Joachim Gudmundsson
Julián Mestre
Taso Viglas (Eds.)

LNCS 7434

Computing and Combinatorics

18th Annual International Conference, COCOON 2012
Sydney, Australia, August 2012
Proceedings

Springer

Lecture Notes in Computer Science 7434

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Joachim Gudmundsson Julián Mestre
Taso Viglas (Eds.)

Computing and Combinatorics

18th Annual International Conference
COCOON 2012
Sydney, Australia, August 20-22, 2012
Proceedings

Springer

Volume Editors

Joachim Gudmundsson
Julián Mestre
Taso Viglas

University of Sydney
School of IT, Building J12
Sydney, NSW 2006, Australia

E-mail: joachim.gudmundsson@sydney.edu.au
E-mail: julian.mestre@sydney.edu.au
E-mail: taso.viglas@sydney.edu.au

ISSN 0302-9743
ISBN 978-3-642-32240-2
DOI 10.1007/978-3-642-32241-9

e-ISSN 1611-3349
e-ISBN 978-3-642-32241-9

Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012942913

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

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

The papers in this volume were selected for publication in the 18th Annual International Computing and Combinatorics Conference, held during August 20–22, 2012, in Sydney, Australia. The Annual International Computing and Combinatorics Conference is a forum for researchers working in the areas related to theoretical aspects of computing.

Typical topics covered by this conference include (but are not restricted to): algorithms, data structures, algorithmic game theory, online algorithms, automatas, languages, logic, computability, complexity theory, computational learning theory, knowledge discovery, cryptography, parallel and distributed computing, reliability and security, database theory, computational biology, computational algebra, computational geometry, graph drawing, information visualization, graph theory, communication networks and optimization.

This year the conference received 121 submissions. Each paper received a minimum of three independent expert reviews by the Program Committee members or reviewers. The reviews were thoroughly discussed by the Program Committee, and 50 papers were selected for presentation during the conference.

In addition to these presentations, the program also included three invited presentations by Kamal Jain, Joseph Mitchell, and János Pach.

We would like to express our gratitude for the authors of all papers submitted to COCOON 2012, the Program Committee members and the reviewers, for their contribution to making this conference possible.

Finally, we would like to acknowledge and thank the sponsors of this event, Google and National ICT Australia, for their generous support for COCOON 2012.

June 2012

Joachim Gudmundsson
Julián Mestre
Taso Viglas

Organization

Program Committee

Eric Allender	Rutgers University, USA
Giorgos Christodoulou	University of Liverpool, UK
Giovanni Di Crescenzo	Telcordia, USA
David Eppstein	University of California, Irvine, USA
Rudolf Fleischer	GUtech, Oman
Mordecai Golin	Hong Kong UST, Hong Kong, SAR China
Joachim Gudmundsson	University of Sydney, Australia
Anupam Gupta	Carnegie Mellon University, USA
Thore Husfeldt	IT University of Copenhagen and Lund University, Denmark and Sweden
Kazuo Iwama	Kyoto University, Japan
Julián Mestre	University of Sydney, Australia
Peter Bro Miltersen	Aarhus University, Denmark
Pat Morin	Carleton University, Canada
Gonzalo Navarro	University of Chile, Chile
Kunihiko Sadakane	National Institute of Informatics, Tokyo, Japan
Saket Saurabh	The Institute of Mathematical Sciences, Chennai, India
Christian Sohler	TU Dortmund, Germany
Xiaoming Sun	Tsinghua University, China
Kavitha Telikepalli	Indian Institute of Science, India
Anke Van Zuylen	Max Planck Institute for Informatics, Germany
Anastasios Viglas	University of Sydney, Australia
Dorothea Wagner	University of Karlsruhe, Germany
Gerhard Woeginger	TU Eindhoven, The Netherlands
Prudence Wong	University of Liverpool, UK

Additional Reviewers

Aggarwal, Divesh
Asahiro, Yuichi
Badanidiyuru, Ashwinkumar
Barbay, Jérémy
Baum, Moritz
Bläsius, Thomas
Bogdanov, Andrej
Bollig, Beate
Bonsma, Paul
Bose, Prosenjit
Bu, Dongbo
Burcea, Mihai
Campos, Sérgio
Canzar, Stefan
Cheng, Yongxi
Chiu, Man Kwun
Chrobak, Marek
Cicalese, Ferdinando

Collins, Andrew
Cormode, Graham
Crowston, Robert
Data, Deepesh
Dibbelt, Julian
Dujmovic, Vida
Dutta, Kunal
Ehsanfar, Ebrahim
Ferreira, Rui
Fortunato, Santo
Fotakis, Dimitris
Frati, Fabrizio
Gao, Xi Alice
Gaspers, Serge
Gemsa, Andreas
Giannakopoulos, Yiannis
Gille, Marc
Gorry, Thomas
Halldorsson, Magnus M.
Hellweg, Frank
Hernandez, Cecilia
Jager, Tibor
Jansson, Jesper
Jin, Jiongxin
Jones, Mark
Jordan, Tibor
Kane, Daniel M.
Kawarabayashi, Ken-Ichi
Keszegh, Balázs
Kida, Takuya
Kiyomi, Masashi
Klauck, Hartmut
Kloks, Ton
Kolay, Sudeshna
Kovacs, Annamaria
Kratsch, Stefan
Krivosija, Amer
Krug, Marcus
Kupferman, Orna
Lam, Chi Kit
Lambert, Nicolas S.
Lammersen, Christiane
Lan, Yu
Leung, Henry C.M.
Levin, Asaf
Li, Rongbin
Liedloff, Mathieu
Lin, Chuang-Chieh
M.S., Ramanujan
Ma, Bin
Madry, Aleksander
Maheshwari, Anil
Markakis, Evangelos
Martin, Russell
Mchedlidze, Tamara
Mertzios, George
Mikalački, Mirjana
Misra, Pranabendu
Miyata, Hiroyuki
Mnich, Matthias
Munteanu, Alexander
Mustafa, Nabil
Muthu, Rahul
Narayanan, Narayanan
Nasre, Meghana
Nekrich, Yakov
Niedermeier, Rolf
Nikiforov, Vladimir
Noellenburg, Martin
O'Donnell, Ryan
Ojiaku, Jude-Thaddeus
Oren, Sigal
Otachi, Yota
Pajor, Thomas
Papadopoulos, Charis
Papakonstantinou, Periklis
Pavan, Aduri
Piliouras, Georgios
Popa, Alexandru
Pérez-Lantero, Pablo
Rajagopalan, S. Raj
Raman, Venkatesh
Randall, Dana
Russo, Luis M.S.
Rutter, Ignaz
Sabharwal, Yogish
Sach, Benjamin
Saeidinvar, Reza
Saitoh, Toshiki
Schalekamp, Frans

Schmidt, Melanie
Schumm, Andrea
Schwiegelshohn, Chris
Shalom, Mordechai
Shi, Yaoyun
Skopalik, Alexander
Smid, Michiel
Sprugnoli, Renzo
Stamoulis, Georgios
Stewart, Iain
Suchy, Ondrej
Suzuki, Yasuhiro
Tamaki, Suguru
Tang, Bo
Tantau, Till
Tazari, Siamak
Telelis, Orestis
Trapnell, Cole
van Leeuwen, Erik Jan
van Stee, Rob
Varma, Nithin Mahendra
Ventre, Carmine
Vollmer, Heribert
Voloshin, Ariella
Wahlström, Magnus
Wan, Andrew
Wang, Yajun
Wiese, Andreas
Wolfler-Calvo, Roberto
Wuhrer, Stefanie
Wulff-Nilsen, Christian
Xia, Lirong
Xiao, Mingyu
Yamazaki, Koichi
Yon, Juyoung
Yu, Wei
Zhang, Shengyu

Table of Contents

A Linear Time Algorithm for Computing Minmax Regret 1-Median on a Tree

Binay Bhattacharya* and Tsunehiko Kameda**

School of Computing Science, Simon Fraser University, Canada
{binay,tiko}@sfu.ca

Abstract. In a model of facility location problem, the uncertainty in the weight of a vertex is represented by an interval of weights, and minimizing the maximum "regret" is the goal. The most efficient previously known algorithm for finding the minmax regret 1-median on trees with positive vertex weights takes $O(n \log n)$ time. We improve it to $O(n)$, solving the open problem posed by Brodal et al. in [3].

1 Introduction

Deciding where to locate facilities to minimize the communication or transportation costs is known as the *facility location problem*. For a recent review of this subject, the reader is referred to [9]. The cost function is formulated as the sum of the distances from the nearest facility weighted by the weights of the vertices. In the *minmax regret* version of this problem, there is uncertainty in the weights of the vertices and/or edge lengths, and only their ranges are known [6,11]. Chen and Lin (Theorem 1 in [6]) proved that in solving this problem, the edge lengths can be set to their maximum values. Therefore, we assume that the (positive) edge lengths are fixed and uncertainty is only in the vertex weights. A particular *realization* (assignment of a weight to each vertex) is called a *scenario*. Intuitively, the minmax regret 1-median problem can be understood as a 2-person game as follows. The first player picks a location x to place a facility. The opponent's move is to pick a scenario s. The payoff to the second player is the cost of x minus the cost of the median, both under s, and he wants to pick the scenario s that maximizes his payoff. Our objective (as the first player) is to select x that minimizes this payoff in the worst case (i.e., over all scenarios).

The problem of finding the minmax regret median on a graph, and a tree in particular, has been attracting great research interest in recent years, and many researchers have worked on this problem. Kouvelis et al. [11] formulated the problem of finding the minmax regret 1-median on a tree and proposed an $O(n^4)$ solution, where n is the number of vertices. Chen and Lin [6] improved it to $O(n^3)$. Averbakh and Berman then found a simple $O(n^2)$ algorithm [1] and improved it later to $O(n \log^2 n)$ [2]. Yu et al. [13] proposed an $O(n \log n)$

* Supported in part by the NSERC of Canada.
** Supported in part by the NSERC of Canada.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 1–12, 2012.

implementation of the algorithm in [2]. More recently Brodal et al. also came up with a simpler $O(n \log n)$ algorithm [3]. When the vertices can have negative weights, Burkard et al. have an $O(n^2)$ algorithm [4]. In this paper, we present an $O(n)$ time algorithm for trees when the edge lengths are fixed, and each vertex has a weight from an interval of positive values. This settles the open problem posed in [3]. We achieve this by eliminating, in each round, a fraction of scenarios that become irrelevant ("dominated") in the future, as well as reducing the size of the subtree in which the optimal location lies. Previous authors only reduced the subtree in which the optimal location lies, paying no special attention to the scenarios that are "dominated". Our pruning algorithm was inspired by a similar method used by Megiddo [12].

The rest of this paper is organized as follows. In the next section, we first review definitions and some known facts, and then prove a lemma which restricts the scenarios we need to consider. Section 3 describes methods for computing the medians for the scenarios of interest, and for computing their costs. Section 4 is devoted to the detailed discussion of our main algorithm. Finally, Section 5 concludes the paper.

2 Preliminaries

2.1 Definitions

Let $T = (V, E)$ be a tree with n vertices. We also use T to denote the set of all points (vertices and points on edges) on T. Each vertex $v \in V$ is associated with an interval of positive[1] weights $W(v) = [\underline{w}_v, \overline{w}_v]$, where $0 < \underline{w}_v \leq \overline{w}_v$, and each edge $e \in E$ is associated with a positive length (or distance). For any two points $x, y \in T$, $d(x, y)$ denotes the shortest distance between x and y on T. If x and/or y is on an edge, then the distance is a prorated fraction of its length. Let $\mathcal{S}$ denote the Cartesian product of all $W(v)$, $v \in V$:

$$\mathcal{S} = \prod_{v \in V} [\underline{w}_v, \overline{w}_v].$$

Under a scenario $s \in \mathcal{S}$, we define the *cost* of a point $x \in T$ by

$$F^s(x) = \sum_{v \in V} d(v, x) w_v^s. \tag{1}$$

A location x that minimizes (1) is a *1-median* under s. Throughout this paper we use 1-median and *median* synonymously. We call

$$R^s(x) = F^s(x) - F^s(m(s)) \tag{2}$$

the *regret* of x under s, where $m(s)$ denotes a median under s. A scenario s is said to *dominate* another scenario s' at x if $R^s(x) \geq R^{s'}(x)$ holds. We finally

[1] We need this assumption in the proof of Lemma 11.

define the *maximum regret* $R_{max}(x)$ of x as the regret of the dominating scenario at x.

$$R_{max}(x) = \max_{s \in \mathcal{S}} R^s(x). \tag{3}$$

Note that $R_{max}(x)$ is the maximum payoff with respect to x that we mentioned in the Introduction. We seek location $x^* \in T$, called the *minmax regret median*, that minimizes $R_{max}(x)$. We sometimes refer to x^* as the *optimal location*. Let $s = \hat{s}(x)$ maximize (2) for a given $x \in T$. We call $\hat{s}(x)$ and $m(\hat{s}(x))$ a *worst case scenario* and a *worst case alternative* for x, respectively.

2.2 Properties of Median and Minmax Regret Median in a Tree

Let $v \in V$ be a vertex with degree d connected to vertices $u_1, u_2, \ldots, u_d$. If we remove v and all edges incident to it from T, then d subtrees, $T(u_1), T(u_2), \ldots, T(u_d)$ result. Let $W(T(u_i))$ denote the total weight of the vertices in $T(u_i)$. Vertex v is said to be a *weight centroid* or *w-centroid* [10] if

$$W(T(u_i)) \leq W(T)/2, \tag{4}$$

holds for all $i = 1, 2, \ldots, d$, where $W(T)$ denotes the total weight of the vertices in T.

Lemma 1. [8]

(a) If the total vertex weights on both sides of an edge are the same, then any point on that edge, including the end vertices, is a median.
(b) If there is no such edge, there is a unique median that is a vertex. □

By Lemma 1, we shall assume that a median is always at a vertex. There are at most two such vertices under a given scenario, and they are w-centroids. However, the minmax regret median may not be at a vertex [11]. If there exists a vertex that is a median under all the scenarios, then clearly it is the minmax regret location. In such a case, the problem instance is said to be *degenerate*.

Theorem 1. *In a general graph, we have $R_{max}(x^*) = 0$ if and only if the problem instance is degenerate.*

Proof. The if part is obvious. SInce $R_{max}(x^*) = 0$ implies that $R^s(x^*) = F^s(x^*) - F^s(m(s)) = 0$ holds for any scenario $s \in \mathcal{S}$, x^* is a median under s. □

Lemma 2. ([6], Theorem 1(a)) *Given any point $x \in T$, there exists a worst-case scenario $\hat{s}(x)$ such that $w_v^{\hat{s}(x)} = \underline{w}_v$ if $d(x, v) < d(m(\hat{s}(x)), v)$ and $w_v^{\hat{s}(x)} = \overline{w}_v$ otherwise.* □

Let $e = (u, u') \in E$, and let $T(u)$ (resp. $T(u')$) denote the maximal subtree of T that does not contain e but contains u (resp. u'). Let $s \in \mathcal{S}$ be such that $w_v^s = \overline{w}_v$ (v is *max-weighted*) for each vertex $v \in T(u)$, and $w_v^s = \underline{w}_v$ (v is *min-weighted*) for each $v \in T(u')$. Such a scenario s is called a *bipartite* scenario, and u is the *front* of s, denoted by $f(s)$. Let $\mathcal{S}^* \subset \mathcal{S}$ be the set of all bipartite

scenarios. Under a bipartite scenario, we call the component consisting of the max-weighted vertices the *max-weighted component* and the other component the *min-weighted component*. Note that scenario $\hat{s}(x)$ in Lemma 2 is bipartite. It is known that [1]

$$\forall x \in T : R_{max}(x) = \max_{s \in \mathcal{S}^*} R^s(x). \tag{5}$$

Let $\tilde{\mathcal{S}} \subset \mathcal{S}^*$ be the set of scenarios under which a median is in the max-weighted component. If we remove a point $a \in T$ from T, a number of connected components result. An *a-branch* [2] of T is any such component with point a restored to it.

Lemma 3.

$$\forall x \in T : R_{max}(x) = \max_{s \in \tilde{\mathcal{S}}} R^s(x). \tag{6}$$

Proof. Let $R_{max}(x) = \max_{s \in \mathcal{S}^*} R^s(x) = R^{\hat{s}(x)}(x)$. Note that $\hat{s}(x)$ is a function of x. For $x \neq m(\hat{s}(x))$, this lemma follows directly from Lemma 2. So assume that $x = m(\hat{s}(x))$, hence $R_{max}(x) = 0$. Then this x is the optimal x^*, and we have $R_{max}(x^*) = 0$, which implies (Theorem 1) that $x = m(\hat{s}(x))$ is a 1-median for every scenario in $\mathcal{S}^*$. Clearly, there is a scenario in $\tilde{\mathcal{S}}$ whose max-weighted component contains $x = m(\hat{s}(x))$. □

Lemma 4. *All scenarios in $\tilde{\mathcal{S}}$ have a common median, if and only if the problem instance is degenerate.*

Proof. The if part follows from definition. Assume that all scenarios in $\tilde{\mathcal{S}}$ have a common median m, but there is a scenario $s' \in \mathcal{S} \setminus \tilde{\mathcal{S}}$ under which m is not a median, so that $R^{s'}(m) > 0$ holds. We modify s' according to Lemma 2 (with $x = m$), obtaining a scenario $s'' \in \tilde{\mathcal{S}}$ that satisfies $R^{s''}(m) \geq R^{s'}(m)$. This is a contradiction, since $R^{s''}(m) = 0$. □

3 Medians and Their Costs

3.1 Computing Medians $m(s)$ for Every $s \in \tilde{\mathcal{S}}$

By Lemma 3, we only consider the scenarios in $\tilde{\mathcal{S}}$ in the rest of this paper. Here we compute the medians of all scenarios in $\tilde{\mathcal{S}}$ in linear time. We pick an arbitrary vertex as the root r, and consider the rooted tree T. Let $T(v)$ denote the subtree of T at v, and let us call $T^c(v) = T \setminus T(v)$ the *complement* of $T(v)$. For convenience, we define two arrays for *subtree weights*, $\overline{W}_t[\cdot]$ and $\underline{W}_t[\cdot]$, and two arrays for *complement weights*, $\overline{W}_c[\cdot]$ and $\underline{W}_c[\cdot]$, as follows:[2]

$$\overline{W}_t[v] = \sum_{u \in T(v)} \overline{w}_u, \quad \underline{W}_t[v] = \sum_{u \in T(v)} \underline{w}_u,$$

$$\overline{W}_c[v] = \sum_{u \in T^c(v)} \overline{w}_u, \quad \underline{W}_c[v] = \sum_{u \in T^c(v)} \underline{w}_u.$$

[2] "Subtree" and "complement" are relative to the root r of tree T chosen, but the totality of information these arrays represent are independent of the choice of the root, because they contain data on both $T(v)$ and $T \setminus T(v)$.

We can compute $\overline{W}_t[\cdot]$ and $\underline{W}_t[\cdot]$ in $O(n)$ time [5]. Once they have been computed, to compute $\overline{W}_c[v]$, for example, we simply use the relation $\overline{W}_c[v] = \overline{W}_t[r] - \overline{W}_t[v]$. For two points $a, b \in T$, let $\pi(a, b)$ denote the shortest path between a and b.

Lemma 5. *Let $T(u)$ be the max-weighted component under $s \in \tilde{\mathcal{S}}$ so that $u = f(s)$. Let $m(s)$ be the median vertex that is farthest from u. Then, for each vertex v on $\pi(m(s), u)$, $\overline{W}_t[v]$ is at least half of the total vertex weight under s.*

Proof. Clearly, it suffices to prove the assertion for the case $v = m(s)$. Since $m(s)$ is a median under s, by (4) we have

$$\overline{W}_t[u] - \overline{W}_t[m(s)] + \underline{W}_c[u] \leq (\overline{W}_t[u] + \underline{W}_c[u])/2,$$

where the left hand side is the weight under s of $T^c(v)$. We thus obtain

$$\overline{W}_t[m(s)] \geq (\overline{W}_t[u] + \underline{W}_c[u])/2.$$

□

Theorem 2. *We can compute $m(s)$ for every $s \in \tilde{\mathcal{S}}$ in $O(n)$ time.*[3]

Proof. We perform a post-order depth first traversal, checking the condition (Eq. (4)) of Lemma 1 on each vertex visited. During the traversal, we compute $m(s)$ for scenario s whose max-weighted component is $T(u)$, where u is the vertex being visited ($u = f(s)$).

If $\underline{W}_c[u] > \overline{W}_t[u]$ then $m(s) \notin T(u)$ by Lemma 1, and $s \notin \tilde{\mathcal{S}}$. So assume $\underline{W}_c[u] \leq \overline{W}_t[u]$, hence $m(s) \in T(u)$. Let $u_1, u_2, \ldots, u_k$ be the child vertices of u, and for $j = 1, 2, \ldots, k$ let s_j be the scenario such that $T(u_j)$ is the max-weighted component of s_j, i.e., $f(s_j) = u_j$. Note that in general $m(s_j)$ may not be in $T(u_j)$. If $m(s_j) \notin T(u_j)$ for all j, then we have $m(s) = u$. This is because by increasing vertex weights in other subtrees $T(u_i)$ $(i \neq j)$, $m(s_j)$ cannot move to $T(u_j)$. Let us now assume that there is j with $m(s_j) \in T(u_j)$, which implies $\underline{W}_c[u_j] \leq \overline{W}_t[u_j]$ and $s_j \in \tilde{\mathcal{S}}$. Lemma 5 implies that $m(s)$ lies in a subtree with the largest $\overline{W}_t[u_j]$. It also implies that $m(s)$ is on $\pi(m(s_j), u_j)$ or at u. To identify one or two vertices $m(s)$, starting from $m(s_j)$, we test each vertex on this path until the condition of Lemma 1 is satisfied. Note that each vertex is visited at most twice, once in the post-order traversal, and the second time if it is on $\pi(m(s_j), u)$ for a certain j. When u is visited, its k children need to be examined. Thus each vertex is examined also as a child vertex when its parent is being visited. But this happens at most twice, and the total time required is $O(n)$.

We also need to compute $m(s)$ for scenario s whose min-weighted component is $T(u)$, where u is the vertex being visited. In this case, we have $f(s) = p(u)$, where $p(u)$ is the parent of u. We now consider the subtrees rooted at the child vertices of $p(u)$, except $T(u)$. The only difference from what we discussed above is that we now need to consider $T^c(p(u))$ also as a subtree under $p(u)$. But with a minor change, we can use the method discussed above to compute the remaining medians. □

[3] As commented after Lemma 1, there may be two median vertices under a scenario.

3.2 Computing $F^s(m(s))$ for Every $s \in \tilde{\mathcal{S}}$

We first define the *subtree costs* (with subscript t) and *complement costs* (with subscript c) relative to the root r as follows:

$$\overline{C}_t[v] = \sum_{u \in T(v)} d(u,v)\overline{w}_u, \ \underline{C}_t[v] = \sum_{u \in T(v)} d(u,v)\underline{w}_u,$$
$$\overline{C}_c[v] = \sum_{u \in T^c(v)} d(u,v)\overline{w}_u, \ \underline{C}_c[v] = \sum_{u \in T^c(v)} d(u,v)\underline{w}_u.$$

Arrays $\overline{C}_t[\cdot]$, $\underline{C}_t[\cdot]$, $\overline{C}_c[\cdot]$, and $\underline{C}_c[\cdot]$ can be computed in $O(n)$ time [4].

Theorem 3. *We can compute* $\{F^s(m(s)) \mid s \in \tilde{\mathcal{S}}\}$ *in* $O(n)$ *time.*

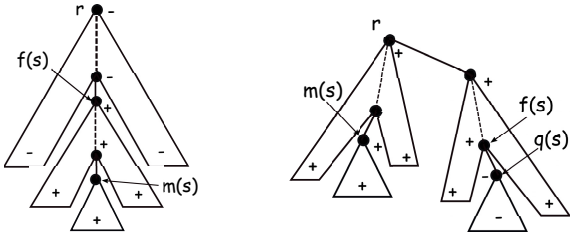


Fig. 1. Vertex v with weight $\overline{w}_v$ (resp. $\underline{w}_v$) is indicated by a + (resp. −): (a) $m(s)$ and $f(s)$ are in the same subtree; (b) $m(s)$ and $f(s)$ are in different subtrees

Proof. We can compute $m(s)$ for all $s \in \tilde{\mathcal{S}}$ in $O(n)$ time by Theorem 2. We now compute costs $F^s(m(s))$ for all $s \in \tilde{\mathcal{S}}$ in two possible cases. Assume first that $m(s)$ and $f(s)$ belong to the same subtree directly under the root r.

Case (a) $[m(s) = f(s)]$: It is easy to see that

$$F^s(m(s)) = \overline{C}_t[m(s)] + \underline{C}_c[m(s)]. \tag{7}$$

Case (b) $[m(s) \neq f(s)]$: Fig. 1(a) illustrates scenario s. Let us consider "contributions" to $F^s(m(s))$ from different parts of tree T.

1. From $T(m(s))$: $\overline{C}_t[m(s)]$.
2. From $T(f(s)) \backslash T(m(s))$: $\overline{C}_c[m(s)] - \{\overline{C}_c[f(s)] + d(f(s), m(s))\overline{W}_c[f(s)]\}$.
3. From $T^c(f(s))$: $\underline{C}_c[f(s)] + d(f(s), m(s))\underline{W}_c[f(s)]$.

It is clear that, using arrays $\overline{C}_*[\cdot]$, $\underline{C}_*[\cdot]$, $\overline{W}_c[\cdot]$, and $\underline{W}_c[\cdot]$, where $* \in \{t, c\}$, we can compute the above three terms in constant time. If $m(s)$ and $f(s)$ don't belong to the same subtree directly under the root r, as in Fig. 1(b), we can similarly compute $F^s(m(s))$ in constant time. □

It follows from Theorem 2 and Lemma 4 that

Lemma 6. *We can decide in* $O(n)$ *time if the given problem instance is degenerate.* □

4 Optimal Facility Location

4.1 Preparation

Lemma 7. ([2], Lemma 1) *For any point $x \in T$, the optimal location is in the x-branch in which the worst case alternative $m(\hat{s}(x))$ for x lies.* □

For the time being, we assume that a *pivot* vertex r that is not a leaf is given. We address the issue of how to choose an appropriate r in Subsec. 4.3. Let T^* be the r-branch that contains the optimal location x^*, and let S_r be the set of scenarios those max-weighted components are totally contained in $T \setminus T^* \cup \{r\}$.

Lemma 8. [13] *Let $s \in S_r$. Then $F^s(x)$ for $x \in T^*$ has the following properties:*

(a) It is a non-decreasing linear function of x on each edge in T^*, as x moves away from r.
(b) It is continuous at vertices in T^*. □

Lemma 9. *Let $s, s' \in S_r$, where $s \neq s'$. Then $R^s(x) = R^{s'}(x)$ has at most one solution for x within T^*.*

Proof. The "derivatives" of $F^s(x)$ and $F^{s'}(x)$ (hence of $R^s(x)$ and $R^{s'}(x)$) with respect to x (the positive direction of x is away from r) on edge $e = (u, v)$ are $W^s(u) - (W^s - W^s(u)) = 2W^s(u) - W^s$ and $2W^{s'}(u) - W^{s'}$, respectively, where $W^s(u)$ is the total weight of the subtree[4] of T rooted at u that does not contain v, and W^s is the total weight of the tree under s. Their difference is thus

$$2\{W^{s'}(u) - W^s(u)\} + \{W^s - W^{s'}\}. \tag{8}$$

The term $\{W^s - W^{s'}\}$ clearly does not depend on edge e on which x lies. Moreover, $\{W^{s'}(u) - W^s(u)\}$ is also independent of edge e, hence u. Therefore, $R^s(x)$ and $R^{s'}(x)$ intersect at most once, since x is bounded within T^*. This claim is also valid in the case where the difference in (8) is 0, in which case $R^s(x)$ and $R^{s'}(x)$ do not intersect. □

Lemma 10. *Assume that $\overline{W}_*[\cdot]$, $\underline{W}_*[\cdot]$, $\overline{C}_*[\cdot]$, and $\underline{C}_*[\cdot]$ are known, where $* \in \{t, c\}$. Given x and a set $S \subset \tilde{S}$ of scenarios that do not contain x in their max-weighted components, we can determine in $O(|S|)$ time the scenario $s^* \in S$ that dominates all others in S at x, and the x-branch that contains $m(s^*)$.*

Proof. Similar to the analysis we used in the proof of Theorem 3, case (b). □

4.2 Pruning Steps

We first test the degeneracy of the given instance in $O(n)$ time (Lemma 6). If it degenerate, then we have the optimal location x^* right away. So, we assume it is not degenerate. We start with the set $S = \tilde{\mathcal{S}}$, and remove irrelevant and

[4] It is not a subtree of T rooted at r.

dominated scenarios from S through pruning rounds. We also keep track of the subtree T^* that we know contains x^*. Here is an outline of a round of our pruning algorithm.

Procedure `Prune`

1. Pick a "pivot" vertex r in the subtree that contains the optimal location x^*.
2. Determine the r-branch, T^*, that contains x^*.
3. From S remove each scenario whose max-weighted component contains T^*.
4. Let S_r be the set of scenarios whose max-weighted components are totally contained in $T \backslash T^* \cup \{r\}$. If $|S_r| \geq 2$, pair up scenarios in S_r, and let p be the number of pairs. (If $|S_r|$ is odd, one scenario is left unpaired.)
5. Determine and throw away $\lceil p/4 \rceil$ dominated scenarios from S. □

If we repeatedly execute procedure `Prune`, we will end up with T^* that consists of just one edge. Then we can easily find the optimal location on the edge in linear time. Since the choice of pivot r in step 1 depends on how it is used in subsequent steps, let us assume for the time being that r has been chosen somehow, and steps 1 to 4 have been carried out. It follows from Theorem 1 that $x^* \neq m(\hat{s}(x*))$ holds in the non-degenerate case, and x^* is in the min-weighted component under $\hat{s}(x^*)$. This justifies step 3.

Step 5 computes the intersections of the regret functions $R^s(x)$ and $R^{s'}(x)$ of each pair (s, s') of scenarios constructed in step 4. By Lemma 9, they intersect at most once in T^*. Let $x_1, x_2, \ldots, x_p$ be the p intersections, and $d_1, d_2, \ldots, d_p$ be the corresponding distances from r. Let d_m be the $\lceil p/2 \rceil^{th}$ smallest among them. If all points at distance d_m from r are outside T^*, in each pair of scenarios with intersection point x_i, where $1 \leq i \leq m$, one is dominated by the other everywhere in T^* and can be discarded. If $R^s(x)$ and $R^{s'}(x)$ don't intersect, then again one of s and s' is dominated by the other in T^* and can be discarded. So, assume that there are points in T^* that are at distance d_m from r. Let d^* denote the distance of the optimal location x^*. If $d^* \geq d_m$ (resp. $d^* < d_m$), then from each scenario pair with intersection point x_i such that $1 \leq i \leq m$ (resp. $m < i \leq p$), one of them can be discarded, because the other dominates it for $x \geq d_m$ (resp. $x < d_m$).

We now show how to determine if $d^* \geq d_m$ for a given value $d_m (> 0)$. Let $y_1, y_2, \ldots, y_l$ be the points in T^* at distance d_m from r. For example, see $y_1, \ldots, y_5$ in Fig. 2(a). Let $\overline{T}(y_i)$ be the y_i-branch that is "below" (farther away from r) y_i. Let S_i denote the set of scenarios whose max-weighted components are totally contained in $\overline{T}(y_i)$. Define

$$\overline{R}(y_i) = \max_{s \in S_i} \{F^s(y_i) - F^s(m(s))\}, \tag{9}$$

and let s_i realize $\overline{R}(y_i)$. By Lemma 10, we can determine s_i in $O(|S_i|)$ time. Among them let $\overline{R}(y_k)$ (realized by $s_k \in S_k$) be as large as any other. We now compute $R_{max}(y_k)$ and $\hat{s}(y_k)$ based on the scenarios in S ($\subset \tilde{\mathcal{S}}$), which is the set of all scenarios that have not been thrown away so far. This can be done in $O(|S|)$) time by Lemma 10.

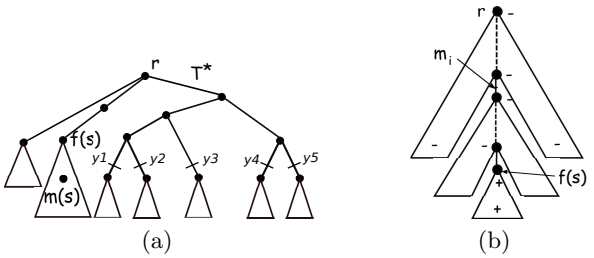


Fig. 2. (a) The edge lengths are not to scale; (b) Computing $F^s(r)$

Lemma 11. *For a given value d_m of x, define $\{y_i \mid i = 1, 2, \ldots, l\}$ and y_k as above.*

(a) If $\hat{s}(y_k) \in S_k$ then the optimal location x^* is in $\overline{T}(y_k)$.
(b) If $\hat{s}(y_k) \notin S_k$ then the optimal location x^* cannot be in $\overline{T}(y_i)$ for any i.

Proof. (a) Follows from Lemma 7, since $\hat{s}(y_k) \in \tilde{S}$, hence $m(\hat{s}(y_k)) \in \overline{T}(y_k)$.

(b) The assertion is trivially true for $i = k$. Assume that the optimal location (over S) was in $\overline{T}(y_j)$ $(j \neq k)$. Then by Lemmas 3 and 7, the scenario that realizes $R_{max}(y_j)$ must be in S_j, and hence $R_{max}(y_j) = \overline{R}(y_j)$. By the definition of k, we have

$$R_{max}(y_j) = \overline{R}(y_j) \leq \overline{R}(y_k). \tag{10}$$

On the other hand, we have

$$\overline{R}(y_k) < R^{s_k}(y_j), \tag{11}$$

by Lemma 8. We have strict inequity here, because y_j is farther away from median $m(s_k)$ than y_k, and there is at least one vertex (with positive weight) between y_k and y_j. By definition, we also have

$$R^{s_k}(y_j) \leq R_{max}(y_j). \tag{12}$$

Eqs. (10), (11) and (12) yield $R_{max}(y_j) < R_{max}(y_j)$, a contradiction. □

As we narrow down on the location x^*, using Lemma 11, we can update the part of tree T containing x^*, i.e., T^*. However, the fronts of the scenarios that we haven't discarded may belong to $T \backslash T^*$. To keep track of those scenarios, we maintain a tree $T' = (N, E')$, called the *auxiliary tree* that contains T^* in it. See Fig. 3, where the nodes[5] in the interior of T^* are indicated by black circles. Note that a vertex of T can be the front vertices of at most two scenarios in S $(\subseteq \tilde{S})$ at any time. If it is the front vertex of two (resp. one) scenarios, we give the corresponding node in T' weight of 2 (resp. 1). We modify T' in each round as follows:

[5] We use the term 'node' to avoid confusion with a vertex of T.

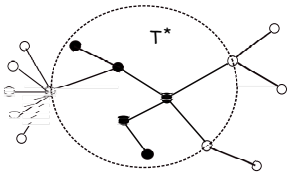


Fig. 3. Auxiliary tree T'

1. In case (a) of Lemma 11, T^* will be inside $\overline{T}(y_k)$. We "shorten" the edge that y_k lies on, removing the part of the edge between point y_k and the vertex (call it v_k) that is closest to y_k and not in $\overline{T}(y_k)$, and connect the shortened edge to v_k by placing y_k at v_k. The "shortening" is just for convenience in representing T', and the "shortened" edge keeps the original length. Further, we attach every node u outside of $\overline{T}(y_k)$ directly to v_k via an edge with length $d(u, v_k)$. In Fig. 3, node v_k is represented by a hollow circle on the boundary of T^*, indicating that it has a weight of 1.
2. In case (b) of Lemma 11, T^* is outside $\overline{T}(y_i)$ for $i = 1, 2, \ldots, l$. For each i, we "shorten" the edge that y_i lies on, removing the part of the edge between y_i and the vertex (call it v_i) that is closest to y_i and in $\overline{T}(y_i)$. The "shortened" edge keeps the original length. We attach every node u in $\overline{T}(y_i)$ directly to v_i via an edge with length $d(u, v_i)$. □

If a scenario whose front is in $T' \backslash T^*$ is discarded, we discard the corresponding node u that now has 0 weight. If u is a leaf, we simply remove the leaf and the edge incident to it. If u is a non-leaf node on the fringe of T^*, then we remove u, and attach all its child nodes (leaves) to the parent of u with appropriate changes in the edge lengths.

4.3 Choosing Pivot Vertex r

By definition, T^* (hence S_r) depends on the choice of r in step 1 of procedure `Prune`. We want to choose r judiciously, so that T^* is as small as we can make it and S_r contains as many scenarios as possible, which in turn maximizes the number $\lceil p/4 \rceil$ of scenarios removed in step 5. We use a w-centroid of T' as r. We can find a w-centroid r of T' in $O(k)$ time, where $k = |N|$ [7]. By definition (of a w-centroid), none of the r-branches minus r, $T^* \backslash \{r\}$ in particular, contains more than 1/2 of the total number of fronts, and therefore, $T \backslash T^* \cup \{r\}$ contains at least 1/2 of the total number of fronts. This fact will be used in (14) below.

Now that we have r, we need to find T^* (step 2 of `Prune`) based on Lemma 7. From Fig. 2(b), we get

$$F^s(r) = \underline{C}_t(r) - \{\underline{C}_t(f(s)) + d(r, f(s))\underline{W}_t(f(s))\} + \{\overline{C}_t(f(s)) + d(r, f(s))\overline{W}_t(f(s))\}. \quad (13)$$

The second term above deletes the min-weight contribution from subtree $T(f(s))$, and the third term replaces it with the max-weight contribution from $T(f(s))$.

This can be evaluated in constant time, since we know r, $f(s)$ and $d(r, f(s))$. We compute $R^s(r) = F^s(r) - F^s(m(s))$ for all scenarios $s \in S$ whose max-weighted components do not contain r and two others such that r is their front. We can then identify the scenario $\hat{s}(r)$ that maximizes $R^s(r)$. The r-branch that contains $m(\hat{s}(r))$ is identified as T^*.

4.4 Time Complexity Analysis

Let $T' = (N, E')$ be the auxiliary tree defined in Subsec. 4.2. Once a w-centroid in T' is found, we classify the nodes in T' into two types, B and C. A node u of type B represents the scenario s such that $f(s) = u$ whose max weighted component is totally contained in $T \backslash T^* \cup \{r\}$. A node u of type C represents two scenarios s and s' such that $f(s) = f(s') = u$, where the max-weighted component of s contains the path from u to T^*, as well as T^*, and the max-weighted component of s' contains u but not the path from u to T^*.

In $T \backslash T^* \cup \{r\}$, let there be b and c nodes of types B and C, respectively. The total weight of the nodes in $T \backslash T^* \cup \{r\}$ is given by

$$b + 2c \geq |S|/2. \tag{14}$$

The inequality follows from the discussion in Subsec. 4.3. Step 3 of `Prune` discards c scenarios from S, and step 5 throws away an additional $(b + c)/8$. The total number of scenarios discarded is thus

$$\begin{aligned} c + (b + c)/8 &= (7/8)c + (b + 2c)/8 \\ &\geq (7/8)c + |S|/16. \end{aligned} \tag{15}$$

The inequality follows from (14). After a round, some nodes of type C may become type B nodes. We attach those nodes directly to r, as in Fig. 3, where each node represented by a hollow (resp. solid) circle is a node of type B (resp. type C).[6] Note that in the future, the w-centroid will never be in $T \backslash T^*$, since this part loses some weight, whereas there is no weight loss from $T^* \backslash \{r\}$.

Let $|S| = k$ and let $\tau(k)$ be the time needed to process the scenarios in S. Eq. (15) means that procedure `Prune` removes at least $|S|/16$ fronts (i.e., scenarios), reducing the size of S by at least $1/16$ in each round. The recurrence relation is

$$\tau(k) \leq \tau(15k/16) + O(k). \tag{16}$$

This recurrence equation has the solution of the form $\tau(k) = O(k)$. We thus have

Theorem 4. *The minmax regret 1-median of a tree can be found in $O(n)$ time, where n is the number of vertices in the given tree.* □

[6] Subtree T^* in the figure is for the previous round. A new round will choose the pivot r, which defines a new T^*, and new sets of type B and C nodes.

5 Conclusion

We have presented an $O(n)$ time algorithm for computing the minmax regret 1-median for trees with positive vertex weights. This improves upon the previously known best complexity of $O(n \log n)$.

References

1. Averbakh, I., Berman, O.: Minmax regret median location on a network under uncertainty. INFORMS Journal of Computing 12(2), 104–110 (2000)
2. Averbakh, I., Berman, O.: An improved algorithm for the minmax regret median problem on a tree. Networks 41, 97–103 (2003)
3. Brodal, G.S., Georgiadis, L., Katriel, I.: An $O(n \log n)$ version of the Averbakh–Berman algorithm for the robust median of a tree. Operations Research Letters 36, 14–18 (2008)
4. Burkard, R.E., Dollani, H.: Robust location problems with pos/neg-weights on a tree. Tech. Rep. Diskrete Optimierung Bericht Nr. 148, Karl-Franzens-Universiät Graz & Technische Universiät Graz (1999)
5. Burkard, R., Krarup, J.: A linear algorithm for the pos/neg-weighted 1-median problem on a cactus. Computing 60, 193–215 (1998)
6. Chen, B., Lin, C.S.: Minmax-regret robust 1-median location on a tree. Networks 31, 93–103 (1998)
7. Goldman, A.: Optimal center location in simple networks. Transportation Science 5, 212–221 (1971)
8. Hakimi, S.: Optimum locations of switching centers and the absolute centers and medians of a graph. Operations Research 12, 450–459 (1964)
9. Hale, T.S., Moberg, C.R.: Location science research: A review. Annals of Operations Research 123, 21–35 (2003)
10. Kariv, O., Hakimi, S.: An algorithmic approach to network location problems, part 2: The p-median. SIAM J. Appl. Math. 37, 539–560 (1979)
11. Kouvelis, P., Vairaktarakis, G., Yu, G.: Robust 1-median location on a tree in the presence of demand and transportation cost uncertainty. Tech. Rep. Working Paper 93/94-3-4, Department of Management Science. The University of Texas, Austin (1993)
12. Megiddo, N.: Linear-time algorithms for linear-programming in R^3 and related problems. SIAM J. Computing 12, 759–776 (1983)
13. Yu, H.I., Lin, T.C., Wang, B.F.: Improved algorithms for the minmax-regret 1-center and 1-median problem. ACM Transactions on Algorithms 4(3), 1–1 (2008)

A Simple D^2-Sampling Based PTAS for k-Means and other Clustering Problems

Ragesh Jaiswal, Amit Kumar, and Sandeep Sen

Department of Computer Science and Engineering,
Indian Institute of Technology Delhi
{rjaiswal,amitk,ssen}@cse.iitd.ac.in

Abstract. Given a set of points $P \subset \mathbb{R}^d$, the k-means clustering problem is to find a set of k *centers* $C = \{c_1, ..., c_k\}, c_i \in \mathbb{R}^d$, such that the objective function $\sum_{x \in P} d(x, C)^2$, where $d(x, C)$ denotes the distance between x and the closest center in C, is minimized. This is one of the most prominent objective functions that have been studied with respect to clustering.

D^2-sampling [1] is a simple non-uniform sampling technique for choosing points from a set of points. It works as follows: given a set of points $P \subseteq \mathbb{R}^d$, the first point is chosen uniformly at random from P. Subsequently, a point from P is chosen as the next sample with probability proportional to the square of the distance of this point to the nearest previously sampled points.

D^2-sampling has been shown to have nice properties with respect to the k-means clustering problem. Arthur and Vassilvitskii [1] show that k points chosen as centers from P using D^2-sampling gives an $O(\log k)$ approximation in expectation. Ailon et. al. [2] and Aggarwal et. al. [3] extended results of [1] to show that $O(k)$ points chosen as centers using D^2-sampling give $O(1)$ approximation to the k-means objective function with high probability. In this paper, we further demonstrate the power of D^2-sampling by giving a simple randomized $(1 + \epsilon)$-approximation algorithm that uses the D^2-sampling in its core.

1 Introduction

Clustering problems arise in diverse areas including machine learning, data mining, image processing and web-search [4,5,6,7]. One of the most commonly used clustering problems is the k-means problem. Here, we are given a set of points P in a d-dimensional Euclidean space, and a parameter k. The goal is to find a set C of k *centers* such that the objective function

$$\Delta(P, C) = \sum_{p \in P} d(p, C)^2$$

is minimized, where $d(p, C)$ denotes the distance from p to the closest center in C. This naturally partitions P into k clusters, where each cluster corresponds to the set of points of P which are closer to a particular center than other centers.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 13–24, 2012.

It is also easy to show that the center of any cluster must be the mean of the points in it. In most applications, the parameter k is a small constant. However, this problem turns out to be NP-hard even for $k = 2$ [8].

One very popular heuristic for solving the k-means problem is the Lloyd's algorithm [9]. The heuristic is as follows : start with an arbitrary set of k centers as seeds. Based on these k centers, partition the set of points into k clusters, where each point gets assigned to the closest center. Now, we update the set of centers as the means of each of these clusters. This process is repeated till we get convergence. Although, this heuristic often performs well in practice, it is known that it can get stuck in local minima [10]. There has been lot of recent research in understanding why this heuristic works fast in practice, and how it can be modified such that we can guarantee that the solution produced by this heuristic is always close to the optimal solution.

One such modification is to carefully choose the set of initial k centers. Ideally, we would like to pick these centers such that we have a center close to each of the optimal clusters. Since we do not know the optimal clustering, we would like to make sure that these centers are not close to each other and yet, are representatives of the set of points. A recently proposed idea [11,1] is to pick the initial centers using D^2-sampling which can be described as follows. The first center is picked uniformly at random from the set of points P. Suppose we have picked a set of $k' < k$ centers – call this set C'. Then a point $p \in P$ is chosen as the next center with probability proportional to $d(p, C')^2$. This process is repeated till we have a set of k centers.

