)

LP-Rounding Algorithms for the Fault-Tolerant Facility Placement Problem[⋆]

(Extended Abstract)

Li Yan and Marek Chrobak

Department of Computer Science, University of California at Riverside
{lyan,marek}@cs.ucr.edu

Abstract. The Fault-Tolerant Facility Placement problem (FTFP) is a generalization of the Uncapacitated Facility Location Problem (UFL). In FTFP we are given a set of facility sites and a set of clients. Opening a facility at site i costs f_i and connecting client j to a facility at site i costs d_{ij}, where the costs d_{ij} satisfy the triangle inequality. Multiple facilities can be opened at any site. Each client j has a demand r_j, which means that it needs to be connected to r_j different facilities. The goal is to minimize the sum of facility opening cost and connection cost. The main result of this paper is a 1.575-approximation algorithm for FTFP, based on LP-rounding.

1 Introduction

In the *Fault-Tolerant Facility Placement* problem (FTFP), we are given a set $\mathbb{F}$ of *sites* at which facilities can be built, and a set $\mathbb{C}$ of *clients* with some demands that need to be satisfied by different facilities. A client j has demand r_j. Building one facility at a site i costs f_i, and connecting one unit of demand from client j to a facility at site $i \in \mathbb{F}$ costs d_{ij}. The connection costs d_{ij} are symmetric and satisfy the triangle inequality. In a feasible solution, some number of facilities, possibly zero, are opened at each site i, and demands from each client are connected to those open facilities, with the constraint that demands from the same client have to be connected to different facilities.

It is easy to see that if all $r_j = 1$ then FTFP reduces to the classic Uncapacitated Facility Location problem (UFL). If we add a constraint that each site can have at most one facility, then the problem is equivalent to the Fault-Tolerant Facility Location problem (FTFL).

Great progress has been achieved lately in designing approximation algorithms for UFL. Shmoys *et al.* [14] proposed an approach based on LP-rounding, achieving a ratio of 3.16. This was then improved by Chudak [5] to 1.736, and later by Sviridenko [15] to 1.582. Byrka [2] gave an improved algorithm with ratio 1.5,

⋆ Research supported by NSF Grant CCF-1217314. Full version available in [20].

by a combination of LP-rounding with dual-fitting techniques. Recently, Li [12] refined the method from [2] to obtain ratio 1.488, which is now the best known approximation result for UFL. Other techniques include the primal-dual algorithm by Jain and Vazirani [10], the dual fitting method by Jain *et al.* [9], and a local search heuristic by Arya *et al.* [1]. On the hardness side, it is known that it is not possible to approximate UFL in polynomial time with ratio less than 1.463, provided that $\mathsf{NP} \not\subseteq \mathsf{DTIME}(n^{O(\log\log n)})$ [6]. An observation by Sviridenko strengthened this assumption to $\mathsf{P} \neq \mathsf{NP}$ [17].

FTFL was first introduced by Jain and Vazirani [11] who gave a primal-dual algorithm with ratio $3\ln(\max_{j\in\mathbb{C}} r_j)$. All subsequently discovered constant-ratio approximation algorithms use variations of LP-rounding, including the work by Guha *et al.* [7], Swamy and Shmoys [16], and Byrka *et al.* [4], who improved the ratio to 1.7245, the best known approximation ratio for FTFL.

FTFP is a natural generalization of UFL. It was first studied by Xu and Shen [18], who presented an approximation algorithm with a ratio claimed to be 1.861. However their algorithm runs in polynomial time only if $\max_{j\in\mathbb{C}} r_j$ is polynomial in $O(|\mathbb{F}| \cdot |\mathbb{C}|)$ and their analysis of the approximation ratio is flawed[1]. To date, the best ratio for FTFP is 3.16 in [19], while the only known lower bound is the 1.463 lower bound for UFL from [6], that applies to FTFP.

The main result of this paper is an LP-rounding algorithm for FTFP with approximation ratio 1.575, matching the best ratio for UFL achieved via the LP-rounding method [3] and significantly improving the bound in [19]. In Section 3 we prove that, for the purpose of LP-based approximations, we can assume that all demand values are polynomial in the number of sites. This *demand reduction* trick itself gives us ratio 1.7245, since we can then reduce the problem to FTFL and use the algorithm from [4]. It also ensures that our algorithms run in polynomial time. If all demand values r_j are equal, the problem can be solved by simple scaling and applying LP-rounding algorithms for UFL. This does not affect the approximation ratio, thus achieving ratio 1.575 for this special case (see also [13]). In Section 4, we demonstrate a technique called *adaptive partitioning*, which splits clients into unit demands and partitions the optimal fractional solution into a fractional solution of the split instance. By exploiting structural properties of the partitioned solution we were able to extend UFL rounding algorithms in [8,5,3], retaining the approximation ratio.

2 The LP Formulation

The FTFP problem has a natural Integer Programming (IP) formulation. Let y_i represent the number of facilities built at site i and let x_{ij} represent the number of connections from client j to facilities at site i. If we relax the integrality constraints, we obtain the following LP and its dual:

[1] Confirmed through private communication with the authors.

$$\begin{array}{llll} \min\ cost(\boldsymbol{x},\boldsymbol{y}) = \sum_{i\in\mathbb{F}} f_i y_i + \sum_{i\in\mathbb{F},j\in\mathbb{C}} d_{ij}x_{ij} & & & \\ & (1) & & \\ \text{s.t. } y_i - x_{ij} \geq 0 & \forall i \in \mathbb{F}, j \in \mathbb{C} & & \\ \quad \sum_{i\in\mathbb{F}} x_{ij} \geq r_j & \forall j \in \mathbb{C} & & \\ \quad x_{ij} \geq 0, y_i \geq 0 & \forall i \in \mathbb{F}, j \in \mathbb{C} & & \end{array}$$

$$\begin{array}{lll} \max & \sum_{j\in\mathbb{C}} r_j\alpha_j & \qquad (2) \\ \text{s.t.} & \sum_{j\in\mathbb{C}} \beta_{ij} \leq f_i & \forall i \in \mathbb{F} \\ & \alpha_j - \beta_{ij} \leq d_{ij} & \forall i \in \mathbb{F}, j \in \mathbb{C} \\ & \alpha_j \geq 0, \beta_{ij} \geq 0 & \forall i \in \mathbb{F}, j \in \mathbb{C} \end{array}$$