There has been lot of recent activity in understanding how good a set of centers picked by D^2-sampling are (even if we do not run the Lloyd's algorithm on these seed centers). Arthur and Vassilvitskii [1] showed that if we pick k centers with D^2-sampling, then the expected cost of the corresponding solution to the k-means instance is within $O(\log k)$-factor of the optimal value. Ackermann et. al. [12] note that the analysis in [1] can be extended to other distance measures such as μ-similar Bregman divergences and use the algorithm to give a coreset construction. Ostrovsky et. al. [11] showed that if the set of points satisfied a separation condition (named (k, ϵ^2)-irreducible as defined in Section 2), then these k centers give a constant factor approximation for the k-means problem. Ailon et. al. [2] proved a bi-criteria approximation property – if we pick $O(k \log k)$ centers by D^2-sampling, then it is a constant approximation, where we compare with the optimal solution that is allowed to pick k centers only. Aggarwal et. al. [3] give an improved result and show that it is enough to pick $O(k)$ centers by D^2-sampling to get a constant factor bi-criteria approximation algorithm.

In this paper, we give yet another illustration of the power of the D^2-sampling idea. We give a simple randomized $(1 + \epsilon)$-approximation algorithm for the k-means algorithm, where $\epsilon > 0$ is an arbitrarily small constant. At the heart of our algorithm is the idea of D^2-sampling – given a set of already selected centers, we pick a small set of points by D^2-sampling with respect to these selected centers. Then, we pick the next center as the centroid of a subset of these small set of points. By repeating this process of picking k centers sufficiently many

times, we can guarantee that with high probability, we will get a set of k centers whose objective value is close to the optimal value. Further, the running time of our algorithm is $O(nd \cdot 2^{\tilde{O}(k^2/\epsilon)})$[1] – for constant value of k, this is a linear time algorithm. It is important to note that PTAS with better running time are known for this problem. Chen [13] give an $O\left(nkd + d^2 n^\sigma \cdot 2^{(k/\epsilon)^{O(1)}}\right)$ algorithm for any $\sigma > 0$ and Feldman et al. [14] give an $O\left(nkd + d \cdot poly(k/\epsilon) + 2^{\tilde{O}(k/\epsilon)}\right)$ algorithm. However, these results often are quite involved, and use the notion of coresets. Our algorithm is simple, and only uses the concept of D^2-sampling.

1.1 Other Related Work

There has been significant research on exactly solving the k-means algorithm (see e.g., [15]), but all of these algorithms take $\Omega(n^{kd})$ time. Hence, recent research on this problem has focused on obtaining fast $(1+\epsilon)$-approximation algorithms for any $\epsilon > 0$. Matousek [16] gave a PTAS with running time $O(n\epsilon^{-2k^2 d} \log^k n)$. Badoiu et al. [17] gave an improved PTAS with running time $O(2^{(k/\epsilon)^{O(1)}} d^{O(1)} n \log^{O(k)} n)$. de la Vega et al. [18] gave a PTAS which works well for points in high dimensions. The running time of this algorithm is $O(g(k,\epsilon)n \log^k n)$ where $g(k,\epsilon) = \exp[(k^3/\epsilon^8)(\ln(k/\epsilon)\ln k]$. Har-Peled et al. [19] proposed a PTAS whose running time is $O(n + k^{k+2}\epsilon^{-(2d+1)k} \log^{k+1} n \log^k \frac{1}{\epsilon})$. Kumar et al. [20] gave the first linear time PTAS for fixed k – the running time of their algorithm is $O(2^{(k/\epsilon)^{O(1)}} dn)$. Chen [13] used the a new coreset construction to give a PTAS with improved running time of $O(ndk + 2^{(k/\epsilon)^{O(1)}} d^2 n^\sigma)$. Recently, Feldman et al. [14] gave a PTAS with running time $O(nkd + d \cdot poly(k/\epsilon) + 2^{\tilde{O}(k/\epsilon)})$ – this is the fastest known PTAS (for fixed k) for this problem.

There has also been work on obtaining fast constant factor approximation algorithms for the k-means problem based on some properties of the input points (see e.g. [11,21]).

1.2 Our Contributions

In this paper, we give a simple PTAS for the k-means problem based on the idea of D^2-sampling. Our work builds on and simplifies the result of Kumar et al. [20]. We briefly describe their algorithm first. It is well known that for the 1-mean problem, if we sample a set of $O(1/\epsilon)$ points uniformly at random, then the mean of this set of sampled points is close to the overall mean of the set of all points. Their algorithm begins by sampling $O(k/\epsilon)$ points uniformly at random. With reasonable probability, we would sample $O(1/\epsilon)$ points from the largest cluster, and hence we could get a good approximation to the center corresponding to this cluster (their algorithm tries all subsets of size $O(1/\epsilon)$ from the randomly sampled points). However, the other clusters may be much smaller, and we may not have sampled enough points from them. So, they need to prune a lot of

[1] $\tilde{O}$ notation hides a $O(\log k/\epsilon)$ factor which simplifies the expression.

points from the largest cluster so that in the next iteration a random sample of $O(k/\epsilon)$ points will contain $O(1/\epsilon)$ points from the second largest cluster, and so on. This requires a non-trivial idea termed as *tightness* condition by the authors. In this paper, we show that the pruning is not necessary if instead of using uniform random sampling, one uses D^2-sampling.

We can informally describe our algorithm as follows. We maintain a set of candidate centers C, which is initially empty. Given a set C, $|C| < k$, we add a new center to C as follows. We sample a set S of $O(k/\epsilon^3)$ points using D^2-sampling with respect to C. From this set of sampled points, we pick a subset T and the new center is the mean of this set T. We add this to C and continue.

From the property of D^2-sampling ([3,2]), with some constant, albeit small probability p', we pick up a point from a hitherto untouched cluster C' of the *optimal clustering*. Therefore by sampling about α/p' points using D^2-sampling, we expect to hit approximately α points from C'. If α is large enough, (c.f. Lemma 1), then the centroid of these α points gives a $(1+\epsilon)$ approximation of the cluster C'. Therefore, with reasonable probability, there will be a choice of a subset T in each iteration such that the set of centers chosen are from C'. Since we do not know T, our algorithm will try out all subsets of size $|T|$ from the sample S. Note that our algorithm is very simple, and can be easily parallelized. Our algorithm has running time $O(dn \cdot 2^{\tilde{O}(k^2/\epsilon)})$ which is an improvement over that of Kumar et al. [20] who gave a PTAS with running time $O\left(nd \cdot 2^{(k/\epsilon)^{O(1)}}\right)$.[2]

Because of the relative simplicity, our algorithm generalizes to measures like Mahalanobis distance and μ-similar Bregman divergence. Note that these do not satisfy triangle inequality and therefore not strict metrics. Ackermann et al. [12] have generalized the framework of Kumar et al. [20] to Bregman divergences but we feel that the D^2-sampling based algorithms are simpler.

We formally define the problem and give some preliminary results in Section 2. In Section 3, we describe our algorithm, and then analyze it subsequently. In Section 4, we discuss PTAS for other distance measures.

2 Preliminaries

An instance of the k-means problem consists of a set $P \subseteq \mathbb{R}^d$ of n points in d-dimensional space and a parameter k. For a set of points (called centers) $C \subseteq \mathbb{R}^d$, let $\Delta(P, C)$ denote $\sum_{p \in P} d(p, C)^2$, i.e., the cost of the solution which picks C as the set of centers. For a singleton $C = \{c\}$, we shall often abuse notation, and use $\Delta(P, c)$ to denote $\Delta(P, C)$. Let $\Delta_k(P)$ denote the cost of the optimal k-means solution for P.

Definition 1. *Given a set of points P and a set of centers C, a point $p \in P$ is said to be sampled using* D^2-sampling *with respect to C if the probability of it being sampled, $\rho(p)$, is given by*

[2] It can be used in conjunction with Chen [13] to obtain a superior running time but at the cost of the simplicity of our approach.

$$\rho(p) = \frac{d(p,C)^2}{\sum_{x\in P} d(x,C)^2} = \frac{\Delta(\{p\},C)}{\Delta(P,C)}.$$

We will also need the following definition from [20].

Definition 2 (Irreducibility or separation condition). *Given k and γ, a set of points P is said to be (k,γ)-irreducible if $\Delta_{k-1}(P) \geq (1+\gamma)\cdot\Delta_k(P)$.*

We will often appeal to the following result [15] which shows that uniform random sampling works well for 1-means[3].

Lemma 1 (Inaba et al. [15]). *Let S be a set of points obtained by independently sampling M points with replacement uniformly at random from a point set P. Then, for any $\delta > 0$,*

$$\Delta(P,\{m(S)\}) \leq \left(1+\frac{1}{\delta M}\right)\cdot\Delta(P,\{m(P)\}),$$

holds with probability at least $(1-\delta)$. Here $m(X) = \left(\frac{\sum_{x\in X} x}{|X|}\right)$ denotes the centroid of a point set X.

Finally, we will use the following property of the squared Euclidean metric. This is a standard result from linear algebra [22].

Lemma 2. *Let $P \subseteq \mathbb{R}^d$ be any point set and let $c \in \mathbb{R}^d$ be any point. Then we have the following:*

$$\sum_{p\in P} d(p,c)^2 = \sum_{p\in P} d(p,m(P))^2 + |P|\cdot d(c,m(P))^2,$$

where $m(P) = \left(\frac{\sum_{p\in P} p}{|P|}\right)$ denotes the centroid of the point set.

Finally, we mention the simple approximate triangle inequality with respect to the squared Euclidean distance measure.

Lemma 3 (Approximate triangle inequality). *For any three points $p,q,r \in \mathbb{R}^d$ we have: $d(p,q)^2 \leq 2\cdot(d(p,r)^2 + d(r,q)^2)$.*

3 PTAS for *k*-Means

We first give a high level description of the algorithm. We will also assume that the instance is (k,ϵ)-irreducible for a suitably small parameter ϵ. We shall then get rid of this assumption later. The algorithm is described in Figure 1. Essentially, the algorithm maintains a set C of centers, where $|C| \leq k$. Initially C is empty, and in each iteration of Step 2(b), it adds one center to C till its

[3] It turns out that even minor perturbations from uniform distribution can be catastrophic and indeed in this paper we had to work around this.

size reaches k. Given a set C, it samples a set S of N points from P using D^2-sampling with respect to C (in Step 2(b)). Then it picks a subset T of S of size $M = O(1/\epsilon)$, and adds the centroid of T to C. The algorithm cycles through all possible subsets of size M of S as choices for T, and for each such choice, repeats the above steps to find the next center, and so on. To make the presentation clearer, we pick a k-tuple of M-size subsets $(s_1, \ldots, s_k)$ in advance, and when $|C| = i$, we pick T as the s_i^{th} subset of S. In Step 2(i), we cycle through all such k-tuples $(s_1, \ldots, s_k)$. In the analysis, we just need to show that *one* such k-tuple works with reasonable probability.

We develop some notation first. For the rest of the analysis, we will fix a tuple $(s_1, \ldots, s_k)$ – this will be the "desired tuple", i.e., the one for which we can show that the set C gives a good solution. As our analysis proceeds, we will argue what properties this tuple should have. Let $C^{(i)}$ be the set C at the end of the i^{th} iteration of Step 2(b). To begin with $C^{(0)}$ is empty. Let $S^{(i)}$ be the set S sampled during the i^{th} iteration of Step 2(b), and $T^{(i)}$ be the corresponding set T (which is the s_i^{th} subset of $S^{(i)}$).

Let $O_1, \ldots, O_k$ be the optimal clusters, and c_i denote the centroid of points in O_i. Further, let m_i denote $|O_i|$, and wlog assume that $m_1 \geq \ldots \geq m_k$. Note that $\Delta_1(O_i)$ is same as $\Delta(O_i, \{c_i\})$. Let r_i denote the average cost paid by a point in O_i, i.e.,

$$r_i = \frac{\sum_{p \in O_i} d(p, c_i)^2}{m_i}.$$

We will assume that the input set of points P are (k, ϵ)-irreducible. We shall remove this assumption later. Now we show that any two optimal centers are far enough.

Lemma 4. *For any* $1 \leq i, j \leq k, i \neq j$, $d(c_i, c_j)^2 \geq \epsilon \cdot (r_i + r_j)$.

Proof. Suppose $i < j$, and hence $m_i \geq m_j$. For the sake of contradiction assume $d(c_i, c_j)^2 < \epsilon \cdot (r_i + r_j)$. Then we have,

$$\begin{aligned}
\Delta(O_i \cup O_j, \{c_i\}) &= m_i \cdot r_i + m_j \cdot r_j + m_j \cdot d(c_i, c_j)^2 \quad \text{(using Lemma 2)} \\
&\leq m_i \cdot r_i + m_j \cdot r_j + m_j \cdot \epsilon \cdot (r_i + r_j) \\
&\leq (1 + \epsilon) \cdot m_i \cdot r_i + (1 + \epsilon) \cdot m_j \cdot r_j \quad \text{(since } m_i \geq m_j\text{)} \\
&\leq (1 + \epsilon) \cdot \Delta(O_i \cup O_j, \{c_i, c_j\})
\end{aligned}$$

This implies that the centers $\{c_1, ..., c_k\} \backslash \{c_j\}$ give a $(1+\epsilon)$-approximation to the k-means objective. This contradicts the fact that P is (k, ϵ)-irreducible. □

We give an outline of the proof. Suppose in the first $i - 1$ iterations, we have found centers which are close to the centers of some $i - 1$ clusters in the optimal solution. Conditioned on this fact, we show that in the next iteration, we are likely to sample enough number of points from one of the remaining clusters (c.f. Corollary 1). Further, we show that the samples from this new cluster are close

Find-k-means(P)
Let $N = (51200 \cdot k/\epsilon^3)$, $M = 100/\epsilon$, and $R = \binom{N}{M}$
1. **Repeat** 2^k times and output the set of centers C that give least cost
2. **Repeat** for all k-tuples $(s_1, ..., s_k) \in [R] \times [R] \times .... \times [R]$ and pick the set of centers C that gives least cost
(a) $C \leftarrow \{\}$
(b) For $i \leftarrow 1$ to k
Sample a set S of N points with D^2-sampling (w.r.t. centers C)
Let T be the s_i^{th} subset of S [a]
$C \leftarrow C \cup \{m(T)\}$. [b]

[a] For a set of size N we consider an arbitrary ordering of the subsets of size M of this set.
[b] $m(T)$ denote the centroid of the points in T.

Fig. 1. The k-means algorithm that gives $(1 + \epsilon)$-approximation for any (k, ϵ)-irreducible data set. Note that the inner loop is executed at most $2^k \cdot \left(\binom{N}{M}\right)^k \sim 2^k \cdot 2^{\tilde{O}(k/\epsilon)}$ times.

to uniform distribution (c.f. Lemma 6). Since such a sample does not come from exactly uniform distribution, we cannot apply Lemma 1 directly. In fact, dealing with the slight non-uniformity turns out to be non-trivial (c.f. Lemmas 7 and 8).

We now show that the following invariant will hold for all iterations : let $C^{(i-1)}$ consist of centers $c'_1, \ldots, c'_{i-1}$ (added in this order). Then, with probability at least $\frac{1}{2^{i-1}}$, there exist distinct indices $j_1, \ldots, j_{i-1}$ such that for all $l = 1, \ldots, i-1$,

$$\Delta(O_{j_l}, c'_l) \leq (1 + \epsilon/20) \cdot \Delta(O_{j_l}, c_{j_l}) \tag{1}$$

Suppose this invariant holds for $C^{(i-1)}$ (the base case is easy since $C^{(0)}$ is empty). We now show that this invariant holds for $C^{(i)}$ as well. In other words, we just show that in the i^{th} iteration, with probability at least $1/2$, the algorithm finds a center c'_i such that

$$\Delta(O_{j_i}, c'_i) \leq (1 + \epsilon/20) \cdot \Delta(O_{j_i}, c_{j_i}),$$

where j_i is an index distinct from $\{j_1, \ldots, j_{i-1}\}$. This will basically show that at the end of the last iteration, we will have k centers that give a $(1 + \epsilon)$-approximation with probability at least 2^{-k}.

We now show that the invariant holds for $C^{(i)}$. We use the notation developed above for $C^{(i-1)}$. Let I denote the set of indices $\{j_1, \ldots, j_{i-1}\}$. Now let $j_i \notin I$ be the index for which $\Delta(O_{j_i}, C^{(i-1)})$ is maximum. Intuitively, conditioned on sampling from the set $\cup_{j \notin I} O_j$ using D^2-sampling, it is likely that enough points from O_{j_i} will be sampled. The next lemma shows that there is good chance that elements from the sets O_j for $j \notin I$ will be sampled.

Lemma 5. *If* $0 < \epsilon \leq 1/2$, *then* $\frac{\sum_{l \notin I} \Delta(O_l, C^{(i-1)})}{\sum_{l=1}^{k} \Delta(O_l, C^{(i-1)})} \geq \epsilon/2$.

Proof. Suppose, for the sake of contradiction, the above statement does not hold. Then,

$$\begin{aligned}
\Delta(P, C^{(i-1)}) &= \sum_{l \in I} \Delta(O_l, C^{(i-1)}) + \sum_{l \notin I} \Delta(O_l, C^{(i-1)}) \\
&< \sum_{l \in I} \Delta(O_l, C^{(i-1)}) + \frac{\epsilon/2}{1-\epsilon/2} \cdot \sum_{l \in I} \Delta(O_l, C^{(i-1)}) \quad \text{(by our assumption)} \\
&= \frac{1}{1-\epsilon/2} \cdot \sum_{l \in I} \Delta(O_l, C^{(i-1)}) \\
&\leq \frac{1+\epsilon/20}{1-\epsilon/2} \cdot \sum_{l \in I} \Delta_1(O_l) \quad \text{(using the invariant for } C^{(i-1)}\text{)} \\
&\leq (1+\epsilon) \cdot \sum_{l \in I} \Delta_1(O_l) \quad \text{(since } \epsilon \leq 1/2\text{)} \\
&\leq (1+\epsilon) \cdot \sum_{l \in [k]} \Delta_1(O_l)
\end{aligned}$$

But this contradicts the fact that P is (k, ϵ)-irreducible. □

We get the following corollary easily.

Corollary 1. *If* $0 < \epsilon \leq 1/2$, *then* $\frac{\Delta(O_{j_i}, C^{(i-1)})}{\sum_{l=1}^{k} \Delta(O_l, C^{(i-1)})} \geq \frac{\epsilon}{2k}$.

The above Lemma and its Corollary say that with probability at least $\frac{\epsilon}{2k}$, points in the set O_{j_i} will be sampled. However the points within O_{j_i} are not sampled uniformly. Some points in O_{j_i} might be sampled with higher probability than other points. In the next lemma, we show that each point will be sampled with certain minimum probability.

Lemma 6. *For any* $l \notin I$ *and any point* $p \in O_l$, $\frac{d(p, C^{(i-1)})^2}{\Delta(O_l, C^{(i-1)})} \geq \frac{1}{m_l} \cdot \frac{\epsilon}{64}$.

Proof. Fix a point $p \in O_l$. Let $j_t \in I$ be the index such that p is closest to c'_t among all centers in $C^{(i-1)}$. We have

$$\begin{aligned}
\Delta(O_l, C^{(i-1)}) &\leq m_l \cdot r_l + m_l \cdot d(c_l, c'_t)^2 \quad \text{(using Lemma 2)} \\
&\leq m_l \cdot r_l + 2 \cdot m_l \cdot \left(d(c_l, c_{j_t})^2 + d(c_{j_t}, c'_t)^2 \right) \quad \text{(using Lemma 3)} \\
&\leq m_l \cdot r_l + 2 \cdot m_l \cdot \left(d(c_l, c_{j_t})^2 + \frac{\epsilon r_t}{20} \right), \qquad (2)
\end{aligned}$$

where the third inequality follows from the invariant condition for $C^{(i-1)}$. Also, we know that

$$\begin{aligned}
d(p, c'_t)^2 &\geq \frac{d(c_{j_t}, c_l)^2}{8} - d(c_{j_t}, c'_t)^2 \quad \text{(using Lemma 3)} \\
&\geq \frac{d(c_{j_t}, c_l)^2}{8} - \frac{\epsilon}{20} \cdot r_t \quad \text{(using the invariant for } C^{(i-1)}\text{)} \\
&\geq \frac{d(c_{j_t}, c_l)^2}{16} \quad \text{(Using Lemma 4)} \qquad (3)
\end{aligned}$$

So, we get

$$\frac{d(p,C^{(i-1)})^2}{\Delta(O_l,C^{(i-1)})} \geq \frac{d(c_{j_t},c_l)^2}{16\cdot m_l\cdot\left(r_l+2\left(d(c_{j_t},c_l)^2+\frac{\epsilon r_l}{20}\right)\right)} \quad \text{(using (2) and (3))}$$

$$\geq \frac{1}{16\cdot m_l}\cdot\frac{1}{(1/\epsilon)+2+1/10} \geq \frac{\epsilon}{64\cdot m_l} \quad \text{(using Lemma 4)}$$

□

Recall that $S^{(i)}$ is the sample of size N in this iteration. We would like to show that that the invariant will hold in this iteration as well. We first prove a simple corollary of Lemma 1.

Lemma 7. *Let Q be a set of n points, and γ be a parameter, $0<\gamma<1$. Define a random variable X as follows : with probability γ, it picks an element of Q uniformly at random, and with probability $1-\gamma$, it does not pick any element (i.e., is null). Let $X_1,\ldots,X_\ell$ be ℓ independent copies of X, where $\ell=\frac{400}{\gamma\epsilon}$. Let T denote the (multi-set) of elements of Q picked by $X_1,\ldots,X_\ell$. Then, with probability at least $3/4$, T contains a subset U of size $\frac{100}{\epsilon}$ which satsifies*

$$\Delta(P,m(U)) \leq \left(1+\frac{\epsilon}{20}\right)\Delta_1(P) \tag{4}$$

Proof. Define a random variable I, which is a subset of the index set $\{1,\ldots,\ell\}$, as follows $I=\{t : X_t \text{ picks an element of } Q, \text{ i.e., it is not null}\}$. Conditioned on $I=\{t_1,\ldots,t_r\}$, note that the random variables $X_{t_1},\ldots,X_{t_r}$ are independent uniform samples from Q. Thus if $|I|\geq\frac{100}{\epsilon}$, then Lemma 1 implies that with probability at least 0.8, the desired event (4) happens. But the expected value of $|I|$ is $\frac{400}{\epsilon}$, and so, $|I|\geq\frac{100}{\epsilon}$ with high probability, and hence, the statement in the lemma is true. □

We are now ready to prove the main lemma.

Lemma 8. *With probability at least $1/2$, there exists a subset $T^{(i)}$ of $S^{(i)}$ of size at most $\frac{100}{\epsilon}$ such that*

$$\Delta(O_{j_i},m(T^{(i)})) \leq (1+\frac{\epsilon}{20})\cdot\Delta_1(O_{j_i}).$$

Proof. Recall that $S^{(i)}$ contains $N=\frac{51200k}{\epsilon^3}$ independent samples of P (using D^2-sampling). We are interested in $S^{(i)}\cap O_{j_i}$. Let $Y_1,\ldots,Y_N$ be N independent random variables defined as follows : for any t, $1\leq t\leq N$, Y_t picks an element of P using D^2-sampling with respect to $C^{(i-1)}$. If this element is not in O_{j_i}, it just discards it (i.e., Y_t is null). Let γ denote $\frac{\epsilon^2}{128k}$. Corollary 1 and Lemma 6 imply that Y_t picks a particular element of O_{j_i} with probability at least $\frac{\gamma}{m_{j_i}}$. We would now like to apply Lemma 7 (observe that $N=\frac{400}{\gamma\epsilon}$). We can do this by a simple coupling argument as follows. For a particular element $p\in O_{j_i}$, suppose Y_t assigns probability $\frac{\gamma(p)}{m_{j_i}}$ to it. One way of sampling a random variable X_t as

in Lemma 7 is as follows – first sample using Y_t. If Y_t is null then, X_t is also null. Otherwise, suppose Y_t picks an element p of O_{j_i}. Then, X_t is equal to p with probability $\frac{\gamma}{\gamma(p)}$, null otherwise. It is easy to check that with probability γ, X_t is a uniform sample from O_{j_i}, and null with probability $1-\gamma$. Now, observe that the set of elements of O_{j_i} sampled by $Y_1, \ldots, Y_N$ is always a superset of $X_1, \ldots, X_N$. We can now use Lemma 7 to finish the proof. □

Thus, we will take the index s_i in Step 2(i) as the index of the set $T^{(i)}$ as guaranteed by the Lemma above. Finally, by repeating the entire process 2^k times, we make sure that we get a $(1+\epsilon)$-approximate solution with high probability. Note that the total running time of our algorithm is $\left(nd \cdot 2^k \cdot 2^{\tilde{O}(k/\epsilon)}\right)$.

Removing the (k,ϵ)-irreducibility assumption : We now show how to remove this assumption. First note that we have shown the following result.

Theorem 1. *Let $0 < \epsilon \leq 1/2$. If a given point set is $(k, \frac{\epsilon}{(1+\epsilon/2)\cdot k})$-irreducible, then there is a randomized algorithm that runs in time $O(nd \cdot 2^{\tilde{O}(k^2/\epsilon)})$ and gives a $(1+\frac{\epsilon}{(1+\epsilon/2)\cdot k})$-approximation to the k-means objective with probability at least $1/2$.*

Proof. The proof can be obtained by replacing ϵ by $\frac{\epsilon}{(1+\epsilon/2)\cdot k}$ in the above analysis.

Suppose the point set P is not $(k, \frac{\epsilon}{(1+\epsilon/2)\cdot k})$-irreducible. In that case it will be sufficient to find fewer centers that $(1+\epsilon)$-approximate the k-means objective. The next lemma shows this more formally.

Theorem 2. *Let $0 < \epsilon \leq 1/2$. There is a randomized algorithm that runs in time $O(nd \cdot 2^{\tilde{O}(k^2/\epsilon)})$ and gives a $(1+\epsilon)$-approximation to the k-means objective with probability at least $1/2$.*

Proof. Let P denote the set of points. Let $1 < i \leq k$ be the largest index such that P is $(i, \frac{\epsilon}{(1+\epsilon/2)\cdot k})$-irreducible. If no such i exists, then

$$\Delta_1(P) \leq \left(1 + \frac{\epsilon}{(1+\epsilon/2)\cdot k}\right)^k \cdot \Delta_k(P) \leq (1+\epsilon)\cdot \Delta_k(P),$$

and so picking the centroid of P will give a $(1+\epsilon)$-approximation.

Suppose such an i exists. In that case, we consider the i-means problem and from the previous lemma we get that there is an algorithm that runs in time $O(nd \cdot 2^i \cdot 2^{\tilde{O}(i^2/\epsilon)})$ and gives a $(1+\frac{\epsilon}{(1+\epsilon/2)\cdot k})$-approximation to the i-means objective. Now we have that

$$\Delta_i \leq \left(1 + \frac{\epsilon}{(1+\epsilon/2)\cdot k}\right)^{k-i} \cdot \Delta_k \leq (1+\epsilon)\cdot \Delta_k.$$

Thus, we are done. □

4 Other Distance Measures

In the previous sections, we looked at the k-means problem where the dissimilarity or distance measure was the square of Euclidean distance. There are numerous practical clustering problem instances where the dissimilarity measure is not a function of the Euclidean distance. In many cases, the points are not generated from a metric space. In these cases, it makes sense to talk about the general k-median problem that can be defined as follows:

Definition 3 (k-median with respect to a dissimilarity measure). *Given a set of n objects $P \subseteq \mathcal{X}$ and a dissimilarity measure $D : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$, find a subset C of k objects (called medians) such that the following objective function is minimized:*

$$\Delta(P, C) = \sum_{p \in P} \min_{c \in C} D(p, c)$$

In this section, we will show that our algorithm and analysis can be easily generalized and extended to dissimilarity measures that satisfy some simple properties. We will look at some interesting examples. Due to lack of space, we just give our main results for this section. The entire discussion could be found in the full version of the paper.[4]

Theorem 3 (k-median w.r.t. Mahalanobis distance). *Let $0 < \epsilon \leq 1/2$. There is a randomized algorithm that runs in time $O(nd \cdot 2^{\tilde{O}(k^2/\epsilon)})$ and with probability at least $1/2$, gives a $(1+\epsilon)$-approximation to the k-median objective function w.r.t. Mahalanobis distances for any point set $P \in \mathbb{R}^d, |P| = n$.*

Theorem 4 (k-median w.r.t. μ-similar Bregman divergences). *Let $0 < \mu \leq 1$ and $0 < \epsilon \leq 1/2$. There is a randomized algorithm that runs in time $O\left(nd \cdot 2^{\tilde{O}\left(\frac{k^2}{\mu \cdot \epsilon}\right)}\right)$ and with probability at least $1/2$, gives a $(1+\epsilon)$-approximation to the k-median objective function w.r.t. μ-similar Bregman divergence for any point set $P \in \mathbb{R}^d, |P| = n$.*

References

1. Arthur, D., Vassilvitskii, S.: k-means++: the advantages of careful seeding. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 1027–1035 (2007)
2. Ailon, N., Jaiswal, R., Monteleoni, C.: Streaming k-means approximation. In: Advances in Neural Information Processing Systems, vol. 22, pp. 10–18 (2009)
3. Aggarwal, A., Deshpande, A., Kannan, R.: Adaptive Sampling for k-Means Clustering. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX 2009. LNCS, vol. 5687, pp. 15–28. Springer, Heidelberg (2009)
4. Broder, A., Glassman, S., Manasse, M., Zweig, G.: Syntactic clustering of the web
5. Faloutsos, C., Barber, R., Flickner, M., Hafner, J.: Efficient and effective querying by image content. Journal of Intelligent Information Systems (1994)

[4] Link for full version of the paper: http://arxiv.org/abs/1201.4206v1

6. Deerwester, S., Dumais, S., Landauer, T., Furnas, G., Harshman, A.: Indexing by latent semantic analysis. Journal of the American Society for Information Science (1990)
7. Swain, M., Ballard, D.: Color indexing. International Journal of Computer Vision (1991)
8. Dasgupta, S.: The hardness of k-means clustering. Technical Report CS2008-0916, Department of Computer Science and Engineering. University of California San Diego (2008)
9. Lloyd, S.: Least squares quantization in PCM. IEEE Transactions on Information Theory 28(2), 129–137 (1982)
10. Arthur, D., Vassilvitskii, S.: How slow is the k-means method? In: Proc. 22nd Annual Symposium on Computational Geometry, pp. 144–153 (2006)
11. Ostrovsky, R., Rabani, Y., Schulman, L.J., Swamy, C.: The effectiveness of lloyd-type methods for the k-means problem. In: Proc. 47th IEEE FOCS, pp. 165–176 (2006)
12. Ackermann, M.R., Blömer, J.: Coresets and approximate clustering for bregman divergences. In: ACM SIAM Symposium on Discrete Algorithms, pp. 1088–1097 (2009)
13. Chen, K.: On k-median clustering in high dimensions. In: SODA, pp. 1177–1185 (2006)
14. Feldman, D., Monemizadeh, M., Sohler, C.: A ptas for k-means clustering based on weak coresets. In: Symposium on Computational Geometry, pp. 11–18 (2007)
15. Inaba, M., Katoh, N., Imai, H.: Applications of weighted voronoi diagrams and randomization to variance based k-clustering. In: Proceedings of the Tenth Annual Symposium on Computational Geometry, pp. 332–339 (1994)
16. Matousek, J.: On approximate geometric k-clustering. In: Discrete and Computational Geometry (2000)
17. Badoiu, M., Har-Peled, S., Indyk, P.: Approximate clustering via core-sets. In: STOC, pp. 250–257 (2002)
18. de la Vega, W.F., Karpinski, M., Kenyon, C., Rabani, Y.: Approximation schemes for clustering problems. In: ACM Symposium on Theory of Computing, pp. 50–58 (2003)
19. Har-Peled, S., Mazumdar, S.: On coresets for k-means and k-median clustering. In: ACM Symposium on Theory of Computing, pp. 291–300 (2004)
20. Kumar, A., Sabharwal, Y., Sen, S.: Linear-time approximation schemes for clustering problems in any dimensions. J. ACM 57(2) (2010)
21. Awasthi, P., Blum, A., Sheffet, O.: Stability yields a ptas for k-median and k-means clustering. In: FOCS, pp. 309–318 (2010)
22. Har-Peled, S., Sadri, B.: How fast is the k-means method? In: ACM SIAM Symposium on Discrete Algorithms, pp. 877–885 (2005)

Speed Scaling for Maximum Lateness

Evripidis Bampis[1,⋆], Dimitrios Letsios[1,2,⋆],
Ioannis Milis[3], and Georgios Zois[1,3,⋆⋆]

[1] LIP6, Université Pierre et Marie Curie, France
{Evripidis.Bampis,Georgios.Zois}@lip6.fr
[2] IBISC, Université d' Évry, France
dimitris.letsios@ibisc.univ-evry.fr
[3] Dept. of Informatics, Athens University of Economics and Business, Greece
milis@aueb.gr

Abstract. We consider the power-aware problem of scheduling non-preemptively a set of jobs on a single speed-scalable processor so as to minimize the maximum lateness. We consider two variants of the problem: In the *budget* variant we aim in finding a schedule minimizing the maximum lateness for a given budget of energy, while in the *aggregated* variant our objective is to find a schedule minimizing a linear combination of maximum lateness and energy. We present polynomial time algorithms for both variants of the problem without release dates and we prove that both variants become strongly $\mathcal{NP}$-hard in the presence of arbitrary release dates. Moreover, we show that, for arbitrary release dates, there is no $O(1)$-competitive online algorithm for the budget variant and we propose a 2-competitive one for the aggregated variant.

1 Introduction

We consider the problem of scheduling a set of jobs, each one associated with a release date, a due date and an amount of work, to be executed *non-preemptively* on a single *speed-scalable* processor in order to minimize the *maximum lateness*. The *lateness* of a job is defined as the difference between its completion time and its due date. In general, high processor's speeds imply low maximum lateness at the price of high energy consumption. Different approaches can be used in order to find a tradeoff between these two conflicting objectives. The one considered in the seminal paper of Yao et al. [13], consists of fixing the maximum lateness to zero and minimizing the energy consumption. They proposed a polynomial time algorithm for the *preemptive* single-processor case. Here, we study two variants of our problem: In the *budget variant*, we aim in minimizing the maximum lateness for a fixed budget of energy, while in the *aggregated variant*, our objective is to minimize a linear combination of maximum lateness and energy.

⋆ Research supported by the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010, and by GDR du CNRS, RO.
⋆⋆ Supported by the European Social Fund and Greek national resources (Program: Heracleitus II).

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 25–36, 2012.

Formally, we denote a set of n jobs by $J = \{1, 2, \ldots, n\}$. Each job i is associated with a release date r_i, a due date d_i and a work w_i. Whenever the speed-scalable processor runs at speed s, it consumes power $P(s) = s^\alpha$, where $\alpha > 2$, and it executes an amount of work w in w/s time units. Then, the consumed energy is $E = w \cdot s^{\alpha-1}$. For a given schedule the lateness of job i is defined as $L_i = C_i - d_i$, where C_i is the completion time of job i and the maximum lateness is defined as $L_{\max} = \max_{1 \leq i \leq n}\{L_i\}$. For convenience, we adopt an equivalent and more intuitive model where every job i is associated with a delivery time $q_i = -d_j + K \geq 0$, where K is a sufficiently large constant, instead of its due date. The delivery time represents the time needed for job i to be delivered after its completion assuming that the delivery of different jobs may take place simultaneously [7]. Now, the lateness of job i becomes $L_i = C_i + q_i$. Jobs that attain the maximum lateness in a schedule are referred as *critical* jobs. Extending the standard field notation of [9], the budget variant of our problem is denoted by $S1 \mid r_j \mid L_{\max}(E)$, while the agregated variant is denoted by $S1 \mid r_j \mid L_{\max} + \beta E$, where $\beta \geq 0$ and $S1$ stands for a single scalable processor.

Related Work and Our Results. As already mentioned, Yao et al. [13], proposed a polynomial time algorithm for the preemptive single-processor energy minimization problem and they also considered its online version.

Bunde [5] studied the budget variant of the *non-preemptive* makespan minimization problem for the single-processor as well as the multiprocessor case with jobs of unit work. He also proved the $\mathcal{NP}$-hardness of the multiprocessor case whenever the jobs have arbitrary works. Pruhs et al. [12] studied the budget variant of the *non-preemptive* multiprocessor makespan minimization problem in the presence of precedence constraints, and proposed an approximation algorithm. They also gave a PTAS for the case with no precedence constraints.

Albers et al. [3] studied online and offline versions of an aggregated variant of the *non-preemptive* problem of minimizing the total flow time of unit work jobs on a single-processor. The flow time of a job is defined as the difference between its completion time and its release date. Note that Pruhs et al. [11] have studied the offline version of the budget variant of this problem. Bansal et al. [4] proved that there is no $O(1)$-competitive algorithm, for the budget variant, even if all jobs have unit works. The interested reader may find recent reviews in [1,2,8].

Our results for the power-aware maximum lateness problem are organized as follows. For the budget variant we propose, in Section 2, an optimal algorithm for the *non-preemptive* single-processor case without release dates. We also prove that the problem, in the presence of release dates, becomes strongly $\mathcal{NP}$-hard and it does not admit an $O(1)$-competitive algorithm. In Section 3, we move to the aggregated variant, and we give an optimal algorithm for the single-processor problem without release dates and a strongly $\mathcal{NP}$-hardness proof for arbitrary release dates. Moreover, we propose a 2-competitive algorithm for the online case.

2 Budget Variant

In this section we present a polynomial algorithm for the $S1 \mid\mid L_{max}(E)$ problem. Our algorithm is based on a number of structural properties of an optimal schedule deduced by formulating our problem as a convex program and applying the KKT (Karush, Kuhn, Tucker) conditions. Next, we consider the same problem with release dates, i.e., $S1 \mid r_j \mid L_{max}(E)$, and we prove that it becomes strongly $\mathcal{NP}$-hard. Moreover, we show that there is no $O(1)$-competitive algorithm for its online version.

2.1 The Problem without Release Dates

A Convex Programming Formulation. A convex programming formulation of our problem stems from two basic properties of an optimal schedule. First, because of the convexity of the speed to power function, each job i runs at a constant speed s_i. Second, jobs are scheduled according to the EDD (Earliest Due Date First) rule, or equivalently in non-increasing order of their delivery times; this can be easily shown by a standard exchange argument. Hence, we propose the following formulation where all jobs are considered to be released at time zero and numbered according to the EDD order:

$$\min L$$

$$C_i + q_i \leq L \qquad 1 \leq i \leq n \tag{1}$$

$$\frac{w_1}{s_1} \leq C_1 \tag{2}$$

$$C_{i-1} + \frac{w_i}{s_i} \leq C_i \qquad 2 \leq i \leq n \tag{3}$$

$$\sum_{i=1}^{n} w_i s_i^{\alpha-1} \leq E \tag{4}$$

$$L, C_i, s_i \geq 0 \qquad 1 \leq i \leq n \tag{5}$$

Our objective is to minimize the maximum lateness, L, among all feasible schedules. Constraints (1) ensure that the lateness of each job is at most L, constraints (2) and (3) enforce the jobs to be scheduled according to the EDD rule in non-overlapping time intervals, constraint (4) does not allow to exceed the given energy budget E and constraints (5) ensure that the maximum lateness, the completion times and the speeds of jobs are non-negative. Constraint (4), for $\alpha > 2$, is convex while all other constraints and the objective function are linear.

This convex program already implies a polynomial algorithm for our problem, as convex programs can be solved to arbitrary precision by the Ellipsoid algorithm [10]. Since the Ellipsoid algorithm is rather impractical, we will exploit this convex program to derive a fast combinatorial algorithm.

Properties of an Optimal Schedule. In what follows we deduce a number of structural properties of an optimal schedule by applying the KKT conditions to the above convex program. Note that these properties can be also derived through exchange arguments based on the convexity of the speed to power function. We associate to each set of constraints from (1) up to (4), dual variables $\beta_i, \gamma_1, \gamma_i, \delta$, respectively. W.l.o.g. the variables L, C_i and s_i are positive and, by the complementary slackness conditions, the dual variables associated to the constraints (5) are equal to zero.

Proposition 1. *By the KKT conditions, the following hold for the primal and dual variables of the above convex program:*

$$\sum_{i=1}^{n} \beta_i = 1 \quad (6) \qquad \beta_i(C_i + q_i - L) = 0,\ 1 \leq i \leq n \quad (10)$$

$$\beta_i = \gamma_i - \gamma_{i+1},\ 1 \leq i \leq n-1 \quad (7) \qquad \gamma_1(\tfrac{w_1}{s_1} - C_1) = 0 \quad (11)$$

$$\beta_n = \gamma_n \quad (8) \qquad \gamma_i(C_{i-1} + \tfrac{w_i}{s_i} - C_i) = 0,\ 2 \leq i \leq n \quad (12)$$

$$(\alpha - 1)\delta = \tfrac{\gamma_i}{s_i^{\alpha}} \quad (9) \qquad \delta\left(\sum_{i=1}^{n} w_i s_i^{\alpha-1} - E\right) = 0 \quad (13)$$

Proof. Stationarity conditions give that

$$\nabla L + \sum_{i=1}^{n} \beta_i \nabla(C_i + q_i - L) + \gamma_1 \nabla(\frac{w_1}{s_1} - C_1)$$
$$+ \sum_{i=2}^{n} \gamma_i \nabla(C_{i-1} + \frac{w_i}{s_i} - C_i) + \delta \nabla(\sum_{i=1}^{n} w_i s_i^{a-1} - E) = 0 \ \Rightarrow$$
$$(1 - \sum_{i=1}^{n} \beta_i)\nabla L + \sum_{i=1}^{n-1} (\beta_i - \gamma_i + \gamma_{i+1})\nabla C_i$$
$$+ (\beta_n - \gamma_n)\nabla C_n + \sum_{i=1}^{n} (-\gamma_i w_i s_i^{-2} + (a-1)\delta w_i s_i^{a-2})\nabla s_i = 0$$

which implies the equations (6)-(9). The equations (10)-(13), follow by the complementary slackness conditions applied to constraints (1)-(4).

Proposition 1 leads to a number of structural properties of an optimal schedule, which are summarized in the following lemma.

Lemma 1. *For the maximum lateness problem with an energy budget E, there is always an optimal schedule that satisfies all the following properties.*
(i) Each job i runs at a constant speed s_i.
(ii) Jobs are scheduled according to the EDD rule.
(iii) There are no idle periods in the schedule.
(iv) The last job is critical, i.e., $L_n = L_{max}$.
(v) Every non-critical job i has equal speed with the job $i+1$, i.e., $s_i = s_{i+1}$.

(vi) Jobs are executed in non-increasing speeds, i.e., $s_i \geq s_{i+1}$.
(vii) All the energy budget is consumed.

Proof. (i)-(ii) They have been already discussed above.

(iii) First, note that $\delta \neq 0$. If $\delta = 0$ then by (9), we get that $\gamma_i = 0$ for each $1 \leq i \leq n$. This, combined with (7) and (8) yields that $\sum_{i=1}^{n} \beta_i = 0$, which is a contradiction because of (6). Since $\delta \neq 0$, we get by (9) that $\gamma_i \neq 0$ for each $1 \leq i \leq n$. Then, equations (11) and (12) give that there is no idle time in any optimal schedule since $C_1 = \frac{w_1}{s_1}$ and $C_i = C_{i-1} + \frac{w_i}{s_i}$, for $2 \leq i \leq n$, respectively.

(iv) Since $\delta \neq 0$, by (9), it follows that $\gamma_n \neq 0$ and finally, because of (8), $\beta_n \neq 0$. So, the last job to finish is always a critical job, by (10).

(v) Note that for every non-critical job i, it holds that $C_i + q_i < L$ and (10) implies that $\beta_i = 0$ for every such job. Hence, if a job i is non-critical $\beta_i = 0 \Rightarrow \gamma_i = \gamma_{i+1} \Rightarrow s_i = s_{i+1}$, by (7) and (9), respectively.

(vi) By the dual feasibility conditions and the equations (7) and (9) we get, respectively, that $\beta_i \geq 0 \Rightarrow \gamma_i \geq \gamma_{i+1} \Rightarrow s_i \geq s_{i+1}$. Thus, the jobs are executed with non-increasing speeds.

(vii) If the energy budget is not entirely consumed, then by (13), $\delta = 0$, which is a contradiction, since, as we have already proved, $\delta \neq 0$.

We refer to any schedule satisfying the properties of Lemma 1 as a *regular* schedule. By (i, j) we denote a sequence of consecutive jobs $i, i+1, \ldots, j$. Any regular schedule can be partitioned into groups of jobs, of the form (i, j), where the jobs $i-1$ and j are critical and the jobs $i, i+1, \ldots, j-1$ are not. By Lemma 1(v), all jobs of such a group are executed at the same speed. We denote this common speed by s_j and the total amount of work of jobs in (i, j) by $w(i, j) = \sum_{k=i}^{j} w_k$. Then, the next proposition follows easily from Lemma 1.

Proposition 2. *Let* i, j, *be two consecutive critical jobs of a regular schedule. The speed of each job in the group* $(i+1, j)$ *equals to* $s_j = \frac{w(i+1,j)}{q_i - q_j}$.

Up to this point, we have shown that there exists a regular schedule which is optimal. By the next lemma, these properties are also proved sufficient for optimality as, in fact, there is a unique regular schedule.

Lemma 2. *For a given budget of energy, there exists a unique regular schedule for the* $S1 \mid\mid L_{max}(E)$ *problem.*

An Optimal Combinatorial Algorithm. So far, we have derived a clear image of the structure of the unique regular optimal schedule for the $S1 \mid\mid L_{max}(E)$ problem. Next, we propose Algorithm BUD which constructs this schedule in polynomial time. Note that a regular schedule is fully specified by the speeds of the jobs. The rough idea of our algorithm is the following: First, it constructs a preliminary schedule by finding groups of jobs running in non-increasing speeds without taking care of the energy consumption. Second, the algorithm manages the energy consumption w.r.t. the energy budget E and determines the final

speeds of all jobs. Let E' be the energy consumption of the current schedule at any point of the execution of the algorithm.