In the paper we fix some optimal solutions of the LPs (1) and (2) that we denote by $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ and $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$, respectively. We then define the optimal facility cost as $F^* = \sum_{i\in\mathbb{F}} f_i y_i^*$ and the optimal connection cost as $C^* = \sum_{i\in\mathbb{F},j\in\mathbb{C}} d_{ij}x_{ij}^*$. Then $\mathrm{LP}^* = cost(\boldsymbol{x}^*, \boldsymbol{y}^*) = F^* + C^*$ is the joint optimal value of (1) and (2). We can also associate with each client j its fractional connection cost $C_j^* = \sum_{i\in\mathbb{F}} d_{ij}x_{ij}^*$. We use notation OPT for the optimal integral solution of (1).

Define $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ to be *complete* if $x_{ij}^* > 0$ implies that $x_{ij}^* = y_i^*$ for all i, j. As shown in [5], we can modify the given instance by adding at most $|\mathbb{C}|$ sites to obtain an equivalent instance that has a complete optimal solution.

3 Reduction to Polynomial Demands

This section presents a *demand reduction* trick that reduces the problem for arbitrary demands to a special case where demands are bounded by $|\mathbb{F}|$. The reduction is based on a complete optimal fractional solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ of LP (1). We split this solution into two parts, namely $(\boldsymbol{x}^*, \boldsymbol{y}^*) = (\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) + (\dot{\boldsymbol{x}}, \dot{\boldsymbol{y}})$, where $\hat{y}_i \leftarrow \lfloor y_i^* \rfloor$, $\hat{x}_{ij} \leftarrow \lfloor x_{ij}^* \rfloor$ and $\dot{y}_i \leftarrow y_i^* - \lfloor y_i^* \rfloor$, $\dot{x}_{ij} \leftarrow x_{ij}^* - \lfloor x_{ij}^* \rfloor$ for all i, j. Now we construct two FTFP instances $\hat{\mathcal{I}}$ and $\dot{\mathcal{I}}$ with the same parameters as the original instance, except that the demand of each client j is $\hat{r}_j = \sum_{i\in\mathbb{F}} \hat{x}_{ij}$ in $\hat{\mathcal{I}}$ and $\dot{r}_j = \sum_{i\in\mathbb{F}} \dot{x}_{ij} = r_j - \hat{r}_j$ in $\dot{\mathcal{I}}$. It is obvious that if we have integral solutions to both $\hat{\mathcal{I}}$ and $\dot{\mathcal{I}}$ then, when added together, they form an integral solution to the original instance. Moreover, it can be shown that (i) $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$ is a feasible integral solution to instance $\hat{\mathcal{I}}$. (ii) $(\dot{\boldsymbol{x}}, \dot{\boldsymbol{y}})$ is a feasible fractional solution to instance $\dot{\mathcal{I}}$. (iii) $\dot{r}_j \leq |\mathbb{F}|$ for every client j. From these properties, we derive the following theorem (proof omitted).

Theorem 1. *Suppose that there is a polynomial-time algorithm $\mathcal{A}$ that, for any instance of* FTFP *with maximum demand bounded by $|\mathbb{F}|$, computes an integral solution that approximates the fractional optimum of this instance within factor $\rho \geq 1$. Then there is a ρ-approximation algorithm $\mathcal{A}'$ for* FTFP.

4 Adaptive Partitioning

In this section we develop our second technique, which we call *adaptive partitioning.* Given an FTFP instance and an optimal fractional solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ to LP (1), we split each client j into r_j individual *unit demand points* (or just *demands*), and we split each site i into no more than $|\mathbb{F}| + 2R|\mathbb{C}|^2$ *facility points* (or *facilities*), where $R = \max_{j\in\mathbb{C}} r_j$. We denote the demand set by $\overline{\mathbb{C}}$ and the facility set by $\overline{\mathbb{F}}$, respectively. We will also partition $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ into a fractional solution $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ for the split instance. We will typically use symbols ν and μ to index demands and facilities respectively, that is $\bar{\boldsymbol{x}} = (\bar{x}_{\mu\nu})$ and $\bar{\boldsymbol{y}} = (\bar{y}_\mu)$. The *neighborhood of a demand* ν is $\overline{N}(\nu) = \{\mu \in \overline{\mathbb{F}} : \bar{x}_{\mu\nu} > 0\}$. We will use notation $\nu \in j$ to mean that ν is a demand of client j; similarly, $\mu \in i$ means that facility μ is on site i. Different demands of the same client (that is, $\nu, \nu' \in j$) are called *siblings*. Further, we use the convention that $f_\mu = f_i$ for $\mu \in i$, $\alpha^*_\nu = \alpha^*_j$ for $\nu \in j$ and $d_{\mu\nu} = d_{\mu j} = d_{ij}$ for $\mu \in i$ and $\nu \in j$. We define $C^{\text{avg}}_\nu = \sum_{\mu\in\overline{N}(\nu)} d_{\mu\nu}\bar{x}_{\mu\nu} = \sum_{\mu\in\overline{\mathbb{F}}} d_{\mu\nu}\bar{x}_{\mu\nu}$. One can think of C^{avg}_ν as the average connection cost of demand ν, if we chose a connection to facility μ with probability $\bar{x}_{\mu\nu}$. Our partition algorithm resembles superficially to the uncrossing technique in Guha *et al.*'s [7] $O(1)$-approximation algorithm for FTFL, nonetheless our construction and resulted structure is more delicate.

Some demands in $\overline{\mathbb{C}}$ will be designated as *primary demands* and the set of primary demands will be denoted by P. In addition, we will use the overlap structure between demand neighborhoods to define a mapping that assigns each demand $\nu \in \overline{\mathbb{C}}$ to some primary demand $\kappa \in P$. As shown in the rounding algorithms in later sections, for each primary demand we guarantee exactly one open facility in its neighborhood, while for a non-primary demand we estimate its connection cost by the distance to the facility opened by its assigned primary demand. For this reason the connection cost of a primary demand must be "small" compared to the non-primary demands assigned to it. To have a bound on the facility cost, we can think of the fractional x^*_{ij} as the budget of how much a client j can use a site i, so (PS.2) means all demands of a client j together can only use as much as x^*_{ij}. The reason we need (PS.1) is two-fold: (i) to have a proper distribution when select one of facility in a primary demand's neighborhood and, (ii) to obtain an estimate on the probability that none of a demand's neighbor is open. We also need sibling demands assigned to different primary demands to satisfy the fault-tolerance requirement. The properties are detailed below.

(PS) *Partitioned solution.* Vector $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is a partition of $(\boldsymbol{x}^*, \boldsymbol{y}^*)$, with unit-value demands, that is:
1. $\sum_{\mu\in\overline{\mathbb{F}}} \bar{x}_{\mu\nu} = 1$ for each demand $\nu \in \overline{\mathbb{C}}$.
2. $\sum_{\mu\in i,\nu\in j} \bar{x}_{\mu\nu} = x^*_{ij}$ for each site $i \in \mathbb{F}$ and client $j \in \mathbb{C}$.
3. $\sum_{\mu\in i} \bar{y}_\mu = y^*_i$ for each site $i \in \mathbb{F}$.

(CO) *Completeness.* Solution $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is complete, that is $\bar{x}_{\mu\nu} \neq 0$ implies $\bar{x}_{\mu\nu} = \bar{y}_\mu$, for all $\mu \in \overline{\mathbb{F}}, \nu \in \overline{\mathbb{C}}$.

(PD) *Primary demands.* Primary demands satisfy the following conditions:
1. For any two different primary demands $\kappa, \kappa' \in P$ we have $\overline{N}(\kappa) \cap \overline{N}(\kappa') = \emptyset$.
2. For each site $i \in \mathbb{F}$, $\sum_{\mu\in i}\sum_{\kappa\in P} \bar{x}_{\mu\kappa} \leq y^*_i$.
3. Each demand $\nu \in \overline{\mathbb{C}}$ is assigned to one primary demand $\kappa \in P$ such that
   (a) $\overline{N}(\nu) \cap \overline{N}(\kappa) \neq \emptyset$, and

(b) $C^{\text{avg}}_{\nu} + \alpha^*_{\nu} \geq C^{\text{avg}}_{\kappa} + \alpha^*_{\kappa}$.

(SI) *Siblings.* For any pair ν, ν' of different siblings we have
1. $\overline{N}(\nu) \cap \overline{N}(\nu') = \emptyset$.
2. If ν is assigned to a primary demand κ then $\overline{N}(\nu') \cap \overline{N}(\kappa) = \emptyset$. In particular, by Property (PD.3(a)), this implies that different sibling demands are assigned to different primary demands.

As we shall demonstrate in later sections, these properties allow us to extend known UFL rounding algorithms to obtain an integral solution to our FTFP problem with a matching approximation ratio. (For the 1.575-approximation algorithm in Section 7, these properties will need to be slightly refined.)

Implementation of Adaptive Partitioning. Recall that $\overline{\mathbb{F}}$ and $\overline{\mathbb{C}}$, respectively, denote the sets of facilities and demands that will be created in this stage, and $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is the partitioned solution to be computed. The partitioning algorithm consists of two phases: Phase 1 is called the partition phase and Phase 2 is called the augmenting phase. Phase 1 is done in iterations, where in each iteration we find the "best" client j and create a new demand ν out of it. This demand either becomes a primary demand itself, or it is assigned to some existing primary demand. We call a client j *exhausted* when all its r_j demands have been created and assigned to some primary demands. Phase 1 completes when all clients are exhausted. In Phase 2 we ensure that every demand has a total connection values equal to 1, that is condition (PS.1).

For each site i we will initially create one "big" facility μ with initial value $\bar{y}_{\mu} = y^*_i$. While we are partitioning the instance, creating new demands and connections, this facility may end up being split into more facilities to preserve completeness of the fractional solution. Also, we will gradually decrease the fractional connection vector for each client j, to account for the demands already created for j and their connection values. These decreased connection values will be stored in an auxiliary vector $\widetilde{\boldsymbol{x}}$. The intuition is that $\widetilde{\boldsymbol{x}}$ represents the part of $\boldsymbol{x}^*$ that still has not been allocated to existing demands and future demands can use $\widetilde{\boldsymbol{x}}$ for their connections. For technical reasons, $\widetilde{\boldsymbol{x}}$ will be indexed by facilities (rather than sites) and clients, that is $\widetilde{\boldsymbol{x}} = (\widetilde{x}_{\mu j})$. At the beginning, we set $\widetilde{x}_{\mu j} \leftarrow x^*_{ij}$ for each $j \in \mathbb{C}$, where $\mu \in i$ is the single facility created initially at site i. At each step, whenever we create a new demand ν for a client j, we will define its values $\bar{x}_{\mu\nu}$ and appropriately reduce the values $\widetilde{x}_{\mu j}$, for all facilities μ. We will deal with two types of neighborhoods, with respect to $\widetilde{\boldsymbol{x}}$ and $\bar{\boldsymbol{x}}$, that is $\widetilde{N}(j) = \{\mu \in \overline{\mathbb{F}} : \widetilde{x}_{\mu j} > 0\}$ for $j \in \mathbb{C}$ and $\overline{N}(\nu) = \{\mu \in \overline{\mathbb{F}} : \bar{x}_{\mu\nu} > 0\}$ for $\nu \in \overline{\mathbb{C}}$. During this process, the following properties will hold for every facility μ after every iteration:

(c1) For each demand ν either $\bar{x}_{\mu\nu} = 0$ or $\bar{x}_{\mu\nu} = \bar{y}_{\mu}$.
(c2) For each client j, either $\widetilde{x}_{\mu j} = 0$ or $\widetilde{x}_{\mu j} = \bar{y}_{\mu}$.