Algorithm BUD starts from job n which is always a critical job and considers all jobs, but the first, in reverse order (note that Proposition 2 does not apply for the first job). When a job i, $2 \leq i \leq n$, is considered for the first time, its speed s_i is set according to Proposition 2, assuming that jobs $i-1$ and i are critical. If $s_i \geq s_j$, for $i+1 \leq j \leq n$, then s_i is called *eligible* speed and it is assigned to job i. If this speed is not eligible, i is a non-critical job and it is merged with the $(i+1)$'s group. More specifically, if c is the last job of this group, then the speeds of jobs $i, i+1, \ldots, c$ are calculated by applying Proposition 2, assuming that $i-1$ and c are critical while $i, i+1, \ldots, c-1$ are not. Next, the algorithm examines whether the new value of s_i is eligible. If this is the case, then it considers the job $i-1$. Otherwise, a further merging, of the i's group with the $(c+1)$'s group, is performed, as before. That is, if c' is the last job of the $(c+1)$'s group, all jobs $i, i+1, \ldots, c'$ are assigned the same speed assuming that jobs $i-1$ and c' are critical, while $i, i+1, \ldots, c'-1$ are not. This speed, according to the Proposition 2, is equal to $s(i,c') = \frac{w(i,c')}{q_{i-1}-q_{c'}}$. Note that the job c is no longer critical in this case. This merging procedure is repeated until job i is assigned an eligible speed. In a degenerate case, jobs $i, i+1, \ldots, n$ are merged into one group. When the algorithm has assigned an eligible speed to all jobs $2, 3, \ldots, n$, it sets $s_1 = s_2$ and its first part completes.

Next, Algorithm BUD takes into account the available budget of energy E. If $E - E' \geq 0$, the current schedule's energy consumption does not exceed the budget of energy, and the surplus $E - E'$ is assigned to the first job. Otherwise, the current schedule is regular, except that it consumes an amount of energy greater than E. Then, the algorithm reduces the consumed energy until it becomes equal to E. In fact, it decreases the speed of the first group, by merging groups with the first one if necessary. This merging procedure is different from the one of the first part of the algorithm and it is as follows: let i be the critical job of maximal index with $s_i = s_1$ in the current schedule. Observe that $s_i > s_{i+1}$. The algorithm sets the speed of jobs $1, 2, \ldots, i$ equal to s_{i+1}. This causes a reduction to E' and there are two cases to distinguish: either $E' \leq E$ or $E' > E$. In the first case, the algorithm adds an amount of energy $E - E'$ to jobs $1, 2, \ldots, i$ by increasing their speeds uniformly, i.e. so that they are all executed with the same speed. In the second case, at least one further merging step has to be performed. When the algorithm terminates, it is obvious that $E' = E$.

Theorem 1. *Algorithm BUD is optimal for the* $S1 \mid\mid L_{max}(E)$ *problem.*

Proof. We shall prove that the algorithm satisfies the properties of Lemma 1, i.e., it produces a regular schedule. For convenience, we distinguish two parts in the algorithm: *Part I*, corresponding to lines 1-6 and *Part II*, corresponding to lines 7-16, respectively.

Property (i)-(ii): The algorithm gives a single constant speed to each job and keeps their initial EDD order.

Property (iii): In Part I, the speeds of jobs are assigned according to Proposition 2. Specifically, the algorithm fixes two consecutive critical jobs i and j,

Algorithm 1. BUD

```
1: Sort the jobs according to the EDD order.
2: for i = n to 2 do
3:    Set s_i assuming that i and i − 1 are critical.
4:    while s_i is not eligible do
5:       Merge the i's group with the next group.
6: Set s_1 = s_2
7: Let E' be the current energy consumption.
8: if E > E' then
9:    Assign energy E − E' to job 1.
10: else
11:    while E < E' do
12:       Set the speed of the first group equal to the speed of the following group.
13:       Update E'.
14:       if E < E' then
15:          Merge the first group with the next one.
16:    Assign E − E' energy uniformly to the first group.
```

$i < j$, with, potentially, some non-critical jobs between them. Then the speed of the non-critical jobs and the one of critical job j is defined such that there is no idle between the jobs. In Part II, no idle period is added between any jobs.

Property (iv) - (v): When the speed of job n is initialized, this is done by assuming that it is critical. Next, consider the current schedule just after the completion of Part I. This schedule can be partitioned into sequences of jobs, $a+1, a+2, \ldots, b$, with $a \geq 1$, such that the jobs of each sequence are executed with the same speed which has been assigned by applying Proposition 2, assuming that the jobs a and b are critical. In fact, jobs a and b attain equal lateness. In order for such a sequence to be a group, we should also prove that all but the last jobs are non-critical while the last job is critical.

Let $a+1, a+2, \ldots, b$ be a sequence of jobs. We claim that $L_i < L_b$, for $a+1 \leq i \leq b-1$. Assume, by contradiction, that there exists a job j, where $a+1 \leq j \leq b-1$, such that $L_j \geq L_b$, or equivalently, $q_j - q_b \geq \sum_{i=j+1}^{b} \frac{w_i}{s_b}$. Since the last job of a sequence attains equal lateness with the last job of the sequence that follows, we have that $L_a = L_b$. This yields that $q_a - q_b = \sum_{i=a+1}^{b} \frac{w_i}{s_b}$. Therefore, $q_a - q_j \leq \sum_{i=a+1}^{j} \frac{w_i}{s_b}$.

Obviously, for any job i, $a+1 \leq i \leq b-1$, we must have a speed $s_i > \frac{w_i}{q_{i-1}-q_i}$, since otherwise, it wouldn't have been merged with another group. That is, $q_{i-1} - q_i > \frac{w_i}{s_i}$. If we sum the last inequalities for $a+1 \leq i \leq j$, we get that $q_a - q_j > \sum_{i=a+1}^{j} \frac{w_i}{s_b}$, a contradiction.

At this point, we have showed that when Part I completes, if a job i, $2 \leq i \leq n$, is critical, then it must be the right extremity of a sequence. Moreover, among all jobs $2, 3, \ldots, n$, the last jobs of all sequences, including job n, attain equal lateness and the remaining jobs attain smaller lateness. In addition, job 1 attains equal lateness with the last job of the sequence that follows. Recall that, at this point, we set $s_1 = s_2$. Job 1 would have equal lateness with the last job of the

sequence that follows for any $s_1 > 0$ since the speed of the second group is set by applying Proposition 2, assuming that 1 is critical. So, at the end of Part I, job 1, job n and every last job of a sequence are critical. Therefore, after Part I finishes, Properties (iv) and (v) hold.

In Part II, if no merging step is performed, then the processing time of job 1 is decreased by some $t \geq 0$ and its lateness decreases by t, while the processing times and speeds of the other jobs are not modified. So, the lateness of every other job also decreases by t. Hence, the Properties (iv) and (v) hold.

If at least one merging step is performed, then the speed of the jobs in the first group decreases and their processing time increases. Then, in the first group, every non-critical job i has equal speed with the job $i+1$ that follows, while the speeds of the jobs in other groups remain unchanged. Now, let t_i be the total increase in the processing time of job i, $1 \leq i \leq n$. Note that this quantity is positive only for jobs belonging to the first group of the current schedule. Then, the lateness of any job i, $1 \leq i \leq n$, increases by $\sum_{j=1}^{i} t_j$; if c_1 is the critical job of the first group, it remains critical after the merging step since its lateness and the lateness of every other job that follows, increases by the same quantity, equal to $\sum_{j=1}^{c_1} t_j$. Note, that if a further merging step is performed, we consider the first two groups as one group. Moreover, the lateness of any job increases by no more than the increase of the lateness of job n, and thus, in the final schedule, job n remains critical and Property (iv) holds. Furthermore, each non-critical job has equal speed with the job that follows and Property (v) holds as well.

Property (vi): At the end of Part I, the speeds of jobs are non-increasing since otherwise, a merging step would be performed. Moreover, during Part II, no speed of a job becomes less than the speed of a subsequent job.

Property (vii): Recall that E' is the total energy consumed when Part I completes. If E' is less than the energy budget, then the energy of the first job increases until the schedule consumes exactly E units of energy, while if E' is greater than the energy budget E, then the energy consumption of the schedule is gradually decreased until it becomes equal to E.

Let us now consider the complexity of the algorithm. Initially, jobs are sorted according to the EDD rule in $O(n \log n)$ time. The first part of the algorithm may take $O(n^2)$ time since each merging step takes $O(n)$ time and there can be $O(n)$ merging steps. Also, the algorithm's second part takes $O(n^2)$ time since the speed of each job may change at most $O(n)$ times. Therefore, the overall complexity of the algorithm is $O(n^2)$. Using a more careful analysis, based on the use of a stack data structure, it can be shown that the algorithm may be implemented in $O(n \log n)$ time.

2.2 The Problem with Release Dates

We now consider the $S1 \mid r_j \mid L_{max}(E)$ problem where the jobs have arbitrary release dates and we show that it becomes strongly $\mathcal{NP}$-hard.

The $\mathcal{NP}$-hardness of $S1 \mid r_j \mid L_{max}(E)$ is established through a reduction from the 3-PARTITION problem which is known to be strongly $\mathcal{NP}$-hard [6]. Our reduction is inspired by the $\mathcal{NP}$-hardness proof for the classical $1 \mid r_j \mid L_{max}$

problem [6]. In the latter problem, we are given a set of jobs where each job i has a release date r_i, a due date d_i, a processing time p_i and we seek a schedule minimizing the maximum lateness. This problem can be viewed as a variant of our problem where the speed of each job is part of the instance. Specifically, we consider that each job i has an amount of work $w_i = p_i$ and it is executed at a constant speed $s_i = 1$. Based on this idea, we extend the existing $\mathcal{NP}$-hardness reduction by fixing an energy budget, so that all jobs have to be executed at the same speed $s_i = 1$ in order to get a feasible schedule.

Theorem 2. *$S1 \mid r_j \mid L_{max}(E)$ problem is strongly $\mathcal{NP}$-hard.*

Let us now turn our attention to the online version of the $S1 \mid r_j \mid L_{max}(E)$ problem. Bansal et al. [4] gave an adversarial strategy for proving that there is no $O(1)$-competitive algorithm for the problem of minimizing the total flow of a set of unit work jobs on a single speed-scalable processor. Following the same strategy it can be proved that the makespan minimization problem, for a given budget of energy, i.e., $S1 \mid r_j,\ w_j = 1 \mid C_{max}(E)$, does not admit an $O(1)$-competitive algorithm. As the latter problem is a special case of our lateness problem (with $q_i = 0,\ 1 \leq i \leq n$), the next theorem follows.

Theorem 3. *There is no $O(1)$-competitive algorithm for the online version of the $S1 \mid r_j \mid L_{max}(E)$ problem, even when jobs have unit works.*

3 Aggregated Variant

In this section we deal with the aggregated variant of the maximum lateness problem, where the objective is the minimization of a linear combination of maximum lateness and energy, i.e., $L_{\max} + \beta E$, where $\beta \geq 0$. In the case with no release dates, we present an optimal combinatorial algorithm. When the jobs are subject to release dates, we prove that the offline problem becomes strongly $\mathcal{NP}$-hard and we present a 2-competitive algorithm for its online version.

3.1 The Problem with No Release Dates

In order to derive an optimal algorithm for the $S1 \mid \mid L_{\max} + \beta E$ problem, we follow the same line as for the budget variant: by formulating the problem as a convex program and applying the KKT conditions, the Properties (i)-(vi) of Lemma 1 will also hold while Property (vii) is replaced by "*(vii) The job executed first runs at speed* $s_1 = (\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$". Moreover, there is again a unique regular schedule and both the optimal algorithm and its analysis for the aggregated variant become simpler than those of the budget variant.

Description of the Algorithm. It can be shown that the regular schedule σ_{opt} for the $S1 \mid \mid L_{\max} + \beta E$ problem attains the same maximum lateness as the schedule σ_c, that executes all jobs at the same a constant speed $s = (\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$.

Algorithm 2. LAGER

1: Order jobs according to the EDD order.
2: Assign to each job the speed $(\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$.
3: Let k be the highest-index critical job in the current schedule.
4: **while** $k < n$ **do**
5: **for** $i = k$ to n **do**
6: Compute v_i assuming that k and i are consecutive critical jobs.
7: Set the speed of jobs $k, k+1, \ldots, n$ equal to $v_{max} = \max_{k \leq i \leq n}\{v_i\}$.
8: Let ℓ be the highest-index critical job in the current schedule.
9: $k = \ell$

This observation implies that if i is the highest-index critical job in σ_c, then, by Properties (v) and (vi), all jobs $1, 2, \ldots, i$ are executed at this speed s in σ_{opt}. Based on this fact, we proceed to the description of our algorithm.

In the first step, the algorithm assigns to every job i a speed s_i equal to $(\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$. In this way, we identify the value of the maximum lateness and the set of jobs executed with speed $(\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$ in the optimal schedule. This can be done by determining the highest-index critical job k in σ_c. All jobs $1, 2, \ldots, k$ are executed with speed $(\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$ in σ_{opt}. Moreover, all jobs with index greater than k have lateness strictly less than the maximum lateness of the optimal schedule. Therefore, we can decrease their speeds in order to reduce their energy consumption without affecting the maximum lateness of the schedule. This is done as follows: At the beginning, the algorithm has already assigned a speed to jobs $1, 2, \ldots, k$. For each job i, $k+1 \leq i \leq n$, the algorithm defines a *candidate speed* of i, which we denote v_i. This speed is such that job i becomes critical given that k is critical and all jobs $k+1, k+2, \ldots, i$ are executed at the same speed. By Proposition 2, $v_i = \frac{1}{q_k - q_i} \sum_{j=k+1}^{i} w_j$. Then, among the candidate speeds, we choose the maximum one $v_{\max} = \max_i\{v_i\}$ and let ℓ be the job with the highest index, with $v_l = v_{max}$. We set the speed of jobs $k+1, k+2, \ldots, \ell$ equal to v_ℓ. Then, we set $k = \ell$ to be the highest index critical job in the current schedule and we proceed to the next step. The algorithm terminates when job n becomes critical. The optimality of the algorithm follows by induction on the number of its steps and its complexity is $O(n^2)$, since each iteration of the while loop takes time at most $O(n)$.

3.2 The Problem with Release Dates

When jobs have arbitrary release dates, then the problem becomes strongly $\mathcal{NP}$-hard, as for the budget variant. The reduction is again inspired by the $\mathcal{NP}$-hardness proof for the classical $1 \mid r_j \mid L_{max}$ [6] and uses a lower bound on the objective of any optimal schedule.

The Online Case. Let us now present a 2-competitive online algorithm for the $S1 \mid r_j \mid L_{max} + \beta E$ problem. The algorithm schedules the jobs in a number of phases by repeatedly applying the optimal offline algorithm LAGER for the $S1 \mid \mid L_{max} + \beta E$ problem. We denote by $\sigma^*(J, t)$ the optimal schedule of a set of jobs J with a common release date t.

Algorithm ALE. Let J_0 be the set of jobs that arrive at time $t_0 = 0$. In phase 0, jobs in J_0 are scheduled according to $\sigma^*(J_0, 0)$. Let t_1 be the time at which the last job of J_0 is finished, i.e., the end of phase 0, and J_1 be the set of jobs released during $(t_0, t_1]$. In phase 1, jobs in J_1 are scheduled as in $\sigma^*(J_1, t_1)$ and so on. In general, if t_i is the end of phase $i-1$, we denote J_i to be the set of jobs released during $(t_{i-1}, t_i]$. Jobs in J_i are scheduled by computing $\sigma^*(J_i, t_i)$. Next, we analyze the competitive ratio of the algorithm.

Theorem 4. *Algorithm ALE is 2-competitive for the online version of the* $S1 \mid r_j \mid L_{max} + \beta E$ *problem.*

Proof. Assume that Algorithm ALE produces a schedule in $\ell + 1$ phases. Recall that the jobs of the i-th phase, i.e., the jobs in J_i, are released during $(t_{i-1}, t_i]$ and scheduled as in $\sigma^*(J_i, t_i)$. Let $L_{max,i} + \beta E_i$ be the cost of $\sigma^*(J_i, t_i)$, where $L_{max,i}$ is the maximum lateness among the jobs in J_i and E_i be the energy consumed by the jobs of J_i. The objective value of the algorithm's schedule is

$$SOL = \max_{0 \le i \le \ell} \{L_{max,i}\} + \beta \sum_{i=0}^{\ell} E_i \tag{14}$$

Now, we consider the optimal schedule. To lower bound the objective value OPT of an optimal schedule, we round down the release dates of the jobs; the release dates of the jobs in phase i, are rounded down to t_{i-1}. Let σ_d^* and OPT_d be the optimal schedule for the rounded instance and its cost, respectively. Clearly, any feasible schedule for the initial instance is also feasible for the rounded one. Thus, $OPT \ge OPT_d$.

To lower bound OPT_d we consider a restricted speed-scaling scheduling problem, i.e., a problem where each job can only be executed by a subset of the available processors. The instance of this problem consists of $\ell + 1$ available speed-scalable processors $M_0, M_1, \ldots, M_\ell$ and the set of jobs J, with their release dates rounded down, as before. Jobs in J_0 can only be assigned to the processor M_0 and every job in J_i can only be executed by one of the processors M_0 or M_i, $1 \le i \le \ell$. Moreover, it is required that all jobs in J_i, $0 \le i \le \ell$, are executed by the same processor. Let σ_m^*, OPT_m be the optimal schedule and its cost, respectively, for this restricted problem. Obviously, $OPT_d \ge OPT_m$ since σ_d^* is feasible for the restricted scheduling problem.

Let us now describe an optimal schedule σ_m^*. Through a simple exchange argument, it can be shown that the jobs of J_i, $0 \le i \le \ell$, in an optimal schedule, are executed by the processor M_i. Moreover, jobs in J_i, for $1 \le i \le \ell$, are scheduled according to $\sigma^*(J_i, t_{i-1})$, while for $i = 0$, according to $\sigma^*(J_0, t_0)$.

Assume that the maximum lateness of the above schedule, is attained by a job of the set J_k, $0 \leq k \leq \ell$, in the processor M_k. So, let $L^*_{max} = L^*_{max,k}$, where L^*_{max}, $L^*_{max,k}$ is the maximum lateness of the schedules $\sigma^*_m, \sigma^*(J_i, t_{i-1})$, respectively. Let E^*_i be the energy consumption of schedule $\sigma^*(J_i, t_{i-1})$. Then,

$$OPT_m = L^*_{max,k} + \beta \sum_{i=0}^{\ell} E^*_i \tag{15}$$

By considering the schedules $\sigma^*(J_i, t_{i-1})$ and $\sigma^*(J_i, t_i)$, it can be easily shown that $L^*_{max,i} = L_{max,i} - (t_i - t_{i-1})$ and $E^*_i = E_i$. Then, by (14) and (15) it yields that $OPT_m = SOL - (t_k - t_{k-1})$. Note that $t_k - t_{k-1}$ is the total processing time of the jobs in J_{k-1}, in the schedule produced by ALE, which is equal to the processing time of the jobs in J_{k-1} in σ^*_m. Recall also that the last job of each set J_i attains $L_{max,i}$. Thus, $t_k - t_{k-1} \leq L^*_{max,k-1} \leq OPT$. Therefore, $SOL \leq 2OPT$ and Algorithm ALE is 2-competitive for the $S1 \mid r_j \mid L_{max} + \beta E$ problem.

References

1. Albers, S.: Energy-efficient algorithms. Communications of ACM 53(5), 86–96 (2010)
2. Albers, S.: Algorithms for dynamic speed scaling. In: Symposium on Theoretical Aspects of Computer Science, pp. 1–11 (2011)
3. Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. ACM Transactions on Algorithms 3(4), 49 (2007)
4. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. SIAM Journal on Computing 39(4), 1294–1308 (2009)
5. Bunde, D.P.: Power-aware scheduling for makespan and flow. Journal of Scheduling 12(5), 489–500 (2009)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)
7. Hall, L.A.: Approximation algorithms for scheduling. In: Hochbaum, D.S. (ed.) Approximation Algorithms for NP-hard problems, pp. 1–45. PWS, Boston (1997)
8. Irani, S., Pruhs, K.: Algorithmic problems in power management. SIGACT News 36(2), 63–76 (2005)
9. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: Sequencing and scheduling: algorithms and complexity. In: Handbooks in Operations Research and Management Science, vol. 4, pp. 445–522, North Holland (1976)
10. Nemirovski, A., Nesterov, I., Nesterov, Y.: Interior Point Polynomial Algorithms in Convex Programming. Society for Industrial and Applied Mathematics (1994)
11. Pruhs, K., Uthaisombut, P., Woeginger, G.J.: Getting the best response for your erg. ACM Transactions on Algorithms 4 (2008)
12. Pruhs, K., van Stee, R., Uthaisombut, P.: Speed scaling of tasks with precedence constraints. Theory of Computing Systems 43, 67–80 (2008)
13. Yao, F.F., Demers, A.J., Shenker, S.: A scheduling model for reduced cpu energy. In: IEEE Symposium on Foundations of Computer Science, pp. 374–382 (1995)

Induced Subgraph Isomorphism: Are Some Patterns Substantially Easier Than Others?

Peter Floderus[1], Mirosław Kowaluk[2,⋆], Andrzej Lingas[3,⋆⋆], and Eva-Marta Lundell[3]

[1] The Centre for Mathematical Sciences, Lund University, 22100 Lund, Sweden
Peter.Floderus@maths.lth.se
[2] Institute of Informatics, Warsaw University, Warsaw, Poland
kowaluk@mimuw.edu.pl
[3] Department of Computer Science, Lund University, 22100 Lund, Sweden
{Andrzej.Lingas,Eva-Marta.Lundell}@cs.lth.se

Abstract. The complexity of the subgraph isomorphism problem where the pattern graph is of fixed size is well known to depend on the topology of the pattern graph. For instance, the larger the maximum independent set of the pattern graph is the more efficient algorithms are known. The situation seems to be substantially different in the case of induced subgraph isomorphism for pattern graphs of fixed size. We present two results which provide evidence that no topology of an induced subgraph of fixed size can be easier to detect or count than an independent set of related size. We show that:

- Any fixed pattern graph that has a maximum independent set of size k that is disjoint from other maximum independent sets is not easier to *detect* as an induced subgraph than an independent set of size k. It follows in particular that an induced path on k vertices is not easier to detect than an independent set on $\lceil k/2 \rceil$ vertices, and that an induced even cycle on k vertices is not easier to detect than an independent set on $k/2$ vertices. In view of linear time upper bounds on induced paths of length three and four, our lower bound is tight. Similar corollaries hold for the detection of induced complete bipartite graphs and induced complete split graphs.
- For an arbitrary pattern graph H on k vertices with no isolated vertices, there is a simple subdivision of H, resulting from splitting each edge into a path of length four and attaching a distinct path of length three at each vertex of degree one, that is not easier to *detect* or *count* than an independent set on k vertices, respectively.

Finally, we show that the so called diamond, paw and C_4 are not easier to detect as induced subgraphs than an independent set on three vertices.

⋆ Research supported by the grant no. \ N206 566740 of the National Science Center.
⋆⋆ Research supported in part by Swedish Research Council grant 621-2008-4649.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 37–48, 2012.

1 Introduction

The *induced subgraph isomorphism* problem is to detect if a host graph has an induced subgraph that is isomorphic to a pattern graph. Its counting variant asks for the number of induced subgraphs of the host graph isomorphic to the pattern graph. The well known independent set and clique problems are special cases of the induced subgraph isomorphism problem which consequently is generally NP-complete [8]. When the pattern graph is of fixed size, induced subgraph isomorphism can be solved in polynomial time even by exhaustive search.

In the literature, there are only a few examples of pattern graphs of fixed size k for which the induced subgraph isomorphism admits lower asymptotic time upper bound in terms of the number of vertices of the host graph than those known for the k-clique problem (for general host graphs). The oldest and most striking example is P_4, a path on four vertices, which can be detected in $O(n + m)$ time, where n, m stand for the number of vertices and edges in the host graph [4]. The other is P_3, the path on three vertices which can be detected in $O(n + m)$ time [17], the third example is the diamond, obtained by removing a single edge from K_4, which can be detected in $O(n + m^{3/2})$ time [6] (cf. [12]). The fourth example is a paw which is a triangle connected to the fourth vertex by an edge, i.e., $K_3 + e$. It can be detected in $O(n^{2.376})$ time (In fact, by considering the complement graph, the analogous bounds hold for the pattern graphs consisting of two adjacent vertices and one or two isolated vertices, or two incident edges and one isolated vertex, respectively.) In comparison, K_3 and K_4 can be detected and counted in $O(n^{2.376})$ time [11] and $O(n^{3.334})$ time [6], respectively. Interestingly, such a gap between the time upper bounds for P_4 and K_4, for P_3 and K_3, and for the diamond and K_4, respectively, is not possible in the counting variant by [12,13].

In the extreme case, when the pattern graph is a set of k isolated vertices, the induced subgraph isomorphism is equally hard as the k-clique problem if the time complexity is a function of the number of vertices. (This is in sharp contrast with the general subgraph isomorphism problem which for the aforementioned pattern becomes trivial.) Therefore, we can naturally pose the following conjecture:
There exists a constant C such that the time complexity of the problems of detecting (counting) the induced subgraphs isomorphic to a given k-vertex pattern graph in an n-vertex host graph is lower bounded by that of detecting (counting) the induced subgraphs of an n-vertex graph isomorphic to an independent set on k/C vertices.
In the counting variant, one could strongly conjecture $C = 1$ while in the detection variant the smallest value of C that one could conjecture is 2 in view of the result on P_4 [4].

By time complexity in the conjecture and throughout the paper, we mean the worst-case asymptotic time complexity in terms of the number n of vertices in the host graph under the assumption that the size of pattern graph is fixed. This allows for reductions of fixed independent set problems to induced fixed subgraph problems which are linear with respect to the number of vertices but

not necessarily preserve graph sparsity so their time complexity can be even quadratic in the number of vertices.

Importantly, we assume arbitrary host graphs in the conjecture. Otherwise, one can easily come up with examples of classes of host graphs for which the topology of the pattern graph on k vertices affects the complexity of induced subgraph isomorphism. E.g., counting independent sets of size k for $k \geq 5$ in a planar graph does not seem to be an easy task while counting occurrences of K_k for $k \geq 5$ in a planar graph is trivial.

A related conjecture would be to claim that the hardness of induced subgraph isomorphism depends on the maximum sizes of an independent set and a clique in the pattern graph.

In the context of our conjecture, let us recall that the problems of detecting an independent set on k vertices and detecting a clique on k vertices in a host graph on n vertices are known to be W[1]-hard in the theory of parametrized complexity and believed to require $n^{\Omega(k)}$ time [5].

Known Results Supporting the Conjecture. Already in 1985, Nešetřil and Poljak showed in [14] that the detection and counting versions of the induced subgraph isomorphism with fixed pattern graph on k vertices are easily reducible to the corresponding versions of the k-clique problem (or, equivalently, the k-independent set problem) in $O(kn^2)$ time, where n is the number of vertices of the host graph.

More recently, Chen and Flum [1] adapted the reduction of log clique to log chordless path due to Papadimitriou and Yannakakis [16] to show that detecting an induced path of length $4k-1$ is not easier than (i.e., its time complexity is lower bounded by that of) detecting an independent set on k vertices. They also showed that an induced cycle on $4k$, C_{4k}, is not easier to detect than an independent set on k vertices. A stronger result for C_5 showing that it is not easier to detect than K_3 (equivalently, $3K_1$) is a folklore. Also, it is easy to observe that the claw on four vertices, i.e., a graph consisting of three edges all sharing the same vertex, is not easier to detect than K_3.

In [12], Kloks, Kratsch and Müller showed that in the induced case if the occurrences of some pattern graph on 4 vertices can be counted in $T(n)$ time then the occurrences of any other pattern graph on 4 vertices can be counted in $O(n^\omega + T(n))$ time, where ω is the exponent of fast matrix multiplication known to be not greater than 2.376 [3]. Recently, Kowaluk et al. generalized the aforementioned result in [13] by showing that the knowledge of the number of occurrences of any pattern graph on k vertices as an induced subgraph is sufficient to compute the number of occurrences of any other pattern graph on k vertices both as induced and non-necessarily induced subgraph in time $O(n^{\omega(\lceil (k-2)/2 \rceil, 1, \lfloor (k-2)/2 \rfloor)})$, where $\omega(p,q,r)$ is the exponent of fast arithmetic matrix multiplication of an $n^p \times n^q$ matrix by an $n^q \times n^r$ matrix [2,10].

The aforementioned generalization is interesting solely for fairly small k in view of the following fact: the detection and counting versions of the induced subgraph isomorphism problem for k-vertex pattern graphs can be solved in time $O(n^{\omega(\lfloor k/3 \rfloor, \lceil (k-1)/3 \rceil, \lceil k/3 \rceil)})$ [6] (cf. [12,14]).

Table 1. Known upper time bounds for detecting induced subgraphs on 4 vertices in an undirected, unweighted graph on n vertices and m edges. The complement pattern graphs are given in parentheses.

subgraph	time complexity	reference
K_4 ($4K_1$)	$\mathcal{O}(n^{3.334})$ ($\mathcal{O}(m^{1.682})$)	Eisenbrand-Grandoni [6]
$K_4 \setminus e$ ($K_2 + 2K_1$)	$\mathcal{O}(m^{3/2})$	Eisenbrand-Grandoni [6]
C_4 ($2K_2$)	$\mathcal{O}(n^{3.334})$	Eisenbrand-Grandoni [6]
$K_3 + e$ ($P_3 + K_1$)	$\mathcal{O}(n^{\omega})$	Olariu [15]
$K_{1,3}$ ($K_3 + K_1$)	$\mathcal{O}(m^{(\omega+1)/2})$ $\mathcal{O}(n^{3.334})$	Kloks et al. [12] Eisenbrand-Grandoni [6]
P_4 (P_4)	$\mathcal{O}(n + m)$	Corneil et al. [4]

Examples of Surprisingly Fast Algorithms for Fixed Size Induced Subgraph Isomorphism. The most striking result is clearly that on detection of induced P_4 in $O(n + m)$ time [4].

The aforementioned $O(n + m)$-time algorithm for the detection of induced P_3 [17] and $O(n + m^{3/2})$-time algorithm for the detection of induced diamond have been generalized to an $O(n + m^{(k-1)/2})$-time algorithm for the detection of induced K_k with a single missing edge, denoted by $K_k \setminus e$, by Vassilevska in [17]. Observe that P_3 and the diamond can be denoted as $K_3 \setminus e$ and $K_4 \setminus e$, respectively. By considering the complement graph, we obtain also an analogous time bound $O(n^{k-1})$ for the detection of the pattern graph consisting of a pair of adjacent vertices and $k-2$ isolated vertices, denoted by $K_2 + (k-2)K_1$. The generalized upper bound $O(n^{k-1})$ for $K_k \setminus e$ and $K_2 + (k-2)K_1$ is however subsumed for $k > 5$ by that universal upper bound $O(n^{\omega(\lfloor k/3 \rfloor, \lceil (k-1)/3 \rceil, \lceil k/3 \rceil)})$ [6] valid for all pattern graphs on k vertices.

Furthermore, in a recent manuscript [9], the authors claim an analogous $O(n + m^{(k-1)/2})$-time bound for the problem of detection of induced path on k vertices, P_k.

Our Contributions. We present two main results on the hardness (i.e., the time complexity) of detecting and counting induced subgraphs of fixed size.

For *detection*, we provide a substantially more general and stronger result than those on chordless induced path and cycle from [1].

We show that any fixed pattern graph with a maximum independent set of size k that is disjoint from other maximum independent sets is not easier to detect as an induced subgraph than an independent set of size k. It follows in particular that an induced path on k vertices is not easier to detect than an independent set on $\lceil k/2 \rceil$ vertices and that an induced even cycle on k vertices is not easier to detect than an independent set on $k/2$ vertices. In view of the aforementioned results on P_3 and P_4 our lower bound is tight. We can also conclude that an induced complete bipartite graph $K_{p,q}$ is not easier to detect than an independent set on $\max\{p, q\}$ vertices. A similar corollary holds for complete split graphs.

Our second main result is concerned with both *detection* and *counting*. It can be regarded as a generalization of the aforementioned results on chordless induced path and cycle [1], basically showing that no pattern topology is easier to detect or count.

For an arbitrary pattern graph H on k vertices with no isolated vertices, let H' be the subdivision of H obtained from H by splitting each edge into a path of length four and attaching a distinct path of length three at each vertex of degree one. We show that H' is not easier to detect or count than an independent set on k vertices, respectively.

Finally, we show that the diamond, paw and C_4 are not easier to detect as induced subgraphs than an independent set on three vertices. Our result on C_4 resolves an open question from [7].

Organization. In the next section, we present our lower bound on detecting induced subgraphs isomorphic to restricted pattern graphs in terms of the size of independent set that is not easier to detect. In Section 3, we provide our lower bound on detecting and counting induced subgraphs isomorphic to restricted pattern graphs in terms of the size of independent set that is not easier to detect or count, respectively. In Section 4, we present simple lower bounds implying that the diamond, paw and C_4 are not easier to detect as induced subgraphs than a triangle. We conclude with final remarks.

2 Lower Bounds on Detecting Induced Subgraphs

Chen and Flum demonstrated the hardness of the induced path and induced cycle problems in [1]. We can state precisely their results as follows.

Fact 1. *Let G be an arbitrary graph on n vertices and m edges. In $O(kn^2+k^2m)$ time, one can construct a graph G' on $O(kn)$ vertices and $O(kn^2+k^2m)$ edges such that G' has an induced subgraph isomorphic to a path on $4k-1$ vertices iff G has an independent set of cardinality k.*

Similarly, one can construct a graph G'' on $O(kn)$ vertices and $O(kn^2+k^2m)$ edges such that G'' has an induced subgraph isomorphic to a cycle on $4k$ vertices iff G has an independent set of cardinality k.

In this section, we provide a general equivalence which works in case of detection for arbitrary pattern graphs with a maximum independent set disjoint from other maximum independent sets. It supports our conjecture if such a maximum independent set is relatively large. Our equivalence also subsumes that of Chen and Flum in the particular case of paths and cycles.

Theorem 1. *Let G be an arbitrary graph on n vertices and m edges, and let H be a pattern graph on h vertices. Suppose that there is a maximum independent set of size k that is disjoint from all other maximum independent sets in H. In $O(kn^2+hkn+k^2m)$ time, one can construct a graph G^* on $O(h+kn)$ vertices*

and $O(kn^2+hkn+k^2m)$ edges such that G^* has an induced subgraph isomorphic to H iff G has an independent set of cardinality k.

Proof. Let $G = (V, E)$ and $H = (V_H, E_H)$. Next, let S be a maximum independent set that is disjoint from the other maximum independent sets in H.

G^* consists of $k = |S|$ cliques $G^*(i)$ on $V \times \{i\}$, where $i \in S$, and the subgraph H' of H induced by all vertices in V_H outside S. (Note that $H \cap G^* = H'$.) Additionally, G^* contains the following edges between the k cliques and H'. Two vertices (v, i), (u, j) from two different cliques $G^*(i)$ and $G^*(j)$ form an edge if $\{v, u\} \in E$ or $v = u$. Each vertex l of H' is connected by an edge with each vertex of each clique $G^*(i)$, where $\{l, i\} \in E_H$ and $i \in S$. There are no other edges in G^*. See Fig. 1 for an example.

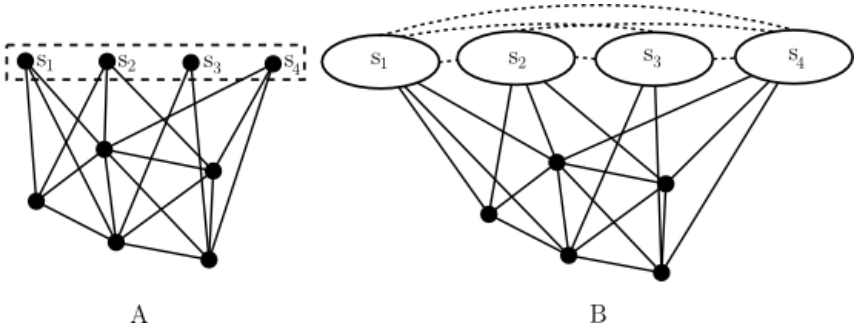


Fig. 1. An example of a pattern graph (A) with a maximum independent set disjoint from others marked and the corresponding graph G^* (B), where the large vertices represent cliques and the dotted lines symbolize the edge connection between them

Suppose that G has an independent set $\{v_1, v_2, ..., v_k\}$ on k vertices. Then, we map each vertex $i \in S$ on the vertex (v_i, i). Next, we map each vertex in $V_H \setminus S$ on itself. The image of H under this mapping is easily seen to induce a subgraph isomorphic to H in G^*.

Conversely, suppose that G^* has an induced subgraph H^* such that there is an isomorphism between H and H^*.

Consider a maximum independent set U of H^*. Let U'' be the subset of U outside of H' and let U' be the subset of U within H'.

Since U'' is an independent set, then its vertices are in disjoint cliques $G^*(i)$. Let U''_S be the set of i for which there is a node of U'' in $G^*(i)$.

Now consider H. Observe that $V_H \setminus S$ is the set of vertices of H'. $U' \subset V_H \setminus S$ and $U''_S \subset S$ together form a maximum independent set of H. It properly intersects S, which yields a contradiction, unless $U''_S = S$ or $U''_S = \emptyset$. In the former case, we are done.

It remains to consider the situation where for each maximum independent set U of H^*, $U''_S = \emptyset$. This would however mean that H^* has all its maximum independent sets in the common subgraph H' of G^* and H. Consequently, H

would have at least one more maximum independent set (S is outside H') than H^*. This would contradict the isomorphism between H^* and H.

□

Note that a path on $4k-1$ vertices as well as a cycle on $4k$ vertices have an independent set of cardinality $2k$. Paths as well as even cycles have at most two maximum independent sets, and they are always disjoint. Thus, Theorem 1 provides stronger lower bounds in terms of the size of independent set than Fact 1 in the particular case of induced paths and cycles.

Corollary 1. *If H is a fixed pattern graph with a maximum independent set of cardinality k which is disjoint from other maximum independent sets (e.g., a path on $2k+1$ or $2k$ vertices or a cycle on $2k$ vertices) then the asymptotic complexity of the detection of an induced subgraph isomorphic to H in terms of the number of vertices of the host graph is not less than that of an independent set on k vertices.*

It is folklore that the detection of K_3 can be easily reduced to that of claw, i.e. $K_{1,3}$, by considering the complement graph expanded by an auxiliary vertex connected to all vertices in the complement graph.

We obtain immediately the following much more general corollary from Theorem 1 and Corollary 1.

Corollary 2. *The time complexity of the detection of an induced subgraph isomorphic to the complete bipartite graph $K_{q,r}$ is lower bounded by that of an independent set on $\max\{q,r\}$ vertices.*

By a *complete split* graph, we shall mean a graph that can be decomposed into an independent set and a clique so it contains all possible edges between the vertices in the two subgraphs. By considering also the complement graphs, we obtain the next corollary from Theorem 1 and Corollary 1.

Corollary 3. *The time complexity of the detection of an induced subgraph isomorphic to the complete split graph with the independent part on q vertices and the clique part on r vertices is lower bounded by that of an independent set on $\max\{q,r\}$ vertices.*

3 Lower Bounds on Detecting and Counting Induced Subgraphs

We can also generalize the equivalence of Fact 1 to work not only for paths and cycles but for subdivisions of arbitrary pattern graphs without isolated vertices too, importantly both in case of detection and counting. The subdivisions replace each edge with a path with three additional inner vertices and attach an additional path at each vertex of degree one. Our next result basically shows that no pattern topology is easier to detect or count.

Theorem 2. *Let G be an arbitrary graph on n vertices and m edges, and let H be a pattern graph with h vertices and l edges and no isolated vertices. Next, let H_d be the subdivision of H obtained by placing three auxiliary vertices on each edge of H, and attaching at each leaf, i.e., vertex of degree 1, of H a distinct additional path of length three. In $O(hn^2 + ln + h^2m)$ time, one can construct a graph $G(h)$ on $O(hn)$ vertices and $O(hn^2+ln+h^2m)$ edges such that the induced subgraphs of $G(h)$ isomorphic to H_d are in one-to-one correspondence with the independent sets in G of cardinality h.*

Proof. We form a graph $G(h)$ which basically consists of h copies of a clique on V, linked according to G. The h copies are additionally linked via auxiliary vertices in one-to-one correspondence with the edges of H. Furthermore, a path on three additional vertices is attached to each clique copy that corresponds to a leaf of H.

Let $G = (V, E)$ and $H = (V_H, E_H)$, and let L be the set of leaves in H. The vertex set of the i-th clique copy is $V \times \{i\}$ for $i \in V_H$. The set $V(h)$ of vertices of $G(h)$ is the union of $V \times V_H$ with the sets $\{a_{ij}, b_{ij}, c_{ij}\}$, where i and j, $i < j$, are adjacent vertices of H, and the sets $\{a_i, b_i, c_i\}$, where i is a leaf of H.

The set $E(h)$ of edges of $G(h)$ is the union of the following edge sets (see Fig. 2 for an illustration): $\bigcup_{i \in V_H} \{\{(u,i),(v,i)\} | u, v \in V \& u \neq v\}$

$$\bigcup_{\{i,j\} \subset V_H} \{\{(u,i),(v,j)\} | i \neq j \& u, v \in V \& (u = v \vee \{u,v\} \in E)\}$$

$$\bigcup_{\{i,j\} \subset V_H} \{\{(u,i), a_{ij}\} | i < j \& (a_{ij}\ defined)\}$$

$$\bigcup_{\{i,j\} \subset V_H} \{\{a_{ij}, b_{ij}\} | (a_{ij}\ defined) \& (b_{ij}\ defined)\}$$

$$\bigcup_{\{i,j\} \subset V_H} \{\{b_{ij}, c_{ij}\} | (b_{ij}\ defined) \& (c_{ij}\ defined)\}$$

$$\bigcup_{\{i,j\} \subset V_H} \{\{c_{ij}, (v,j)\} | i < j \& (c_{ij}\ defined)\}$$

$$\bigcup_{i \in L} \{\{(u,i), a_i\}\} \quad \bigcup_{i \in L} \{\{a_i, b_i\}\} \quad \bigcup_{i \in L} \{\{b_i, c_i\}\}$$

Claim. *Suppose that an embedding ϕ of H_d in $G(h)$ satisfies the following three conditions:*

- *for $l \in V_H$, $\phi(l)$ is a vertex in $V \times \{l\}$,*
- *for any two adjacent vertices i and j of H, where $i < j$, ϕ maps the three vertices between i and j on the path onto the three vertices in $\{a_{ij}, b_{ij}, c_{ij}\}$ so to form a path $\{\phi(i), a_{ij}\}$, $\{a_{ij}, b_{ij}\}$, $\{b_{ij}, c_{ij}\}$, $\{c_{ij}, \phi(j)\}$,*
- *for any leaf of H_d, the path leading to the associated leaf i of H is mapped on $\{c_i, b_i\}$, $\{b_i, a_i\}$, $\{a_i, \phi(i)\}$.*

Then,
(a) $\phi(H_d)$ is a subgraph of $G(h)$ isomorphic to H_d,
(b) $\phi(H_d)$ is an induced subgraph in $G(h)$ iff $\bigcup_{l\in V_H}\{\phi_1(l)\}$ is an independent set in G, where $\phi_1(l)$ stands for the first coordinate of $\phi(l)$,
(c) each induced subgraph of $G(h)$ isomorphic to H_d can be defined as the image of such an embedding ϕ composed with an automorphism of H_d,
(d) if μ is another embedding of H_d in $G(h)$ satisfying the aforementioned conditions and both $\phi(H_d)$ and $\mu(H_d)$ are induced subgraphs of $G(h)$ then $\bigcup_{l\in V_H}\{\phi_1(l)\}$ and $\bigcup_{l\in V_H}\{\mu_1(l)\}$ are different independent sets.
The (a) part follows directly from the specification of ϕ. Also by the specification, the image $\phi(H_d)$ is not an induced subgraph of $G(h)$ iff for some $i,\ j \in V_{H_d}$, the vertices $\phi(i)$ and $\phi(j)$ are adjacent in $G(h)$. Since each of them belongs to a different copy of the clique on V, this can only happen if $\phi_1(i) = \phi_1(j)$ or $\phi_1(i)$ is adjacent to $\phi_1(j)$ in G. In the first case, the set $\{\phi_1(1), ..., \phi_1(h)\}$ has size less than h, in the second one, it is not an independent set.

To prove part (c), consider an induced subgraph F of $G(h)$ isomorphic to H_d. The following observations will be useful:
(1) No vertex of F whose degree is at least three can be of the form $a_{i,j}$ or $b_{i,j}$, or $c_{i,j}$, or a_i, or b_i, or c_i since then either it would form a triangle with two vertices in the i-th or j-th copy of the clique on V or it would have degree at most two.
(2) All the vertices a_{ij}, b_{ij}, c_{ij} as well as all the vertices a_i, b_i, c_i, where i is a leaf of H, have to belong to F. It follows in particular that each leaf of H_d has to be mapped on some c_i in the isomorphism.

To see (2), denote by V_i the set $V \times \{i\}$ extended by the adjacent vertices a_{ij}, when $i < j$, and the adjacent vertices c_{ki}, when $i > k$, and halves of the in between vertices b_{ij}, as well as the vertices a_i, b_i, c_i in case $i \in L$.

Note that $V \times \{i\}$ can accommodate at most two vertices of F because the triangles do not occur in H_d. However, if $V \times \{i\}$ contains two vertices of F then the only additional vertices of F that can be accommodated by V_i are those placed at b_{ij}s counted as halves, again because of the absence of triangles, as well as b_i and c_i in case i is a leaf of H. If only one vertex of F is in $V \times \{i\}$ then V_i can accommodate additionally $1.5deg_H(i)$ vertices, plus three vertices in case i is a leaf of H, by fully using the vertices in $V_i \setminus V \times \{i\}$. The latter number of accommodated vertices is larger than that when $V \times \{i\}$ contains two vertices of

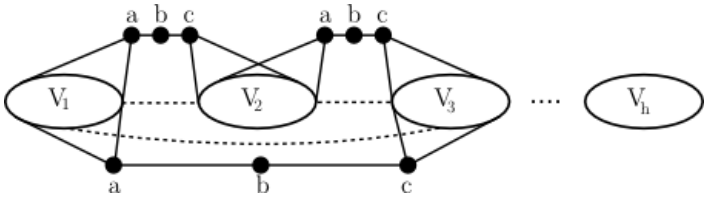


Fig. 2. Example of the vertices and edges of $G(h)$, the large vertices $V_1, V_2, \ldots V_h$ represents cliques of size n and the dotted lines symbolize the edge connection between them

F but for the case where $deg(i) = 1$, when the numbers are equal. In fact, each V_i has to accommodate the aforementioned maximum number in order to cover all vertices of F. Therefore, in particular all the vertices a_i, b_i, c_i for leaves i of H have to be used by F. Hence, no b_{ij} can be used as a vertex of degree 1 which implies that each $V \times \{i\}$ contains exactly one vertex of F, and consequently all the vertices a_{ij}, b_{ij}, c_{ij} have to be used by F. This proves (2).

Suppose first that H is different from a simple cycle.