It may appear that (c1) is the same property as (CO), yet we repeat it here as (c1) needs to hold after every iteration, while (CO) only applies to the final partitioned fractional solution $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$. A full description of the algorithm is given in Pseudocode 1. The two helper routines NearestUnitChunk and

AugmentToUnit can be found in the full version [20]. At a high level, NearestUnitChunk orders facilities in $\widetilde{N}(j)$ by nonincreasing order of distance to j and returns a closest set of facilities such that the total connection value $\widetilde{x}_{\mu j}$ is equal to 1. AugmentToUnit works by adding facilities to $\overline{N}(\nu)$ for each demand ν until every one of them has a total connection value equal 1.

Pseudocode 1. Algorithm: Adaptive Partitioning

```
Input: $\mathbb{F}$, $\mathbb{C}$, $(\boldsymbol{x}^*, \boldsymbol{y}^*)$
Output: $\overline{\mathbb{F}}$, $\overline{\mathbb{C}}$, $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$        ▷ Unspecified $\bar{x}_{\mu\nu}$'s and $\widetilde{x}_{\mu j}$'s are assumed to be 0
 1: $\widetilde{\boldsymbol{r}} \leftarrow \boldsymbol{r}, U \leftarrow \mathbb{C}, \overline{\mathbb{F}} \leftarrow \emptyset, \overline{\mathbb{C}} \leftarrow \emptyset, P \leftarrow \emptyset$        ▷ Phase 1
 2: for each site $i \in \mathbb{F}$ do
 3:     create a facility $\mu$ at $i$ and add $\mu$ to $\overline{\mathbb{F}}$, $\bar{y}_\mu \leftarrow y_i^*$ and $\widetilde{x}_{\mu j} \leftarrow x_{ij}^*$ for each $j \in \mathbb{C}$
 4: while $U \neq \emptyset$ do
 5:     for each $j \in U$ do
 6:         $\widetilde{N}_1(j) \leftarrow$ NearestUnitChunk$(j, \overline{\mathbb{F}}, \widetilde{\boldsymbol{x}}, \bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$
 7:         $\mathrm{tcc}(j) \leftarrow \sum_{\mu \in \widetilde{N}_1(j)} d_{\mu j} \cdot \widetilde{x}_{\mu j}$
 8:     $p \leftarrow \arg\min_{j \in U}\{\mathrm{tcc}(j) + \alpha_j^*\}$
 9:     create a new demand $\nu$ for client $p$
10:     if $\widetilde{N}_1(p) \cap \overline{N}(\kappa) \neq \emptyset$ for some primary demand $\kappa \in P$ then
11:         assign $\nu$ to $\kappa$, $\bar{x}_{\mu\nu} \leftarrow \widetilde{x}_{\mu p}$ and $\widetilde{x}_{\mu p} \leftarrow 0$ for each $\mu \in \widetilde{N}(p) \cap \overline{N}(\kappa)$
12:     else
13:         make $\nu$ primary, $P \leftarrow P \cup \{\nu\}$, set $\bar{x}_{\mu\nu} \leftarrow \widetilde{x}_{\mu p}$ and $\widetilde{x}_{\mu p} \leftarrow 0$ for each $\mu \in \widetilde{N}_1(p)$
14:     $\overline{\mathbb{C}} \leftarrow \overline{\mathbb{C}} \cup \{\nu\}, \widetilde{r}_p \leftarrow \widetilde{r}_p - 1$
15:     if $\widetilde{r}_p = 0$ then $U \leftarrow U \setminus \{p\}$
16: for each client $j \in \mathbb{C}$ do        ▷ Phase 2
17:     for each demand $\nu \in j$ do        ▷ each client $j$ has $r_j$ demands
18:         if $\sum_{\mu \in \overline{N}(\nu)} \bar{x}_{\mu\nu} < 1$ then AugmentToUnit$(\nu, j, \overline{\mathbb{F}}, \widetilde{\boldsymbol{x}}, \bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$
```

We start with $|\mathbb{F}|$ facilities and in each iteration each client causes at most one split. We have at most $R|\mathbb{C}|$ iterations as in each iteration we create one demand. (Recall that $R = \max_j r_j$.) In Phase 2 the augmenting operation creates no more than $R|\mathbb{C}|$ new facilities. So the total number of facilities will be at most $|\mathbb{F}| + R|\mathbb{C}|^2 + R|\mathbb{C}| \leq |\mathbb{F}| + 2R|\mathbb{C}|^2$, which is polynomial in $|\mathbb{F}| + |\mathbb{C}|$ due to our bound on R.

Correctness. We now show that all the required properties (PS), (CO), (PD) and (SI) are satisfied. (CO) is implied by the completeness condition (c1). (PS.1) is a result of calling Procedure AugmentToUnit() in Line 18. To see that (PS.2) holds, note that at each step the algorithm maintains the invariant that, for every $i \in \mathbb{F}$ and $j \in \mathbb{C}$, we have $\sum_{\mu \in i} \sum_{\nu \in j} \bar{x}_{\mu\nu} + \sum_{\mu \in i} \widetilde{x}_{\mu j} = x_{ij}^*$. In the end, we will create r_j demands for each client j, with each demand $\nu \in j$ satisfying (PS.1), and hence $\sum_{\nu \in j} \sum_{\mu \in \overline{\mathbb{F}}} \bar{x}_{\mu\nu} = r_j$. As a result we have $\widetilde{x}_{\mu j} = 0$ for every facility $\mu \in \overline{\mathbb{F}}$, and (PS.2) follows. (PS.3) holds because every time we split a facility μ into μ' and μ'', the sum of $\bar{y}_{\mu'}$ and $\bar{y}_{\mu''}$ is equal to the old value of $\bar{y}_\mu$.

Now we deal with properties in group (PD). First, (PD.1) follows directly from the algorithm, Pseudocode 1 (Line 13), since every primary demand has its neighborhood fixed when created, and that neighborhood is disjoint from those of the existing primary demands. Property (PD.2) follows from (PD.1), (CO) and (PS.3). In more detail, it can be justified as follows. By (PD.1), for each $\mu \in i$ there is at most one $\kappa \in P$ with $\bar{x}_{\mu\kappa} > 0$ and we have $\bar{x}_{\mu\kappa} = \bar{y}_\mu$ due to (CO). Let $K \subseteq i$ be the set of those μ's for which such $\kappa \in P$ exists, and denote this κ by κ_μ. Then, using conditions (CO) and (PS.3), we have $\sum_{\mu\in i}\sum_{\kappa\in P}\bar{x}_{\mu\kappa} = \sum_{\mu\in K}\bar{x}_{\mu\kappa_\mu} = \sum_{\mu\in K}\bar{y}_\mu \le \sum_{\mu\in i}\bar{y}_\mu = y_i^*$. Property (PD.3(a)) follows from the way the algorithm assigns primary demands. When demand ν of client p is assigned to a primary demand κ in Line 11 of Pseudocode 1, we move all facilities in $\widetilde{N}(p) \cap \overline{N}(\kappa)$ (the intersection is nonempty) into $\overline{N}(\nu)$, and we never remove a facility from $\overline{N}(\nu)$. We postpone the proof for (PD.3(b)) to Lemma 3.

Finally we argue that the properties in group (SI) hold. (SI.1) is easy, since for any client j, each facility μ is added to the neighborhood of at most one demand $\nu \in j$, by setting $\bar{x}_{\mu\nu}$ to $\bar{y}_\mu$, while other siblings ν' of ν have $\bar{x}_{\mu\nu'} = 0$. Note that right after a demand $\nu \in p$ is created, its neighborhood is disjoint from the neighborhood of p, that is $\overline{N}(\nu) \cap \widetilde{N}(p) = \emptyset$, by Line 11 of Pseudocode 1. Thus all demands of p created later will have neighborhoods disjoint from the set $\overline{N}(\nu)$ before the augmenting phase 2. Furthermore, Procedure AugmentToUnit() preserves this property, because when it adds a facility to $\overline{N}(\nu)$ then it removes it from $\widetilde{N}(p)$, and in case of splitting, one resulting facility is added to $\overline{N}(\nu)$ and the other to $\widetilde{N}(p)$. Property (SI.2) is shown below in Lemma 1.

Lemma 1. *Property (SI.2) holds after the Adaptive Partitioning stage.*

We need one more lemma before proving our last property (PD.3(b)). For a client j and a demand ν, we use notation $\mathrm{tcc}_\nu(j)$ for the value of $\mathrm{tcc}(j)$ at the time when ν was created. (It is not necessary that $\nu \in j$ but we assume that j is not exhausted at that time.)

Lemma 2. *Let η and ν be two demands, with η created no later than ν, and let $j \in \mathbb{C}$ be a client that is not exhausted when ν is created. Then we have (a) $tcc_\eta(j) \le tcc_\nu(j)$, and (b) if $\nu \in j$ then $tcc_\eta(j) \le C_\nu^{\mathrm{avg}}$.*

Lemma 3. *Property (PD.3(b)) holds after the Adaptive Partitioning stage.*

We have thus proved that properties (PS), (CO), (PD) and (SI) hold for our partitioned solution $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$. In the following sections we show how to use these properties to round the fractional solution to an integral solution. (In the 1.575-approximation in Section 7, this partitioning will need to be slightly refined.)

5 Algorithm EGUP with Ratio 3

We first describe a simple algorithm that achieves ratio 3, in order to illustrate how the properties of our partitioned fractional solution are used in rounding to obtain an integral solution with cost close to an optimal solution. The rounding approach is an extension to the method for UFL in [8].

Algorithm EGUP**.** In Algorithm EGUP, we apply a rounding process, guided by the fractional values $(\bar{y}_\mu)$ and $(\bar{x}_{\mu\nu})$, that produces an integral solution. For each primary demand $\kappa \in P$, we open one facility $\phi(\kappa) \in \overline{N}(\kappa)$. To this end, we use randomization: for each $\mu \in \overline{N}(\kappa)$, we choose $\phi(\kappa) = \mu$ with probability $\bar{x}_{\mu\kappa}$, ensuring that exactly one $\mu \in \overline{N}(\kappa)$ is chosen. Note that $\sum_{\mu\in\overline{N}(\kappa)} \bar{x}_{\mu\kappa} = 1$, so this distribution is well-defined. We open this facility $\phi(\kappa)$ and connect κ to this facility $\phi(\kappa)$, as well as all non-primary demands that are assigned to κ.

In our description above, the algorithm is presented as a randomized algorithm. It can be de-randomized using the method of conditional expectations.

Lemma 4. *The expectation of facility cost F_{EGUP} of our solution is at most F^*.*

Lemma 5. *The expectation of connection cost C_{EGUP} of our solution is at most $C^* + 2 \cdot \text{LP}^*$.*

Theorem 2. *Algorithm* EGUP *is a* 3*-approximation algorithm.*

Proof. The feasibility follows from (SI.2) and (PD.1). As for the total cost, Lemma 4 and Lemma 5 imply that the total cost is at most $F^* + C^* + 2 \cdot \text{LP}^* = 3 \cdot \text{LP}^* \leq 3 \cdot \text{OPT}$. □

6 Algorithm ECHS with Ratio 1.736

We now improve the approximation ratio to $1 + 2/e \approx 1.736$. The basic idea to improve the ratio, following the approach of Chudak and Shmoys [5], is to connect a non-primary demand to its nearest neighbor when one is available and only use indirect connections when none of its neighbor is open.

Algorithm ECHS**.** As before, the algorithm starts by solving the linear program and applying the adaptive partitioning algorithm described in Section 4 to obtain a partitioned solution $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$. Then we apply the rounding process to compute an integral solution (see Pseudocode 2).

Analysis. We first show feasibility. The only constraint that is not explicitly enforced by the algorithm is the fault-tolerance requirement; namely that each client j is connected to r_j different facilities. Let ν and ν' be two different sibling demands of client j and let their assigned primary demands be κ and κ' respectively. Due to (SI.2) we know $\kappa \neq \kappa'$. From (SI.1) we have $\overline{N}(\nu) \cap \overline{N}(\nu') = \emptyset$. From (SI.2), we have $\overline{N}(\nu) \cap \overline{N}(\kappa') = \emptyset$ and $\overline{N}(\nu') \cap \overline{N}(\kappa) = \emptyset$. From (PD.1) we have $\overline{N}(\kappa) \cap \overline{N}(\kappa') = \emptyset$. It follows that $(\overline{N}(\nu) \cup \overline{N}(\kappa)) \cap (\overline{N}(\nu') \cup \overline{N}(\kappa')) = \emptyset$. Since the algorithm connects ν to some facility in $\overline{N}(\nu) \cup \overline{N}(\kappa)$ and ν' to some facility in $\overline{N}(\nu') \cup \overline{N}(\kappa')$, ν and ν' will be connected to different facilities.