Consider a maximal path P with inner vertices of degree at most 2 in F. Suppose first that P has both endpoints of degree at least three. Let p_1 be the vertex on P within distance four from an endpoint p of P. By (1), p is in $V \times \{i\}$ for some i. If P does not continue from p through some a_{ij}, b_{ij}, c_{ij} or vice versa then by (2) the degree of p in H has to be larger than the number of such paths linked to $V \times \{i\}$. This in turn means that there is $l \in V_H$, where no all paths of this form linked to $V \times \{l\}$ are used by F. We obtain a contradiction by (2). We conclude that P continues to p_1 in $V \times \{j\}$ by a path in one of the two aforementioned forms. By iterating this argument for p_1 etc., we infer that the vertices of P corresponding to the vertices of H are in distinct $V \times \{l\}$, whereas the vertices between are mapped on some triples a_{ij}, b_{ij}, c_{ij}.

If P has an endpoint of degree 1 then it has some vertex c_l as an endpoint. Hence, the vertices b_l, a_l and some vertex p in $V \times \{l\}$ corresponding to a leaf of H have to follow. Then, we can continue with p similarly as in the previous case. In case the other endpoint is also of degree 1 then when P reaches a vertex in some $V \times \{q\}$ corresponding to another leaf of H, it has to have a_q, b_q and c_q as a suffix.

In case H_d is a simple cycle, we pick an arbitrary vertex of F in some $V \times \{l\}$ as the start and endpoint of a path P with inner vertices of degree 2 and proceed analogously as in the previous cases. Then every fourth following vertex of F will be also in some $V \times \{l\}$. If these vertices are not images of the original vertices of H, we need to compose an embedding ϕ satisfying the three conditions with an automorphism (shift) of H_d.

This completes the proof of the (c) part of the claim.

Now (d) follows by (b) from the fact that an embedding ϕ satisfying the three conditions is uniquely determined by the choice of the second coordinates of the clique vertices. The claim yields the theorem. □

Corollary 4. *Let H be a fixed graph on h vertices, and let its subdivision H_d be defined as in Theorem 2. The problems of detecting and counting induced subgraphs isomorphic to H_d have asymptotic time complexity in terms of the number of vertices of the host graph not less than those for the corresponding problems for independent set on h vertices, respectively.*

4 Simple Lower Bounds

We can expand the list of lower bounds on detection for pattern graphs on four vertices in terms of K_3 (for $K_{1,3}$ cf. Corollary 2) by the following ones for the

diamond $K_4 \setminus e$, the paw $K_3 + e$, and C_4. Because of space considerations, the proofs are brief.

Theorem 3. *Let $k \geq 4$. The time complexity of the detection of an induced subgraph isomorphic to $K_k \setminus e$ is lower bounded by that of an independent set on $k-1$ vertices (K_{k-1} equivalently).*

Proof. Augment an arbitrary host graph G with single copies of its vertices. For a copy v' of a vertex v, add edges between v' and all neighbors of v in G. Let G' denote the resulting graph, where the copy vertices form an independent set.

If G contains a K_{k-1} induced by $(u, ..., w)$ then G' contains the subgraph induced by $(u, ..., w, u')$ which is a $K_k \setminus e$. Conversely, if G' contains an induced $K_k \setminus e$ then the latter either includes a K_{k-1} of G or it is induced by a sequence $(u', v, ..., w, z')$. In the latter case, G contains the subgraphs induced by $(u, v, ..., w)$ and $(v, ..., w, z)$, both are K_{k-1}. □

By K_k+e, we denote a graph consisting of a clique on k vertices and an additional vertex connected by a single edge with the clique.

Theorem 4. *Let $k \geq 3$. The time complexity of the detection of an induced subgraph isomorphic to $K_k + e$ is lower bounded by that of an independent set on k vertices (K_k equivalently).*

Proof. Augment an arbitrary host graph G with single copies of its vertices adjacent solely to their original counterparts to form a graph G'. Observe that G contains a K_k iff G' contains $K_k + e$. □

Our lower bound for C_4 seems most interesting as neither K_3 nor $3K_1$ are induced subgraphs of C_4. In the introduction to [7], Eschen et al. write "Whether there is a C_4-free graph recognition algorithm that beats matrix multiplication and/or a reduction from triangle-free graph recognition remains open."

Theorem 5. *The time complexity of the detection of an induced subgraph isomorphic to C_4 is lower bounded by that of an independent set on three vertices (K_3 equivalently).*

Proof. Let $G = (V, E)$. Insert an auxiliary vertex $x_{\{u,v\}}$ on each edge $\{u, v\} \in E$. Add a set V' of single copies of vertices in V, and form a clique on V'. For each edge $\{u, v\} \in E$, besides the edges $\{u, x_{\{u,v\}}\}$, $\{x_{\{u,v\}}, v\}$, form the edges $\{u', v\}$ and $\{u, v'\}$. Let G' be the resulting graph.

Clearly, if G contains a triangle (u, v, w) then G' contains the induced cycle $(u, x_{\{u,v\}}, v, w')$. Suppose conversely that G' contains an induced C_4. Since G' is a clique on V', the induced C_4 can have at most two vertices in V'. It follows from the definition of G' that at least three other vertices outside V' are necessary to form a cycle. Thus, exactly two vertices in V' are not possible. If there is one vertex $w' \in V'$ in the induced C_4, then the latter has the form $(u, x_{\{u,v\}}, v, w')$ which implies that (u, v, w) is a triangle in G. Finally, there is no way to close a cycle on less than six vertices in G' outside V'. □

Corollary 2, Theorems 3, 4, 5 and the graph complement yield the following.

Corollary 5. *For any pattern graph H on four vertices different from P_4, the time complexity of the detection of an induced subgraph isomorphic to H is lower bounded by that of an independent set on three vertices (K_3 equivalently).*

Acknowledgments. The authors are very grateful to unknown referees whose comments helped to improve a preliminary version of the paper. Special thanks go to the referee that rephrased our conjecture, simplified the proof of Theorem 1 and provided many other valuable comments.

References

1. Chen, Y., Flum, J.: On Parametrized Path and Chordless Path Problems. In: Proc. IEEE Conference on Computational Complexity, pp. 250–263 (2007)
2. Coppersmith, D.: Rectangular matrix multiplication revisited. Journal of Complexity 13, 42–49 (1997)
3. Coppersmith, D., Winograd, S.: Matrix Multiplication via Arithmetic Progressions. J. of Symbolic Computation 9, 251–280 (1990)
4. Corneil, D.G., Perl, Y., Stewart, L.K.: A Linear Recognition Algorithm for Cographs. SIAM J. Comput. 14(4), 926–934 (1985)
5. Downey, R.G., Fellows, M.R.: Parametrized Complexity. Springer, New York (1999)
6. Eisenbrand, F., Grandoni, F.: On the complexity of fixed parameter clique and dominating set. Theoretical Computer Science 326, 57–67 (2004)
7. Eschen, E.M., Hoàng, C.T., Spinrad, J., Sritharan, R.: On graphs without a C4 or a diamond. Discrete Applied Mathematics 159(7), 581–587 (2011)
8. Garey, M.R., Johnson, D.S.: Computers and Intractability. A Guide to the Theory of NP-completeness. W.H. Freeman and Company, New York (2003)
9. Hoàng, C.T., Kaminski, M., Sawada, J., Sritharan, R.: Finding and listing induced paths and cycles. Manuscript (2010)
10. Huang, X., Pan, V.Y.: Fast rectangular matrix multiplications and applications. Journal of Complexity 14(2), 257–299 (1998)
11. Itai, A., Rodeh, M.: Finding a minimum circuit in a graph. SIAM Journal on Computing 7(4), 413–423 (1978)
12. Kloks, T., Kratsch, D., Müller, H.: Finding and counting small induced subgraphs efficiently. Information Processing Letters 74(3-4), 115–121 (2000)
13. Kowaluk, M., Lingas, A., Lundell, E.M.: Counting and detecting small subgraphs via equations and matrix multiplication. In: Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1468–1476 (2011)
14. Nešetřil, J., Poljak, S.: On the complexity of the subgraph problem. Commentationes Mathematicae Universitatis Carolinae 26(2), 415–419 (1985)
15. Olariu, S.: Paw-Free Graphs. Information Processing Letters 28, 53–54 (1988)
16. Papadimitriu, C.H., Yannakakis, M.: On limited nondeterminism and the complexity of the VC-dimension. Journal of Computer and System Sciences 53, 161–170 (1996)
17. Vassilevska, V.: Efficient Algorithms for Path Problems in Weighted Graphs. PhD thesis, CMU, CMU-CS-08-147 (2008)

Contiguous Minimum Single-Source-Multi-Sink Cuts in Weighted Planar Graphs

Ivona Bezáková and Zachary Langley

Rochester Institute of Technology, Rochester, NY, USA
{ib,zbl9222}@cs.rit.edu

Abstract. We present a fast algorithm for uniform sampling of contiguous minimum cuts separating a source vertex from a set of sink vertices in a weighted undirected planar graph with n vertices embedded in the plane. The algorithm takes $O(n)$ time per sample, after an initial $O(n^3)$ preprocessing time during which the algorithm computes the number of all such contiguous minimum cuts. Contiguous cuts (that is, cuts where a naturally defined boundary around the cut set forms a simply connected planar region) have applications in computer vision and medical imaging [6,14].

1 Introduction

Graph cuts have become a popular tool in computer vision over the past decade, see, e. g., [7,6,14]. The goal of image segmentation is to partition a given image into meaningful segments, for example, to isolate an object in the foreground from the background, or to find the boundary of an organ on an ultrasound image.

The image is represented by a graph of pixels (the vertices), edges connect neighboring pixels, and edge weights represent (dis)similarity between the endpoints. In the simplest scenario a user selects a point in the object (the source) and a point in the background (the sink) and a minimum cut between the source and the sink is used to isolate the object from the background.

However, thin objects such as blood vessels are often hard to isolate when using a cut between only two points, see, e. g., [14]. This is because the minimum cut might be clustered around the selected point in the object – for example, opting to sever the point from the rest of the blood vessel – instead of forming a needle-like shape with numerous cut edges of small weight. To avoid this problem, the user may select additional points (seeds) in the object and/or the background, and keep selecting such points until the desired segmentation is achieved – this is known as interactive image segmentation. However, with multiple seeds the cut might consist of several planar regions – for example, regions around the seeds, severing the blood vessel multiple times – instead of a desired single region. A natural solution to this problem is to enforce "contiguity" of the cut; similar concepts are known as a "connectivity prior" [14] and a "topology preserving cut" [6,15].

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 49–60, 2012.

A cut separating the source vertices S from the sink vertices T is a set of vertices containing every vertex in S and no vertex from T – we refer to these cuts as (S,T)-cuts. An (S,T)-cut is minimum if the sum of the weights of edges connecting a vertex in the cut set to a vertex outside the cut set is the smallest possible across all (S,T)-cuts. For planar graphs embedded in the plane, we consider a cut to be contiguous if a region formed by connecting the neighborhoods around each cut vertex along edges and through faces shared by these vertices is simply connected, see Figure 1 and the formal definition in Section 2.

We present an $O(n)$ algorithm that produces a uniformly random contiguous minimum cut separating a single source vertex from a set of sink vertices in a positively weighted undirected planar graph embedded in the plane. The algorithm uses $O(n^3)$ preprocessing time during which it computes the number of all contiguous minimum (s,T)-cuts for a source s and a set of sink vertices T. Note that there could be exponentially many such cuts. Optimization problems with multiple optimum solutions have been recognized as drawbacks in computer vision, see, e. g., [10]. In such cases, random sampling can be used to gather various statistical data on the solutions, or the user can be given a choice between several randomly generated solutions.

We note that heuristics and approximation algorithms have been proposed for finding regions of various connectivity requirements [14,15]; however, to the best of our knowledge, our work is the first polynomial-time provably correct exact algorithm for such a problem.

The earliest works considering the problem of counting minimum cuts in a graph date back to the 1980's. Ball and Provan [1] showed that for a single source and a single sink, the problem reduces to the problem of counting maximal antichains in a poset. In particular, this poset is the directed acyclic graph obtained by finding an acyclic maximum flow, constructing the corresponding residual graph, and contracting each strongly connected component into a single vertex. This implies that the problem is $\#P$-complete for general graphs [13]. Recently, building on [1], a polynomial-time algorithm was developed for the single-source-single-sink variant for planar graphs [2], using, as the first step, the same reduction to the maximal antichains. However, the reduction can not be applied in the contiguous multi-sink case, as the contractions can "bypass" vertices lying in the region defined by the contracted component.

We present a novel reduction that preserves the contiguity of the cuts by selectively contracting certain edges within the strongly connected components, as well as edges that connect vertices from different strongly connected components. This yields a planar directed acyclic multi-graph in which we need to count antichains satisfying a contiguity requirement – we call them contiguous forward cuts. Additionally, we present a new contiguity variant of the cut-cycle duality, where we represent contiguous forward cuts as special kinds of tours of the dual graph – we refer to them as non-crossing. This notion is similar to the so-called non-self-crossing cycle, but with the restriction that the tour forms a star-like shape with respect to every face.

Then we form an acyclic subgraph of the dual graph by "cutting" the primal graph along a tree connecting the source and sink vertices. We decompose each tour into paths in this subgraph where every pair of consecutive paths is joined by a single edge in the dual graph. Moreover, the paths can be sampled independently and are guaranteed to not cross. This allows us to use dynamic programming to obtain the final count of all non-crossing tours. While the proof of correctness is quite involved, the final algorithm is reasonably simple, as summarized in Algorithms 1 and 2.

For completeness, we mention recent works dealing with maximum flows and minimum cuts in planar graphs. Borradaile and Klein [3] gave an $O(n \log n)$ algorithm for the single-source-single-sink acyclic maximum flow. Borradaile, Sankowski, and Wulff-Nilsen [5] produce a minimum single-source-single-sink cut for any source-sink pair in time proportional to the size of the cut, after an initial $O(n \text{ polylog } n)$ preprocessing time. Italiano, Nussbaum, Sankowski, and Wulff-Nilsen [11] give algorithms for undirected planar graphs that break the $O(n \log n)$ time barrier. Recently, Borradaile, Klein, Mozes, Nussbaum, and Wulff-Nilsen [4] gave an $O(n \log^3 n)$ algorithm for maximum flow from multiple sources to multiple sinks. While all these algorithms are very ingenious, as far as we know, none of them produce the respective cut counts (or samples).

Finally, we remark that we do not know of any polynomial-time algorithms counting or sampling all minimum single-source multi-sink cuts in planar graphs (i. e., not just contiguous cuts). Similarly, the problem is open for simple cuts (i. e., cuts where the graph induced by the cut vertices is connected). In both cases Ball and Provan's reduction can be applied but it is unclear how to count the corresponding sets of antichains. In the case of general cuts, an antichain might correspond to a set of several tours, not just one. In the case of simple cuts, an antichain corresponds to a cycle and our dynamic programming technique does not guarantee to not repeat vertices across different path segments. The case of contiguous cuts with multiple sources and multiple sinks is also open.

The paper is organized as follows. Section 2 contains preliminaries, Section 3 describes how to reduce the problem to the problem of counting contiguous forward cuts in a planar directed acyclic multi-graph, Section 4 describes the representation of contiguous forward cuts via non-crossing tours in the dual graph, and Section 5 describes the main dynamic programming algorithm, followed by the sampling procedure. The proofs are omitted due to space constraints.

2 Preliminaries

Let $G = (V, E, w)$ be a weighted undirected connected planar graph with edge weights $w : E \to \mathbf{R}^+$. Let $s \in V$ and $T \subseteq V$, $s \notin T$. Our objective is to count[1] all minimum (s, T)-cuts of G where an (s, T)-*cut* is a set of vertices $C \subseteq V$ such that $s \in C$ and $T \cap C = \emptyset$. The value of the cut C is the sum of all

[1] We first develop the counting algorithm – the sampling part will be discussed in Section 5.

edge weights of edges leading out of C – formally, $\sum_{(u,v)\in E: u\in C, v\notin C} w(u,v)$. A *minimum* (s,T)*-cut* has the smallest possible value of all (s,T)-cuts.

Given a directed graph $H=(V_H,E_H)$, we say that a cut $C\subseteq V_H$ is a *forward-cut* if there is no edge leading into C, i. e., there is no edge (u,v) such that $u\notin C$ and $v\in C$. A forward-cut C is a *forward-*(A,b)*-cut* where $A\subseteq V_H$, $b\in V_H$ and $b\notin A$, if $A\subseteq C$ and $b\notin C$.

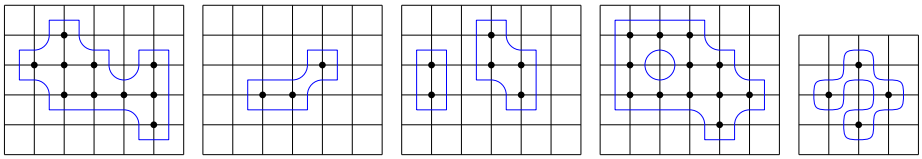
A *flow network* is a directed graph $G=(V,E,c)$ where $c: E\to \mathbf{R}^+$ defines non-negative *edge capacities*. Let $s,t\in V$, $s\neq t$, be two vertices called the *source* and the *sink*, respectively. A *flow* from s to t is a function $f: E\to \mathbf{R}_0^+$ satisfying the following properties: (1) *capacity constraint:* $f(e)\leq c(e)$ for every $e\in E$, and (2) *flow conservation:* $\sum_{u:\ (u,v)\in E} f(u,v) = \sum_{u:\ (v,u)\in E} f(v,u)$ for every $v\in V\setminus\{s,t\}$. The *value of the flow* f is the sum of the values of flow edges out of s minus the sum of the values of the flow edges into s, i. e., $\sum_{u:\ (s,u)\in E} f(s,u) - \sum_{u:\ (u,s)\in E} f(u,s)$. A flow is said to be *maximum* if it has the largest possible value among all flows from s to t (we also refer to such flows as s-t flows). A flow is said to be acyclic if the set of edges with positive flow value $\{e\in E \mid f(e)>0\}$ does not contain a directed cycle.

The *residual graph of the flow* f, denoted $G_f=(V,E_f,w_f)$, is a weighted directed graph where E_f contains the following two types of edges: (1) for every $e=(u,v)\in E$ with $f(e)<c(e)$, the set E_f contains a *forward edge* $e=(u,v)$ with weight $w_f(e)=c(e)-f(e)$, and (2) for every $e=(u,v)\in E$ with $f(e)>0$, the set E_f contains a *backward edge* $e'=(v,u)$ with weight $w_f(e')=f(e)$.

The following theorem describes Ball and Provan's reduction.

Theorem 1 ([1,2]). *Let $G=(V,E,w)$ be a connected positively weighted undirected graph and let $s,t\in V$, $s\neq t$. Let $G'=(V,E',c)$ be a flow network obtained by including, for every edge $(u,v)\in V$, two directed edges (u,v) and (v,u) in E' with capacities $c(u,v)=c(v,u)=w(u,v)$. Let f be an acyclic maximum s-t flow in G' and let G'_f be the corresponding residual graph. Let $H=(V_H,E_H)$ be the graph obtained from G'_f by contracting each strongly connected component into a single vertex, omitting duplicate and self-loop edges, and ignoring the edge weights. Hence, vertices of H are sets of vertices of G – let $\hat{s}$ and $\hat{t}$ be the vertices in V_H containing s and t respectively. Then, the set of minimum (s,t)-cuts in G is in bijection with the set of forward-$(\hat{t},\hat{s})$-cuts in H by mapping a forward-$(\hat{t},\hat{s})$-cut $C_H\subseteq V_H$ to the (s,t)-cut $C=\cup_{x\notin C_H} x$. Moreover, if G is connected, then H, viewed as an undirected graph, is connected, and if G is planar, then H is planar. The graph H can be obtained in time $O(|V|^3+|E|^2)$ and time $O(|V|\log|V|)$ if G is planar.*

Next we define contiguous cuts that give rise to a contiguous region in the plane – a concept useful for many segmentation applications, see, e. g., [6,14] for a discussion of several contiguity concepts. For a planar (directed or not) graph $G=(V,E)$ embedded in the plane and a set of vertices $C\subset V$, we define $R(C)$, a set of points in the plane, as follows. We start with the union of all faces that contain a vertex from C on their boundary. Then, for every vertex not in C, we remove an ε-neighborhood around this vertex. Finally, for every edge between



(a): cut set C_1 (b): cut set C_2 (c): cut set C_3 (d): cut set C_4 (e)

Fig. 1. (a)-(d): Contiguous vs. noncontiguous cuts: four possible cut sets (of the 6×7 grid graph). The highlighted vertices are in the respective cut sets $C_1, \ldots, C_4$. The shown boundaries bound the corresponding point sets $R(C_1), \ldots, R(C_4)$. Cut sets C_1 and C_2 are contiguous, cut sets C_3 and C_4 are not. (e): Contiguous cuts vs. non-self-crossing cycles: not a contiguous cut, yet the cut set can be bounded by a non-self-crossing cycle.

two vertices that are both not in C, we remove an ε-neighborhood around this edge. (By ε-neighborhood we mean the set of points in the plane with distance $\leq \varepsilon$ from the vertex or edge. We choose ε so that the ε-neighborhood does not intersect with non-adjacent edges or contain other vertices in the planar drawing of G.) We say that C is *contiguous* if $R(C)$ forms a simply connected region in the plane, i. e., if the boundary of $R(C)$ splits the plane into exactly two regions. See Figure 1(a)-(d). Informally, in the grid graph the contiguity concept means a "corner-connected" region without holes.

To put contiguous cuts in perspective with the standard cut-cycle duality, we note that contiguous cuts are *not* dual with the so-called non-self-crossing cycles. Consider Figure 1(e) where a non-self-crossing cycle separates the cut vertices from the remaining vertices, yet the cut is not contiguous.

Finally, to simplify our language, for a directed planar graph embedded in the plane, we refer to the two faces neighboring an edge $e = (u, v)$ as the *left face of* e (when traversing e from u to v, this face is on the left) and the *right face of* e (the other face). We use the same terminology when describing regions bounded by directed paths/cycles. By a *clockwise traversal* of the boundary of a face (or a planar simply connected region) f we mean listing the edges on the boundary of f in the clockwise order as seen from the viewpoint of somebody standing *inside* f.

3 Reduction to Contiguous Forward Cuts

In this section we present an algorithm that reduces the problem of counting all contiguous minimum (s, T)-cuts to the problem of counting all contiguous forward-$(\hat{T}, \hat{s})$-cuts in a planar directed acyclic (multi)graph. On the surface this statement seems analogous to Theorem 1: we can create a super-sink connected to every sink by an ∞-weighted edge and apply the original reduction. However, this can result in contracting a cycle consisting of edges that are not minimum-cut edges into a single vertex while "bypassing" the area inside the cycle. Hence,

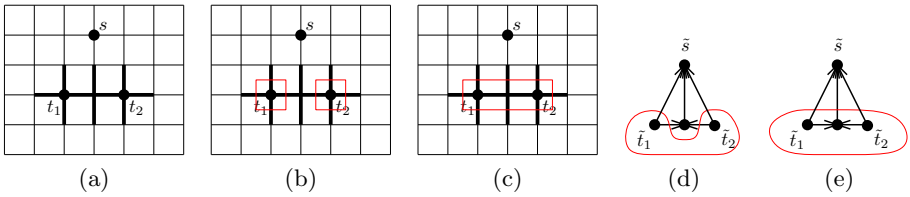


Fig. 2. Contiguous (s, T)-cuts in G vs. contiguous forward-$(\hat{T}, \hat{s})$-cuts in H. Figure (a) shows a source and two sinks, the highlighted edges are of weight 1, all other edges are of weight ∞. Figures (b) and (c) depict two possible minimum (s, T)-cuts (of weight 8) – (b) is not contiguous and (c) is contiguous. Figures (d) and (e) show the graph H and the forward cuts corresponding to the minimum cuts from figures (b) and (c). Both forward cuts are contiguous, even though the cut in figure (b) is not contiguous.

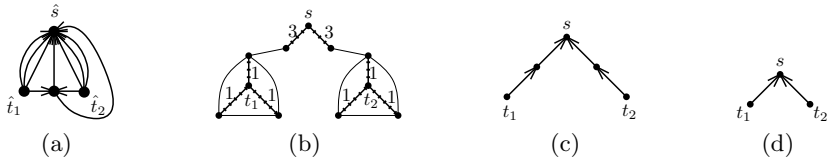


Fig. 3. Applying Algorithm 1: Figure (a) shows the result for the graph from Figure 2(a). Figure (b) shows another possible input graph, the highlighted edges are of weights 1 or 3, the other edges are of weight ∞. Figures (c) and (d) depict the corresponding graphs after applying Ball and Provan's reduction (there are 4 minimum $(s, \{t_1, t_2\})$-cuts and thus 4 corresponding forward cuts) and Algorithm 1 (there is only one forward cut corresponding to the single contiguous minimum $(s, \{t_1, t_2\})$-cut).

noncontiguous cuts become contiguous forward cuts, as demonstrated in Figure 2. To avoid such problems, we designed a new reduction (Algorithm 1) that selectively contracts and removes edges to preserve contiguity.

Theorem 2. *Let $G = (V, E, w)$ be a connected undirected planar graph embedded in the plane, $w > 0$. Let $s \in V$ and $T \subseteq V$, $s \notin T$. Algorithm 1 decides whether there exists a contiguous minimum (s, T)-cut in G. If yes, it constructs a directed acyclic (multi)graph $H' = (V'_H, E'_H)$ embedded in the plane, a vertex $\hat{s} \in V'_H$, and a set of vertices $\hat{T} \subseteq V'_H$ such that the set of all contiguous minimum (s, T)-cuts in G is in bijection with the set of all contiguous forward-$(\hat{T}, \hat{s})$-cuts in H'. The algorithm runs in time $O(|V|^3)$. Moreover, $\hat{T}$ is the set of vertices of indegree 0 in H' and $\hat{s}$ is the only vertex of outdegree 0 in H'.*

What happens when we apply Algorithm 1 to the graph from Figure 2? We will keep two edges from the middle vertex to $\hat{s}$, as shown in Figure 3(a), preventing the "illegal" contiguous forward cut from Figure 2(d). A graph where Algorithm 1 needs to deal with self-loops is shown in Figure 3(b)-(d). Figure 4 sketches the individual self-loop cases on which the proof of Theorem 2 is based.

Algorithm 1. Reduction to contiguous forward cuts

Add an extra vertex, τ, to G, connect it with ∞-weight edges to the vertices in T, and replace every edge by two directed edges of the same weight. Let $G_f = (V_\tau, E_f, w_f)$ be the residual graph of an acyclic s-τ maximum flow f of the new graph. Let $\hat{T}$ be the set of vertices of G that belong to the same strongly connected component of G_f as τ.
Remove τ and its adjacent edges from G_f, and ignore the edge weights, obtaining H'_0 (embedded in the plane analogously to G).
Let $e_1, e_2, \ldots, e_\ell \in E_f$ be the edges that belong to any strongly connected component of H'_0.
for $i = 1, \ldots, \ell$ **do**
  if e_i does not form a self-loop in H'_{i-1} **then**
    Get H'_i from H'_{i-1} by contracting the edge e_i (if it has not been removed earlier).
  else
    Let (a, a) be the self-loop formed by e_i in H'_{i-1}.
    if $s \notin a$ **then**
      The self-loop splits the plane into two regions: let R be the region that does not contain s.
      if all vertices in $\hat{T}$ are in R **then**
        Get H'_i from H'_{i-1} by removing the self-loop (a, a). See Figure 4(a).
      else
        Get H'_i from H'_{i-1} by contracting all vertices in R into a and remove all (a, a) self-loops. See Figure 4(b)-(c).
    else
      if the two regions bounded by the self-loop each contain a vertex in $\hat{T}$ **then**
        Return "no contiguous minimum cuts". See Figure 4(d).
      else
        Let R be the region that contains the vertices in $\hat{T}$.
        Get H'_i from H'_{i-1} by contracting all vertices outside R into a and remove all (a, a) self-loops. See Figure 4(e).
For every $t \in \hat{T}$, contract all predecessors of t into t and omit self-loops.
Return $H' := H'_\ell$ and its planar embedding, $\hat{T}$, and the vertex $\hat{s}$ containing s.

4 Non-crossing Tours

In this section we classify contiguous forward-$(\hat{T}, \hat{s})$-cuts as certain types of *tours* (i. e., cycles that are allowed to repeat vertices) in the dual planar graph. While these tours may revisit faces (i. e., vertices in the dual graph), they cannot "self-cross," as defined below.

First, recall the standard definition of a directed planar dual. For a directed planar (multi)graph $H = (V_H, E_H)$ embedded in the plane, we define the *dual (multi)graph* $H_D = (V_D, E_D)$ as follows: V_D is the set of all faces of H and for every edge $e \in E_H$ we include an edge from f_1 to f_2 in E_D where f_1 and f_2 are the left and the right face of e, respectively.

Next we define a "non-crossing" tour and the "inside" and the "outside" regions defined by the tour.

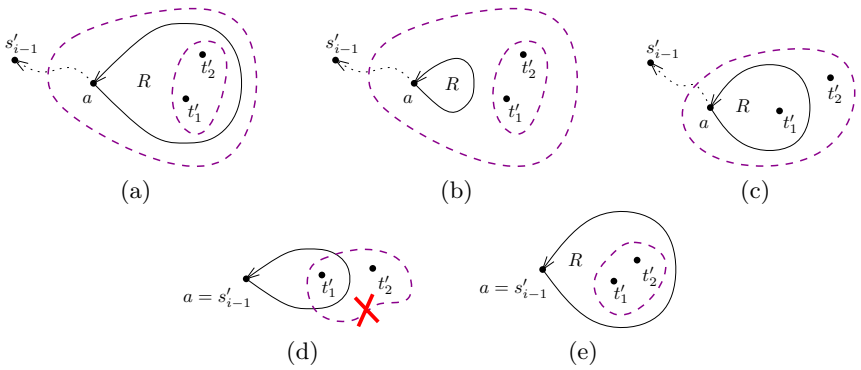


Fig. 4. Demonstrating the cases in Algorithm 1, possible contiguous cuts are dashed

Definition 1. *Let $H' = (V'_H, E'_H)$ be a connected planar directed acyclic (multi) graph embedded in the plane and let $H'_D = (V'_D, E'_D)$ be its dual graph. Let $d_1, d_2, \ldots, d_\ell$, $d_i \in E'_D$ for $i \in \{1, \ldots, \ell\}$, be a tour in H'_D. Let $f_1, \ldots, f_\ell$ be the faces visited by the tour, i. e., the edge d_i goes from f_i to f_{i+1} (where $f_{\ell+1} = f_1$) and let e_i be the edge that gave rise to the edge d_i in the dual graph. We say that the tour is* non-crossing *if the following holds for every face f visited by the tour. Let $f_{j_1}, f_{j_2}, \ldots, f_{j_p}$ for $1 \le j_1 \le j_2 \le \cdots \le j_p \le \ell$ be all the faces on the tour equal to f. Then, the edges $e_{j_1-1}, e_{j_1}, e_{j_2-1}, e_{j_2}, \ldots, e_{j_p-1}, e_{j_p}$ must appear in this order when clockwise traversing the boundary of f (where $e_0 := e_\ell$). See Figure 5.*

The inside region *defined by the tour consists of all the starting endpoints of the edges e_i, $i \in \{1, \ldots, \ell\}$, and all their predecessors. The* outside region *contains all the other vertices of H'.*

Notice that a non-crossing tour can be drawn in the plane in a "non-self-crossing way". Notice also that drawing a tour in a non-self-crossing way does *not* imply that the tour is non-crossing, as demonstrated in Figure 5(c).

Lemma 1. *Let $H'_D = (V'_D, E'_D)$ be the dual of the graph H' from Theorem 2. Then, the set of all contiguous forward-$(\hat{T}, \hat{s})$-cuts of H' is bijection with the set of all non-crossing tours in H'_D such that the inside region defined by the tour contains all vertices from $\hat{T}$ and the outside region contains $\hat{s}$.*

5 Counting and Sampling Contiguous Minimum (s, T)-Cuts

In this section we prove the main theorem of the paper:

Theorem 3. *Let $G = (V, E, w)$ be a connected undirected planar graph with edge weights $w : E \to \mathbf{R}^+$, embedded in the plane. Let $s \in V$ and $T \subset V$, $s \notin T$. The number of contiguous minimum (s, T)-cuts of G can be computed in time*

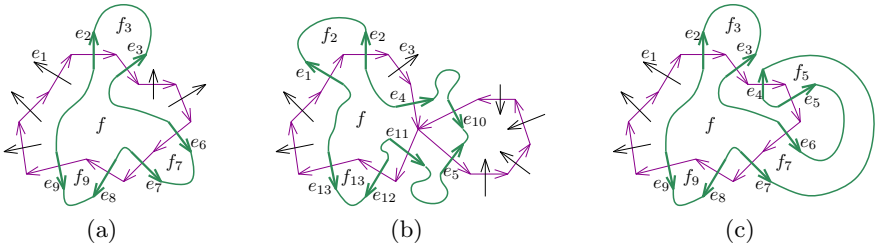


Fig. 5. Illustrating Definition 1 (a non-crossing tour): Face f (the curve-shaped shaped region) is visited by the tour three times in figure (a): $f = f_3 = f_7 = f_9$, four times in figure (b): $f = f_2 = f_5 = f_{11} = f_{13}$, and four times in figure (c): $f = f_3 = f_5 = f_7 = f_9$. The edges e_i appear on the boundary of f in this order: (a) $e_2, e_3, e_6, e_7, e_8, e_9$ – the clockwise order (with respect to the tour), i. e., the tour could be non-crossing (depending on the other faces on the tour); (b) $e_1, e_2, e_4, e_{10}, e_5, e_{11}, e_{12}, e_{13}$, and (c) $e_2, e_3, e_6, e_5, e_4, e_7, e_8, e_9$ – the tours (b) and (c) are definitely not non-crossing.

$O(|V|^3)$. *A uniformly random contiguous minimum* (s, T)*-cut can be produced in additional linear time.*

By Lemma 1, we know that it suffices to count all non-crossing tours separating $\hat{s}$ from $\hat{T}$ in the dual graph H'_D. Even though counting cycles or tours in planar graphs tends to be $\#P$-complete [8,9,12], we show that the problem of counting non-crossing tours in H'_D can be solved in polynomial time. In particular, we decompose the tour into paths (that cannot repeat vertices) and then we count (or sample) each path type separately. The counting algorithm combines the paths using dynamic programming.

Before we state the decomposition lemma, we define a "restricted dual" graph that, unlike the dual graph H'_D, will be guaranteed to be acyclic. The definition will use a "tree" of edges in H' that connects $\hat{T}$ to $\hat{s}$.

Observation 1 *Let* $H' = (V'_H, E'_H)$ *be a planar directed acyclic (multi)graph, let* $\hat{s} \in V'_H$ *be the only vertex of outdegree 0, and let* $\hat{T} \subseteq V'_H$, $\hat{s} \notin \hat{T}$, *be the set of vertices of indegree 0. There exists a set of edges* $A \subseteq E'_H$ *such that for every* $\hat{t} \in \hat{T}$ *there is a unique directed path from* $\hat{t}$ *to* $\hat{s}$ *using only the edges from* A*, and every edge in* A *is on the path from* $\hat{t}$ *to* $\hat{s}$ *for some* $\hat{t} \in \hat{T}$*. Moreover,* A *can be constructed in time* $O(|\hat{T}||V'_H|)$.

Definition 2. *Let* H' *(embedded in the plane),* $\hat{s}$, $\hat{T}$, *and* A *be as in Observation 1. We define the* restricted dual (multi)graph $H'_d = (V'_d, E'_d)$ *of* H' *as follows:* V'_d *is the set of all faces of* H'*, and, for every edge* $e \in E'_H \setminus A$*, we include an edge from* f_1 *to* f_2*, where* f_1 *and* f_2 *are the left and right faces of* e*, respectively.*

Lemma 2. *The graph* $H'_d = (V'_d, E'_d)$ *from Definition 2 is acyclic.*

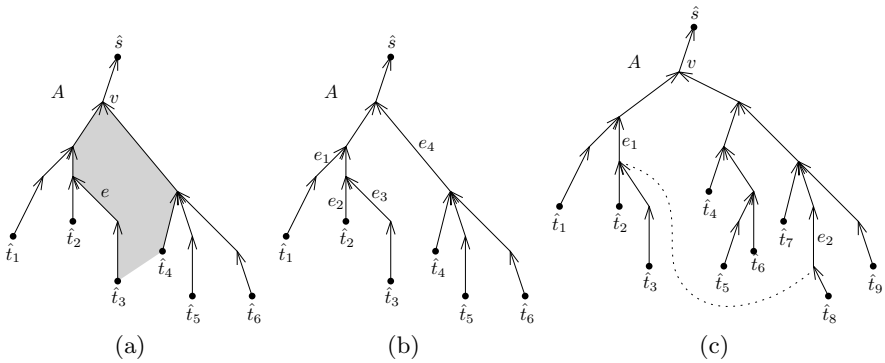


Fig. 6. A-induced order and wedges: (a) The A-induced order of vertices in $\hat{T}$ is $\hat{t}_1, \ldots, \hat{t}_6$. The wedge between $\hat{t}_3$ and $\hat{t}_4$ is highlighted – it is defined by the $\hat{t}_3$-v path (its left path) and the $\hat{t}_4$-v path (its right path). The wedge is on the right of the edge e; the wedge on the left of e is between $\hat{t}_2$ and $\hat{t}_3$. (b) The edges e_1, e_2, e_3, e_4 are listed in the A-induced order. (c) Edge e_1 is five wedges apart from e_2, since the region defined by the e_1-v path, the e_2-v path and the dotted curve contains 4 vertices from $\hat{T}$.

We need some additional terminology before stating the decomposition lemma.

Suppose we reverse the edges of A and perform a depth-first traversal from $\hat{s}$, going through the neighbors of the current vertex in the counterclockwise order, starting from the edge we used to get to the vertex. Let $\hat{t}_1, \hat{t}_2, \ldots, \hat{t}_k$ be the order in which we visited the vertices in $\hat{T}$ (we refer to this order as the *A-induced order of vertices in $\hat{T}$*). Let $\hat{t}_{k+1} := \hat{t}_1$. For every $i \in \{1, \ldots, k\}$ we define a *wedge* as follows. Let v be the first common successor of $\hat{t}_i$ and $\hat{t}_{i+1}$ when restricted only to edges in A. The path from $\hat{t}_i$ to v, using only edges in A, forms the *left path* of the wedge; similarly, the path from $\hat{t}_{i+1}$ to v, using only edges in A, forms the *right path*. Every edge $e \in A$ is in two wedges: the *left* and the *right wedge of* e, for which e lies on the right and the left path, respectively. See Figure 6(a).

In addition to the A-induced order of vertices in $\hat{T}$, we also define the A-induced order of pairwise independent edges in A as follows. Let $e_1, e_2, \ldots, e_q \in A$, where for every $i \neq j$, e_i is not a successor of e_j in A. Suppose that we perform the same depth-first traversal of A as described in the previous paragraph. Then, if we visit the edges $e_1, e_2, \ldots, e_q$ in this order (or its cyclic rotation), we say that the edges are ordered in the *A-induced order of edges*, see Figure 6(b).

We say that edge $e_1 \in A$ is j *wedges apart* from $e_2 \in A$ if there are $j-1$ intermediate vertices from $\hat{T}$ between e_1 and e_2. More precisely, let v be the first common successor of e_1, e_2 when using only edges in A. Connect the starting vertices of e_1 and e_2 by a curve in the plane so that the curve does not touch any of the edges in A. Then, consider the region on the right of the e_1-v path in A, on the left of the e_2-v path in A, and bounded by the curve; if it contains exactly $j-1$ vertices from $\hat{T}$, we say that e_1 is j wedges apart from e_2, see Figure 6(c).

The following lemma describes how to decompose a non-crossing tour that separates $\hat{s}$ from $\hat{T}$ into paths in the restricted dual graph H'_d (and edges in H'_D

Algorithm 2. Counting non-crossing tours in the dual graph H'_D that contain all vertices in $\hat{T}$ in the inside region and $\hat{s}$ in the outside region

1: **for** every $e_1, e_2 \in A$ **do**
2: compute $\xi[e_1, e_2]$, the number of paths in H'_d from the right face of e_1 to the left face of e_2 (in linear time, since H'_d is acyclic)
3: let p be a path from one of the vertices in $\hat{T}$ to $\hat{s}$, using only edges in A
4: **for** every $e_1 \in p$ **do**
5: **for** every $e_2 \in A$ such that e_1 is exactly 1 wedge apart from e_2 in A **do**
6: let $a[e_1, e_2] = \xi[e_1, e_2]$
7: **for** $j = 2$ to $|\hat{T}| - 1$ **do**
8: **for** every $e_2 \in A$ such that e_1 is exactly j wedges apart from e_2 in A **do**
9: let

$$a[e_1, e_2] := \sum_{e' \in p'} a[e_1, e']\xi[e', e_2],$$

where p' is the left path of e_2's left wedge
10: **for** every $e \in p$ **do**
11: let $a[e, e] := \sum_{e' \in p'} a[e, e']\xi[e', e]$, where p' is the left path of e's left wedge
12: return $\sum_{e \in p} a[e, e]$

that connect the paths to form the tour). In essence, the tour is cut into path segments by the tree A.

Lemma 3. *Under the assumptions of Definition 2, let X be a tour in H'_D, the dual graph of H'. The tour X is non-crossing and separates $\hat{s}$ from $\hat{T}$ if and only if all of the following conditions hold: (1) there exist edges $e_1, e_2, \ldots, e_q \in A$ such that for every $i, j \in \{1, \ldots, q\}$, $i \neq j$, e_i is not a successor of e_j in A, (2) $e_1, \ldots, e_q$ is the A-induced order of these edges, (3) for every $\hat{t} \in \hat{T}$ there exists i such that e_i is on the path from $\hat{t}$ to $\hat{s}$ in A, (4) there exists a path p_i in H'_d from the right face of e_i to the left face of e_{i+1} (let $e_{q+1} := e_1$), and (5) $X = p_1, d_1, p_2, d_2, \ldots, p_q, d_q$, where d_i is the dual of the edge e_i.*

Lemma 3 yields a dynamic programming algorithm (Algorithm 2) for counting all non-crossing tours separating $\hat{s}$ from $\hat{T}$. By gradually increasing the wedge distance, it counts walks (paths with repeated vertices) in the dual graph H'_D starting and ending in faces both bordering an edge in A. In particular, $a[e_1, e_2]$ is the number of walks starting with the right face of e_1 and ending with the left face of e_2. For e_1, e_2 at distance 1, $a[e_1, e_2]$ can be computed by a topological traversal of H'_d. For larger distances, the computation goes through an intermediate edge e' on the left wedge of e_2, "responsible" for separating $\hat{s}$ from the $\hat{t}_i$ on this wedge (see step 9). Correctness of Algorithm 2, along with Theorem 2 and Lemma 1 imply the counting claim of Theorem 3.

For the sampling part, we first choose e on the path p proportionally to $a[e, e]$. Then we choose e' proportionally to $a[e, e']\xi[e', e]$, etc., getting the edges $e_1, e_2, \ldots, e_q$ from Lemma 3. Then we independently sample each path p_i (see [2] for the details of this step), obtaining a non-crossing tour and the corresponding contiguous minimum (s, T)-cut.

We conclude the paper with two remarks. First, the running time bound $O(n^3)$ is tight. This can be seen by forming a graph with three paths of length $n/3$ starting at the same vertex s and ending at t_1, t_2, t_3, respectively (except for s, the paths are disjoint). Second, the $O(n^3)$ preprocessing time might seem prohibitively large for larger data sets. We note that in practice the graph H', formed by contracting a typically very sizable set of edges, is likely going to be significantly smaller than the original graph. Combining this with a faster network flow algorithm [4], the overall running time becomes much more practical.

References

1. Ball, M.O., Provan, J.S.: Calculating bounds on reachability and connectnedness in stochastic networks. Networks 13, 253–278 (1983)
2. Bezáková, I., Friedlander, A.J.: Counting and sampling minimum (s,t)-cuts in weighted planar graphs in polynomial time. Theor. Comp. Sci. 417, 2–11 (2012)
3. Borradaile, G., Klein, P.N.: An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. J. ACM 56(2) (2009)
4. Borradaile, G., Klein, P.N., Mozes, S., Nussbaum, Y., Wulff-Nilsen, C.: Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. In: Proceedings of the 52nd IEEE Symposium on Foundations of Computer Science (FOCS), pp. 170–179 (2011)
5. Borradaile, G., Sankowski, P., Wulff-Nilsen, C.: Min st-cut oracle for planar graphs with near-linear preprocessing time. In: Proceedings of the 51st IEEE Symposium on Foundations of Computer Science (FOCS), pp. 601–610 (2010)
6. Boykov, Y., Veksler, O.: Graph cuts in vision and graphics: Theories and applications. In: Paragios, N., Chen, Y., Faugeras, O. (eds.) Handbook of Mathematical Models in Computer Vision. Springer (2006)
7. Boykov, Y., Veksler, O., Zabih, R.: Fast approximate energy minimization via graph cuts. IEEE Trans. Pattern Anal. Mach. Intell. 23(11), 1222–1239 (2001)
8. Creed, P.: Counting and sampling problems on Eulerian graphs. Ph.D. Dissertation, University of Edinburgh (2010)
9. Ge, Q., Štefankovič, D.: The complexity of counting Eulerian tours in 4-regular graphs. Algorithmica 63(3), 588–601 (2012)
10. Grady, L.: Minimal surfaces extend shortest path segmentation methods to 3D. IEEE Trans. Pattern Anal. Mach. Intell. 32(2), 321–334 (2010)
11. Italiano, G.F., Nussbaum, Y., Sankowski, P., Wulff-Nilsen, C.: Improved algorithms for min cut and max flow in undirected planar graphs. In: Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC), pp. 313–322 (2011)
12. Liskiewicz, M., Ogihara, M., Toda, S.: The complexity of counting self-avoiding walks in subgraphs of two-dimensional grids and hypercubes. Theoretical Computer Science 304(1-3), 129–156 (2003)
13. Provan, J.S., Ball, M.O.: The complexity of counting cuts and of computing the probability that a graph is connected. SIAM J. Comput. 12(4), 777–788 (1983)
14. Vicente, S., Kolmogorov, V., Rother, C.: Graph cut based image segmentation with connectivity priors. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR (2008)
15. Zeng, Y., Samaras, D., Chen, W., Peng, Q.: Topology cuts: A novel min-cut/max-flow algorithm for topology preserving segmentation in n-d images. Computer Vision Image Understanding 112, 81–90 (2008)

Online Knapsack Problem with Removal Cost

Xin Han[1,⋆], Yasushi Kawase[2,⋆⋆], and Kazuhisa Makino[2,⋆⋆]

[1] Dalian University of Technology
hanxin@dlut.edu.cn
[2] University of Tokyo
{yasushi_kawase,makino}@mist.i.u-tokyo.ac.jp

Abstract. In this paper, we study the online knapsack problem with removal cost. The input is a sequence of items $u_1, u_2, \ldots, u_n$, each of which has a size and a value, where the value of each item is assumed to be equal to the size. Given the ith item u_i, we either put u_i into the knapsack or reject it with no cost. When u_i is put into the knapsack, some items in the knapsack are removed with removal cost if the sum of the size of u_i and the total size in the current knapsack exceeds the capacity of the knapsack. Here the removal cost means a cancellation charge or disposal fee. Our goal is to maximize the profit, i.e., the sum of the values of items in the last knapsack minus the total removal cost occurred.

In this paper, we consider two kinds of removal cost: unit and proportional cost. For both models, we provide their competitive ratios. Namely, we construct optimal online algorithms and prove that they are best possible.

1 Introduction

The knapsack problem is one of the most classical problems in combinatorial optimization and has a lot of applications in the real world [10]. The knapsack problem is that: given a set of items with values and sizes, we are asked to maximize the total value of selected items in the knapsack satisfying the capacity constraint.

In this paper, we study the online version of the knapsack problem with removal cost. Here, "online" means i) the information of the input (i.e., the items) is given gradually, i.e., after a decision is made on the current item, the next item is given; ii) the decisions we have made are irrevocable, i.e., once a decision has been made, it cannot be changed. Given the ith item u_i, we either accept u_i (i.e., put u_i into the knapsack) or reject it with no cost. When u_i is put into the knapsack, some items in the knapsack are removed with removal cost if the sum of the size of u_i and the total size in the current knapsack exceeds

⋆ Partially supported by NSFC(11101065).
⋆⋆ Supported by a Grant-in-Aid for Scientific Research from the Ministry of Education, Culture, Sports, Science and Technology of Japan and the Global COE "The Research and Training Center for New Development in Mathematics."

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 61–73, 2012.

1, i.e., the capacity of the knapsack. Here the removal cost means a cancellation charge or disposal fee. Our goal is to maximize the profit, i.e., the sum of the values of items in the last knapsack minus the total removal cost occurred.

1.1 Related Work

The online knapsack problem (under no removal condition) was first studied on average case analysis by Marchetti-Spaccamela and Vercellis [12]. They proposed a linear time approximation algorithm such that the expected difference between the optimal profit and the one obtained by the algorithm is $O(\log^{3/2} n)$ under the condition that the capacity of the knapsack grows proportionally to the number of items n. Lueker [11] improved the expected difference to $O(\log n)$ under a fairly general condition on the distribution.

Iwama and Taketomi [8] studied the online knapsack problem on worst case analysis. They obtained a $\frac{1+\sqrt{5}}{2} \approx 1.618$-competitive algorithm for the online knapsack when (1) the removable condition (without removal cost) is allowed and (2) the value of each item is equal to the size, and showed that this is best possible by providing a lower bound 1.618 for the case. We remark that the problem has unbounded competitive ratio, if at least one of the conditions (1) and (2) is not satisfied [8,9]. For other models such as minimum knapsack problem and knapsack problem with limited cuts, refer to papers in [6,7,13].

The *removal cost* has introduced in the buyback problem [1–5]. In the problem, we observe a sequence of bids and decide whether to accept each bid at the moment it arrives, subject to constraints on accepted bids such as single item and matroid constraints. Decisions to reject bids are irrevocable, whereas decisions to accept bids may be canceled at a cost which is a fixed fraction of the bid value. Babaioff *et al.* [3] showed that the buyback problem with matroid constraint has $\left(1 + 2f + 2\sqrt{f(1+f)}\right)$-competitive ratio, where $f > 0$ is a buyback factor. Ashwinkumar [1] extended their results and show that the buyback problem with the constraint of k matroid intersections has $k(1+f)(1+\sqrt{1-\frac{1}{k(1+f)}})^2$-competitive ratio.

1.2 Our Results

In this paper, we study the worst case analysis of the online knapsack problem with removal cost, when the value of each item is equal to the size. We considers two kinds of models of removal cost: the *proportional* and the *unit* cost model. In the proportional cost model, the removal cost of each item u_i is proportional to its value (and hence size), i.e., it is $f \cdot s(u_i)$, where $s(u_i)$ denotes the size of u_i and $f > 0$ is a fixed constant, called *buyback factor*. Therefore, we can view this model as the buyback problem with knapsack constraints. In the unit cost model, the removal cost of each item is a fixed constant $c > 0$, where we assume that every item has value at least c, since in many applications, the removal cost (i.e., cancellation charge) is not higher than its value. We remark that the

problem has unbounded competitive ratio if no such assumption is satisfied (see Section 3).

We show that the proportional and unit cost models have competitive ratios $\lambda(f)$ and $\mu(c)$ in (1) and (2), respectively, where $\lambda(f)$ and $\mu(c)$ are given in Figure 1. Namely, we construct $\lambda(f)$- and $\mu(c)$-competitive algorithms for the models and prove that they are best possible.

$$\lambda(f) = \begin{cases} 2 & (1/2 \geq f > 0), \\ \frac{1+f+\sqrt{f^2+2f+5}}{2} & (f > 1/2). \end{cases} \tag{1}$$

$$\mu(c) = \begin{cases} \max\{\eta(k), \xi(k+1)\} & (1 - \sqrt{\frac{k+1}{k+2}} \leq c \leq 1 - \sqrt{\frac{k}{k+1}},\ k = 1, 2, \dots), \\ \xi(1) & (1 - \frac{1}{\sqrt{2}} \leq c \leq 1/2), \\ 1/c & (c \geq 1/2), \end{cases} \tag{2}$$

where

$$\eta(k) = \frac{k(c+1) + \sqrt{k^2(1-c)^2 + 4k}}{2k(1-kc)}, \quad \xi(k) = \frac{1}{2} + \frac{1}{2}\sqrt{1 + \frac{4}{kc}}. \tag{3}$$

The main ideas of our algorithms for both models are: i) we may reject items (with no cost) many times, but in at most one round, we remove items which from the knapsack. ii) some items are removed from the knapsack, only when the total value in the resulting knapsack gets high enough to guarantee the optimal competitive ratio.

The rest of the paper is organized as follows. In the next section, we consider the proportional cost model, and in Section 3, we consider the unit cost model. Due to the space limitation, some of the proofs are omitted.

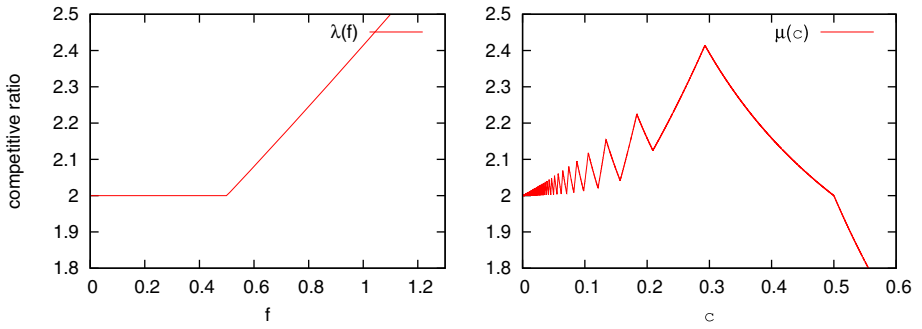


Fig. 1. The competitive ratios $\lambda(f)$ and $\mu(c)$ for the proportional and unit cost models

2 Proportional Cost Model

In this section, we consider the proportional cost model, where each item u_i has removal cost $f \cdot s(u_i)$ for some positive constant f. We first show that $\lambda(f)$ is a lower bound of the competitive ratio of the problem, and then propose a $\lambda(f)$-competitive algorithm, where $\lambda(f)$ is given in (1).

2.1 Lower Bound

In this subsection, we show a lower bound of the competitive ratio $\lambda(f)$ for the problem.

Theorem 1. *There exists no online algorithm with competitive ratio strictly less than $\rho = \lambda(f)$ for the knapsack problem with proportional removal cost, when the value of each item is equal to the size.*

Proof. According to the value of f, we separately consider the following two cases.

Case 1: $1/2 \geq f > 0$. Let A denote an online algorithm chosen arbitrarily. For a sufficiently small $\varepsilon\,(> 0)$, our adversary requests the sequence of items whose sizes are

$$\frac{1}{2}+\varepsilon,\ \frac{1}{2}+\frac{\varepsilon}{2},\ \ldots,\ \frac{1}{2}+\frac{\varepsilon}{\lceil 1/f\rceil+1}, \tag{4}$$

until A rejects some item in (4). If A rejects the item with size $\frac{1}{2}+\varepsilon$, then the adversary stops the input sequence. On the other hand, if it rejects the item with size $\frac{1}{2}+\frac{\varepsilon}{k}$ for some $k > 1$, then the adversary requests an item with size $\frac{1}{2}-\frac{\varepsilon}{k}$ and stops the input sequence.

We first note that algorithm A must take the first item, since otherwise the competitive ratio of A becomes infinite. After the first round, A always keeps exactly one item in the knapsack, since all the items in (4) have size larger than $\frac{1}{2}$ (i.e., a half of the knapsack capacity) and for any $j < k$ we have $(\frac{1}{2}+\frac{\varepsilon}{j})+(\frac{1}{2}-\frac{\varepsilon}{k})$ is larger than 1. This implies that A removes the old item from the knapsack to accept a new item. If A rejects $\frac{1}{2}+\frac{\varepsilon}{k}$ for some $k > 1$, the competitive ratio is at least $1/\left(\frac{1}{2}+\frac{\varepsilon}{k}\right)$, which approaches 2 as $\varepsilon \to 0$. Finally, if A rejects no item in (4), then its profit is

$$\frac{1}{2}+\frac{\varepsilon}{\lceil 1/f\rceil+1}-f\sum_{k=1}^{\lceil 1/f\rceil}\left(\frac{1}{2}+\frac{\varepsilon}{k}\right) \leq \frac{1}{2}-f\sum_{i=1}^{\lceil 1/f\rceil}\frac{1}{2} \leq 0 \tag{5}$$

while the optimal profit for the offline problem is $\frac{1}{2}+\varepsilon$, which completes the proof for $1/2 \geq f > 0$.

Case 2: $f > 1/2$. Let A denote an online algorithm chosen arbitrarily, and let $x = \frac{3+f-\sqrt{f^2+2f+5}}{2(1+f)}$. For a sufficiently small $\varepsilon\,(> 0)$, our adversary requests the following sequence of items

$$x,\ 1-x+\varepsilon,\ 1-x, \tag{6}$$

until A rejects some item in (6), and if A rejects the item then the adversary immediately stops the input sequence.

Note that A must accept the first item x, since otherwise the competitive ratio becomes infinite. If A rejects the second item, then the competitive ratio is at least

$$\frac{1-x+\varepsilon}{x} \geq \frac{1-x}{x} = \lambda(f). \tag{7}$$

If A takes the second item $1-x+\varepsilon$ (and removes the first item),the competitive ratio is at least $\frac{1}{1-x+\varepsilon-f\cdot x}$, which approaches to $\lambda(f)\,(=\frac{1}{1-x-f\cdot x})$ as $\varepsilon \to 0$, which completes the proof for $f > 1/2$. □

2.2 Upper Bound

In this subsection, we propose a $\lambda(f)$-competitive algorithm. Note that the total profit becomes small (even negative), if we remove items from the knapsack many times. Intuitively, our algorithm accepts the item if the knapsack has room to put it. If we can make the profit sufficiently high by accepting the item and removing some items from the current knapsack, then our algorithm follows this, and after this iteration, it rejects all the items. Otherwise, we simply rejects the item.

Let $\rho = \lambda(f)$, and let u_i be the item given in the ith round. Define by B_{i-1} the set of items in the knapsack at the beginning of ith round, and by $s(B_{i-1})$ the total size in B_{i-1}.

Algorithm 1.

1. **if** $s(u_i) + s(B_{i-1}) \leq 1$ **then** $B_i \leftarrow B_{i-1} \cup \{u_i\}$ and **if** $s(B_i) \geq 1/\rho$ **then** STOP
2. **else if** $\exists B'_i \subseteq B_{i-1}$ s.t. $\frac{1}{\rho} + f \cdot (s(B_{i-1}) - s(B'_i)) < s(B'_i) + s(u_i) \leq 1$ **then** $B_i \leftarrow B'_i \cup \{u_i\}$ and STOP
3. **else** $B_i \leftarrow B_{i-1}$

Here STOP denotes that the algorithm rejects the items after this round.

Lemma 2. *If $s(u_i) + s(B_{i-1}) > 1$ and some $B'_i \subseteq B_{i-1}$ satisfies $\rho \cdot s(B_{i-1}) < s(B'_i) + s(u_i) \leq 1$, then the second line is executed in the ith round.*

Proof. Since $s(u_i) + s(B_{i-1}) > 1$ and $\rho \cdot s(B_{i-1}) < s(B'_i) + s(u_i)$, we obtain

$$\begin{aligned}
\frac{1}{\rho} + f \cdot (s(B_{i-1}) - s(B'_i)) &< \frac{s(u_i) + s(B_{i-1})}{\rho} + f \cdot (s(B_{i-1}) - s(B'_i)) \\
&< \frac{s(u_i)}{\rho} + \frac{s(B'_i) + s(u_i)}{\rho^2} + f \cdot \frac{s(B'_i) + s(u_i)}{\rho} - f s(B'_i) \\
&= \frac{1 + f\rho - f\rho^2}{\rho^2} s(B'_i) + \frac{1 + f\rho + \rho}{\rho^2} s(u_i).
\end{aligned} \tag{8}$$

As $\rho^2 \geq 1 + f\rho + \rho$ by the definition of ρ, we have

$$\frac{1 + f\rho - f\rho^2}{\rho^2} \leq \frac{1 + f\rho - f\rho^2}{1 + f\rho + \rho} < 1 \quad \text{and} \quad \frac{1 + f\rho + \rho}{\rho^2} \leq 1. \tag{9}$$

□

Let OPT denote an optimal solution for the offline problem whose input sequence is $u_1, \ldots, u_i$.

Lemma 3. *If $s(B_i) < 1/\rho$ then we have $|\mathrm{OPT} \setminus B_i| \leq 1$.*

Proof. B_i contains all the items smaller than $1/2$, since $s(B_i) < 1/\rho \leq 1/2$. Any item $u \in \mathrm{OPT} \setminus B_i$ has size greater than $1 - 1/\rho \geq 1/2$. Therefore, $|\mathrm{OPT} \setminus B_i| \leq 1$ holds by $s(\mathrm{OPT}) \leq 1$. □

Theorem 4. *The online algorithm given in this section is $\lambda(f)$-competitive.*

Proof. Suppose that the second line is executed in round k. Then it holds that $\frac{1}{\rho} + f \cdot (s(B_{k-1}) - s(B'_k)) < s(B'_k) + s(u_k) = s(B_k)$ holds. Since $s(B_i) = s(B_k)$ holds for all $i \geq k$, we have

$$\frac{s(\mathrm{OPT})}{s(B_i) - f \cdot (s(B_{k-1}) - s(B'_k))} \leq \frac{1}{s(B_k) - f \cdot (s(B_{k-1}) - s(B'_k))} < \rho\,(= \lambda(f)). \tag{10}$$

We next assume that the second line has never been executed. If $s(B_i) \geq 1/\rho$, we have the competitive ratio $s(\mathrm{OPT})/s(B_i) \leq 1/s(B_i) \leq \rho$. On the other hand, if $s(B_i) < 1/\rho$, $|\mathrm{OPT} \setminus B_i| = 0$ or 1 holds by Lemma 3. If $|\mathrm{OPT} \setminus B_i| = 0$, we obtain the competitive ratio 1. Otherwise (i.e., $\mathrm{OPT} \setminus B_i = \{u_k\}$ for some k), Lemma 2 implies that $\rho \cdot s(B_{k-1}) \geq s(B'_k) + s(u_k)$ for $B'_k = \mathrm{OPT} \cap B_{k-1}$ Therefore we obtain

$$\begin{aligned} \frac{s(\mathrm{OPT})}{s(B_i)} &\leq \frac{s(B'_k) + s(u_k) + s(B_i \setminus B_{k-1})}{s(B_{k-1}) + s(B_i \setminus B_{k-1})} \\ &\leq \max\left\{ \frac{s(B'_k) + s(u_k)}{s(B_{k-1})}, \frac{s(B_i \setminus B_{k-1})}{s(B_i \setminus B_{k-1})} \right\} \leq \rho\,(= \lambda(f)). \end{aligned} \tag{11}$$

□

Before concluding this section, we remark that the condition in the second line can be checked in $O(|B_{i-1}| + 2^{\rho^2})$ time.

3 Unit Cost Model

In this section, we consider the unit cost model, where it costs us a fixed constant $c > 0$ to remove each item from the knapsack. Recall that every item has size at least c. In this section, we show that the knapsack problem with unit cost

is $\mu(c)$-competitive, where $\mu(c)$ is defined in (2). We note that $\mu(c)$ attains the maximum $1+\sqrt{2}$ when $c = 1 - 1/\sqrt{2}$.

Remark: If items are allowed to have size smaller than c, the problem becomes unbounded competitive. To see this, for a positive number r, let ε denote a positive number such that $\varepsilon < 1/(\lceil 1/c \rceil \cdot r)$. For an online algorithm A chosen arbitrarily, our adversary keeps requesting the items with size ε, until A accepts $\lceil 1/c \rceil$ items or rejects $r \cdot \lceil 1/c \rceil$ items. If A rejects $r \cdot \lceil 1/c \rceil$ items (before accepting $\lceil 1/c \rceil$ items), the adversary stops the input sequence; otherwise, it requests an item with size 1 and stops the input sequence. In the former case, the competitive ratio is at least $\frac{r\lceil 1/c \rceil \varepsilon}{\lceil 1/c \rceil \varepsilon} = r$. In the latter case, the competitive ratio becomes $\frac{1}{\lceil 1/c \rceil \cdot \varepsilon} > r$ if A rejects the last item (with size 1). Otherwise, A removes the $\lceil 1/c \rceil$ items to take the last item. This implies that the profit is $1 - \lceil 1/c \rceil \cdot c \leq 0$. Therefore, without the assumption, no online algorithm attains a bounded competitive ratio.

3.1 The Case $c \geq 1/2$

We first consider the case where $c \geq 1/2$. In this case, it is not difficult to see that the problem is $1/c\,(= \mu(c))$-competitive.

Theorem 5. *If the unit removal cost c of the knapsack problem is at least $1/2$, then there exists no online algorithm with competitive ratio strictly less than $1/c$ for the problem when the value of each item is equal to the size.*

Proof. For an online algorithm A chosen arbitrarily, our adversary first requests an item with size c. If A does not accept it, the adversary stop the input sequence. Otherwise, it next request an item with size 1 and stop the input sequence. It is clear that A must take the first item, since otherwise the competitive ratio becomes infinite. If A rejects the second item, then we have the competitive ratio $1/c$. Otherwise (i.e., A accepts the second item by removing the first item), the competitive ratio is $1/(1-c) \geq 1/c$, since $c \geq 1/2$. □

Theorem 6. *There exists a $1/c$-competitive algorithm for the knapsack problem with unit removal cost, when the value of each item is equal to the size.*

Proof. Consider an online algorithm which takes the first item u_1 and rejects the remaining items. Since $s(u_1) \geq c$ and the optimal value of the offline problem is at most 1, the competitive ratio is at most $1/c$. □

3.2 The Case $c < 1/2$

In this section we consider the case in which $c < 1/2$.

3.2.1 Lower Bound

For $0 < c < 1/2$, we show that $\mu(c)$ is a lower bound of the competitive ratio for the problem by starting with several propositions needed later.

Proposition 7. *For any positive integer k, we have*

$$\frac{1}{2k+4} < 1 - \sqrt{\frac{k+1}{k+2}}, \quad 1 - \sqrt{\frac{k}{k+1}} < \frac{1}{2k+1}. \tag{12}$$

Definition 8. *We define x_k and y_k as follows:*

$$x_k = \frac{k+2-kc-\sqrt{k^2(1-c)^2+4k}}{2}, \quad y_k = \frac{kc+\sqrt{k^2c^2+4kc}}{2}. \tag{13}$$

Proposition 9. *$\eta(k)$ and $\xi(k)$ in (3) satisfy the following equalities.*

$$\eta(k) = \frac{1}{1-x_k-kc} = \frac{1-x_k}{kx_k} = \frac{k(c+1)+\sqrt{k^2(1-c)^2+4k}}{2k(1-kc)}, \tag{14}$$

$$\xi(k) = \frac{1}{y_k-kc} = \frac{y_k}{kc} = \frac{1}{2} + \frac{1}{2}\sqrt{1+\frac{4}{kc}}. \tag{15}$$

We provide two kinds of adversaries.

Theorem 10. *Assume that removal cost c satisfies $1-\sqrt{\frac{k+1}{k+2}} \le c \le 1-\sqrt{\frac{k}{k+1}}$ for a positive integer k. Then there exists no online algorithm with competitive ratio strictly less than $\eta(k)$ for the knapsack problem with unit removal cost, when the value of each item is equal to the size.*

Proof. Let $x_k = \frac{k+2-kc-\sqrt{k^2(1-c)^2+4k}}{2}$. For an online algorithm A chosen arbitrarily, our adversary keeps requesting the items with size x_k until A accepts k items or rejects $\lceil 1/x_k \rceil$ items. If A rejects $\lceil 1/x_k \rceil$ items before accepting k items, the adversary stops the input sequence (1). Otherwise (i.e., A accepts k items), then the adversary next requests an item with size $1-x_k+\varepsilon$ where ε is a sufficiently small positive number; if A rejects it, the adversary stops the input sequence (2), and otherwise, the adversary next requests an item with size $1-x_k$ and stops the input sequence (3). Note that all the items have size at least c, since $1-\sqrt{\frac{k+1}{k+2}} \le c \le 1-\sqrt{\frac{k}{k+1}}$ implies $x_k \ge c$ and $1-x_k \ge c$.

In the case of (1), we have the competitive ratio at least $\frac{1-x_k}{(k-1)x_k} > \frac{1-x_k}{kx_k} = \eta(k)$, where the last equality follows from Proposition 9. In the case of (2), the competitive ratio is at least $\frac{1-x_k+\varepsilon}{kx_k} > \frac{1-x_k}{kx_k} = \eta(k)$ by Proposition 9. Finally, in the case of (3), the competitive ratio is at least $\frac{1}{1-x_k+\varepsilon-kc}$. Proposition 9 implies that this approaches $\eta(k)$ $(= \frac{1}{1-x_k-kc})$ as $(\varepsilon \to 0)$. □

Theorem 11. *Assume that removal cost c satisfies $1-\sqrt{\frac{k}{k+1}} \le c < \frac{1}{2k}$ for a positive integer k. Then there exists no online algorithm with competitive ratio strictly less than $\xi(k)$ for the knapsack problem with unit removal cost, when the value of each item is equal to the size.*

Proof. Let A denote an online algorithm chosen arbitrarily. Then our adversary keeps requesting the items with size c until A accepts k items or rejects $\lceil 1/c \rceil$

items. If A rejects $\lceil 1/c \rceil$ items before accepting k items, the adversary stops the input sequence (1). Otherwise (i.e., A accepts k items), the adversary requests an item with size $y_k = \frac{kc+\sqrt{k^2c^2+4kc}}{2}$ which is at least $1-c > c$, since $1-\sqrt{\frac{k}{k+1}} \leq c < \frac{1}{2k}$; if A rejects it, the adversary stops the input sequence (2), and otherwise, the adversary requests an item with size $1-c$ and stops the input sequence (3).

In the case of (1), the competitive ratio is at least $\frac{1-c}{(k-1)c} \geq \frac{1}{kc} \geq \frac{y_k}{kc} = \xi(k)$, where the last equality follows from Proposition 9. In the case of (2), the competitive ratio is $\frac{y_k}{kc} = \xi(k)$ by Proposition 9. Finally, in the case of (3), the competitive ratio is at least $\frac{1}{y_k - kc} = \xi(k)$, which again follows from Proposition 9. □

By Theorems 10 and 11, it holds that $\mu(c)$ is a lower bound of the competitive ratio for $0 < c < 1/2$.

3.2.2 Upper Bound

In this subsection, we show $\mu(c)$ is also an upper bound for the competitive ratio of the problem when $0 < c < 1/2$. We start with several propositions needed later.

Proposition 12. *For a positive integer k, let c satisfy $0 < c \leq 1 - \sqrt{\frac{k}{k+1}}$. Then we have*

$$\max\{\eta(k), \xi(k+1)\} \geq 2. \tag{16}$$

Proposition 13. *For a positive integer k, let c satisfy $1 - \sqrt{\frac{k+1}{k+2}} \leq c \leq 1 - \sqrt{\frac{k}{k+1}}$. Then we have*

$$\max\left\{\max_{\alpha \in \{1,2,\ldots,k\}} \eta(\alpha), \xi(k+1)\right\} = \max\{\eta(k), \xi(k+1)\} = \mu(c). \tag{17}$$

Proposition 14. *For a positive integer k, let c satisfy $1 - \sqrt{\frac{k+1}{k+2}} \leq c \leq 1 - \sqrt{\frac{k}{k+1}}$. Then for any positive integer $\alpha \leq k$ and real $x \in (0, 1-\alpha c)$, it holds that*

$$\min\left\{\frac{1}{1-x-\alpha c}, \frac{1-x}{\alpha x}\right\} \leq \eta(\alpha) \leq \mu(c). \tag{18}$$

Proof. Since $\frac{1}{1-x-\alpha c}$ and $\frac{1-x}{\alpha x}$ are respectively monotone increasing and decreasing in x, the first inequality holds by Proposition 9. The second inequality is obtained by Proposition 13. □

Proposition 15. *For a positive integer k, let c satisfy $1 - \sqrt{\frac{k+1}{k+2}} \leq c \leq 1 - \sqrt{\frac{k}{k+1}}$. Then for any real $y \in ((k+1)c, 1]$, we have*

$$\min\left\{\frac{1}{y-(k+1)c}, \frac{y}{(k+1)c}\right\} \leq \xi(k+1) \leq \mu(c). \tag{19}$$

Proof. Since $\frac{1}{y-(k+1)c}$ and $\frac{y}{(k+1)c}$ are respectively monotone decreasing and increasing in y, the first inequality holds by Proposition 9. The second inequality follows from the definition of $\mu(c)$. □

We are now ready to prove that $\mu(c)$ is an upper bound for the competitive ratio. According to the size of c, we make use of two algorithms described below.

Theorem 16. *If* $1 - \frac{1}{\sqrt{2}} \leq c \leq \frac{1}{2}$, *there exists an online algorithm with competitive ratio* $\mu(c)$ *for the knapsack problem with unit removal cost, when the value of each item is equal to the size.*

Proof. We consider the following algorithm, where B_{i-1} denotes the set of items in the knapsack at the beginning of the ith round, and and $s(B_{i-1})$ denotes the total size in B_{i-1}. Let u_i be the item given in the ith round.

Algorithm 2.

1. **if** $s(B_{i-1}) + s(u_i) \leq 1$ **then** $B_i \leftarrow B_{i-1} \cup \{u_i\}$
2. **else if** $|B_{i-1}| = 1$ and $s(u_i) \geq \frac{c+\sqrt{c^2+4c}}{2}$ **then** $B_i \leftarrow \{u_i\}$ and STOP
3. **else** $B_i \leftarrow B_{i-1}$

Here STOP denotes that the algorithm rejects the items after this round.

Let OPT denote an optimal solution for the offline problem whose input sequence is $u_1, \ldots, u_i$. If the algorithm stops at the second line, the competitive ratio is at most $1/\left(\frac{c+\sqrt{c^2+4c}}{2} - c\right) = \frac{c+\sqrt{c^2+4c}}{2c} = \mu(c)$, since $s(\mathrm{OPT}) \leq 1$. Assume that the algorithm has never stopped at the second line and $|B_i| = 1$. Then if $s(B_i) \geq 1/2$, the competitive ratio is at most $\frac{1}{1/2} = 2 \leq \mu(c)$. Otherwise, the item in B_i has size smaller than $1/2$, while the item u_j with $j < i$ and $u_j \notin B_i$ has size at least $1/2$. This implies that $|\mathrm{OPT}| = 1$ and the competitive ratio is smaller than $\mu(c)$, since $s(B_i) \geq c$ and $s(\mathrm{OPT}) < \frac{c+\sqrt{c^2+4c}}{2}$. If the algorithm has never stopped at the second line and $|B_i| > 1$, the competitive ratio is at most $\frac{1}{2c} < \mu(c)$, since $c \geq 1 - 1/\sqrt{2} > 1/6$ implies $c + \sqrt{c^2 + 4c} > 1$. □

Theorem 17. *If* $1 - \sqrt{\frac{k+1}{k+2}} \leq c \leq 1 - \sqrt{\frac{k}{k+1}}$, *there exists an online algorithm with competitive ratio* $\mu(c)$ *for the knapsack problem with unit removal cost, when the value of each item is equal to the size.*

Proof. We show that the following algorithm satisfies the desired property. In the algorithm, let $B_{i-1} = \{b_1, b_2, \ldots, b_m\}$ be the set of items in the knapsack at the beginning of the ith round, such that $s(b_1) \geq s(b_2) \geq \cdots \geq s(b_m)$. Let u_i be the item given in the ith round.

Algorithm 3.

1. **if** $s(B_{i-1}) + s(u_i) \leq 1$ **then** $B_i \leftarrow B_{i-1} \cup \{u_i\}$
2. **else**
3. $\quad B'_{i-1} \leftarrow \emptyset$
4. $\quad$ **for** $j = 1$ **to** m **if** $s(B'_{i-1}) + s(b_j) \leq 1 - s(u_i)$ **then** $B'_{i-1} \leftarrow B'_{i-1} \cup \{b_j\}$
5. $\quad$ **if** $s(B'_{i-1}) + s(u_i) - |B_{i-1} \setminus B'_{i-1}|c \geq 1/\mu(c)$ **then** $B_i \leftarrow B'_{i-1} \cup \{u_i\}$ and STOP
6. $\quad$ **else** $B_i \leftarrow B_{i-1}$

Here STOP denotes that the algorithm rejects the items after this round.

Let OPT denote an optimal solution for the offline problem whose input sequence is $u_1, \ldots, u_i$. If the algorithm stops at the fifth line in round $l \leq i$, $s(B_i) = s(B_l) = s(B'_{l-1}) + s(u_l)$ and the profit of the algorithm is $s(B'_{l-1}) + s(u_l) - |B_{l-1} \setminus B'_{l-1}|c$. Therefore, the competitive ratio is at most $\frac{1}{s(B'_{l-1}) + s(u_l) - |B_{l-1} \setminus B'_{l-1}|c} \leq \mu(c)$, since $s(\mathrm{OPT}) \leq 1$. Otherwise, the algorithm has never removed old items from the knapsack. If $s(B_i) \geq 1/2$, then the competitive ratio is at most $\frac{1}{1/2} = 2 \leq \mu(c)$. On the other hand, if $s(B_i) < 1/2$, then any item in B_i has size at most $1/2$. while any item in $OPT \setminus B_i$ has size larger than $1/2$. This implies $|\mathrm{OPT} \setminus B_i| \leq 1$ by $s(\mathrm{OPT}) \leq 1$. If $|\mathrm{OPT} \setminus B_i| = 0$, then we have $\mathrm{OPT} = B_i$, which implies that the competitive ratio is 1. Thus we assume that $|\mathrm{OPT} \setminus B_i| = 1$. For the cardinality of B_i, we have $|B_i| \leq k + 1$, since any $b \in B_i$ satisfies $s(b) \geq c \geq 1 - \sqrt{\frac{k+1}{k+2}} \geq \frac{1}{2k+4}$, where the last inequality follows from Proposition 7. Since the algorithm has never removed items, $|B_l| \leq k + 1$ also holds for each l with $l \leq i$. Let

$$\{u_l\} = \mathrm{OPT} \setminus B_i,\ \alpha = |B_{l-1} \setminus B'_{l-1}|,\ x = 1 - (s(u_l) + s(B'_{l-1})). \tag{20}$$

Since $B_{l-1} \setminus B'_{l-1} \neq \emptyset$, we have

$$\alpha > 0 \ \text{ and } \ x < \sqrt{\frac{k}{k+1}} < 1 - \alpha c. \tag{21}$$

Since $s(B_i) = s(B_{l-1}) + s(B_i \setminus B_{l-1})$ and $s(\mathrm{OPT}) \leq s(u_l) + s(B_{l-1} \cap \mathrm{OPT}) + s(B_i \setminus B_{l-1})$, the competitive ratio is at most

$$\frac{s(u_l) + s(B_{l-1} \cap \mathrm{OPT}) + s(B_i \setminus B_{l-1})}{s(B_{l-1}) + s(B_i \setminus B_{l-1})} \leq \max\left\{ \frac{s(u_l) + s(B_{l-1} \cap \mathrm{OPT})}{s(B_{l-1})}, 1 \right\}.$$

We claim that $\frac{s(u_l) + s(B_{l-1} \cap \mathrm{OPT})}{s(B_{l-1})} \leq \mu(c)$.

Let $B_l = \{b_1, b_2, \ldots, b_m\}$ satisfy $s(b_1) \geq s(b_2) \geq \cdots \geq s(b_m)$. To see this claim, we separately consider the following two cases:

Case 1. Consider the case in which there exists $b_j \in B'_{l-1}$ such that $b_h \notin B'_{l-1}$ holds for some $h > j$. Let us take b_j as the largest such item, i.e., $b_j \in B'_{l-1}$ and $b_g \notin B'_{l-1}$ for all $g\,(< j)$.

In this case, we obtain the following inequalities:

$$\frac{s(u_l) + s(B_{l-1} \cap \mathrm{OPT})}{s(B_{l-1})} \leq \frac{s(b_h) + 1 - x}{s(b_h) + \alpha x} \leq \max\left\{ 1, \frac{1 - x}{\alpha x} \right\}. \tag{22}$$

Here the numerator and denominator in the left hand side of (22) respectively satisfy $s(u_l) + s(B_{l-1} \cap \mathrm{OPT}) \leq 1 < s(b_h) + s(u_l) + s(B'_{l-1}) = s(b_h) + 1 - x$ and $s(B_{l-1}) = s(B'_{l-1}) + s(B_{l-1} \setminus B'_{l-1}) \geq s(b_h) + \alpha x$, since $b_h \notin B'_{l-1}$ and $s(b) > x$ holds for any $b \in B_{l-1} \setminus B'_{l-1}$. Finally, we show $\frac{1-x}{\alpha x} \leq \mu(c)$, which completes the claim.

Since the algorithm has not stopped at the fifth line and $1-x-\alpha c > 0$ by (21), we have $\frac{1}{1-x-\alpha c} = \frac{1}{s(B'_{l-1})+s(u_l)-\alpha c} > \mu(c)$. Note that $\alpha \leq |B_{l-1} \setminus \{b_h\}| \leq k$, since $|B_{l-1}| \leq k+1$. Therefore, we obtain $\frac{1-x}{\alpha x} \leq \mu(c)$ by Proposition 14.

Case 2. We next consider the case in which $b_j \in B'_{l-1}$ implies $b_h \in B'_{l-1}$ for all $h\,(> j)$, i.e., B'_{l-1} consists of the $|B'_{l-1}|$ smallest items of B_{l-1}. Then we have $s(b) > 1 - s(u_l)$ for any $b \in B_{l-1} \setminus B'_{l-1}$. This implies $B_{l-1} \cap \mathrm{OPT} \subseteq B'_{l-1}$, and $s(B_{l-1} \setminus B'_{l-1}) > \alpha x$ holds by (20).

If $\alpha \leq k$, thus, the competitive ratio is at most

$$\frac{s(u_l) + s(B_{l-1} \cap \mathrm{OPT})}{s(B_{l-1})} \leq \frac{s(u_l) + s(B'_{l-1})}{s(B_{l-1} \setminus B'_{l-1})} \leq \frac{1-x}{\alpha x} \leq \mu(c), \tag{23}$$

where the last inequality follows from a similar argument to **Case 1**. On the other hand, if $\alpha = k+1$, let $y = s(u_l) + s(B'_{l-1})$. Then we have

$$\frac{s(u_l) + s(B_{l-1} \cap \mathrm{OPT})}{s(B_{l-1})} \leq \frac{y}{(k+1)c}, \tag{24}$$

where the inequality follows from the fact that $s(u_l) + s(B_{l-1} \cap \mathrm{OPT}) \leq s(u_l) + s(B'_{l-1}) = y$ and $s(B_{l-1}) \geq s(B'_{l-1}) \geq (k+1)c$, since $B_{l-1} \cap \mathrm{OPT} \subseteq B'_{l-1}$ and any item has size at least c. Finally, since $y > (k+1)c$ and the algorithm has not stopped at the fifth line, it holds that $\frac{1}{y-(k+1)c} = \frac{1}{s(B'_{l-1})+s(u_l)-(k+1)c} > \mu(c)$. This together with Proposition 15 implies $\frac{y}{(k+1)c} \leq \mu(c)$. □

References

1. Ashwinkumar, B.V.: Buyback Problem - Approximate Matroid Intersection with Cancellation Costs. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6755, pp. 379–390. Springer, Heidelberg (2011)
2. Ashwinkumar, B.V., Kleinberg, R.: Randomized Online Algorithms for the Buyback Problem. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 529–536. Springer, Heidelberg (2009)
3. Babaioff, M., Hartline, J.D., Kleinberg, R.D.: Selling ad campaigns: Online algorithms with cancellations. In: ACM Conference on Electronic Commerce, pp. 61–70 (2009)
4. Biyalogorsky, E., Carmon, Z., Fruchter, G.E., Gerstner, E.: Research note: Overselling with opportunistic cancellations. Marketing Science 18(4), 605–610 (1999)
5. Constantin, F., Feldman, J., Muthukrishnan, S., Pál, M.: An online mechanism for ad slot reservations with cancellations. In: SODA, pp. 1265–1274 (2009)
6. Han, X., Makino, K.: Online Minimization Knapsack Problem. In: Bampis, E., Jansen, K. (eds.) WAOA 2009. LNCS, vol. 5893, pp. 182–193. Springer, Heidelberg (2010)

7. Han, X., Makino, K.: Online removable knapsack with limited cuts. Theoretical Computer Science 411, 3956–3964 (2010)
8. Iwama, K., Taketomi, S.: Removable online knapsack problems. LNCS, pp. 293–305 (2002)
9. Iwama, K., Zhang, G.: Optimal Resource Augmentations for Online Knapsack. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) RANDOM 2007 and APPROX 2007. LNCS, vol. 4627, pp. 180–188. Springer, Heidelberg (2007)
10. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer (2004)
11. Lueker, G.S.: Average-case analysis of off-line and on-line knapsack problems. Journal of Algorithms 29(2), 277–305 (1998)
12. Marchetti-Spaccamela, A., Vercellis, C.: Stochastic on-line knapsack problems. Mathematical Programming 68, 73–104 (1995)
13. Noga, J., Sarbua, V.: An online partially fractional knapsack problem. In: ISPAN, pp. 108–112 (2005)

An Improved Exact Algorithm for TSP in Degree-4 Graphs*

Mingyu Xiao[1] and Hiroshi Nagamochi[2]

[1] School of Computer Science and Engineering,
University of Electronic Science and Technology of China, China
myxiao@gmail.com
[2] Graduate School of Informatics, Kyoto University, Japan
nag@amp.i.kyoto-u.ac.jp

Abstract. The paper presents an $O^*(1.716^n)$-time polynomial-space algorithm for the traveling salesman problem in an n-vertex edge-weighted graph with maximum degree 4, which improves the previous results of the $O^*(1.890^n)$-time polynomial-space algorithm by Eppstein and the $O^*(1.733^n)$-time exponential-space algorithm by Gebauer.

Keywords: Traveling Salesman Problem, Exact Exponential Algorithm, Graph Algorithm.

1 Introduction

The famous traveling salesman problem (TSP) was first formulated as a mathematical problem in 1930s and is one of the most intensively studied problems in optimization. A large number of heuristics and exact methods for it are known. A classical dynamic programming solution with running time $O^*(2^n)$ for it was discovered in early 1960s, where n is the number of vertices in the graph. Despite the great progress in the pass half a century on exact exponential algorithms and their worst-case analysis for other basic NP-hard optimization problems, such as the maximum independent set problem, the CNF satisfiability problem and so on, it seems very hard to break the barrier of 2 in the basic of the exponential part for TSP [8]. To make steps toward the long-standing open problem of designing an $O^*(1.99^n)$ algorithm for TSP, researchers have interests in finding fast solutions to the problem in special classes of graphs, especially degree bounded graphs. Eppstein [5] presented a branch-and-search method to solve TSP in $O^*(1.260^n)$ time for degree-3 graphs and in $O^*(1.890^n)$ time for degree-4 graphs. Both of the algorithms run in polynomial space. Later, Iwama and Nakashima [7] refined Eppstein's algorithm for degree-3 graphs and improved the result to $O^*(1.251^n)$. Gebauer [6] designed an $O^*(1.733^n)$-time exponential-space algorithm for TSP in degree-4 graphs. Bjorklund *et al.* [2] also showed TSP in degree bounded graph can be solved in $O^*((2-\varepsilon)^n)$ time, where $\varepsilon > 0$ depends on the degree bound only. There is a Monte Carlo algorithm to decide a

* Supported in part by Grant 60903007 of NSFC, China.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 74–85, 2012.

graph is Hamiltonian or not in $O^*(1.657^n)$ time [1]. We also note that there are sub-exponential algorithms for planar TSP and Euclidean TSP based on small separators [3]. In this paper, we present a deterministic branch-and-search algorithm for TSP in degree-4 graphs that runs in $O^*(1.716^n)$ time and improves previous exact algorithms.

Similar to most previous branch-and-search algorithms for TSP, the basic idea of our algorithm is to branch on an edge by either including it into the solution or not. To effectively analyze our algorithm, we use the measure and conquer method, in which we set a weight to each vertex in the graph to distinguish them and then analyze how much total weight can be reduced in each branch. In previous measure-and-conquer algorithms, we usually set the same weight to vertices of the same degree in a graph. However this setting may not be useful for branch-and-search algorithms for TSP. In our algorithm, we set two different weights for vertices of the same degree. Our result is the first time to successfully apply the measure and conquer method to TSP to get a nontrivial improvement.

The paper is organized as follows. Section 2 reviews basic properties of infeasible instances and a polynomially solvable case, and derives reduction rules. Section 3 sets a weight function on vertices for analyzing the time bound of our algorithm, and introduces a notion of "chains" which is used to simplify the case analysis for weight decrease in branching operations. After Section 4 describes a set of branching rules used in our algorithm, Section 5 analyzes some of the branching rules (the analysis of the rest of branching rules are omitted due to space limitation). Section 6 makes some concluding remarks.

2 Preliminaries

An instance $I = (G, F)$ consists of a simple undirected graph G with an edge cost and a subset $F \subseteq E$ of edges, called *forced*. A Hamiltonian cycle of G is called a *tour* if it passes though all the forced edges in F. We will consider a generation of TSP, named the *forced traveling salesman problem*, which asks to find a minimum cost tour of an instance (G, F).

For a graph H, let $V(H)$ and $E(H)$ denote the sets of vertices and edges in H, respectively.

For a vertex subset X (or a subgraph X), let $\partial_{\mathrm{f}}(X)$ (resp., $\partial_{\mathrm{u}}(X)$) denote the set of forced (resp., unforced) edges between X and $V(G) - X$. For an edge subset Y (or a subgraph Y), $d_Y(v)$ denote the *degree* of a vertex v in the graph $(V(G), Y))$ (or Y). For a subset $E' \subseteq E(G)$, a component in $(V(G), E')$ containing at least one edge is called an E'*-component*. Let $U = E(G) - F$ denote the set of unforced edges. For any U-component H, it holds $\partial_{\mathrm{u}}(H) = \emptyset$. A vertex is called *forced* if exactly one incident edge is forced, and is called *unforced* if no forced edge is incident to it. A neighbor u of a vertex v adjacent via an unforced edge uv is called a *good* neighbor.

2.1 Sufficient Conditions for Infeasibility

In general, whether (G, F) admits a tour or not is an NP-hard problem. We here introduce some sufficient conditions for an instance to be infeasible (i.e., there is no tour).

Lemma 1. *An instance (G, F) is infeasible if one of the following holds:*

(i) *G is not biconnected;*
(ii) *$d_F(v) \geq 3$ for some vertex $v \in V$;*
(iii) *$(V(G), F)$ contains a cycle shorter than a Hamiltonian cycle;*
(iv) *$(V(G), U)$ contains a U-component H such that $|\partial_{\mathrm{f}}(H)|$ is odd.*

Let us call an instance *pseudo-feasible* if none of the conditions in Lemma 1 holds.

2.2 A Solvable Case

A special case when TSP with forced edges is polynomially solvable is identified by Eppstein [5]. A U-component H is called an *i-cycle* U-component if H is a cycle of length i with no chord of unforced edge (possibly a forced edge joins two vertices in H).

Lemma 2. [5] *If every U-component is a 4-cycle U-component, then a minimum cost tour of the instance can be found in polynomial time.*

2.3 Reductions

A reduction is an operation that transforms an instance into a smaller instance without changing the optimality of the instance.

Lemma 3. *Each of the following transformations preserves a minimum cost tour of an instance.*

(i) *Remove any unforced edge incident to a vertex v with $d_F(v) = 2$;*
(ii) *Add to F any unforced edge incident to a vertex v with $d_G(v) = 2$;*
(iii) *Assume that* (i) *is no longer applicable. Let H be a U-component with a bridge uv where H will be separated into H_1 and H_2 by removing uv. Then remove uv from G if $|\partial_{\mathrm{f}}(H_1)|$ is even, and add uv to F otherwise.*
(iv) *For three vertices v_i of $d_G(v_i) = 3$, $i = 1, 2, 3$ in a 3-cycle $v_1v_2v_3$ with unforced edges v_1v_2 and v_3v_1 such that v_2v_3 or t_1v_1 is a forced edge (where t_i denotes the neighbor of v_i not in the 3-cycle), remove vertices v_2 and v_3 and add two new unforced edges v_1t_2 and v_1t_3 such that the cost of v_1t_2 (resp., v_1t_3) is the sum of those of v_1v_3 and v_2t_2 (v_1v_2 and v_3t_3).*

Proof. (i)-(iii) Immediate from the definition of tours.