This integral solution can be shown to have expected facility cost bounded by F^* and connection cost bounded by $C^* + (2/e) \cdot \text{LP}^*$ As a result the expected total cost is bounded by $(1 + 2/e) \cdot \text{LP}^*$. We state this as the theorem below.

Theorem 3. *Algorithm* ECHS *is a* $(1+2/e)$*-approximation algorithm for* FTFP*.*

Pseudocode 2. Algorithm ECHS: Constructing Integral Solution

```
 1: for each $\kappa \in P$ do
 2:     choose one $\phi(\kappa) \in \overline{N}(\kappa)$, with each $\mu \in \overline{N}(\kappa)$ chosen as $\phi(\kappa)$ with probability
        $\bar{y}_\mu$ (note $\bar{x}_{\mu\kappa} = \bar{y}_\mu$ for all $\mu \in \overline{N}(\kappa)$)
 3:     open $\phi(\kappa)$ and connect $\kappa$ to $\phi(\kappa)$
 4: for each $\mu \in \overline{\mathbb{F}} - \bigcup_{\kappa \in P} \overline{N}(\kappa)$ do
 5:     open $\mu$ with probability $\bar{y}_\mu$ (independently)
 6: for each non-primary demand $\nu \in \overline{\mathbb{C}}$ do
 7:     if any facility in $\overline{N}(\nu)$ is open then
 8:         connect $\nu$ to the nearest open facility in $\overline{N}(\nu)$
 9:     else
10:         connect $\nu$ to $\phi(\kappa)$ where $\kappa$ is $\nu$'s primary demand
```

7 Algorithm EBGS with Ratio 1.575

In this section we give our main result, a 1.575-approximation algorithm for FTFP, where 1.575 is the rounded value of $\min_{\gamma \geq 1} \max\{\gamma, 1 + 2/e^\gamma, \frac{1/e+1/e^\gamma}{1-1/\gamma}\}$. This matches the ratio of the best known LP-rounding algorithm for UFL by Byrka *et al.* [3]. Our approach is a combination of the ideas in [3] with the techniques of demand reduction and adaptive partitioning that we introduced earlier. However, our adaptive partitioning technique needs to be carefully modified because now we will be using a more intricate neighborhood structure.

We begin by describing properties that our partitioned fractional solution $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ needs to satisfy. The neighborhood $\overline{N}(\nu)$ of each demand ν will be divided into two disjoint parts. The first part, called the *close neighborhood* $\overline{N}_{\text{cls}}(\nu)$, contains the facilities in $\overline{N}(\nu)$ nearest to ν with the total connection value equal $1/\gamma$. The second part, called the *far neighborhood* $\overline{N}_{\text{far}}(\nu)$, contains the remaining facilities in $\overline{N}(\nu)$ (see below). The respective average connection costs from ν for these sets are defined by $C^{\text{avg}}_{\text{cls}}(\nu) = \gamma \sum_{\mu \in \overline{N}_{\text{cls}}(\nu)} d_{\mu\nu}\bar{x}_{\mu\nu}$ and $C^{\text{avg}}_{\text{far}}(\nu) = \frac{\gamma}{\gamma-1} \sum_{\mu \in \overline{N}_{\text{far}}(\nu)} d_{\mu\nu}\bar{x}_{\mu\nu}$. We will also use notation $C^{\max}_{\text{cls}}(\nu) = \max_{\mu \in \overline{N}_{\text{cls}}(\nu)} d_{\mu\nu}$ for the maximum distance from ν to its close neighborhood.

Our partitioned solution $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ must satisfy properties (PS) and (CO) in Section 4. In addition, it must satisfy new neighborhood property (NB) and modified properties (PD') and (SI'), listed below.

(NB) For each demand ν, its neighborhood is divided into *close* and *far* neighborhood, that is $\overline{N}(\nu) = \overline{N}_{\text{cls}}(\nu) \cup \overline{N}_{\text{far}}(\nu)$, where
- $\overline{N}_{\text{cls}}(\nu) \cap \overline{N}_{\text{far}}(\nu) = \emptyset$,
- $\sum_{\mu \in \overline{N}_{\text{cls}}(\nu)} \bar{x}_{\mu\nu} = 1/\gamma$, and
- if $\mu \in \overline{N}_{\text{cls}}(\nu)$ and $\mu' \in \overline{N}_{\text{far}}(\nu)$ then $d_{\mu\nu} \leq d_{\mu'\nu}$.

Note that the second condition, together with (PS.1), implies that $\sum_{\mu \in \overline{N}_{\text{far}}(\nu)} \bar{x}_{\mu\nu} = 1 - 1/\gamma$.

(PD') *Primary demands.* Primary demands satisfy the following conditions:
1. For any two different primary demands $\kappa, \kappa' \in P$ we have $\overline{N}_{\text{cls}}(\kappa) \cap \overline{N}_{\text{cls}}(\kappa') = \emptyset$.

2. For each site $i \in \mathbb{F}$, $\sum_{\kappa \in P} \sum_{\mu \in i \cap \overline{N}_{\text{cls}}(\kappa)} \bar{x}_{\mu\kappa} \le y_i^*$.
3. Each demand $\nu \in \overline{\mathbb{C}}$ is assigned to one primary demand $\kappa \in P$ such that
   (a) $\overline{N}_{\text{cls}}(\nu) \cap \overline{N}_{\text{cls}}(\kappa) \neq \emptyset$, and
   (b) $C^{\text{avg}}_{\text{cls}}(\nu) + C^{\text{max}}_{\text{cls}}(\nu) \ge C^{\text{avg}}_{\text{cls}}(\kappa) + C^{\text{max}}_{\text{cls}}(\kappa)$.

(SI') *Siblings.* For any pair ν, ν' of different siblings we have
1. $\overline{N}(\nu) \cap \overline{N}(\nu') = \emptyset$.
2. If ν is assigned to a primary demand κ then $\overline{N}(\nu') \cap \overline{N}_{\text{cls}}(\kappa) = \emptyset$.