(iv) Since $d_G(v_i) = 3$ for all $i = 1, 2, 3$ and v_2v_3 or t_1v_1 is in F, any tour must pass either $\{v_1v_2, v_2v_3, v_3t_3\}$ or $\{v_1v_3, v_3v_2, v_2t_3\}$. Therefore, the resulting instance preserves the feasibility and the optimality of tours. ∎

For an edge e in an instance $I = (G, F)$, `delete`(e) denotes an operation of removing e from the graph G followed by applications of reductions as much as possible. If the resulting irreducible instance is not pseudo-feasible, then e is called *f-reducible*, which means that every tour must pass through e. Similarly, `force`(e) denotes an operation of adding e to F followed by applications of reductions as much as possible. If the resulting irreducible instance is not pseudo-feasible, then e is called *d-reducible*, which means that e is not contained any tour. Note that we can test whether an instance has an f- or d- reducible edge or not in polynomial time.

A pseudo-feasible instance is called *irreducible* if none of the reductions in Lemma 3 is applicable and there is no f- or d-reducible edge. In any irreducible instance, there is no vertex v with $d_G(v) \leq 1$, and every F-component is a path P to which no unforced edge is incident except for the end points of P. Hence, we regard each F-component as a single edge joining its end point unless confusion arises. Then F-components form a matching in the resulting graph wherein the degree of every vertex is at least 3..

3 Weight Setting

In what follows, we assume that the maximum degree in G is at most 4. For simplicity, we call a forced (resp., unforced) degree-i vertex an fi*-vertex* (resp., ui*-vertex*). Let (G, F) be an irreducible instance, which has four kinds of vertices u4-, f4-, u3- and f3-vertices.

We solve a given irreducible instance $I = (G, F)$ by a search tree algorithm. We first choose an unforced edge uv in $= (G, F)$; branch on it; i.e., generate two cases (called *branches*) by adding uv to F or by removing uv from G; and apply reductions as much as possible to obtain two new smaller irreducible instances I' and I''. Thus, we execute `force`(uv) and `delete`(uv) to obtain such new instances, where both I' and I'' are pseudo-feasible (otherwise I would have a d- or f-reducible edge). The performance of the algorithm depends on how to choose an "effective" unforced edge uv so that the size of the new instances decreases quickly. We discuss how to choose such unforced edges to branch on in the next section.

To efficiently analyze our search tree algorithms, we adopt an amortized transfer technique, called the measure and conquer method. We set a vertex weight function $\omega : V \to \mathbb{R}_+$ in the graph and use the sum $W = \sum_{v \in V(G)} \omega(v)$ of the weight of all vertices in the graph as the measure. Since we will require the weight of each vertex is not greater than 1, the measure W is not greater than the number n of vertices. A running time bound related to measure W will imply a running time bound related to n.

For irreducible instances, we set weights of these vertices as follows. Since we know that an instance with only 4-cycle U-components will be solved in polynomial time, we set the weight of any vertices in a 4-cycle U-component

to be 0. In U-components which are not 4-cycle U-components, let w_4 be the weight of a u4-vertex v, $w_{4'}$ be the weight of a f4-vertex, w_3 be the weight of a u3-vertex, and $w_{3'}$ be the weight of an f3-vertex. We require that $w_4 = 1$, $0.5 \geq w_3 \geq 0.4$, $w_{3'} = 0.5w_3$,

$$w_{4'} + 0.5w_3 \leq 1, \tag{1}$$

$$w_{4'} \geq 1.5w_3. \tag{2}$$

Denote $\Delta_4 = w_4 - w_{4'}$, $\Delta_3 = w_3 - w_{3'}$, $\Delta_{4-3} = w_4 - w_3$ and $\Delta'_{4-3} = w_{4'} - w_{3'}$. Then (1) and (2) imply

$$\Delta_4 \geq \Delta_3 = 0.5w_3 \text{ and } \Delta_{4-3} \geq \Delta'_{4-3} \geq w_3.$$

We only need to decide the value of w_3 and $w_{4'}$, and then we can decide the weight of all vertices.

A path P consisting of f3-vertices and unforced edges is called a *chain*. Note that when the first edge in a chain is added to F (or deleted from the graph) the other edges in the chain will be alternately deleted from the graph or added to F by Lemma 3(i)-(ii).

Let uv be an unforced edge. When v is an f3-vertex, we consider a maximal chain P starting from v, but not containing edge uv. We say that such a chain *starts from* (uv, v). In an irreducible instance, P ends with an f3-vertex y such that $y = u$ or y is a good neighbor of a non-f3-vertex z. In the former case, u is an f3-vertex, and P and uv form a cycle U-component. In the latter case, we say that P *reaches* z, and denote z by $\gamma(uv, v)$, and let $\Gamma(uv, v)$ denote the set of vertex $z = \gamma(uv, v)$ and the f3-vertices in P. We observe that `delete`(uv) decreases the weight of vertices in $\Gamma(uv, v)$ by at least $w_{3'} + kw_{3'} + 0.5w_3 = (1 + k/2)w_3$, where k is the number of f3-vertices in $\Gamma(uv, v) - \{v\}$ and the weight of z decreases by at least $\min\{\Delta_{4-3}, \Delta'_{4-3}, \Delta_4, \Delta_3\} = 0.5w_3$.

When v is not an f3-vertex, we define $\Gamma(uv, v) = \{v\}$ for a u4- or f4-vertex v and $\Gamma(uv, v) = \{v, v', v''\}$ for a u3-vertex v with its good neighbors v', v'' $(\neq u)$. Note that a chain joins two vertices u and v means that these vertices are both f3-vertices.

Lemma 4. *Let $e_1 = u_1v_1$ and $e_2 = u_2v_2$ be two unforced edges in G such that $v_i \neq u_j, v_j$ for $\{i, j\} = \{1, 2\}$, and v_1 and v_2 are not joined by a chain in $G - \{e_1, e_2\}$. Then the weight of vertices in $\Gamma(e_1, v_1) \cup \Gamma(e_2, v_2)$ by at least $(2 + k/2)w_3$ by removing edges e_1 and e_2 and applying reduction rules, where k is the number of f3-vertices in $\Gamma(e_1, v_1) \cup \Gamma(e_2, v_2) - \{v_1, v_2\}$.*

Proof. If v_i is a degree-4 vertex, then the weight of $\Gamma(e_i, v_i) = \{v_i\}$ decreases by $\min\{\Delta_{4-3}, \Delta'_{4-3}\} \geq w_3$. If v_i is a u3-vertex, then the weight of the three vertices in $\Gamma(e_i, v_i)$ decreases by $w_3 + 2\min\{\Delta_4, \Delta_3\} = 2w_3$.

(i) Both v_1 and v_2 are non-f3-vertices: From the above observation, the weight of vertices in $\Gamma(e_1, v_1) \cup \Gamma(e_2, v_2)$ decreases by $2w_3$, where $k = 0$.

(ii) Both v_1 and v_2 are f3-vertices: For each $i = 1, 2$, the weight of vertices in $\Gamma(e_i, v_i)$ decreases by at least $(1 + k_i/2)w_3$, where k_i is the number of f3-vertices

in $\Gamma(e_i, v_i) - \{v_i\}$. Hence $\Gamma(e_1, v_1) \cup \Gamma(e_2, v_2)$ decreases by $(1 + k_1/2)w_3 + (1 + k_2/2)w_3 = (2 + k/2)w_3$ even if $\gamma(e_1, v_1) = \gamma(e_2, v_2)$ (since the weight of a non-f3-vertex $\gamma(e_1, v_1) = \gamma(e_2, v_2)$ is at least $w_3 \geq 2\Delta_3$).

(iii) Exactly one of v_1 and v_2, say v_2 is an f3-vertex: If $\gamma(e_2, v_2) \neq v_1$, then the weight of v_1 and the vertices in $\Gamma(e_1, v_1) \cup \Gamma(e_2, v_2)$ decreases by at least $w_3 + (1 + k/2)w_3 = (2 + k/2)w_3$. If $\gamma(e_2, v_2) = v_1$ holds and v_1 is a degree-4 vertex, then the weight of v_1 and the vertices in $\Gamma(e_1, v_1) \cup \Gamma(e_2, v_2)$ decreases by at least $\min\{w_{4'}, 1-w_{3'}\}+(0.5+k/2)w_3 \geq w_{4'}+(0.5+k/2)w_3 \geq (2+k/2)w_3$ (by (1) and (2)). If $\gamma(e_2, v_2) = v_1$ holds and v_1 is a u3-vertex, then v_1 has at least one good neighbor $y \in \Gamma(e_1, v_1)$ which is not in the chain from (e_2, v_2), and the weight of v_1, y, v_2 and vertices in $\Gamma(e_2, v_2) - \{v_1, v_2\}$ decreases by at least $w_3 + \Delta_3 + w_{3'} + kw_{3'} = (2 + k/2)w_3$. ∎

4 Branching Rules

We are now ready to describe a set of branching rules in our algorithm for solving TSP in graphs with maximum degree 4. Given an irreducible instance (G, F), our algorithm first selects an f4-vertex v in (G, F), and branches on an unforced edge e incident to v. Such a pair of vertex v and edge e is chosen as the one satisfying one of the conditions in Case-1,2,3 and 4 in Fig. 1. When there is no f4-vertex in (G, F), our algorithm selects an f3-vertex v in a U-component H that is not a 4-cycle component, and branches on an unforced edge e in H according to the one satisfying one of the conditions in Case-5 to -10 in Fig. 1 (recall that the instance is polynomially solvable when every U-component is a 4-cycle component). Also see Fig. 2 for illustrations for Cases-1 to 10.

For each Case-i, we execute the procedure in Case-i only when there is no vertex v satisfying the condition of Case-j with $j < i$. For an f4-vertex (resp., f3-vertex) v in Fig. 1, we always let t_1, t_2 and t_3 (resp., t_1 and t_2) denote the good neighbors of v.

For each Case-i, we derive a recurrence that evaluates how much weight decreases in each of the two instances obtained by branching on an edge e.

5 Cases-1,2 and 3 of Branching at f4-vertices

This section analyzes Cases-1, 2, and 3 for branching at an f4-vertex. The analysis for Cases-4 to -10 can be obtained in a similar manner (the detail is omitted due to space limitation). We prove the following lemma.

Lemma 5. *We can branch on edge e in Case-1,2,3 and 4 with a recurrence which is not worse than one of the following:*

$$C(w) \leq C(w - (w_{4'} + 2.5w_3)) + C(w - (w_{4'} + 1.5w_3)); \tag{3}$$

/* while there is an f4-vertex in an irreducible instance (G, F), choose an f4-vertex v in Cases-1,2,3, and 4, where the three good neighbors of v are denoted by t_1, t_2 and t_3 */

Case-1. v has a u3-vertex as one (say t_1) of its good neighbors of v: Branch on edge $e = vt_1$;

Case-2. v has an f4-vertex as one (say t_1) of its good neighbors of v (where t_4, t_5 ($\neq v$) denote the other good neighbors of t_1): (I) **if** t_2, t_3, t_4 and t_5 are four distinct f3-vertices t_2 (resp., t_3) is a good neighbor of t_4 (resp., t_5) **then** Branch on edge $e = vt_3$; (II) **else** Branch on edge $e = vt_1$;

Case-3. v has a u4-vertex as one (say t_1) of its good neighbors of v: (I) **if** v has an f3-vertex t_2 as one of its good neighbors (where t_2 is assumed to be a good neighbor of t_1 if one of f3-vertices t_2 and t_3 is a good neighbor of t_1) **then** Branch on edge $e = vt_2$; (II) **else** Branch on edge $e = vt_1$;

Case-4. all good neighbors t_1, t_2 and t_3 of v are f3-vertices: Branch on edge $e = vt_1$;

/* when there is no f4-vertex in an irreducible instance (G, F), choose an f3-vertex v in Cases-5,6,7,8,9 and 10, where the two good neighbors of v are denoted by t_1 and t_2 */

Case-5. the two good neighbors t_1 and t_2 of v are both u3-vertices: Branch on edge $e = vt_1$;

Case-6. v is in a U-component that is not a 4-cycle component and the two good neighbors t_1 and t_2 of v are both f3-vertices: Branch on edge $e = vt_1$;

Case-7. v has a u4-vertex and an f3-vertex as its good neighbors t_1 and t_2 of v, respectively: Branch on edge $e = vt_1$;

Case-8. v has an f3-vertex and a u3-vertex as its good neighbors t_1 and t_2 of v, respectively (where y_1 ($\neq v$) denotes the other good neighbor of t_1): (I) **if** t_1 and t_2 have a common good neighbor y_1 ($\neq v$) **then** Branch on edge $e = y_1z$ for a good neighbor z ($\neq t_1, t_2$) of y_1; (II) **else** Branch on edge $e = vt_1$;

Case-9. v has a u4-vertex and a u3-vertex as its good neighbors t_1 and t_2 of v, respectively: (I) **if** t_1 is a good neighbor of t_2 **then** Branch on edge $e = t_1x$ for a good neighbor x ($\neq v, t_2$) of t_1; (II) **else** Branch on edge $e = vt_1$;

Case-10: all good neighbors t_1 and t_2 of v are u4-vertices: Branch on edge $e = vt_1$.

Fig. 1. Branching rules

$$C(w) \leq C(w - (1 + w_{4'} + 1.5w_3)) + C(w - (w_{4'} + 0.5w_3)); \tag{4}$$

$$C(w) \leq C(w - (3 - 2w_3)) + C(w - (1 + w_{4'} - 1.5w_3)); \tag{5}$$

$$C(w) \leq C(w - (w_{4'} + 3.5w_3)) + C(w - (w_{4'} + 0.5w_3)). \tag{6}$$

In Cases-1,2,3 and 4, we let H denote the U-component containing v, and t_1, t_2 and t_3 denote the good neighbor of v. Two good neighbors t_i and t_j of v are

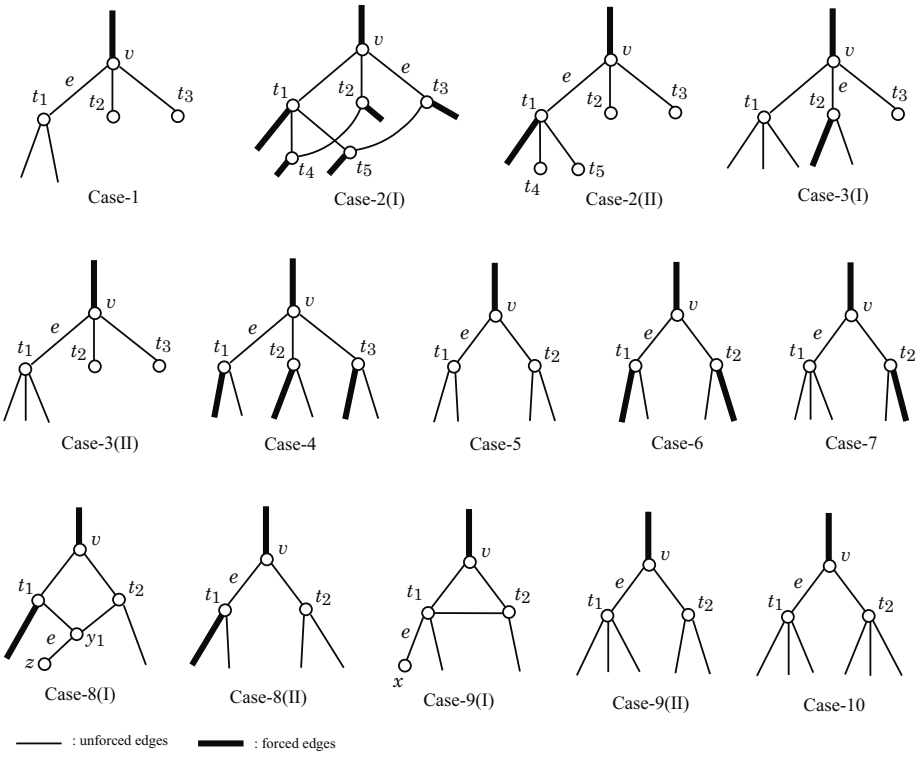


Fig. 2. Illustration for the branching rules in Case-1 to 10

not joined by a chain since otherwise edge vt_k with $k \neq i, j$ would be d- or f-reducible. Hence t_i and t_j are not adjacent via an unforced edge if they are f3-vertices.

Case-1. At least one of good neighbors of v, say t_1 is a u3-vertex: We branch on edge vt_1.

In the branch of `force`(vt_1), we also delete edges vt_2 and vt_3 from G. This decreases the weight of v and t_1 by at least $w_{4'} + \Delta_3$. Recall that t_2 and t_3 are not joined by a chain if they are f3-vertices. By Lemma 4 the weight of vertices in $\Gamma(vt_2, t_2) \cup \Gamma(vt_3, t_3)$ decreases by at least $2w_3$. If $t_1 \notin \Gamma(vt_2, t_2) \cup \Gamma(vt_3, t_3)$ or $\Gamma(vt_2, t_2) \cup \Gamma(vt_3, t_3) - \{t_2, t_3\}$ contains an f3-vertex, then `force`(vt_1) decreases the entire weight by at least $w_{4'} + w_{3'} + 2w_3 = w_{4'} + 2.5w_3$. Assume that $\Gamma(vt_2, t_2) \cup \Gamma(vt_3, t_3) - \{t_2, t_3\}$ contains no f3-vertex and t_1 is contained in $\Gamma(vt_2, t_2) \cup \Gamma(vt_3, t_3)$. Then $t_1 \in \Gamma(vt_2, t_2) \cup \Gamma(vt_3, t_3)$ implies that the weight of t_1 decreases by w_3. If t_2 or t_3 (say t_3) is not an f3-vertex, then its weight decreases by $\min\{\Delta_{4-3}, \Delta'_{4-3}, w_3\} = w_3$, and the weight of v, t_1, t_2 and t_3 decreases by $w_{4'} + w_3 + w_{3'} + w_3 = w_{4'} + 2.5w_3$ in total. Let t_i with each $i = 2, 3$ be an f3-vertex, where $t_1 = \gamma(vt_2, t_2)$ is assumed without loss of generality since $t_1 \in \Gamma(vt_2, t_2) \cup \Gamma(vt_3, t_3)$. Then $t_1 \notin \Gamma(vt_3, t_3)$ holds, since

otherwise ($t_1 = \gamma(vt_2, t_2) = \gamma(vt_3, t_3)$) H contains only vertices v, t_1, t_2 and t_3 and $|\partial_{\mathrm{f}}(H)| = 3$ would hold (recall that $\Gamma(vt_2, t_2) \cup \Gamma(vt_3, t_3) - \{t_2, t_3\}$ contains no f3-vertex). Then the weight of v, t_1, t_2 and vertices in $\Gamma(vt_3, t_3)$ decreases by $w_{4'} + w_3 + w_{3'} + w_3 = w_{4'} + 2.5w_3$ in total. Therefore $\texttt{force}(vt_1)$ always decreases the entire weight by at least $w_{4'} + 2.5w_3$.

In the other branch of $\texttt{delete}(vt_1)$, the weight of v, t_1 and the two other good neighbors of t_1 decreases by $\Delta'_{4-3} + w_3 + \Delta_3 + \Delta_3 = w_{4'} + 1.5w_3$. Then we can get recurrence

$$C(w) \leq C(w - (w_{4'} + 2.5w_3)) + C(w - (w_{4'} + 1.5w_3)),$$

i.e., (3).

Case-2. One good neighbor t_1 of v is a f4-vertex, and neither of the other two t_2 and t_3 is a u3-vertex: Let t_4 and t_5 be the two other good neighbors of t_1 than v. We assume that neither of t_4 and t_5 is a u3-vertex, since otherwise Case-1 would be applicable to the f4-vertex t_1. Recall that t_2 and t_3 are not joined by a chain if they are f3-vertices. For the similar reason, t_4 and t_5 are not joined by a chain either. Note that two pairs $\{t_2, t_4\}$ and $\{t_3, t_5\}$ of f3-vertices are joined by chains P_2 and P_3, the U-component H containing vt_1 has no other unforced edges than $\{vt_1, vt_2, vt_3, t_1t_4, t_1t_5\} \cup E(P_2) \cup E(P_3)$.

(I) t_2, t_3, t_4 and t_5 are four distinct f3-vertices t_2 (resp., t_3) is a good neighbor of t_4 (resp., t_5): That is, two pairs $\{t_2, t_4\}$ and $\{t_3, t_5\}$ of f3-vertices are joined by chains P_2 and P_3 which contain no other f3-vertex. We branch on edge vt_3. Now it holds $V(H) = \{v, t_1, t_2, t_3, t_4, t_5\}$. In the branch of $\texttt{force}(vt_3)$, all unforced edges in H will be deleted or added to F, decreasing the entire weight by $2w_{4'} + 4w_{3'} = 2w_{4'} + 2w_3$. In the other branch of $\texttt{delete}(vt_3)$, we delete vt_3 and t_1t_5, and add t_3t_5 to F, creating a 4-cycle component from H. Thus this also decreases the entire weight by $2w_{4'} + 2w_3$. We get a recurrence $C(w) \leq 2C(w - (2w_{4'} + 2w_3))$, which is covered by (3), since $2w_{4'} + 2w_3 \geq w_{4'} + 2.5w_3$ and $2w_{4'} + 2w_3 \geq w_{4'} + 1.5w_3$ in the corresponding two terms.

(II) (i) $\{t_2, t_3\} \cap \{t_4, t_5\} = \emptyset$ holds, there are two pairs $\{t_2, t_4\}$ and $\{t_3, t_5\}$ of f3-vertices which are joined by chains P_2 and P_3, and there is another f3-vertex $y \neq t_i$ in P_2 and P_3: We branch on edge vt_1. In this case, there must be at least two such vertices y, say y_1 and y_2 (otherwise $|\partial_{\mathrm{f}}(H)|$ would be odd), and in the branch of $\texttt{force}(vt_1)$, the weight of y_1 and y_2 decreases by $2w_{3'}$. In total, $\texttt{force}(vt_1)$ decreases the entire weight by $2w_{4'} + 4w_{3'} + 2w_{3'} = 2w_{4'} + 3w_3$. In the other branch of $\texttt{delete}(vt_1)$, the weight of v and t_1 decreases by $2w_{4'} - w_3$. We get a recurrence

$$C(w) \leq C(w - (2w_{4'} + 3w_3)) + C(w - (2w_{4'} - w_3)),$$

which is covered by (5) since $2w_{4'} + 3w_3 \geq 3 - 2w_3$ and $2w_{4'} - w_3 \geq 1 + w_{4'} - 1.5w_3$.

(ii) $\{t_2, t_3\} \cap \{t_4, t_5\} = \emptyset$, and there is at most one pair of f3-vertices $t_a \in \{t_2, t_3\}$ and $t_b \in \{t_4, t_5\}$ which are joined by a chain: We branch on edge vt_1. In the branch of $\texttt{force}(vt_1)$, we also delete edges vt_2, vt_3, t_1t_4 and t_1t_5 from G. For $j, k \neq a, b$, vertices t_j and t_k are not joined by a chain. Deleting these

edges decreases the weight of vertices in $\Gamma(e_j, t_j) \cup \Gamma(e_k, t_k)$ decreases by $2w_3$ by Lemma 4. Also the weight of v, t_1 t_a and t_b decreases by at least $w_{4'} + w_{4'} + w_{3'} + w_{3'}$. In total, `force`$(vt_1)$ decreases the entire weight by $2w_{4'} + 3w_3$. In the other branch of `delete`(vt_1), the weight of v and t_1 decreases by $2w_{4'} - w_3$. Hence we get recurrence

$$C(w) \leq C(w - (2w_{4'} + 3w_3)) + C(w - (2w_{4'} - w_3)),$$

which is covered by (5).

(iii) $|\{t_2, t_3\} \cap \{t_4, t_5\}| = 1$: Let $t_2 = t_4$, which is of degree 4 (otherwise vt_1 would be d-reducible). We branch on edge vt_1. In the branch of `force`(vt_1), the weight of v, t_1, t_2, t_3 and t_5 decreases by at least $w_{4'} + w_{4'} + w_{4'} + w_{3'} + w_{3'} = 3w_{4'} + w_3$. In the branch of `delete`$(vt_1)$, the weight of v and t_1 decreases by $2\Delta'_{4-3} = 2w_{4'} - w_3$. Hence we have

$$C(w) \leq C(w - (3w_{4'} + w_3)) + C(w - (2w_{4'} - w_3)),$$

which is covered by (4) since $3w_{4'} + w_3 \geq 1 + w_{4'} + 1.5w_3$ and $2w_{4'} - w_3 \geq w_{4'} + 0.5w_3$.

(iv) $\{t_2, t_3\} = \{t_4, t_5\}$: We branch on edge vt_1. As in (ii), each vertex in $\{t_2, t_3\} = \{t_4, t_5\}$ is of degree 4. In the branch of `force`(vt_1), the weight of v, t_1, t_2 and t_3 decreases by at least $w_{4'} + w_{4'} + w_{4'} + w_{4'} = 4w_{4'}$ $(\geq 2w_{4'} + 3w_3)$. In the branch of `delete`(vt_1), the weight of v and t_1 decreases by $2\Delta'_{4-3} = 2w_{4'} - w_3$. Then we get a recurrence covered by (4).

Case-3. One good neighbor t_1 of v is a u4-vertex, and neither of the other two t_2 and t_3 is a u3- or f4-vertex: Each of t_2 and t_3 is a u4- or f3-vertex (otherwise Case-1 or -2 is applicable at v).

(I) Both t_2 and t_3 are f3-vertices: Let y_i, $i = 2, 3$, be the good neighbor of t_i other than v, where $y_2 \neq t_3$ and $y_3 \neq t_2$ since we have observed that no unforced edge joins two f3-vertices t_2 and t_3. If t_2 or t_3 is a good neighbor of t_1, then t_2 is assumed to be a good neighbor of t_1 (i.e., $t_1 = y_2$).

We branch on edge vt_2. In the branch of `force`(vt_2), we delete vt_1, vt_3 and t_2y_2 from G. Note that no chain joins y_2 and t_3 since t_2 and t_3 are not joined by a chain. If $t_1 \notin \Gamma(t_2y_2, y_2) \cup \Gamma(vt_3, t_3)$, then the weight of vertices in $\Gamma(vt_3, t_3) \cup \Gamma(t_2y_2, y_2)$ decreases by at least $2w_3$ by Lemma 4 and the weight of vertices v, t_1 and t_2 decreases by $w_{4'} + \Delta_{4-3} + w_{3'} = 1 + w_{4'} - 0.5w_3$, implying that the weight in G decreases by $1 + w_{4'} + 1.5w_3$. Assume that $t_1 \in \Gamma(t_2y_2, y_2) \cup \Gamma(vt_3, t_3)$; i.e., $t_1 \in \{\gamma(t_2y_2, y_2), y_2\}$ or $t_1 = \gamma(vt_3, t_3)$ holds. If $t_1 \in \{\gamma(t_2y_2, y_2), y_2\}$ and $t_1 = \gamma(vt_3, t_3)$ holds, then the U-component H has a bridge at t_1, and the reduction in Lemma 3(iii) would be applicable. Hence if $t_1 = y_2$ or $t_1 = y_3$, then exactly one of them occurs (i.e., $t_1 = y_2$ by the choice of the indices $i = 2, 3$). When $t_1 = y_2$, the weight of vertices $\{v, t_1, t_2\} \cup \Gamma(vt_3, t_3)$ decreases by $w_{4'} + 1 + w_{3'} + w_3 = 1 + w_{4'} + 1.5w_3$. When $t_1 \notin \{y_2, y_3\}$ and $t_1 = \gamma(t_2y_2, y_2)$ (resp. $t_1 = \gamma(vt_3, t_3)$), an f3-vertex u $(\neq t_2, t_3)$ is contained in $\Gamma(t_2y_2, y_2)$ (resp. $\Gamma(vt_3, t_3)$), and the weight of vertices in $\{v, t_1\} \cup \Gamma(t_2y_2, y_2) \cup \Gamma(vt_3, t_3)$ decreases by $w_{4'} + (1 - w_{3'}) + w_3 + w_3 = 1 + w_{4'} + 1.5w_3$. Therefore `force`$(vt_2)$ decreases the

entire weight by at least $1 + w_{4'} + 1.5w_3$. In the other branch of $\texttt{delete}(vt_2)$, we add t_2y_2 to F, decreasing the weight of vertices v, t_2 and y_2 by $\Delta'_{4-3} + w_{3'} + \Delta_3 = w_{4'} + 0.5w_3$. We get recurrence

$$C(w) \leq C(w - (1 + w_{4'} + 1.5w_3)) + C(w - (w_{4'} + 0.5w_3)),$$

i.e., (4).

(S-I) We here analyze the branching on edge vt_2 in a special case of (I) where both y_2 and y_3 are u4-vertices (this will be used in an analysis for Case-10). We show that $\texttt{force}(vt_2)$ decreases the weight by $2 + w_{4'} - 0.5w_3$. First consider the case of $t_1 \neq y_2$ (hence $t_1 \neq y_3$ by the choice of y_2). By noting that $y_2 = y_3$ is possible, $\texttt{force}(vt_2)$ decreases the weight of v, t_1, t_2, t_3 and $\{y_2, y_3\}$ by $w_{4'} + \Delta_{4-3} + w_{3'} + w_{3'} + \min\{\Delta_{4-3} + \Delta_4, w_4 - w_{3'}\} = 1 + w_{4'} + \min\{2 - w_{4'} - w_3, 1 - 0.5w_3\} = 1 + w_{4'} + 1 - 0.5w_3 = 2 + w_{4'} - 0.5w_3$ (by $\Delta_4 \geq \Delta_3$). Next consider the other case of $t_1 = y_2$, where $t_1 \neq y_3$ holds, as observed. Let z_1 and z_2 be the other neighbors of t_1 than v and t_2. Then all unforced edges incident to t_1 will be deleted or added to F. If $y_3 \notin \{z_1, z_2\}$ (resp., $y_3 \in \{z_1, z_2\}$), then $\texttt{force}(vt_2)$ decreases the weight of vertices v, t_1, t_2 and t_3 by $w_{4'} + w_4 + w_{3'} + w_{3'} = 1 + w_{4'} + w_3$ and the weight of vertices y_3, z_1 and z_2 (resp., z_1 and z_2) by $\Delta_4 + \Delta_3 + \Delta_3 = 1 - w_{4'} + w_3$ (resp., $w_4 + \Delta_3 \geq 1 - w_{4'} + w_3$), in total $2 + 2w_3 \geq 2 + w_{4'} - 0.5w_3$ (by $w_3 \geq 0.4$). The other branch $\texttt{delete}(vt_2)$ decreases the weight of v, t_2 and y_2 by $\Delta'_{4-3} + w_{3'} + \Delta_4 = 1$. Then we get

$$C(w) \leq C(w - (2 + w_{4'} - 0.5w_3)) + C(w - 1). \tag{7}$$

(II) (i) Both t_2 and t_3 are u4-vertices: We branch on edge vt_1. In the branch of $\texttt{force}(vt_1)$, the weight of vertices v, t_1, t_2 and t_3 decreases by $w_{4'} + \Delta_4 + \Delta_{4-3} + \Delta_{4-3} = 3 - 2w_3$. In the other branch of $\texttt{delete}(vt_1)$, the weight of vertices v and t_1 decreases by $\Delta'_{4-3} + \Delta_{4-3} = 1 - w_{4'} - 1.5w_3$. Then we can get recurrence

$$C(w) \leq C(w - (3 - 2w_3)) + C(w - (1 + w_{4'} - 1.5w_3)),$$

i.e., (5).

(ii) Exactly one of t_2 and t_3, say t_2 is an f3-vertex: We branch on edge vt_2. Let y_2 be the good neighbor of t_2 other than v. The branch of $\texttt{delete}(vt_2)$ decreases the entire weight by $w_{4'} + 0.5w_3$, as in (ii). The other branch of $\texttt{force}(vt_2)$ decreases the weight of vertices v, t_1, t_2 and t_3 by $w_{4'} + \Delta_{4-3} + w_{3'} + \Delta_{4-3} = 2 + w_{4'} - 1.5w_3$ and the weight of vertices in $\Gamma(t_2y_2, y_2)$ by w_3. In total we decrease the entire weight by $2 + w_{4'} - 0.5w_3 \geq 1 + w_{4'} + 1.5w_3$ (by $0.5 \geq w_3$). Then we get recurrence (4).

(S-II-ii) We also analyze the branching on edge vt_2 in a special case of (II-ii) where y_2 is a u4-vertex (this will be used in analyzing Case-10). Then $\texttt{delete}(vt_2)$ decreases the weight of v, t_2 and y_2 by $\Delta'_{4-3} + w_{3'} + \Delta_4 = 1$. First consider the case of $y_2 \notin \{t_1, t_3\}$. Then $\texttt{force}(vt_2)$ decreases the weight of v, t_2, t_1, t_3 and y_2 by $w_{4'} + w_{3'} + 3\Delta_{4-3} = 3 + w_{4'} - 2.5w_3$. Next consider the case of $y_2 = t_i$ for $i = 1, 3$. Then the weight of two other good neighbors of t_i will decrease, and $\texttt{force}(vt_2)$ decreases the entire weight by $w_{4'} + w_{3'} + \Delta_{4-3} + w_4 + 2\Delta_3 = 2 + w_{4'} + 0.5w_3 \geq 3 + w_{4'} - 2.5w_3$ (by $w_3 \geq 0.4$). Therefore we get a recurrence $C(w) \leq C(w - (3 + w_{4'} - 2.5w_3)) + C(w - 1)$, which is not worse than (7) in (S-I).

This completes the analysis of Cases-1, 2 and 3 (the analysis for Cases-4 to -10 are omitted due to space limitation). A quasiconvex program is obtained from 28 recurrences in our analysis. By solving this quasiconvex program [4], we get a bound $O(1.7154^w)$ on the running time by setting $w_{4'} = 0.6753$ and $w_3 = 0.4502$ for our problem.

Theorem 1. *TSP in an n-vertex graph G with maximum degree 4 can be solved in* $O^*(1.7154^n)$ *time and polynomial space.*

6 Concluding Remarks

In this paper, we have presented an improved exact algorithm for TSP in degree-4 graphs. The algorithm is analyzed by using the measure and conquer method and the basic operation in the algorithm is to search a solution by either including an edge into the solution or excluding it from the solution. However, to effectively reduce our measure in the algorithm, we need to select a 'good' edge to branch on. This kind of edges are easily identified with our branching rules but the analysis of the running time may not be straightforward.

Note that our algorithm also includes an $O^*(1.260^n)$-time algorithm for TSP in degree-3 graphs. It is easy to verify this result when the input graph is a degree-3 graph. In fact, when the input graph is restricted to degree-3 graphs, the main steps of our algorithm are similar to that of Eppstein's $O^*(1.260^n)$-time algorithm. But the analyses of the running time are different. So our algorithm also provides another analysis of Eppstein's algorithm.

References

1. Bjorklund, A.: Determinant sums for undirected Hamiltonicity. In: Proc. 51st Annual IEEE Symp. on Foundations of Computer Science, pp. 173–182 (2010)
2. Bjorklund, A., Husfeldt, T., Kasaki, P., Koivisto, M.: The Travelling Salesman Problem in Bounded Degree Graphs. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 198–209. Springer, Heidelberg (2008)
3. Dorn, F., Penninkx, E., Bodlaender, H.L., Fomin, F.V.: Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. Algorithmica 58(3), 790–810 (2010)
4. Eppstein, D.: Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms. ACM Trans. on Algorithms 2(4), 492–509 (2006)
5. Eppstein, D.: The traveling salesman problem for cubic graphs. J. Graph Algorithms and Applications 11(1), 61–81 (2007)
6. Gebauer, H.: Finding and enumerating Hamilton cycles in 4-regular graphs. Theoretical Computer Science 412(35), 4579–4591 (2011)
7. Iwama, K., Nakashima, T.: An Improved Exact Algorithm for Cubic Graph TSP. In: Lin, G. (ed.) COCOON 2007. LNCS, vol. 4598, pp. 108–117. Springer, Heidelberg (2007)
8. Woeginger, G.J.: Exact Algorithms for NP-hard Problems: A Survey. In: Jünger, M., Reinelt, G., Rinaldi, G. (eds.) Combinatorial Optimization - Eureka, You Shrink! LNCS, vol. 2570, pp. 185–207. Springer, Heidelberg (2003)

Dynamic Programming for H-minor-free Graphs*

Juanjo Rué[1], Ignasi Sau[2], and Dimitrios M. Thilikos[3]

[1] Instituto de Ciencias Matemáticas, Madrid, Spain
juanjo.rue@icmat.es
[2] AlGCo project-team, CNRS, LIRMM, Montpellier, France
ignasi.sau@lirmm.fr
[3] Department of Mathematics, National and Kapodistrian University of Athens, Greece
sedthilk@math.uoa.gr

Abstract. We provide a framework for the design and analysis of dynamic programming algorithms for H-minor-free graphs with branchwidth at most k. Our technique applies to a wide family of problems where standard (deterministic) dynamic programming runs in $2^{O(k \cdot \log k)} \cdot n^{O(1)}$ steps, with n being the number of vertices of the input graph. Extending the approach developed by the same authors for graphs embedded in surfaces, we introduce a new type of branch decomposition for H-minor-free graphs, called an *H-minor-free cut decomposition*, and we show that they can be constructed in $O_h(n^3)$ steps, where the hidden constant depends exclusively on H. We show that the separators of such decompositions have connected packings whose behavior can be described in terms of a combinatorial object called *ℓ-triangulation*. Our main result is that when applied on H-minor-free cut decompositions, dynamic programming runs in $2^{O_h(k)} \cdot n^{O(1)}$ steps. This broadens substantially the class of problems that can be solved deterministically in *single-exponential* time for H-minor-free graphs.

Keywords: analysis of algorithms, parameterized algorithms, graphs minors, branchwidth, dynamic programming, non-crossing partitions.

1 Introduction

The celebrated theorem of Courcelle [6] states that graph problems expressible in MSOL can be solved in $f(\mathbf{bw}) \cdot n$ steps, where $\mathbf{bw}$ is the branchwidth and n is the

* This research was done during a research visit of the first two authors at the Department of Mathematics of the National and Kapodistrian University of Athens. The authors wish to express their gratitude to the decisive support of 'Pontios' during that visit. The first author was partially supported by grants JAE-DOC (CSIC), MTM2011-22851, and SEV-2011-0087, the second author was partially supported by project AGAPE (ANR-09-BLAN-0159), and the third author was 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: "Thales. Investing in knowledge society through the European Social Fund".

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 86–97, 2012.

number of vertices of the input graph. Using terminology from parameterized complexity, this implies that a large number of graph problems admit fixed-parameter tractable algorithms when parameterized by the branchwidth of their input graph. As the bounds for $f(\mathbf{bw})$ provided by Courcelle's theorem are huge, the design of specific dynamic programming algorithms for graph problems so that $f(\mathbf{bw})$ is a simple function, became an essential ingredient for many results on graph algorithms (see [2,4,10,11,22]). In this paper, we provide a general framework for the design and analysis of dynamic programming algorithms for families of graphs excluding a graph H as a minor, where $f(\mathbf{bw}) = 2^{O(\mathbf{bw})}$. Our framework applies to a family of problems where no deterministic dynamic programming algorithm with single-exponential parameterized dependence on **bw** is known.

Motivation and Previous Work. Dynamic programming is usually applied in a bottom-up fashion on a rooted branch decomposition of the input graph G. Roughly, a branch decomposition of a graph is a way to decompose it into a tree structure of edge bipartitions (the formal definition is in Section 2). Each bipartition defines a separator S of the graph called the *middle set*, of cardinality bounded by the branchwidth of the input graph. The decomposition is routed, in the sense that one of the parts of each bipartition is the "lower part of the middle set", i.e., the so-far processed one. For each graph problem, dynamic programming requires a suitable definition of tables encoding how potential (global) solutions of the problem are restricted to a middle set and the corresponding lower part. The size of these tables reflects the dependence on $k = |S|$ in the running time of the dynamic programming.

Designing the tables for each middle set S may vary considerably among different problems. For simple problems where the tables of dynamic programming encode vertex subsets of the middle set, such as Vertex Cover or Dominating Set, we may easily have a single-exponential dependence on k, as the number of subsets of a set of size k is 2^k. However, there are problems where the tables of the dynamic programming encode vertex pairings, such as Longest Path, Cycle Packing, or Hamiltonian Cycle, or (more generally) vertex packings, such as Connected Vertex Cover, Connected Dominating Set, Feedback Vertex Set, or Steiner Tree. For the latter category of problems, single-exponential bounds on their table size are not known to exist. This complication arises from the fact that, for such problems, the tables should encode at least $2^{\Theta(k \log k)}$ many pairings/packings. Nevertheless, for such problems one can do better for several classes of sparse graphs. This line of research was initiated in [12] and occupied several researchers in parameterized algorithms design (see also [5,7,9,11,16,22]). The current technology of dynamic programming in graphs of bounded decomposability implies single-exponential parametric dependence for problems encodable by pairings in H-minor free graphs [11] and for problems encodable by packings in graphs embedded in surfaces [22]. In this paper we extend both approaches of [11] and [22] to problems encodable by packings in H-minor free graphs.

Our Results and Techniques. We present a general framework that provides single-exponential dynamic programming algorithms for *connected packing-encodable problems* (the formal definition of this class of problems is in Section 2.1) when the input graph excludes a graph H as a minor. The main idea in [22] was to introduce a new type of branch decomposition for graphs on surfaces, called a *surface cut decomposition* (which in turn, extended the concept of sphere cut decompositions for planar graphs introduced in [12,24]). Namely, in [22], it was proved that the number of partial solutions that can be arranged on a surface cut decomposition can be upper-bounded by the number of non-crossing partitions on surfaces with boundary, which have been recently enumerated in [21]. It follows that partial solutions can be represented by a single-exponential number of configurations. This proves that, when applied on surface cut decompositions, dynamic programming for connected packing-encodable problems runs in $2^{O(k)} \cdot n^{O(1)}$ steps.

We follow the same approach to extend this technique to H-minor-free graphs: we define a new type of branch decomposition for graphs excluding an h-vertex graph H as a minor; we call it an *H-minor-free cut decomposition*. In Section 3 we show how to compute an H-minor-free cut decomposition of width $O_h(k)$ in $O_h(n^3)$ steps[1]. This algorithm uses the recent result of Kawarabayashi and Wollan [15] to find in time $O(n^3)$ the tree-like decomposition of an H-minor-free graph G, given by the seminal structure theorem of Robertson and Seymour [20]. Roughly, this result says that each H-minor free graph admits a bounded-adhesion tree decomposition whose bags are nearly embedded in some surface of small genus. We also make use of the algorithm of [22] to find a surface cut decomposition of the surface-embedded part of each bag, then enhance them with the apices and vortices, and finally we glue them appropriately along the clique-sums. But for being able to use the algorithm of [22], we need to prove that there exists a tree-like decomposition of G whose bags have good topological properties. This is done in Section 2.2, and requires a suitable extension of the notion of a polyhedral decomposition introduced in [22].

In order to prove the upper bound on the size of the tables when using an H-minor-free cut decomposition, the main difficulty is to deal with the vortices. From a combinatorial point of view, our main contribution is to capture the behavior of the vortices of an H-minor-free graph in terms of an object called *ℓ-triangulation* (cf. Section 2). Roughly speaking, in order to take into account the number of *simultaneous* crossings of a set of connected subgraphs inside a vortex, ℓ-triangulations seem to be the appropriate combinatorial object to look at (see Section 4 for more details). Finally, we prove our main result in Section 5. That is, by combining all the ingredients mentioned above, we prove that by using H-minor-free cut decompositions, the size of the tables for solving connected packing-encodable problems is single-exponential in the branchwidth. We would like to note that we did not make any effort to optimize the constants depending on H, as they are already huge since we use the Structure Theorem of

[1] Given a computable function f and an integer h, we use the notation $O_h(f(k))$ to denote $O(g(h) \cdot f(k))$ for some computable function g.

the Graph Minors series [15,20]. Due to space limitations, all proofs are omitted in this extended abstract, while sketches of them can be found in [23].

Our results can also be used to derive subexponential parameterized algorithms for connected packing-encodable *bidimensional* problems. That way, we broaden the class of problems where the general framework introduced in [8] can be applied. It is worth mentioning that our results directly imply that Steiner Tree and Connected Dominating Set, among others, can be solved in subexponential time in H-minor-free graphs, which has been recently (and independently) proved by Tazari [25].

Recent Results and Further Research. Recently, Cygan *et al.* [7] have presented a new framework for obtaining *randomized* single-exponential algorithms parameterized by treewidth in general graphs. This framework is based on a dynamic programming technique named Cut&Count, which seems applicable to most connected packing-encodable problems, like Connected Vertex Cover, Connected Dominating Set, Feedback Vertex Set, or Steiner Tree. The randomization in the algorithms of [7] comes from the usage a probabilistic result called the Isolation Lemma [18], whose derandomization is a challenging open problem [3]. Therefore, the existence of *deterministic* single-exponential algorithms parameterized by treewidth for connected packing-encodable problems in general graphs remains wide open.

Our results for minor-free graphs can be seen as an intermediate step towards an eventual positive answer to this question. It may also be the case that this class of graphs establishes a frontier of the existence of *deterministic* single-exponential parameterized algorithms for connected packing-encodable problems although we do not think that this is the case. It would be also interesting to reduce the big polynomial overheads in the algorithms of [7], given by the usage of the Isolation Lemma. In addition, the approach presented in [7] does not seem to be applicable to weighted problems, while our results are easily extendable to weighted connected packing-encodable problems.

Finally, it is worth mentioning that another type of branch decomposition for graphs on surfaces, called *s*urface split decomposition, has been recently introduced by Bonsma [5] to prove that Subgraph Isomorphism can be solved in single-exponential time in graphs on surfaces. It remains open to find single-exponential algorithms for Subgraph Isomorphism in H-minor-free graphs.

2 Preliminaries

In this section we provide some preliminaries required in the sequel. Due to lack of space, we avoid the definitions of some basic graph theoretical concepts related to topological graph theory and tree or branch decompositions. For more details on the missing definitions, see [23].