Modified Adaptive Partitioning. As in Section 4, our modified partitioning algorithm has two phases. Phase 1 runs in iterations. Consider any client j. As before, $\widetilde{N}(j)$ is the neighborhood of j with respect to the yet unpartitioned solution, namely the set of facilities μ such that $\widetilde{x}_{\mu j} > 0$. Order the facilities in this set as $\widetilde{N}(j) = \{\mu_1, ..., \mu_q\}$ in order of non-decreasing distance from j, that is $d_{\mu_1 j} \le d_{\mu_2 j} \le \ldots \le d_{\mu_q j}$, where $q = |\widetilde{N}(j)|$. Without loss of generality, there is an index l for which $\sum_{s=1}^{l} \widetilde{x}_{sj} = 1/\gamma$, since we can always split one facility to have this property. Then we define $\widetilde{N}_\gamma(j) = \{\mu_1, ..., \mu_l\}$. We also use notation $\text{tcc}_\gamma(j) = D(\widetilde{N}_\gamma(j), j) = \sum_{\mu \in \widetilde{N}_\gamma(j)} d_{\mu j} \widetilde{x}_{\mu j}$ and $\text{dmax}_\gamma(j) = \max_{\mu \in \widetilde{N}_\gamma(j)} d_{\mu j}$. In each iteration, we find a not yet exhausted client p that minimizes the value of $\text{tcc}_\gamma(p) + \text{dmax}_\gamma(p)$. Now we have two cases:
Case 1: $\widetilde{N}_\gamma(p) \cap \overline{N}_{\text{cls}}(\kappa) \neq \emptyset$, for some existing primary demand κ. In this case we assign ν to κ. As before, if there are multiple such κ, we pick any of them. We also fix $\bar{x}_{\mu\nu} \leftarrow \widetilde{x}_{\mu p}, \widetilde{x}_{\mu p} \leftarrow 0$ for each $\mu \in \widetilde{N}(p) \cap \overline{N}_{\text{cls}}(\kappa)$. As before, although we check for overlap between $\widetilde{N}_\gamma(p)$ and $\overline{N}_{\text{cls}}(\kappa)$, the facilities we actually move into $\overline{N}(\nu)$ include all facilities in the intersection of $\widetilde{N}(p)$, a bigger set, with $\overline{N}_{\text{cls}}(\kappa)$. We would like to point out that $\overline{N}(\nu)$ is not finalized at this time as we will add more facilities to it in the augment phase. As a result $\overline{N}_{\text{cls}}(\nu)$ is not fixed either, as we could potentially add facilities closer to ν than facilities already in $\overline{N}(\nu)$.
Case 2: $\widetilde{N}_\gamma(p) \cap \overline{N}_{\text{cls}}(\kappa) = \emptyset$, for all existing primary demands κ. In this case we make ν a primary demand. We then fix $\bar{x}_{\mu\nu} \leftarrow \widetilde{x}_{\mu p}$ for $\mu \in \widetilde{N}_\gamma(p)$ and set the corresponding $\widetilde{x}_{\mu p}$ to 0. Note that the total connection value in $\overline{N}_{\text{cls}}(\nu)$ is now exactly $1/\gamma$. The set $\widetilde{N}_\gamma(p)$ turns out to coincide with $\overline{N}_{\text{cls}}(\nu)$ as the facilities in $\widetilde{N}(p) \setminus \widetilde{N}_\gamma(p)$ are all farther away than any facilitity in $\widetilde{N}_\gamma(p)$. In the augmenting phase, Phase 2, we have available only facilities in some subset of $\widetilde{N}(p) \setminus \widetilde{N}_\gamma(p)$. Thus $\overline{N}_{\text{cls}}(\nu)$ is defined when ν is created.

Once all clients are exhausted, Phase 1 concludes. We then do Phase 2, the augmenting phase. For each demand ν of client j with total connection value less than 1, we use our AugmentToUnit() procedure to add additional facilities from $\widetilde{N}(j)$ to ν's neighborhood to make its total connection value equal 1, as before a facility is removed from $\widetilde{N}(j)$ once added to a demand's neighborhood. We do facility split if necessary to make $\overline{N}(\nu)$ have total connection value of 1.

We argue that the fractional solution $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ satisfies all the stated properties. Properties (PS), (CO), (NB), (PD'.1) and (SI'.1) are enforced by the algorithm. The proofs for other properties (PD'.2), (PD'.3(b)) and (SI'.2) are similar to those in Section 4, with the exception of (PD.3(a)), which we justify below.

The argument for (PD.3(a)) is a bit subtle, because of complications arising in Phase 2. For non-primary demands, $\overline{N}(\nu)$, for $\nu \in j$, takes all facilities in $\overline{N}_{\rm cls}(\kappa) \cap \widetilde{N}(j)$, which might be close to κ but far from j. As a result, facilities added in the augmenting phase might appear in $\overline{N}_{\rm cls}(\nu)$, yet they are not in $\overline{N}_{\rm cls}(\kappa)$. It is conceivable that in a worst case, the facilities added in the augmenting phase form $\overline{N}_{\rm cls}(\nu)$ exclusively, which will then be disjoint from $\overline{N}_{\rm cls}(\kappa)$, and we may not have (PD.3(a)) for demand ν. Nevertheless, we show that Property (PD.3(a)) holds.

Consider an iteration when we create a demand $\nu \in p$ and assign it to κ. Then the set $B(p) = \widetilde{N}_\gamma(p) \cap \overline{N}_{\rm cls}(\kappa)$ is not empty. We claim that $B(p)$ must be a subset of $\overline{N}_{\rm cls}(\nu)$ after $\overline{N}(\nu)$ is finalized with a total connection value of 1. To see this, first observe that $B(p)$ is a subset of $\overline{N}(\nu)$, which in turn is a subset of $\widetilde{N}(p)$, after taking into account the facility split. Here $\widetilde{N}(p)$ refers to the neighborhood of client p just before ν was created. For an arbitrary set of facilities A define $\text{dmax}(A, \nu)$ as the minimum distance τ such that $\sum_{\mu \in A \,:\, d_{\mu\nu} \le \tau} \bar{y}_\mu \ge 1/\gamma$. Adding additional facilities into A cannot make $\text{dmax}(A, \nu)$ larger, so it follows that $\text{dmax}(\overline{N}_{\rm cls}(\nu), \nu) \ge \text{dmax}(\widetilde{N}(p), \nu)$, because $\overline{N}_{\rm cls}(\nu)$ is a subset of $\widetilde{N}(p)$. Since we have $d_{\mu\nu} = d_{\mu p}$ by definition, it is easy to see that every $\mu \in B(p)$ satisfies $d_{\mu\nu} \le \text{dmax}(\widetilde{N}(p), \nu) \le \text{dmax}(\overline{N}_{\rm cls}(\nu), \nu)$ and hence they all belong to $\overline{N}_{\rm cls}(\nu)$. We need to be a bit more careful here when we have a tie in $d_{\mu\nu}$ but we can assume ties are always broken in favor of facilities in $B(p)$ when defining $\overline{N}_{\rm cls}(\nu)$. Finally, since $B(p) \neq \emptyset$, we have that the close neighborhood of a demand ν and its primary demand κ must overlap.

Algorithm EBGS. We first solve the linear program and compute the partitioning described earlier. Given the partitioned fractional solution $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ with the desired properties, we open exactly one facility in the close neighborhood of each primary demand, but now each facility μ is chosen with probability $\gamma \bar{y}_\mu$. The facilities μ that do not belong to any set $\overline{N}_{\rm cls}(\kappa)$ are opened independently with probability $\gamma \bar{y}_\mu$ each.

Next, we connect demands to facilities. Each primary demand κ will connect to the only facility $\phi(\kappa)$ open in its cluster $\overline{N}_{\rm cls}(\kappa)$. For each non-primary demand ν, if there is an open facility in $\overline{N}(\nu)$ then we connect ν to the nearest such facility. Otherwise, we connect ν to its *target facility* $\phi(\kappa)$, where κ is the primary demand that ν is assigned to.

Analysis. The feasibility of our integral solution follows from (SI.1), (SI.2), and (PD.1), as these properties ensure that siblings get connected to different facilities. It is possible to show that the expected facility cost of our algorithm is bounded by γF^* and the expected connection cost can be bounded by $C^* \max\{\frac{1/e + 1/e^\gamma}{1 - 1/\gamma}, 1 + \frac{2}{e^\gamma}\}$. Hence the total cost is bounded by $\max\{\gamma, \frac{1/e + 1/e^\gamma}{1 - 1/\gamma}, 1 + \frac{2}{e^\gamma}\} \cdot \text{LP}^*$. Picking $\gamma = 1.575$ we obtain:

Theorem 4. *Algorithm* EBGS *is a* 1.575*-approximation algorithm for* FTFP.

References

1. Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., Pandit, V.: Local search heuristics for k-median and facility location problems. SIAM J. Comput. 33(3), 544–562 (2004)
2. Byrka, J., Aardal, K.: An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. SIAM J. Comput. 39(6), 2212–2231 (2010)
3. Byrka, J., Ghodsi, M.R., Srinivasan, A.: LP-rounding algorithms for facility-location problems. CoRR abs/1007.3611(2010)
4. Byrka, J., Srinivasan, A., Swamy, C.: Fault-tolerant facility location: A randomized dependent LP-rounding algorithm. In: Eisenbrand, F., Shepherd, F.B. (eds.) IPCO 2010. LNCS, vol. 6080, pp. 244–257. Springer, Heidelberg (2010)
5. Chudak, F., Shmoys, D.: Improved approximation algorithms for the uncapacitated facility location problem. SIAM J. Comput. 33(1), 1–25 (2004)
6. Guha, S., Khuller, S.: Greedy strikes back: improved facility location algorithms. In: Proc. 9th SODA, pp. 649–657 (1998)
7. Guha, S., Meyerson, A., Munagala, K.: A constant factor approximation algorithm for the fault-tolerant facility location problem. J. Algorithms 48(2), 429–440 (2003)
8. Gupta, A.: Lecture notes: CMU 15-854b (Spring 2008)
9. Jain, K., Mahdian, M., Markakis, E., Saberi, A., Vazirani, V.: Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. J. ACM 50(6), 795–824 (2003)
10. Jain, K., Vazirani, V.: Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. J. ACM 48(2), 274–296 (2001)
11. Jain, K., Vazirani, V.: An approximation algorithm for the fault tolerant metric facility location problem. Algorithmica 38(3), 433–439 (2003)
12. Li, S.: A 1.488 approximation algorithm for the uncapacitated facility location problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 77–88. Springer, Heidelberg (2011)
13. Liao, K., Shen, H.: Unconstrained and constrained fault-tolerant resource allocation. In: Fu, B., Du, D.-Z. (eds.) COCOON 2011. LNCS, vol. 6842, pp. 555–566. Springer, Heidelberg (2011)
14. Shmoys, D., Tardos, E., Aardal, K.: Approximation algorithms for facility location problems (extended abstract). In: Proc. 29th STOC, pp. 265–274 (1997)
15. Sviridenko, M.I.: An improved approximation algorithm for the metric uncapacitated facility location problem. In: Cook, W.J., Schulz, A.S. (eds.) IPCO 2002. LNCS, vol. 2337, pp. 240–257. Springer, Heidelberg (2002)
16. Swamy, C., Shmoys, D.: Fault-tolerant facility location. ACM Trans. Algorithms 4(4), 1–27 (2008)
17. Vygen, J.: Approximation Algorithms for Facility Location Problems. Forschungsinst. für Diskrete Mathematik (2005)
18. Xu, S., Shen, H.: The fault-tolerant facility allocation problem. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 689–698. Springer, Heidelberg (2009)
19. Yan, L., Chrobak, M.: Approximation algorithms for the fault-tolerant facility placement problem. Inf. Process. Lett. 111(11), 545–549 (2011)
20. Yan, L., Chrobak, M.: LP-rounding Algorithms for the Fault-Tolerant Facility Placement Problem. CoRR abs/1205.1281 (2012)

Author Index