ℓ-Triangulations and Related Constructions. Let $\mathbb{D}_k$ be a disc with k vertices on its border. We assume that these vertices are labeled counterclockwise with labels $1, 2, \ldots, k$. By an *ℓ-triangulation* of $\mathbb{D}_k$ we mean a maximal set of

diagonals with no pairwise crossing-set of size $\ell+1$. In other words, the graph whose vertices are the diagonals of the ℓ-triangulation and there is an edge between two diagonals if and only if they cross in an internal vertex, does not contain $K_{\ell+1}$ as a subgraph. This concept generalizes the classical notion of *triangulation* of a disc. Denote by $T_\ell(k)$ the number of different ℓ-triangulations of $\mathbb{D}_k$. In particular, $T_0(k+2)$ is equal to the k-th Catalan number $C_k = \frac{1}{k+1}\binom{2k}{k}$, a result which is well-known since Euler's time. The study of ℓ-triangulations for $\ell > 1$ is more involved than the study of triangulations. In [13, 19] the authors show that the number of diagonals in an ℓ-triangulation of $\mathbb{D}_k$ is always $\ell(k-2\ell-1)$. More recently a closed expression for $T_\ell(k)$ in terms of a determinant of Catalan numbers has been obtained in [14]. This expression generalizes the enumeration of the number of triangulations of a polygon with k vertices. For the asymptotic of $T_\ell(k)$, observe that the recurrence $C_k = \frac{4k-2}{k+1}C_{k-1}$ makes each entry of the determinant equal to C_k times a rational function of degree at most 2ℓ in ℓ. Using also that $C_k = \frac{1}{\sqrt{\pi}}k^{-3/2}4^k(1+o(1))$ for k large enough, it is easy to get bounds for $T_\ell(k)$. More concretely, $T_\ell(k) \leq_{k\to\infty} \frac{\ell!}{\pi^{\ell/2}}k^{-3\ell/2}4^{\ell k}$ (see [23] for details).

We say that a set of diagonals in $\mathbb{D}_k$ is a *partial ℓ-triangulation* if it a subset of an ℓ-triangulation. Let $\Pi = \{\pi_1, \pi_2, \ldots, \pi_r\}$ be a partition of the set $\{1, 2, \ldots, k\}$ (i.e., $\bigcup_{i=1}^r \pi_i = \{1, 2, \ldots, r\}$, and $\pi_i \cap \pi_j = \emptyset$ if and only if $i \neq j$). We say that each of the subsets π_i, $i \in \{1, 2, \ldots, r\}$ is a *block* of the partition Π. We represent a partition in the following way: we draw each block of Π as a polygon connecting the corresponding vertices. This defines a graph $G(\Pi)$ whose vertices are the blocks of Π, and the edges are defined by the incidences of the blocks (i.e., an edge is drawn between a block π_i and a block π_j if and only if the associated polygons intersect in the graphical representation). We say that a packing (that is, a collection of pairwise disjoint non-empty blocks) of the disc $\mathbb{D}_k$ is an *ℓ-packing* if and only if $G(\Pi)$ does not contain $K_{\ell+1}$ as a subgraph. In particular, if $\ell < \ell'$, then an ℓ-packing is also an ℓ'-packing. The notion of ℓ-packing of a disc is a natural generalization of the notion of non-crossing partition, which corresponds to the case $\ell = 1$, in the same way as ℓ-triangulations generalize triangulations of a disc. In the following lemma we find asymptotic estimates for the number of ℓ-packings of $\mathbb{D}_k$, which we denote by $P_\ell(k)$.

Lemma 1. *The number of ℓ-packings of $\mathbb{D}_k$ satisfies $P_\ell(k) = 2^{O_\ell(k)}$.*

Partitions of An Integer. Let q be a non-negative integer. A *partition* of q is a non-increasing sequence of positive integers $p_1, p_2, \ldots, p_r$ whose sum is q. Let $p(q)$ be the number of partitions of q. The Hardy-Ramanujan-Rademacher estimate for $p(q)$ states that $p(q) = \frac{1}{4\sqrt{3}q}e^{\pi\sqrt{2q/3}}(1+o(1)) = 2^{O(\sqrt{q})}$ (see [1]).

2.1 Connected Packing-Encodable Problems

In this paper we follow the standard dynamic programming approach on branch decompositions. The interested reader can find more details and considerations about dynamic programming for different problems in [22] or in [23].

Before we proceed with the description of the family of problems that we examine in this paper, we need some definitions. Let G be a graph and let S be a set of vertices of G. We denote by $\mathcal{G}$ the collection of all subgraphs of G. Each $H \in \mathcal{G}$ defines a packing $\mathcal{P}_S(H)$ of S such that two vertices $x, y \in S$ belong to the same set of $\mathcal{P}_S(H)$ if x, y belong to the same connected component of H. We say that $H_1, H_2 \in \mathcal{G}$ are *S-equivalent* if $\mathcal{P}_S(H_1) = \mathcal{P}_S(H_2)$, and we denote it by $H_1 \equiv_S H_2$. Let $\overline{\mathcal{G}}_S$ the collection of all subgraphs of G modulo the equivalence relation $\equiv_S$. We define the set of all *connected packings of S with respect to G* as the collection $\Psi_G(S) = \{\mathcal{P}_S(H) \mid H \in \overline{\mathcal{G}}_S\}$. Notice that each member of $\Psi_G(S)$ can indeed be seen as a packing of S, as its sets may not necessarily meet all vertices of S.

In this paper we consider graph problems that can be solved by dynamic programming algorithms on branch decompositions for which the size of $\mathsf{struct}(e)$ is upper-bounded by $2^{O(|\mathbf{mid}(e)|)} \cdot |\Psi_{G_e}(\mathbf{mid}(e))| \cdot n^{O(1)}$. We call these problems *connected packing-encodable*. We stress that our definition of connected packing-encodable problem assumes the existence of *an* algorithm with this property, but there may exist other algorithms whose tables are much bigger. For these problems, dynamic programming has a single-exponential dependence on branch-width if and only if $\Psi_{G_e}(\mathbf{mid}(e))$ contains a single-exponential number of packings, i.e., $|\Psi_{G_e}(\mathbf{mid}(e))| = 2^{O(|\mathbf{mid}(e)|)}$. See [22] for more details.

In general, the number of different connected packings that could be created during the dynamic programming is not necessarily smaller than the number of the non-connected ones. In fact, it typically depends on the k-th Bell number, where k is the branchwidth of the input graph. This implies that, in general, $|\Psi_{G_e}(\mathbf{mid}(e))| = 2^{O(k \log k)}$ is the best upper bound that can be so far achieved for connected packing-encodable problems, at least for deterministic algorithms. The purpose of this paper is to show that, for such problems, this bound can be reduced to a single-exponential one when their input graphs exclude a graph as a minor. In Section 3, we define the concept of an H-minor-free cut decomposition, which is a key tool for the main result of this paper, summarized as follows.

Theorem 1. *Every connected packing-encodable problem whose input is an n-vertex graph G that excludes an h-vertex graph H as a minor, and has branch-width at most k, can be solved by a dynamic programming algorithm on an H-minor-free cut decomposition of G with tables of size $2^{O_h(k)} \cdot n^{O(1)}$.*

In Section 3, we prove (Theorem 3) that, given an H-minor-free graph G, an H-minor-free cut decomposition of G of width $O_h(\mathbf{bw}(G))$ can be constructed in $O_h(n^3)$ steps. Therefore, we conclude the following result.

Theorem 2. *Every connected packing-encodable problem whose input is an n-vertex graph G that excludes an h-vertex graph H as a minor and has branch-width at most k, can be solved in $2^{O_h(k)} \cdot n^{O(1)}$ steps.*

2.2 Polyhedral Decomposition of H-minor-free Graphs

Let Σ be a surface with boundary. An *O-arc* is a subset of Σ homeomorphic to $\mathbb{S}^1$. A subset of Σ meeting the drawing only at vertices of G is called *G-normal*.

If an O-arc is G-normal, then we call it a *noose*. We denote by V_N the set of vertices met by a noose N, i.e., $V(N) = V(G) \cap N$. The *length* N of a noose is the number of the vertices that it meets and is denoted by $|N|$, i.e., $|N| = |V(N)|$. The *facewidth* of a Σ-embedded graph embedding (G, τ) is the smallest length of a non-contractible (i.e., non null-homotopic) noose in Σ and is denoted by $\mathbf{fw}(G)$. We call a Σ-embedded graph G *polyhedral* [17] if G is 3-connected and $\mathbf{fw}(G) \geq 3$, or if G is a clique and $1 \leq |V(G)| \leq 3$.

Definition 1. *Let $\alpha, \beta, \gamma, \delta$ be non-negative integers. A graph G is* $(\alpha, \beta, \gamma, \delta)$-nearly embeddable*, if there is a surface Σ of Euler genus γ and a set of vertices $A \subseteq V(G)$ (called* apices*) of size at most α such that graph $G \setminus A$ is the union of subgraphs $\mathcal{G} = \{G_0, \ldots, G_\delta\}$ (some of them may be the empty graph) with the following properties:*

1. *There is a set $\mathcal{R} = \{\Delta_1, \ldots, \Delta_\delta\}$ of pairwise disjoint open discs in Σ such that G_0 is a graph embedded in Σ in a way that $G_0 \cap \bigcup_{i=1,\ldots,\delta} \Delta_i = \emptyset$ (G_0 is called* underlying graph *of G),*
2. *the graphs $G_1, \ldots, G_\delta$ (called* vortices*) are pairwise disjoint and for $1 \leq i \leq \delta$, $V(G_0) \cap V(G_i) \subseteq \mathbf{bor}(\Delta_i)$ (we call the vertices in $V(G_0) \cap V(G_i) \subseteq \mathbf{bor}(\Delta_i)$* base vertices *of the vortex G_i),*
3. *for $1 \leq i \leq \delta$, let $U_i = \{u^i_1, \ldots, u^i_{m_i}\}$ be the base vertices of G_i appearing in an order obtained by counterclockwise traversing $\mathbf{bor}(\Delta_i)$. Then G_i has a path decomposition $\mathcal{B}_i = (B^i_j)_{1 \leq j \leq m_i}$, of width equal to β such that for $1 \leq j \leq m_i$, we have $u^i_j \in B^i_j$. We also denote $\mathcal{B} = \{\mathcal{B}_1, \ldots, \mathcal{B}_\delta\}$ and for each v^i_j, we call B^i_j the* cloud *of v^i_j.*

If G is an $(\alpha, \beta, \gamma, \delta)$-nearly embeddable graph for some $\mathfrak{E} = (A, \Sigma, \mathcal{G}, \mathcal{R}, \mathcal{B})$, we say that $\mathfrak{E}$ is its *embedding pattern*. If in Definition 1 the embedding of the graph G_0 to be polyhedral, then we say that G is *polyhedrally $(\alpha, \beta, \gamma, \delta)$-nearly embeddable*, and we say that the corresponding pattern is *polyhedral*. The graph G is (*polyhedrally*) *c-nearly embeddable* graph if it is (polyhedrally) $(\alpha, \beta, \gamma, \delta)$-nearly embeddable for some $\alpha, \beta, \gamma, \delta \leq c$. We prove the following.

Proposition 1. *There exists an algorithm that, given an h-vertex graph H and an n-vertex graph G that excludes H as a minor, outputs, in $O_h(n^3)$ steps, a tree decomposition $\mathcal{D} = (\mathcal{X} = \{X^t \mid t \in V(T)\}, T)$ of G of adhesion $O_h(1)$ and such that every $t \in V(T)$, $\overline{X}^t$ is a polyhedrally $O_h(1)$-nearly embeddable graph. Moreover, the same algorithm outputs the corresponding embedding pattern $\mathfrak{E}^t$ of $\overline{X}^t$ for each $t \in V(T)$.*

Given an h-vertex graph H and an H-minor free graph G, we call a tree decomposition $\mathcal{D}$ as the one in Proposition 1, a *c_h-nearly polyhedral decomposition* (where c_h is a constant depending only on h). If in Proposition 1 G is a graph embedded in a surface, then a *c_h-nearly polyhedral decomposition* is what has been defined in [22] as a *polyhedral decomposition*, where the adhesion is at most 2 (now in each embedding pattern $\mathfrak{E}^t = (A^t, \Sigma^t, \mathcal{G}^t, \mathcal{R}^t, \mathcal{B}^t)$, we have that $\mathcal{G}^t_0$ contains only the underlying graph, while $\mathcal{R}^t = \emptyset$ and $\mathcal{B}^t = \emptyset$).

3 H-minor-free Cut Decompositions

In this section we define and show how to construct a special type of branch decompositions for families of graphs excluding a graph H as a minor; we call them *H-minor-free cut decompositions*. Their construction relies on surface cut decompositions, recently introduced in [22].

Let Σ be a surface without boundary, and let $\mathcal{N}$ be a finite set of O-arcs in Σ pairwise intersecting at zero-dimensional subsets of Σ (i.e., points). Then the closure of each connected component of $\Sigma \setminus \mathbf{U}\mathcal{N}$ is called a *pseudo-surface*, where $\mathbf{U}\mathcal{N} = \bigcup_{N \in \mathcal{N}} N$. For a point $p \in \Sigma$, let $\mathcal{N}(p)$ be the number of O-arcs in $\mathcal{N}$ containing p, and let $P(\mathcal{N}) = \{p \in \Sigma : \mathcal{N}(p) \geq 2\}$; note that by assumption $P(\mathcal{N})$ is a finite set of points of Σ. Then we define $\theta(\mathcal{N}) = \sum_{p \in P(\mathcal{N})} (\mathcal{N}(p) - 1)$.

Note that if the O-arcs are pairwise disjoint, then each pseudo-surface is a surface with boundary. Notice that in Definition 1 we can permit Σ to be a pseudo-surface with boundary instead of a surface without boundary. This extension of the definition is necessary for defining the concept of an H-minor-free cut decomposition below.

Definition 2. *Given an h-vertex graph H, an n-vertex H-minor-free graph G, an* H-minor-free cut decomposition *of G is a branch decomposition (T, μ) of G such that there exists an $O_h(1)$-nearly polyhedral decomposition $\mathcal{D} = (\mathcal{X} = \{X^t \mid t \in V(T')\}, T')$ of G such that for each edge $e \in E(T)$, either $|\mathbf{mid}(e)| = O_h(1)$ or there exists a bag $X^t \in \mathcal{X}$ such that*

- $\mathbf{mid}(e) \subseteq V(X^t)$;
- *given that $\mathfrak{E}^t = (A^t, \Sigma^t, \mathcal{G}^t, \mathcal{R}^t, \mathcal{B}^t)$ is the embedding pattern of $\overline{X}^t$ and $\mathcal{G}^t = \{G^t_0, G^t_1, \ldots, G^t_\delta\}$, there exists a set $\mathcal{N}$ of nooses of G^t_0 in Σ^t such the vertices of $\mathbf{mid}(e) \cap V(G^t_0)$ are all met by the nooses in $\mathcal{N}$ in a way that (1) $|\mathcal{N}| = O_h(1)$, (2) the nooses in $\mathcal{N}$ pairwise intersect only at subsets of $\mathbf{mid}(e)$, (3) $\theta(\mathcal{N}) = O_h(1)$, (4) $\Sigma^t \setminus \mathbf{U}\mathcal{N}$ contains exactly two connected components, such that the graph $\overline{G}[V(G_e) \cap V(X^t)]$ is $O_h(1)$-nearly embedded in the closure of one of them.*

If in the above definition we consider that G is embedded in some surface of genus γ and we restrict each $\mathcal{G}^t$ to contain only the underlying graph (i.e., there are no vortices, $\mathcal{R}^t = \emptyset$, and $\mathcal{B}^t = \emptyset$), we have the definition of surface cut decompositions introduced in [22]. Finally, if we further restrict Σ to be a sphere and set $A^t = \emptyset$, we have the notion of sphere cut decompositions introduced in [12,24].

Theorem 3. *There exists an algorithm that, given an h-vertex graph H and an n-vertex graph G that excludes H as a minor and has branchwidth at most k, outputs in $O_h(n^3)$ steps an H-minor-free cut decomposition of G of width $O_h(k)$.*

It is worth noting here that the algorithm that computes a surface cut decomposition for a surface-embedded graph in [22] runs in time $2^{O(k)} \cdot n^3$, because we wanted to optimize the dependence on the genus of the width of the obtained branch decomposition, while keeping the running time single-exponential in k.

4 Combinatorial Behavior of the Vortices

In this section we focus on the combinatorial behavior of the vortices in a graph excluding a graph H as a minor. The main objective is to prove that, in an H-minor-free cut decomposition, the number of ways that connected subgraphs can behave inside a vortex can be upper-bounded by the number of $O_h(1)$-packings (defined in Section 2) of size linear in the branchwidth of the input graph. By Theorem 3, from now on we assume that we have an H-minor-free cut decomposition (T, μ) of G, as well as a tree decomposition $\mathcal{D} = (\mathcal{X} = \{X^t \mid t \in V(T)\}, T)$ of G of adhesion $O_h(1)$, such that each $\mathcal{D}$-closure $\overline{X}^t$ is a polyhedrally $(\alpha, \beta, \gamma, \delta)$-nearly embedded graph, with $\alpha, \beta, \gamma, \delta = O_h(1)$.

In order to have a clearer picture of the behavior of the vortices, we define according to [11] the graph $R_{d,s}$, where $V(R_{d,s}) = V_1 \cup \cdots \cup V_s$ with $|V_i| = d$ for $1 \leq i \leq s$ and $E(R_{d,s}) = \{\{x_j, x_k\} \mid x_j, x_k \in V_i, 1 \leq j \neq k \leq d, 1 \leq i \leq s\} \cup \{\{x_j, y_j\} \mid x_j \in V_{i-1} \text{ and } y_j \in V_i, 1 \leq j \leq d, 1 \leq i \leq s\}$.

We call such a graph $R_{d,s}$ a *normalized vortex*. In the graph $R_{d,s}$ we distinguish a subset $U \subseteq V(R_{d,s})$ containing exactly one vertex from each V_i. The pair $(R_{d,s}, U)$ is called an (d, s)-*vortex pattern*. We say that an (d, s)-vortex pattern has *depth* d. Note that each base vertex belongs to a clique of size d. The d edges between two consecutive cliques are called a *section* of the vortex. Normalized vortices and vortex patterns are useful because any vortex is a minor of a vortex pattern, as stated in the following lemma.

Lemma 2 (Dorn, Fomin, and Thilikos [11]). *Any vortex of a d-nearly embeddable graph with base set J is a minor of a (d, s)-vortex pattern $(R_{d,s}, U)$, where the minor operations map bijectively the vertices of U to the vertices in J in a way that the order of the vortex and the cyclic ordering of U induced by the indices of its elements is respected.*

By Lemma 2, from now on we will only deal with vortex patterns. We say that connected subgraph B of G *meets* a vortex F, if B contains some of the base vertices of F. If U is the set of base vertices of F, the number of times that B meets F is exactly $|V(B) \cap U|$. For the analysis, we need to consider the possibility that a subgraph in a connected packing $\mathcal{P} \in \Psi_G(S)$ meets more than one vortex. This possibility is ruled out in the following lemma.

Lemma 3. *We can assume that each subgraph in a connected packing $\mathcal{P} \in \Psi_G(S)$ meets at most one vortex.*

Loosely speaking, the proof of Lemma 3 is based on the fact that if a subgraph B in a connected packing $\mathcal{P} \in \Psi_G(S)$ meets two vortices of depth at most β, these two vortices can be virtually merged along a path of B into a new vortex of depth at most 2β. As a subgraph may a priori meet an arbitrary number of the vortices, for our analysis we need to consider all possibilities of merging any subset of the δ vortices, which are at most $p(\delta)$ many (see "partitions of an integer" in Section 2). Therefore, potentially some of this merged vortices may have depth up to $\delta \cdot \beta = O_h(1)$. Also, in order to find an upper bound for

the number of connected packings, we will need to incur an additional factor $p(\delta)$. The following lemma, whose proof uses Lemma 3 above, will allow us to simulate the behavior of the vortices with simpler objects of appropriate size, independent of the integer s (recall that we deal with (β, s)-vortex patterns.

Lemma 4. *For each vortex F, we can assume that the total number of times that the subgraphs in a connected packing $\mathcal{P} \in \Psi_G(S)$ meet F is $O_h(k)$.*

Let F be a given (d, s)-vortex pattern with set of base vertices $U = \{u_1, \ldots, u_s\}$, ordered cyclically. A *configuration* in F is a set of vertex-disjoint subgraphs $\mathcal{F} = \{F_1, \ldots, F_\ell\}$ of F. We say that two subgraphs $F_i, F_j \in \mathcal{F}$ *cross* if there exist $u_{i_1}, u_{i_2} \in V(F_i) \cap U$ and $u_{j_1}, u_{j_2} \in V(F_j) \cap U$ such that $i_1 < j_1 < i_2 < j_2$. The *crossing graph* $\mathcal{F}_c$ of a configuration $\mathcal{F}$ has one vertex for each subgraph in $\mathcal{F}$, and an edge between two vertices if and only if their corresponding subgraphs cross. A configuration $\mathcal{F}$ is said to be an *ℓ-configuration* if the maximum size of a clique in $\mathcal{F}_c$ is ℓ. In the following lemma we prove that in a vortex of given depth, the existing configurations can cross only in a *bounded* way. This fact will enable us to upper-bound the number of configurations in a vortex of depth d in terms of the number of d-packings in the circle.

Lemma 5. *A vortex pattern of depth at most β does not contain any β'-configuration with $\beta' > \beta$.*

5 Upper-Bounding the Size of the Tables

In this section we show that by using H-minor-free cut decompositions in order to solve connected packing-encodable problems in H-minor-free graphs, one can guarantee single-exponential upper bounds on the size of the tables of dynamic programming algorithms. Theorem 1 follows directly by the definition of a connected packing-encodable problem and the following lemma, which we will prove in this section.

Lemma 6. *Let G be a graph excluding an h-vertex graph H as a minor, and let (T, μ) be an H-minor-free cut decomposition of G of width at most k. Then for every $e \in E(T)$, $|\Psi_{G_e}(\mathbf{mid}(e))| = 2^{O_h(k)}$.*

Let (T, μ) be an H-minor-free cut decomposition of a graph G. For edges $e \in E(T)$ such that $\mathbf{mid}(e) = O_h(1)$, we trivially have that $|\Psi_{G_e}(\mathbf{mid}(e))| = 2^{O_h(1)}$, and therefore the statement of Lemma 6 is satisfied. Therefore, we only need to deal with edges $e \in E(T)$ such that $\mathbf{mid}(e)$ is contained in a graph which is polyhedrally $O_h(1)$-nearly embedded in a surface Σ.

In order to prove Lemma 6, we will need the lemmata of Section 4 about the combinatorial behavior of the vortices. We will also need the following key result from [22], which bounds the size of the tables for graphs embedded in surfaces with boundary.

Lemma 7. *Let G be a graph containing a set A of vertices such that $G \setminus A$ is embedded in a surface Σ with boundary. Let also S be the set of vertices of G that*

lie on the boundary of Σ and $A' \subseteq A$. Then, if $|S| \leq k$ and $|A|, \gamma(\Sigma), \nu(\Sigma) \leq \gamma$, then $|\Psi_G(S \cup A')| = \gamma^{O(\gamma)} \cdot k^{O(\gamma)} \cdot \gamma^{O(k)}$.

Note that, in the statement of Lemma 7, if $|A|, \gamma(\Sigma), \nu(\Sigma) = O_h(1)$, then it holds that $|\Psi_G(S \cup A')| = 2^{O_h(k)}$. The following lemma gives an upper bound on the number of non-crossing packings on a surface with apices and vortices. Intuitively, our approach consists in considering each vortex as a new virtual noose of length $O_h(k)$ in an H-minor-free cut decomposition, where each vertex of such noose corresponds to an eventual meeting of a subgraph of the connected packing with the corresponding vortex. We then consider all non-crossing packings taking into account the original and the virtual nooses, and finally we merge the subgraphs incident to a virtual noose according to the possible $O_h(1)$-packings corresponding to that vortex. This approach is made more precise in Lemma 8 below, which implies Lemma 6, and therefore also Theorem 1 and Theorem 2. The proof of Lemma 8 makes use of Lemmata 2, 4, 5, and 7.

Lemma 8. *Let G be a graph polyhedrally $(\alpha, \beta, \gamma, \delta)$-nearly embedded in a surface Σ with boundary, with a set of apices A. Let also S be the set of vertices of G that lie on the boundary of Σ. If $|S| \leq k$ and $\alpha, \beta, \gamma, \delta, \nu(\Sigma) \leq \eta$, then $|\Psi_G(S \cup A)| = 2^{O_\eta(k)}$.*

References

1. Andrews, G.: The Theory of Partitions. Cambridge University Press (1984)
2. Arnborg, S.: Efficient algorithms for combinatorial problems on graphs with bounded decomposability – a survey. BIT 25(1), 2–23 (1985)
3. Arvind, V., Mukhopadhyay, P.: Derandomizing the Isolation Lemma and Lower Bounds for Circuit Size. In: Goel, A., Jansen, K., Rolim, J.D.P., Rubinfeld, R. (eds.) APPROX and RANDOM 2008. LNCS, vol. 5171, pp. 276–289. Springer, Heidelberg (2008)
4. Bodlaender, H.L.: Dynamic Programming on Graphs with Bounded Treewidth. In: Lepistö, T., Salomaa, A. (eds.) ICALP 1988. LNCS, vol. 317, pp. 105–118. Springer, Heidelberg (1988)
5. Bonsma, P.: Surface split decompositions and subgraph isomorphism in graphs on surfaces. CoRR, abs/1109.4554 (2011), to appear in the Proc. of STACS 2012
6. Courcelle, B.: The Monadic Second-Order Logic of Graphs: Definable Sets of Finite Graphs. In: van Leeuwen, J. (ed.) WG 1988. LNCS, vol. 344, pp. 30–53. Springer, Heidelberg (1989)
7. Cygan, M., Nederlof, J., Pilipczuk, M., Pilipczuk, M., van Rooij, J.M.M., Wojtaszczyk, J.O.: Solving connectivity problems parameterized by treewidth in single exponential time. In: Proc. of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 150–159 (2011)
8. Demaine, E.D., Fomin, F.V., Hajiaghayi, M.T., Thilikos, D.M.: Subexponential Parameterized Algorithms on Graphs of Bounded Genus and H-Minor-Free Graphs. Journal of the ACM 52(6), 866–893 (2005)
9. Dorn, F., Fomin, F.V., Thilikos, D.M.: Fast Subexponential Algorithm for Nonlocal Problems on Graphs of Bounded Genus. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 172–183. Springer, Heidelberg (2006)

10. Dorn, F., Fomin, F.V., Thilikos, D.M.: Subexponential Parameterized Algorithms. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 15–27. Springer, Heidelberg (2007)
11. Dorn, F., Fomin, F.V., Thilikos, D.M.: Catalan Structures and Dynamic Programming in H-minor-free Graphs. In: Proc. of the 19th Annual ACM-SIAM Symposium on Discrete algorithms (SODA), pp. 631–640 (2008)
12. Dorn, F., Penninkx, E., Bodlaender, H.L., Fomin, F.V.: Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. Algorithmica 58(3), 790–810 (2010)
13. Dress, A., Koolen, J.H., Moulton, V.: On line arrangements in the hyperbolic plane. European Journal of Combinatorics 23(5), 549–557 (2002)
14. Jonsson, J.: Generalized triangulations and diagonal-free subsets of stack polyominoes. Journal of Combinatorial Theory, Series A 112(1), 117–142 (2005)
15. Kawarabayashi, K.-I., Wollan, P.: A simpler algorithm and shorter proof for the graph minor decomposition. In: Proc. of the 43rd ACM Symposium on Theory of Computing (STOC), pp. 451–458 (2011)
16. Lokshtanov, D., Marx, D., Saurabh, S.: Slightly Superexponential Parameterized Problems. In: Proc. of the 22nd annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 760–776 (2011)
17. Mohar, B., Thomassen, C.: Graphs on surfaces. John Hopkins University Press (2001)
18. Mulmuley, K., Vazirani, U.V., Vazirani, V.V.: Matching is as easy as matrix inversion. Combinatorica 7(1), 105–113 (1987)
19. Nakamigawa, T.: A generalization of diagonal flips in a convex polygon. Theoretical Computer Science 235(2), 271–282 (2000)
20. Robertson, N., Seymour, P.D.: Graph minors. XVI. Excluding a Non-planar Graph. Journal of Combinatorial Theory, Series B 77, 1–27 (1999)
21. Rué, J., Sau, I., Thilikos, D.M.: Asymptotic enumeration of non-crossing partitions on surfaces. In: CoRR, abs/1104.2477 a preliminary version appeared in the Proc. of ICALP (2011), Last version, http://www.lirmm.fr/~sau/Pubs/RST11NCP.pdf
22. Rué, J., Sau, I., Thilikos, D.M.: Dynamic programming for graphs on surfaces. CoRR, abs/1104.2486 (2011), to appear in ACM Transactions on Algorithms (TALG), last version available at, http://www.lirmm.fr/~sau/Pubs/RST12TALG.pdf
23. Rué, J., Sau, I., Thilikos, D.M.: Dynamic Programming for H-minor-free Graphs (2012), http://www.lirmm.fr/~sau/Pubs/RSTminor12.pdf
24. Seymour, P.D., Thomas, R.: Call routing and the ratcatcher. Combinatorica 14(2), 217–241 (1994)
25. Tazari, S.: Faster approximation schemes and parameterized algorithms on (odd-) H-minor-free graphs. Theoretical Computer Science 417, 95–107 (2012)

Restricted Max-Min Fair Allocations with Inclusion-Free Intervals*

Monaldo Mastrolilli and Georgios Stamoulis

IDSIA, Lugano, Switzerland
{monaldo,georgios}@idsia.ch

Abstract. We consider the restricted assignment version of the problem of fairly allocating a set of m indivisible items to n agents (also known as the Santa Claus problem). We study the variant where every item has some non-negative value and it can be assigned to an *interval* of players (i.e. to a set of consecutive players). Moreover, intervals are inclusion free. The goal is to distribute the items to the players and fair allocations in this context are those maximizing the minimum utility received by any agent. When every item can be assigned to any player a PTAS is known [Woe97]. We present a 1/2-approximation algorithm for the addressed more general variant with inclusion-free intervals.

1 Introduction

Max-min fair allocation is a resource allocation problem. In this setting we are asked to allocate a set of m indivisible resources denoted by I to a set of n customers denoted by K. Each customer $i \in [n]$ has an *additive* utility function $f_i : 2^R \to \mathbb{R}$ that is defined for every subset of the resources. For simplicity, we may define $f_i(j) = w_{ij} \in \mathbb{Q}$ where $i \in [n]$ and $j \in [m]$. This represents the evaluation of resource j from customer i and allocating resource j to customer i increases the utility of i by w_{ij}. The term additive means that $f_i(I') = \sum_{j \in I'} w_{ij}$ for $I' \subseteq I$. We are asked to find a partition of the resources $I_1 \cup I_2 \cup \cdots \cup I_n = I$ and allocate each bundle of resources to the corresponding customer such that the minimum utility received by any player is as high as possible. In other words the objective function is $\max \min_i f_i(I_i)$.

Recently, the problem was studied also under the name *Santa Claus* in the literature ([BS06]). We are interested in the following variant of the Santa Claus problem:

Definition 1 (Restricted Santa Claus with Inclusion-free Intervals). *Santa Claus has a set I of m presents to distribute to a set K of n kids. Each present has a value $w_j \in \mathbb{Z}^+$. Each kid wants a subset of the presents and the happiness of a kid increases by w_j if the kid is interested in item j, and that kid*

* Supported by the Swiss National Science Foundation Project N.200020-122110/1 "Approximation Algorithms for Machine Scheduling Through Theory and Experiments III" and by Hasler Foundation Grant 11099.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 98–108, 2012.

gets j. We assume that for each present j a set of consecutive kids (an interval) is interested on that present. Moreover, no interval is a proper subset of another interval i.e. if we have an interval $[l, r]$ then there does not exist another interval $[l', r']$ with $l' > l$ and $r' < r$ (but we allow to have a common endpoint).

More precisely, we have a bipartite graph $B = (I, K, E)$ with bipartition I and K, edge set E, $w(i, j) = w_j$, $\forall (i, j) \in E(B)$ and, moreover, an ordering of the vertices of K such that the neighborhood of every vertex in I is an interval in K. The goal is to distribute the presents to the kids such that the least happy kid is as happy as possible.

We assume from now on that our instance respects this ordering and is given in that way i.e. there is an ordering of the kids such that for every present $j \in I$ its neighborhood $N(j) \subseteq K$ is an interval (respecting the properties of the above definition). The addressed problem generalizes the one studied in [Woe97], where the graph is complete bipartite, i.e. every present can be assigned to every kid. Moreover, it incudes all instances (as the previous one) in which all the corresponding intervals of kids have the *same length*. Observe that this case of the Santa Claus problem is *strongly* **NP**-complete: it includes as a special case the case where the assignment graph is *complete* bipartite (a bi-clique), see [Woe97]. This case is known to be *strongly* **NP**-complete by a reduction from 3-partition, see [GJ79].

This model arises naturally and has applications, for example, when resources can be assigned and used in some time intervals and the goal is a fair allocation of the resources over time, i.e. an allocation that maximizes the minimum accumulated resource we collect at each time step (for example in energy production settings where the energy produced at some specific day is available for a fixed amount of time only). Moreover, it is a common scenario for a resource to request a contiguous segment in which it can be allocated. Such scenarios have been considered, for example, in [Sga96] in on-line scheduling settings.

1.1 Related Work

For the general Santa Claus problem no "good" approximation algorithm is known. In [BD05], an additive ratio of $\max_{ij} w_{ij}$ was given which can be arbitrarily bad. By defining a stronger LP formulation, the *Configuration LP*, it was shown in [BS06] that we can get value at least opt/n. Subsequently, in [AS07], [AS10] it was shown that it is possible to round a fractional solution of this LP in such a way that the objective function value that we get is not worse than $\frac{\text{opt}}{\sqrt{n}(\log^3 n)}$, where opt is the optimal objective function value. Recently it was shown, using the same LP, an almost tight approximation factor of $\Omega(\sqrt{\frac{\log\log n}{n \log n}})$ [SS10].

Subsequently, the interest has shifted towards the *restricted* case in which every item has the same value among all players that want it, i.e. $w_{ij} \in \{w_j, 0\}, \forall j \in [m]$. For this case a $\Omega(\frac{\log\log\log n}{\log\log n})$ factor approximation algorithm is known [BS06] and a simple $1/2$ inapproximability result [BD05]. In [Fei08] it was proved

that the value of the optimal solution of the configuration LP defined in [BS06] can be estimated to be at most a constant times better than the optimal value of the problem. In [AFS08] an $\frac{1}{5}$ integrality gap was shown for the same LP which later improved to $\frac{1}{4}$. This is an *estimation* ratio but they failed to provide a polynomial time $\frac{1}{4}$ approximation algorithm. Instead, they provided a local search heuristic with the same guarantee but, unfortunately, the procedure is not known to converge in polynomial time. Very recently it was shown that it can be done in $n^{O(\log n)}$ time (in contrast with the $O(2^n)$ original running time), see [PS12]. In [HSS11] the authors managed to make constructive Feige's original non constructive argument based on Lovász Local Lemma [AS00] so they demonstrated the first constant factor approximation of the restricted Santa Claus problem solving, to some extent, an important question. Notice that the result returns an α-approximation algorithm for some *constant* α. An explicit value of α is not provided. But still there is a gap between the $\frac{1}{2}$ inapproximability result and the constant α approximability result in [HSS11] and finding an explicit constant factor approximation algorithm for the (restricted) Santa Claus problem is considered a major open problem.

For the restricted Santa Claus problem, several special cases of the problem have been studied. To this direction in [KP07] it was considered the case where $w_{ij} \in \{0, 1, \infty\}$ and was established a tradeoff between running time and approximation guarantee. This special case is believed to be as hard as the general, but no formal proof of this is known. In [BCG09] the authors considered the case in which each item has bounded degree D (can be assigned to at most D kids). They proved that when $D \leq 3$ then the problem is as hard as in the general case. For the case of $D \leq 2$ they provided a $1/2$ inapproximability result and a $1/4$ approximation algorithm for the *asymmetric* case (in which the evaluation of this item by the two competing players is different). The authors also provided a simpler LP formulation for the general problem and devised a polylogarithmic approximation algorithm that runs in quasipolynomial time. The same result has been obtained in [CCK09] in which it was also shown a $1/2$ approximation for the $D \leq 2$ case, thus matching the bound proved in [BCG09] for this case.

In [Woe97], the author consider a special case of the problem considered in this paper, namely, the case where the graph is complete bipartite, i.e. every present can be assigned to every kid. It was shown that this version admits a PTAS based on dynamic programming techniques.

1.2 The Approach

Our algorithm works as follows. We first guess the value t of the optimal solution. For each t we present a polynomial time algorithm that either returns that no solution of value t exists or return a solution of value at least $t/2$ (this means, a solution such that each kid is satisfied to an extent of at least $t/2$). In order to find the maximum such t we simply perform a binary search in the range $[0, W]$, $W = \sum_j w(j)$. This results in a $1/2$-approximate solution. We first need a definition.

Definition 2 (t-assignment). *For any value of t a (fractional) t-assignment is a (fractional) assignment such that every kid is happy to the extent of at least t, i.e. the total value of the presents (fractionally) given to any child is at least t.*

For each specific value of t we perform the following steps (that structure our input):

Clipping: Firstly, we clip the value of every present with value greater that t down to t so that every present has value in the range of $(0, t]$. This is done without any loss because the notion of a t-assignment is preserved i.e. if a t assignment exists *before* this step, then a t-assignment exists *after* this clipping step.

Scaling step: Then, we *scale* the values of the presents such that any present has value in $(0, 1]$. This is done as follows: we simply divide their values by t. Every present now has value in the interval $(0, 1]$. We observe that this step is again performed without any loss: a t-assignment becomes 1-assignment.

Rounding step: Next we proceed to the *rounding* step. This step consists on partitioning the set of the presents into two sets: the *large* (or big) presents (with value $\geq 1/2$) and the *small* presents (with value $< 1/2$). After this partition, we *round* all the large presents to 1 and leave the values of all small presents unchanged. This step is done with some loss. Note that if a kid gets a (rounded) large present then after restoring the original value we know that he/she gets value at least $1/2$.

Our goal is to integrally assign the large presents and fractionally the small presents, in polynomial time. With this aim, we propose a Linear Programming (LP) formulation for identifying the set of kids that can be satisfied with large presents in such way that the remaining kids can be satisfied with a fractional assignment of small items (i.e. the remaining small presents are enough to satisfy the rest of the kids *fractionally*). Interestingly enough the constraint matrix defined by the LP formulation is *totally unimodular*, which allows us to *integrally* allocate the big presents and fractionally the small ones to the rest of the kids in polynomial time. Then we apply a result from [BD05] to obtain an integral allocation of the small presents to the rest of the kids so that every kid gets presents of value at least $1/2$:

Theorem 1. *There is an algorithm which returns in polynomial time in the input size an assignment (allocation) of presents to kids such that each kid is at least half as happy as it would be in the optimal assignment. In other words, there is a 1/2-approximation algorithm for the Santa Claus problem with inclusion free intervals.*

2 Satisfying All the Kids

As explained in Section 1, we transform our instance such that every present is either *large* (i.e. its value is 1) or *small* (its value is $< 1/2$). Our goal is to identify

which kids will be satisfied with large presents such that there are enough small items left for the remaining children to be satisfied fractionally. With this aim, we use the following well-known condition which is a sufficient and necessary condition for a bipartite graph to have a perfect matching:

Lemma 1 (Hall's condition, [HV50]). *Given a bipartite graph G with bipartition V and U, we say that G has a perfect matching (on V) if and only if $\forall S \subseteq V \ |S| \leq |N(S)|$ where $N(S)$ is the neighborhood of the set S i.e. the following set: $N(S) = \{u \in U : \ (v,u) \in E(G)$ for some $v \in S \ \}$.*

We say that a present j can be assigned to a kid i if there is an edge in our graph between j and i. For the kids that get large presents the above condition translates as follows: let K_{large} be the kids that we decided to assign large presents and $|N_{\text{large}}(K_{\text{large}})|$ is the number of large presents in their neighborhood. Then, we must have that

$$\forall S \subseteq K_{\text{large}}, \ |S| \leq |N_{\text{large}}(S)|. \tag{1}$$

Basically, this condition says that for a subset S of children we cannot assign more large items to S than the number we have in their neighborhood.

For the remaining children (those that get small items), if we define K_{small} in similar way, the following condition is necessary and sufficient for getting a fractional 1-assignment:

$$\forall S \subseteq K_{\text{small}}, \ |S| \leq w(N_{\text{small}}(S)) = \sum_{j \in N_{\text{small}}(S)} w(j) \tag{2}$$

where $w(j)$ is the value of present j (after the rounding).

In general a t-assignment exists if and only if

$$\begin{aligned} \forall S \subseteq K_{\text{small}}, \ |S| &\leq \frac{1}{t} \cdot w(N_{\text{small}}(S)) \\ &= \frac{1}{t} \cdot \sum_{j \in N_{\text{small}}(S)} w(j) \end{aligned}$$

where $w(j)$ is the value of the (small) present j.

Remark: Observe that this fractional version of Hall's condition is in fact *equivalent* to the integral Hall's condition: assuming, without any loss, that t and $w(j)$ are (positive) integers, replace every (small) present j of value $w(j)$ with $w(j)$ copies of value 1. Replace each child i with t new copies of it and connect each such copy to every copy of the presents that i was initially connected with. Observe that we have $\sum_i t = t|K|$ many new kids. It is easy to verify that the fractional version of Hall's condition is true in the initial graph if and only if the integral version of Hall's condition is true in the new graph.

2.1 A Linear Program to Assign Large Presents Integrally and Small Presents Fractionally

Basically, as we explained, we need to decide which kids will be satisfied integrally by large presents and which will be satisfied (fractionally) by small presents (i.e. to identify a set $K_{\text{large}} \subseteq K$ such that we have enough large presents for this set and enough small presents to satisfy the kids in $K \setminus K_{\text{large}}$ fractionally). The idea is to "translate" conditions (1) and (2) as linear constraints. Towards this goal, we define the following set of (binary) variables:

$$L_i = \begin{cases} 1 & \text{if kid } i \text{ gets a large present} \\ 0 & \text{otherwise.} \end{cases}$$

Let ℓ_{lr} denote the number of the *large* presents that can be assigned to kids in the *interval* $[l, r]$ for $l \leq r$. To satisfy Hall's condition on large presents we need, for any interval of kids $[l, r]$ to assign no more than ℓ_{lr} large presents to that interval. This translates to the following constraints:

$$\sum_{i=l}^{r} L_i \leq \ell_{lr}, \qquad \forall\ l \leq r \tag{3}$$

We need to do the same for the small presents i.e. for any possible subset of kids that we decided to make them happy with small presents, the available amount of small presents that can be assigned to them should be enough to satisfy them all (fractionally):

$$\sum_{i=l}^{r}(1 - L_i) \leq \sum_{j \in N_{\text{small}}([l,r])} w(j) = w(N_{\text{small}}([l, r])), \qquad \forall\ l \leq r \tag{4}$$

Since we have polynomially many possible intervals (namely $\mathcal{O}(n^2)$), this means we have polynomially many constraints of the above type. We will show that the constraint matrix, as defined above, is a special matrix:

Definition 3 ([Sch03], [Sch98]). *An integer square matrix A is called **unimodular** matrix if its determinant is either +1 or -1. Equivalently, a square matrix is unimodular if it is invertible over the integers. A matrix M is a **totally unimodular** matrix if every square non-singular submatrix of M is unimodular.*

Totally unimodular matrices have some very interesting and well known properties:

Lemma 2 ([Sch03], [Sch98]). *If $A \in \{-1, 0, 1\}^{m \times n}$ is a totally unimodular matrix and $\beta \in \mathbb{Z}^m$ is an integral vector, then linear programs of form $\min c^T x$ s.t. $Ax \leq \beta$ have integral optima, for any $c \in \mathbb{Z}^n$.*

Of a particular interest in our case is a special case of totally unimodular matrices, called *interval* matrices:

Definition 4 ([Sch03]). *A matrix A is called an **interval** matrix if it is a binary matrix (i.e. all entries are either 0 or 1) and, moreover, in every row all the ones (if any) are appearing consecutively. We also say that such a matrix has the consecutive ones property.*

It is well known fact that interval matrices are totally unimodular (see the above references).

We observe that the matrix defined by inequalities (3) and (4) is an interval matrix, thus the polyhedron defining the solution space of this linear program is *integral* (see Lemma 2). This means that if the the LP (after relaxing the binary variables to be $L_i \in [0,1]$) defined by all the constraints (3) and (4) is feasible, the solution returned will have integral coordinates i.e. $L_i \in \{0,1\}\ \forall i$ and in the next sub-section e will show how to assign presents to kids in such a way that every kid gets value at least $1/2$. Moreover, if no feasible solution exists that respects constraints (3) and (4) then no solution for the Santa claus problem exists with value 1 and we continue with our binary search procedure.

2.2 Allocating the Presents

What we need to show first is that the decision of which kids will be satisfied by large presents is done such that it does not violate (the integral version of) Hall's condition (in the following, by $N(S)$ for $S \subseteq K$, we mean $N_{\text{large}}(S)$):

Lemma 3. *Let $\Lambda = \{i \in K :\ L_i = 1\}$. Then the (integral) Hall's condition is satisfied for Λ i.e. $\forall S \subseteq \Lambda :\ |S| \leq |N(S)|$.*

Proof. Let $S \subseteq \Lambda$. Assume that α is the first kid in the set S and β the last kid in it. Since we obtained the values for the L_i's by solving the Integer Linear Program (defined by constraints (3) and (4)), we know these values satisfy the constraint $\sum_{i\in[\alpha,\beta]} L_i \leq \ell_{\alpha\beta}$. Now, we need to distinguish between two cases regarding the structure of S:

- S **is an interval:** In this case $S = [\alpha, \beta]$ and Hall's condition is satisfied because of constraint (1).
- S **is not an interval:** Then, is this case, S can be represented as a union of (disjoint) intervals: $S = S_1 \cup S_2 \cup \cdots \cup S_\gamma$ where all the S_i's, $i \in [\gamma]$, are all intervals. By the previous case we know that Hall's condition is satisfied for any interval S_i, i.e. $|S_i| \leq |N(S_i)|$. First, we will show that for any two consecutive subintervals S_i and S_{i+1} where $1 \leq i \leq \gamma - 1$ Hall's condition is satisfied as well, i.e. $|S'| = |S_i| + |S_{i+1}| \leq |N(S')|$. Without any loss of generality, assume that these two subintervals are the first two: S_1 and S_2. Let H be the "hole" (set of kids) between S_1 and S_2. Let $S' = S_1 \cup S_2$. Then $N(S') = N(S_1) \cup N(S_2)$ and $|N(S')| = |N(S_1)| + |N(S_2)| - |N(S_1) \cap N(S_2)|$. Let $\Phi = N(H) \setminus (N(S_1) \cap N(S_2))$.
  Assume first that $\Phi = \emptyset$. This means that there does not exist a present that sees an interval entirely inside H. Then we have that $|N(S_1 \cup S_2 \cup H)| = |N(S_1 \cup S_2)|$. Now let α be the first kid in S' and β the last one.

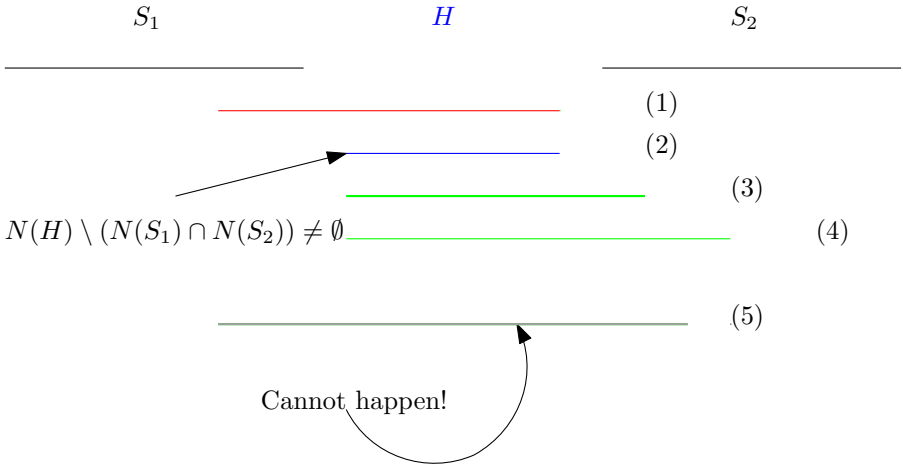


Fig. 1. Second case. We know that the blue interval (number 2) exists. So, any other interval that intersects it cannot be proper subinterval or proper super-interval of it (i.e the interval number 5 cannot exist). But they can share one common endpoint, as in intervals 1,3 and 4.

From the previous we have that the two subintervals S_1, S_2 cover that entire neighborhood of the interval $[\alpha, \beta] = S_1 \cup S_2 \cup H$ and so $|N(S')| = |N(S_1 \cup S_2)| = |N([\alpha, \beta])|$ which by definition equals to $\ell_{\alpha\beta}$. But also we know that $|S'| \leq \ell_{\alpha\beta}$. This is because $S' \subset \Lambda = \{k : L_k = 1\}$ so by the LP constraint $\sum_{k \in [\alpha,\beta]} L_k \leq \ell_{\alpha\beta}$ and moreover $S' \subset [\alpha, \beta]$ (because of the presence of the hole H). So, we conclude that $|S'| \leq \ell_{\alpha,\beta} = |N(S')|$.

If $\Phi \neq \emptyset$ then this means that there exists a present that sees an interval entirely inside H. Because of the assumption that all intervals are sorted such that they are inclusion free intervals, this means that there cannot exist a present that can be assigned in kids both S_1 and S_2 i.e. no present's interval extends from S_1 to S_2, or else, it would violate the requirements of the ordering of our instance (inclusion free but can share a common endpoint), since there is a present with interval entirely inside H. So, we conclude that $|N(S_1) \cap N(S_2)| = 0$ and so $|N(S')| = |N(S_1)| + |N(S_2)| - |N(S_1) \cap N(S_2)| = |N(S_1)| + |N(S_2)| \geq |S_1| + |S_2| = |S|'$ (because S_1 and S_2 are intervals). So, we conclude that $|S'| = |S_1| + |S_2| \leq |N(S_1)| + |N(S_2)| = |N(S')|$ and Hall's condition is satisfied.

Now, like before, assume that $S = S_1 \cup S_2 \cup \cdots \cup S_\gamma$. Let H_i be the "hole" between the subintervals S_i and S_{i+1}. So we have a collections of holes $\mathcal{H} = \{H_1, H_2, \ldots, H_{\gamma-1}\}$. Define, like before, $\Phi_i = N(H_i) \setminus N((S_i \cup S_{i+1}) \cap H_i)$. A present belongs to the set Φ_i if the interval of kids that can be assigned is *entirely* inside H_i or inside $H_i \cup S_i$ or $H_i \cup S_{i+1}$ but cannot be assigned to a kid in S_i and in S_{i+1} (see Figure 1). Now, compute the largest index j such that $\Phi_i = \emptyset \; \forall i \leq j$. This means that $|N(S_1 \cup H_1 \cup S_2 \cup H_2 \cup \cdots \cup H_{j-1} \cup S_j)|$

$= |N(S_1 \cup S_2 \cup \cdots \cup S_j)| = |N([\alpha_1, \beta_j])|$, where α_i (β_i) is the first (last) kid in the interval S_i. But $|N([\alpha_1, \beta_j])| = \ell_{\alpha_1\beta_j} = |N(S_1 \cup S_2 \cup \cdots \cup S_j)|$. And, as before, from the constraint of the LP we have that $|S_1 \cup \cdots \cup S_j| \leq \ell_{\alpha_1\beta_j}$. So $|S_1 \cup S_2 \cup \cdots \cup S_j| = |S_1| + \cdots |S_j| \leq |N(S_1 \cup S_2 \cup \cdots \cup S_j)|$.

Define $C_1 = S_1 \cup S_2 \cup \cdots \cup S_j$. From the computation of the index j we know that $\Phi_{j+1} \neq \emptyset$. Again compute the largest index $j' > j$ such that $\Phi_{j'+1} \neq \emptyset$ and define in similar way C_2 and so on. So, we have defined the set $\mathcal{C} = \{C_1, C_2, \ldots, C_p\}$, for some p computed in the way just described. For every $C_q \in \mathcal{C}$ we know that $|C_q| \leq |N(C_q)|$. Between any pair of C_l and C_{l+1}, $1 \leq l \leq p-1$ there is a hole with index lets say x such that $\Phi_x \neq \emptyset$. With an argument identical as before (when we had two subintervals to consider) we have that $|N(C_l) \cap N(C_{l+1})| = 0$ (since there is no "crossing interval") and so $|N(C_1 \cup C_2 \cup \cdots \cup C_p)| = |N(C_1| + \cdots |N(C_p)| \geq |C_1| + \cdots |C_p| = |S|$.

□

We need to show the same for the small items i.e. that (the fractional version of) Hall's condition on small presents and the rest of the kids is satisfied:

Lemma 4. *Let $\bar{\Lambda} = \{i \notin \Lambda\} = \{i \in K : \ L_i = 0\}$. For any possible subset S of $\bar{\Lambda}$ we have that $|S| \leq w_{small}(N(S))$.*

Proof. The proof of this is identical as the proof of the previous lemma and we omit it.

□

Corollary 1. *The decision of which kids will be assigned large presents is such that Hall's condition is preserved for both the large and the non large items and can be done in polynomial time.*

Now that we have decided which kids will take large items, we need a way of actually *assigning* large items to those kids. This can be easily done by the following simple argument: let $B_{\text{large}}(K_{\text{large}}, I_{\text{large}}, E)$ be the bipartite subgraph induced by the large presents and all the "large" kids (i.e. all the kids i for which $L_i = 1$). In this subgraph find a *perfect matching* (perfect on the side of the kids). This means that every kid i with $L_i = 1$ receives a large present. Lemma 3 guarantees that Hall's condition is satisfied in the graph B_{large} and so such a perfect matching always exists.

Since every large present has value $\geq 1/2$, when we restore the original value of this present, we see that each kid $i \in K_{\text{large}}$ is satisfied to an extent of at least $1/2$. Next we will show how we can obtain the same guarantee for the rest of the kids as well (kids that get satisfied by only small presents).

2.3 Assigning Small Presents to the Rest of the Kids

So now we are left with all the kids i for which $L_i = 0$ and we need to satisfy them as much as possible with the remaining small presents. In other words, we are left with the following instance $B' = (\{i \in K : L_i = 0\}, I_{\text{small}}, E')$, i.e.

with the subgraph induced by the "small" kids (for which $L_i = 0$) and the small presents. The only guarantee we have so far is that (fractional) Hall's condition is true in B' (Lemma 4), so this means that we can *fractionally* satisfy every kid to an extent of at least 1.

To satisfy these kids integrally, we use an observation made in [BD05]. There, using ideas from [ST93] (see also [LST87]), it was shown how to obtain a feasible integral solution of value at least $\text{opt} - w_{\max}$, where $w_{\max} = \max_{j \in I} w(j)$. Since in our case we have $w_{\max} < 1/2$ (only small items) and $\text{opt} = 1$ (Hall's condition is true and we are seeking a 1-assignment) we immediately get that every kid is satisfied to an extent of at least $1/2$.

Finally, for convenience, we summarize our algorithm:

Algorithm 1. A $\frac{1}{2}$-approximation Algorithm for Max-Min Fair Allocation with Inclusion free Intervals

---

Input: An instance of (restricted) max-min fair allocation $B = (I, K, E)$ as described in Definition 1.
Output: A *feasible* $\frac{1}{2}$-approximate allocation.

1. Guess the value t of the optimal solution.
2. For a particular t, structure the input according to the described Clipping, Scaling and Rounding steps.
3. Define and solve the ILP defined by the constraints (3) and (4).
4. **if** the ILP is feasible **then**
   (a) Consider the induced subgraph $B_{\text{large}}(K_{\text{large}}, I_{\text{large}}, E)$ where $K_{\text{large}} = \{i \in K : L_i = 1\}$.
   (b) Find a perfect matching on the side of K_{large}.
   (c) Consider the *rest* of the graph (induced by the small presents and the set of kids $K_{\text{small}} = \{i \in K : L_i = 0\}$). Let B_s this graph.
   (d) Apply the algorithm in [BD05] on B_s to obtain a $\text{opt} - w_{\max} \geq \frac{1}{2}$ integral allocation.

Corollary 2. *If a (fractional) t-assignment exists for some t i.e. if the ILP defined by all inequalities described in (3) and (4) is feasible for some particular value of t, then we can compute an (integral) $t/2$-assignment in polynomial time.*

Future Work: We note that the problem studied in this article is not known to be **APX**-hard, and so a PTAS is not ruled out. We leave this as a research direction for the future.

Acknowledgments. We would like to thank Arash Rafiey for many usefull discussions.

References

[AFS08] Asadpour, A., Feige, U., Saberi, A.: Santa Claus Meets Hypergraph Matchings. In: Goel, A., Jansen, K., Rolim, J.D.P., Rubinfeld, R. (eds.) APPROX and RANDOM 2008. LNCS, vol. 5171, pp. 10–20. Springer, Heidelberg (2008)

[AS00] Alon, N., Spencer, J.H.: The probabilistic method, 2nd edn. Wiley-Interscience, New York (2000)

[AS07] Asadpour, A., Saberi, A.: An approximation algorithm for max-min fair allocation of indivisible goods. In: STOC, pp. 114–121. ACM (2007)

[AS10] Asadpour, A., Saberi, A.: An approximation algorithm for max-min fair allocation of indivisible goods. SIAM J. Comput. 39(7), 2970–2989 (2010)

[BCG09] Bateni, M.H., Charikar, M., Guruswami, V.: Maxmin allocation via degree lower-bounded arborescences. In: STOC. ACM (2009)

[BD05] Bezáková, I., Dani, V.: Allocating indivisible goods. SIGecom Exchanges 5(3), 11–18 (2005)

[BS06] Bansal, N., Sviridenko, M.: The santa claus problem. In: STOC, pp. 31–40. ACM (2006)

[CCK09] Chakrabarty, D., Chuzhoy, J., Khanna, S.: On allocating goods to maximize fairness. In: FOCS, pp. 107–116 (2009)

[Fei08] Feige, U.: On allocations that maximize fairness. In: SODA, pp. 287–293. ACM-SIAM (2008)

[GJ79] Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)

[HSS11] Haeupler, B., Saha, B., Srinivasan, A.: New constructive aspects of the lovász local lemma. J. ACM 58(6), 28 (2011)

[HV50] Halmos, P.R., Vaughan, H.E.: The marriage problem. American Journal of Mathematics, 214–215 (1950)

[KP07] Khot, S., Ponnuswami, A.K.: Approximation Algorithms for the Max-Min Allocation Problem. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) RANDOM 2007 and APPROX 2007. LNCS, vol. 4627, pp. 204–217. Springer, Heidelberg (2007)

[LST87] Lenstra, J.K., Shmoys, D.B., Tardos, É.: Approximation algorithms for scheduling unrelated parallel machines. In: FOCS, pp. 217–224. IEEE (1987)

[PS12] Polacek, L., Svensson, O.: Quasi-polynomial local search for restricted max-min fair allocation. In: ICALP 2012 (2012)

[Sch98] Schrijver, A.: Theory of Linear and Integer Programming. John Wiley & Sons (1998)

[Sch03] Schrijver, A.: Combinatorial optimization: polyhedra and efficiency. Springer (2003)

[Sga96] Sgall, J.: Randomized on-line scheduling of parallel jobs. J. Algorithms 21(1), 149–175 (1996)

[SS10] Saha, B., Srinivasan, A.: A new approximation technique for resource-allocation problems. In: ICS, pp. 342–357. Tsinghua University Press (2010)

[ST93] Shmoys, D.B., Tardos, É.: An approximation algorithm for the generalized assignment problem. Math. Program. 62, 461–474 (1993)

[Woe97] Woeginger, G.: A polynomial-time approximation scheme for maximizing the minimum machine completion time. Operation Research Letters 20(4), 149–154 (1997)

An Improved Algorithm for Packing T-Paths in Inner Eulerian Networks

Maxim A. Babenko*, Kamil Salikhov, and Stepan Artamonov

Moscow State University Yandex LLC
{maxim.babenko,salikhov.kamil}@gmail.com,
swatmad@list.ru

Abstract. A digraph $G = (V, E)$ with a distinguished set $T \subseteq V$ of *terminals* is called *inner Eulerian* if for each $v \in V - T$ the numbers of arcs entering and leaving v are equal. By a T-*path* we mean a simple directed path connecting distinct terminals with all intermediate nodes in $V - T$. This paper concerns the problem of finding a maximum T-path packing, i.e. a maximum collection of arc-disjoint T-paths.

A min-max relation for this problem was established by Lomonosov. The capacitated version was studied by Ibaraki, Karzanov, and Nagamochi, who came up with a strongly-polynomial algorithm of complexity $O(\phi(V, E) \cdot \log T + V^2 E)$ (hereinafter $\phi(n, m)$ denotes the complexity of a max-flow computation in a network with n nodes and m arcs).

For unit capacities, the latter algorithm takes $O(\phi(V, E) \cdot \log T + VE)$ time, which is unsatisfactory since a max-flow can be found in $o(VE)$ time. For this case, we present an improved method that runs in $O(\phi(V, E) \cdot \log T + E \log V)$ time. Thus plugging in the max-flow algorithm of Dinic, we reduce the overall complexity from $O(VE)$ to $O(\min(V^{2/3}E, E^{3/2}) \cdot \log T)$.

1 Preliminaries

1.1 Introduction

Computing a maximum integer flow, i.e. a maximum packing of paths connecting a given pair of terminals subject to edge capacities, is widely regarded as a central problem in combinatorial optimization. This problem has myriads of applications, both theoretical and practical.

Given a graph $G = (V, E)$ (either directed or undirected) and arbitrary integer capacities $e\colon E \to \mathbb{Z}_+$, one of the best strongly-polynomial max-flow algorithm [12] runs in $O(VE \log(V^2/E))$ time. (Hereinafter, in notation involving functions of numerical arguments or time bounds, we indicate sets for their cardinalities.) More efficient methods are known for the special case of unit capacities. The oldest one belongs to Dinic [3] and runs in $O(\min(E^{3/2}, V^{2/3}E))$ time (as shown independently by Karzanov [17] and Even and Tarjan [5]). Better results

* Supported by RFBR grants 10-01-93109 and 12-01-00864.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 109–120, 2012.

were recently discovered conerning max-flows in undirected unit-capacitated networks. Karger proposed an $M^*(V^{3/2}E^{2/3})$-time randomized algorithm [14,15], Goldberg and Rao gave an $O(V^{3/2}E^{1/2})$-time deterministic algorithm [11], and finally Karger and Levine presented a deterministic $O(V^{7/6}E^{2/3})$-time algorithm and a randomized $M^*(V^{20/9})$-time algorithm [16] (here $M^*(\cdot)$ denotes expected time with omitted polylogarithmic factors).

Multiflows are similar to usual flows but involve a set of terminals T that can be arbitrarily large. Most versions of the maximum integer multiflow problem are NP-hard. Still if the network is undirected and paths in a multiflow are allowed to connect arbitrary pairs of (distinct) terminals then the problem is tractable [24]. (This case is sometimes referred to as a *free multiflow*.) For directed networks one must additionally require capacities at all inner (non-terminal) nodes to be *balanced*. Considering this version of the problem, Lomonosov derived a simple min-max formula.

Ibaraki, Karzanov, and Nagamochi [13] applied a "divide and conquer" approach (originally introduced by Karzanov [19]) and devised a strongly-polynomial $O(\phi(V,E)\cdot \log T + V^2E)$-time algorithm for T-path packing problem in a capacitated digraph (V,E). The latter algorithm incorporates an arbitrary max-flow routine that runs in $\phi(n,m)$ time for a network with n nodes and m arcs. Since $\phi(V,E) = O(VE\log(V^2/E))$ (as, e.g., in the algorithm of Goldberg and Tarjan [12]), the term $O(V^2E)$ becomes a bottleneck. This issue was partially resolved in [1], where the total complexity was decreased to $O((\phi(V,E)+VE)\cdot\log T + VE\log(V^2/E))$.

The present paper concerns unit-capacitated networks. For this case, the algorithm from [13] takes $O(\phi(V,E)\cdot\log T+VE)$ time and since $\phi(V,E)=o(VE)$ the second term remains a bottleneck.

We introduce a novel *discrepancy scaling* technique and prove the following:

Theorem 1. *A maximum integer multiflow in an inner Eulerian digraph $G=(V,E)$ with terminals $T\subseteq V$ can be found in $O(\phi(V,E)\cdot\log T + E\log V)$ time.*

Our approach also extends to integer capacities as follows:

Theorem 2. *A maximum integer multiflow in an inner Eulerian digraph $G=(V,E)$ with terminals $T\subseteq V$ and integer capacities not exceeding C can be found in $O(\phi(V,E)\cdot\log T + E\log V\log T + E\log(V^2/E)\log(VC))$ time.*

The latter improves the bound $O(\phi(V,E)\log T + VE)$ from [1] provided that $\phi(n,m)=o(mn)$ (e.g. if the weakly-polynomial algorithm of Goldberg and Rao [11] is applied). In sense of capacity scaling, this improvement is ultimate since for all currently known scaling max-flow algorithms the first term is dominant.

1.2 Basic Notation and Facts

We shall use some standard definitions and notation. For a graph G, its set of nodes and edges (or arcs) are denoted by $V(G)$ and $E(G)$, respectively. A similar notation is used for paths and cycles.

Let G be a digraph and X be a subset of nodes. Then $G[X]$ denotes the subgraph of G induced by X. Also $\delta^{\text{in}}_G(X)$ (respectively $\delta^{\text{out}}_G(X)$) denotes the set of arcs entering X (respectively leaving X) and $\delta_G(X) := \delta^{\text{in}}_G(X) \cup \delta^{\text{out}}_G(X)$. When G is clear from the context, it is omitted. Also for $X = \{v\}$ we write just $\delta^{\text{in}}(v)$ and $\delta^{\text{out}}(v)$.

For an arbitrary real-valued function $h\colon U \to \mathbb{R}$ and $U' \subseteq U$ we write $h(U')$ to denote $\sum_{u \in U'} h(u)$.

A digraph $G = (V, E)$ with a distinguished subset $T \subseteq V$ is said to be *inner Eulerian* if for each $v \in V - T$ the in-degree and out-degree of v are equal. Nodes in T and in $V - T$ are called *terminals* and *inner* nodes, respectively. A simple path in G is called a *T-path* if its endpoints are distinct terminals and the other nodes are inner.

By a network we mean a triple $N = (G, T, c)$ consisting of a digraph $G = (V, E)$, a set of terminals T, and a non-negative function $c\colon E \to \mathbb{Z}_+$ of arc *capacities*. The notion of inner Eulerianess is extended to N in a natural way, namely $c(\delta^{\text{in}}(v)) = c(\delta^{\text{out}}(v))$ should hold for all $v \in V - T$. In case of unit capacities we omit c from notation and deal with the pair $N = (G, T)$.

A collection $\mathcal{P}$ consisting of T-paths P_i endowed with real weights $\alpha_i \in \mathbb{R}_+$ such that

$$\sum(\alpha_i : e \in E(P_i)) \le c(e) \qquad \text{for all } e \in E \tag{1}$$

is called a *free multiflow* or a *T-path packing* (the adjective "free" is used to emphasize that any pair of distinct terminals is allowed to be connected, i.e., the commodity graph in the corresponding multiflow problem is complete). The *value* of $\mathcal{P}$ is the sum

$$\text{val}(\mathcal{P}) := \alpha_1 + \ldots + \alpha_k$$

and $\mathcal{P}$ is called maximum if $\text{val}(\mathcal{P})$ is maximum.

If all numbers α_i are integers then $\mathcal{P}$ is called *integer*. Integer multiflows can be viewed as multisets of T-paths. If $c(e) = 1$ for all $e \in E$ then T-paths forming an integer multiflow are arc-disjoint. This case will be our primary focus.

The maximum integer multiflow problem in inner Eulerian networks admits a min-max relation, which is due to Lomonosov. (See also [18, Sec. 4], [8] and [25, p. 1289].) For $t \in T$, call $X \subset V$ a *t-cut* if $X \cap T = \{t\}$. Denote by $\lambda^{\text{out}}(t)$ the minimum of $c(\delta^{\text{out}}(X))$ over all t-cuts X.

Theorem 3 (Lomonosov (unpublished, 1978), Frank [8]). *For $N = (G, T, c)$ as above, there exists a maximum integer directed multiflow $\mathcal{P}$ in N with*

$$\text{val}(\mathcal{P}) = \sum_{t \in T} \lambda^{\text{out}}(t).$$

For $t \in T$, let X_t be a t-cut with $c(\delta^{\text{out}}(X_t))$ minimum. Note that inner Eulerianness of N implies

$$c(\delta^{\text{out}}(X)) - c(\delta^{\text{in}}(X)) = c(\delta^{\text{out}}(t)) - c(\delta^{\text{in}}(t)) \quad \text{for each } t\text{-cut } X. \tag{2}$$

Thus the minima of $c(\delta^{\text{out}}(X_t))$ and $c(\delta^{\text{in}}(X_t))$ are obtained simultaneously. Also note that

$$\sum_{t\in T}\left(c(\delta^{\text{out}}(t)) - c(\delta^{\text{in}}(t))\right) = 0$$

and hence

$$\sum_{t\in T} c(\delta^{\text{out}}(X_t)) = \sum_{t\in T} c(\delta^{\text{in}}(X_t)).$$

Therefore every optimal $\mathcal{P}$ saturates the cuts $\delta^{\text{out}}(X_t)$ and $\delta^{\text{in}}(X_t)$ for $t \in T$.

We shall also rely on analogous facts concerning *undirected* networks. A network $N = (G, T, c)$ consisting of an undirected graph $G = (V, E)$, a set of terminals $T \subseteq V$, and integer capacities $c\colon E \to \mathbb{Z}_+$ is called *inner Eulerian* if $c(\delta(t))$ is even for all $t \in V - T$. The notions of T-paths, t-cuts ($t \in T$), and free multiflows in N are defined similar to the directed case. For $t \in T$, let $\lambda(t)$ be the capacity of a minimum t-cut X_t. Lovász and Cherkassky, independently, established the following min-max relation:

Theorem 4 (Lovász [22], Cherkassky [2]). *For $N = (G, T, c)$ as above, there exists a maximum integer undirected multiflow $\mathcal{P}$ in N with*

$$\text{val}(\mathcal{P}) = \frac{1}{2}\sum_{t\in T}\lambda(t).$$

As above, every optimal multiflow $\mathcal{P}$ saturates cuts $\delta(X_t)$ for $t \in T$.

2 Algorithm

2.1 Outline

Modern efficient methods for computing maximum free multiflows rely on "divide and conquer" approach, which was originally applied in [19] to find, in $O(\phi(V, E) \cdot \log T)$ time, a *half-integer* maximum multiflow in a undirected graph $G = (V, E)$ with integer edge capacities. Subsequently, this method was improved and extended in [13] so as to find an integer maximum free multiflow in an inner Eulerian undirected network in the same time $O(\phi(V, E) \cdot \log T)$, and in an inner Eulerian directed network in $O(\phi(V, E) \cdot \log T + V^2 E)$ time.

Consider a directed unit-capacitated network $N = (G, T)$ with $|T| \geq 4$. The problem for N is recursively reduced to problems in two networks $N' = (G', T')$ and $N'' = (G'', T'')$ such that $|T'|, |T''| \leq \lceil |T|/2 \rceil + 1$. First, fix an arbitrarily partition $\{S', S''\}$ of T into parts of almost equal sizes. Second, compute an S'-cut X' (i.e. a subset $X' \subset V(G)$ with $X' \cap T = S'$) of minimum capacity $c(\delta^{\text{out}}(X'))$ by running a max-flow routine and considering nodes S' as sources and nodes S'' as sinks. Form graph G' by contracting X' in G into a new *composite* node (denoted by t'). We identify arcs in graphs resulting from contractions with their pre-images in original graphs. Define the set of terminals in G' as $T' := \{t'\} \cup S''$. Similarly $N'' = (G'', T'')$ is constructed by contracting $X'' := V(G) - X'$ in G into a new node t'' and setting $T'' := \{t''\} \cup S'$.

Remark 1. The complexity of this *separation* procedure is clearly dominated by the min-cut computation. Note that in view of (2) (which holds for S'-cuts as well) the requested minimum cut can be found by computing a max-flow in the unit-capacitated underlying *undirected* graph (obtained from G by dropping arc directions). This enables to use faster max-flows algorithms designed to handle undirected graphs (e.g. [16]).

Next, the algorithm proceeds to N' and N'' recursively and computes optimal solutions $\mathcal{P}'$ and $\mathcal{P}''$ to N' and N'', respectively. Finally, the *aggregation* procedure combines $\mathcal{P}'$ and $\mathcal{P}''$ into a maximum integer multiflow $\mathcal{P}$ in N as follows. We shall assume that $\mathcal{P}'$ and $\mathcal{P}''$ are given explicitly, i.e. as collections of arc-disjoint T'- and T''-paths, respectively. Define $\mathcal{P}'_0 := \{P \in \mathcal{P}' \mid t' \text{ is not an endpoint of } P\}$ and, symmetrically, $\mathcal{P}''_0 := \{P \in \mathcal{P}'' \mid t'' \text{ is not an endpoint of } P\}$. By the maximality of $\mathcal{P}'$ and $\mathcal{P}''$ and the minimality of X', paths in $\mathcal{P}' - \mathcal{P}'_0$ saturate the cut $\delta_{G'}(t')$ and, symmetrically, paths in $\mathcal{P}'' - \mathcal{P}''_0$ saturate the cut $\delta_{G''}(t'')$. This enables to recombine these paths into a collection $\mathcal{P}_1$ of T-paths (connecting terminals in S' with terminals in S''). The final packing $\mathcal{P}$ in N is defined as $\mathcal{P} := \mathcal{P}'_0 \cup \mathcal{P}''_0 \cup \mathcal{P}_1$. For the proof of maximality of $\mathcal{P}$ and a more detailed exposition, see [13]. The aggregation procedure runs in $O(V + E)$ time, which, compared to the separation phase, is negligible.

For $|T| = 3$, the above method is inapplicable (since it yields $|T'| = |T''| = 3$) so this basic case is handled separately, as explained below in Subsection 2.3.

Let $T(n, m, k)$ denote the complexity of the algorithm in a network N with n nodes, m arcs, and k terminals. Then for $k \geq 4$

$$T(n, m, k) = T(n', m', k') + T(n'', m'', k'') + \phi(n, m) + O(m + n), \qquad (3)$$

where (n', m', k') and (n'', m'', k'') denote analogous size parameters for networks N' and N'', respectively, and $\phi(n, m)$ is the complexity of a max-flow routine in a network with n nodes and m arcs. (As indicated earlier, the latter routine can be assumed to deal with an undirected unit-capacitated graph.) Also, as we shall show in Subsection 2.3, for $k \leq 3$

$$T(n, m, k) = O(\phi(n, m) + m \log n). \qquad (4)$$

Assuming that $\phi(n, m)$ is "reasonable" one can solve (3) and (4) as follows (see [13] for a detailed proof):

$$T(n, m, k) = O(\phi(n, m) \log k + m \log n).$$

It remains to show how to solve the problem for a directed unit-capacitated network $N = (G, T)$ with $|T| \leq 3$ terminals in $O(\phi(V, E) + E \log V)$ time. For this we shall need some terminology and basic facts concerning *skew-symmetric graphs* (which were earlier introduced as a convenient tool for solving flow and matching problems; see [10,1] for a survey).

2.2 Skew-Symmetric Graphs

A *skew-symmetric graph* is a digraph $G = (V, E)$ endowed with two bijections σ_V, σ_E such that: σ_V is an involution on the nodes (i.e., $\sigma_V(v) \neq v$ and $\sigma_V(\sigma_V(v)) = v$ for each $v \in V$), σ_E is an involution on the arcs, and for each arc e from u to v, $\sigma_E(e)$ is an arc from $\sigma_V(v)$ to $\sigma_V(u)$. For brevity, we combine the mappings σ_V, σ_E into one mapping σ on $V \cup E$ and call σ the *symmetry* (rather than skew-symmetry) of G. For a node (arc) x, its symmetric node (arc) $\sigma(x)$ is also called the *mate* of x, and we will often use notation with primes for mates, denoting $\sigma(x)$ by x'. Obviously $\delta^{\text{in}}(v)' = \delta^{\text{out}}(v')$ and $\deg^{\text{in}}(v) = \deg^{\text{out}}(v')$ for each $v \in V$.

We admit parallel arcs, but not loops in G. Observe that if G contains an arc e from a node v to its mate v', then e' is also an arc from v to v' (so the number of arcs of G from v to v' is even and these parallel arcs are partitioned into pairs of mates).

The symmetry σ is extended in a natural way to paths, circuits, subsets etc. Namely, two paths are symmetric to each other if the elements of one of them are symmetric to those of the other and go in the reverse order: for a path $P = (v_0, e_1, v_1, \ldots, e_k, v_k)$, the symmetric path P' is $(v'_k, e'_k, v'_{k-1}, \ldots, e'_1, v'_0)$.

2.3 Case $|T| \leq 3$

Consider a directed unit-capacitated network $N = (G = (V, E), T)$ with $|T| \leq 3$. Adding an isolated terminal (if needed) one may assume that $|T| = 3$. The best known algorithm that finds a maximum free multiflow in N runs in $O(\phi(V, E) + VE)$ time [13] . We shall improve this to $O(\phi(V, E) + E \log V)$ as follows.

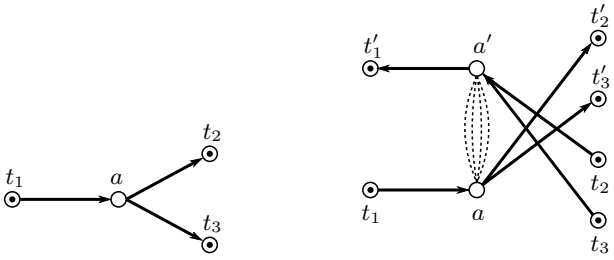
Stage 1: Define $T = \{t_1, t_2, t_3\}$ and let $\overline{G}$ be the underlying undirected graph of G. Apply the algorithm for inner Eulerian undirected graphs from [13, Sec.2.1] to find a maximum integer free multiflow $\overline{\mathcal{P}}$ in $\overline{N} = (\overline{G}, T)$ (endowed with unit capacities). This takes $O(\phi(V, E))$ time.

Remark 2. The latter algorithm involves computing two max-flows, say g_1 and g_2, where only g_1 is undirected. Namely, g_1 is a max-flow in $\overline{G}$ from t_1 to $\{t_2, t_3\}$, and g_2 is a max-flow from t_2 to t_3 in the *residual network* w.r.t. g_1. Hence the undirected max-flow algorithm of Karger [16] may seem inapplicable here. Fortunately, Karger's algorithm also works if a small number of edges is directed (which is the case for the residual network since g_1 is acyclic and hence uses few edges, see [16]).

Stage 2: For reasons that will soon become clear, we need $\overline{\mathcal{P}}$ to saturate all edges of $\overline{G}$. Let E_0 be the set of edges in $\overline{G}$ that are not used by paths in $\overline{\mathcal{P}}$.

Lemma 1. $\deg_{\overline{G}_0}(v)$ *is even for all* $v \in V$.

Proof. This is clear for $v \in V - T$ since $\deg_{\overline{G}}(v)$ is even, paths in $\overline{\mathcal{P}}$ are edge-disjoint, and every path in $\overline{\mathcal{P}}$ uses an even number of edges incident to v. Consider



(a) Graph G (b) Graph $\widehat{G}$

Fig. 1. Graph G with terminals $\{t_1, t_2, t_3\}$ and the corresponding graph $\widehat{G}$ (directions of auxiliarly arcs are not shown)

$v \in T$. Since $\overline{\mathcal{P}}$ is maximum, by Theorem 4 there exist t_i-cuts X_i such that $\overline{\mathcal{P}}$ saturates $\delta_{\overline{G}}(X_i)$ $(i = 1, 2, 3)$. Hence in $\overline{G}_0[X_i]$ at most one node, namely t_i, can have odd degree. In every undirected graph the number of nodes of odd degree is even, therefore $\deg_{\overline{G}_0}(v)$ is also even, as claimed. □

By Lemma 1 the algorithm can decompose E_0 into a collection of undirected circuits and attach these circuits to arbitrary paths in $\overline{\mathcal{P}}$. This takes $O(V + E)$ time and ensures that $\overline{\mathcal{P}}$ covers all edges of $\overline{G}$, as desired.

Stage 3: Construct an auxiliary skew-symmetric graph as follows. First take a disjoint symmetric copy $V' := \{v' \mid v \in V\}$ of V. For each arc $(u, v) \in E$ add two symmetric *regular* arcs (u, v) and (v', u'). Adjust the endpoints of regular arcs to ensure that each regular arc incident to t $(t \in T)$ leaves t and, symmetrically, each regular arc incident to t' enters t'. To this aim, replace every arc (x, t), $x \in V \cup V'$, by (x, t') and, symmetrically, replace every arc (t', x), $x \in V \cup V'$, by (t, x). Finally, for each $v \in V - T$ add four *auxiliary* arcs: two (symmetric) arcs (v, v') and two (also symmetric) arcs (v', v). Denote the resulting skew-symmetric graph by $\widehat{G}$. An example is depicted on Fig. 1.

Remark 3. The need for adding *two* auxiliary arcs (v, v') instead of just one is dictated by the definition of skew-symmetric graphs and is thus purely technical.

Stage 4: Multiflow $\overline{\mathcal{P}}$ in $\overline{G}$ gives rise to a certain weighted collection $\widehat{\mathcal{P}}$ of directed paths in $\widehat{G}$. Consider a path $\overline{P} \in \overline{\mathcal{P}}$ from, say, t_i to t_j $(i \neq j)$:

$$\overline{P} = (t_i = v_0, e_1, v_1, \ldots, e_l, v_l = t_j),$$

where $v_i \in V(G)$ $(i = 0, \ldots, l)$, $e_i \in E(\overline{G})$, e_i connects nodes v_{i-1} and v_i $(i = 1, \ldots, l)$.

Since edge capacities are 1 and $\overline{\mathcal{P}}$ is integer, $\overline{P}$ has weight 1. We transform $\overline{P}$ into a directed path $\widehat{P}$ (also of weight 1) in $\widehat{G}$ by taking appropriate regular arcs (corresponding to edges e_i) and inserting auxiliary arcs where needed. More formally, for each edge $e_i = \{v_{i-1}, v_i\}$, $\widehat{G}$ contains a unique regular arc $a_i =$

(x, y), where $x = v_{i-1}$ or $x = v'_{i-1}$ and $y = v_i$ or $y = v'_i$. Consider the sequence $(a_1, \dots, a_l)$ and turn it into a directed path $\widehat{P}$ by inserting auxiliary arcs (x, x'), $x \in V \cup V'$, between a_{i-1} and a_i if a_{i-1} ends at x and a_i starts at x'. Then $\widehat{\mathcal{P}}$ consists of paths $\widehat{P}$ and their symmetric mates $\widehat{P}'$.

Note the following:

(5) (i) $\widehat{\mathcal{P}}$ is symmetric, i.e. $\widehat{P} \in \widehat{\mathcal{P}}$ implies $\widehat{P}' \in \widehat{\mathcal{P}}$;
(ii) Each regular arc belongs to at most one path in $\widehat{\mathcal{P}}$;
(iii) Each path $P \in \widehat{\mathcal{P}}$ connects a node t_i with a node t'_j, where $i \neq j$.

A collection $\widehat{\mathcal{P}}$ of directed paths in $\widehat{G}$ obeying (5) will be referred to as a *integer skew-symmetric multiflow*. Suppose the following property holds for $\widehat{\mathcal{P}}$:

(6) Paths in $\widehat{\mathcal{P}}$ do not contain auxiliary arcs.

Then such $\widehat{\mathcal{P}}$ induces an integer multiflow $\mathcal{P}$ in G obeying $\mathrm{val}(\mathcal{P}) = \frac{1}{2}\mathrm{val}(\widehat{\mathcal{P}})$. Indeed, consider pairs of symmetric paths $\{\widehat{P}, \widehat{P}'\}$ forming $\widehat{\mathcal{P}}$ and let us show that each such pair corresponds to a T-path in G and these T-paths are arc-disjoint. Let us say that arcs e', e'' form a *transit pair* if e', e'' have a common endpoint v, one of e', e'' enters x, and the other leaves v. Consider $\widehat{P}$ as a sequence of arcs $\widehat{\tau} = (\widehat{e}_1, \dots, \widehat{e}_l)$. For each $i = 1, \dots, l-1$, arcs $\widehat{e}_i, \widehat{e}_{i+1}$ in $\widehat{G}$ form a transit pair and thus their pre-images e_i, e_{i+1} in G also form a transit pair (as it follows from the construction of $\widehat{G}$). Therefore either the sequence of pre-images $\tau = (e_1, \dots, e_l)$ or the reverse one $\tau^{-1} = (e_l, \dots, e_1)$ gives rise to a directed T-path in G. These paths are arc-disjoint by (5)(ii).

Stage 5: At this final stage we rearrange paths in $\widehat{\mathcal{P}}$ while maintaining (5) and preserving $\mathrm{val}(\widehat{\mathcal{P}})$ to get rid of auxiliary arcs. A similar subtask was earlier addressed in [1]. The latter algorithm scans nodes $v \in V - T$ one by one and removes auxiliary arcs between v and v'. Handling each node involves computing certain flow decompositions and takes $O(V+E)$ time (assuming unit capacities), which gives $O(VE)$ in total. We choose a different way of dealing with auxiliary arcs. Instead of processing inner nodes one at a time we apply certain global transformations aimed to decrease the total flow on auxiliary arcs.

For $1 \leq i, j \leq 3$, $i \neq j$, combine all t_i–t'_j paths in $\widehat{\mathcal{P}}$ and form an integer t_i–t'_j flow f_{ij} (see Fig. 2). For a function h on $E(\widehat{G})$, define $h'(e) := h(e')$. The symmetry of $\widehat{\mathcal{P}}$ implies $f'_{ij} = f_{ji}$ for all $1 \leq i, j \leq 3$, $i \neq j$. This property will be maintained throughout the iterations.

For each $v \in V - T$, define

$$\alpha(v) := f_{1,2}[v, v'], \quad \beta(v) := f_{1,3}[v, v'], \quad \gamma(v) := f_{2,3}[v, v'],$$

where $f_{ij}[v, v']$ is the *flow between* v *and* v', i.e. the sum of f_{ij}-values on auxiliary arcs (v, v') minus the sum of f_{ij}-values on auxiliary arcs (v', v).

Lemma 2. *$\alpha(v) + \beta(v) + \gamma(v) = 0$ for all $v \in V - T$.*

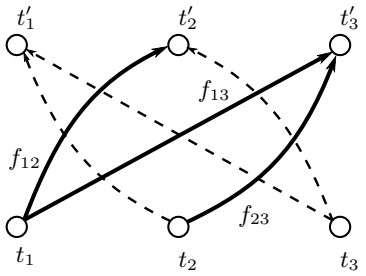


Fig. 2. Network $\widehat{G}$ and flows f_{12}, f_{13}, and f_{23}. Symmetric flows f_{21}, f_{31}, and f_{32} are shown by dashed lines.

Proof. Consider a node $v \in V - T$. Since G is inner balanced, $\deg^{\text{in}}_G(v) = \deg^{\text{out}}_G(v)$. From the construction of $\widehat{G}$ it follows that v has equal numbers of incoming and outcoming regular arcs. Define $g := \sum (f_{ij} : 1 \le i, j \le 3, i \ne j)$. Recall that $\overline{\mathcal{P}}$ saturates all edges of $\overline{G}$. Hence g saturates all regular arcs of $\widehat{G}$ and thus the total flow on incoming regular arcs equals the total flow on outcoming regular arcs. Therefore $g[v, v'] = 0$ and thus $\alpha(v) + \beta(v) + \gamma(v) = 0$. □

Define the *discrepancy* at v by $\Delta(v) := |\alpha(v)| + |\beta(v)| + |\gamma(v)|$ and the *total discrepancy* by $\Delta := \sum (\Delta(v) : v \in V - T)$. Note that 2Δ is exactly the sum of flows f_{ij}, $1 \le i, j \le 3$, $i \ne j$, on all auxiliary arcs (factor 2 comes from the symmetry) and thus (6) is equivalent to $\Delta = 0$.

The algorithm executes a series of scaling phases. Each phase decreases Δ by at least a factor of $\frac{11}{12}$. Since Δ is integer and initially $\Delta \le E/2 = O(V^2)$ (recall that G has unit capacities and each unit of discrepancy corresponds to an inner node of a T-path in $\overline{\mathcal{P}}$) it follows that $O(\log V)$ scaling phases are sufficient to achieve $\Delta = 0$. Flows f_{ij} are finally decomposed into an integer skew-symmetric multiflow $\widehat{\mathcal{P}}$ obeying (6).

A phase works as follows. Call a node $v \in V - T$ *active* if $|\alpha(v)| \ge |\beta(v)| \ge |\gamma(v)|$. By permuting terminals t_1, t_2, and t_3 (and thus values α, β, and γ) one can assume w.l.o.g. that

$$\sum (\Delta(v) : v \text{ is active}) \ge \frac{1}{6}\Delta. \tag{7}$$

To decrease Δ, define $h := f_{12} + f_{13}$ and cancel flows on oppositely directed auxiliary arcs. Note that h is an integer t_1–$\{t'_2, t'_3\}$ flow in $\widehat{G}$ and is thus decomposable into a sum of a t_1–t'_2 flow h_{12} and a t_1–t'_3 flow h_{13}. Computing h and decomposing it into h_{12} and h_{13} takes $O(V + E)$ time. The algorithm resets $(f_{12}, f_{21}, f_{13}, f_{31}) := (h_{12}, h'_{12}, h_{13}, h'_{13})$ and then proceeds to the next phase.

To estimate the decrease of Δ on each phase consider an arbitrary node $v \in V - T$ and let $\alpha'(v)$, $\beta'(v)$, and $\gamma'(v)$ $(= \gamma(v))$ be the corresponding values for the updated triple flows f_{ij}. Clearly $|\alpha'(v)| + |\beta'(v)| + |\gamma'(v)| \le |\alpha(v)| + |\beta(v)| + |\gamma(v)|$. The update also maintains the property given in Lemma 2.

Now suppose that v is active. Note that $\alpha(v) + \beta(v) = \alpha'(v) + \beta'(v)$. Also $\alpha(v)$ and $\beta(v)$ are of different signs (since $\alpha(v)$, $\beta(v)$, and $\gamma(v)$ add up to zero, and $\alpha(v)$ has the largest magnitude) while $\alpha'(v)$ and $\beta'(v)$ are of the same sign (since h_{12} and h_{13} come from a decomposition of h and thus use auxiliary arcs of the same direction). Therefore

$$\begin{aligned}
&\big(|\alpha(v)| + |\beta(v)| + |\gamma(v)|\big) - \big(|\alpha'(v)| + |\beta'(v)| + |\gamma'(v)|\big) = \\
&|\alpha(v) - \beta(v)| + |\alpha'(v) + \beta'(v)| = \\
&|\alpha(v) - \beta(v)| + |\alpha(v) + \beta(v)| = \\
&2\,|\beta(v)|\,.
\end{aligned}$$

Since $|\beta(v)| \geq \frac{1}{2}\,|\alpha(v)| \geq \frac{1}{4}\Delta(v)$ (which follows from $\alpha(v) + \beta(v) + \gamma(v) = 0$ and $|\alpha(v)| \geq |\beta(v)| \geq |\gamma(v)|$) the total discrepancy decreases by at least

$$\sum \left(\frac{1}{2}\Delta(v) : v \textit{ is active} \right) \geq \frac{1}{12}\Delta.$$

This concludes the proof of Theorem 1. □

3 General Capacities

The above approach also extends to the case when arc capacities are integers in range $[0, C]$ and leads to a weakly-polynomial $o(VE)$-time algorithm, as claimed in Theorem 2. The detailed proof is rather technical so we shall only give a brief sketch here.

During the course of the algorithm we maintain each multiflow $\mathcal{P}$ as a collection $\{(A_1, B_1, f_1), \ldots, (A_q, B_q, f_q)\}$, where for $i = 1, \ldots, q$, $A_i \cap B_i = \emptyset$, $A_i, B_i, \subset T$, and f_i is an acyclic integer A_i–B_i flow. Flows f_i are kept in a compact form, i.e. as lists $\{(e, f_i(e)) \mid f_i(e) \neq 0\}$. As in [13], such a representation of $\mathcal{P}$ additionally obeys $q = O(\log T)$ and occupies $O(E \log T)$ space. To merge a pair of such representations on each recursion level we solve $O(\log T)$ flow decomposition problems (each taking linear time due to acyclicity), extract flows corresponding to $\mathcal{P}' - \mathcal{P}'_0$ and $\mathcal{P}'' - \mathcal{P}''_0$, merge these flows, and finally decycle the result (which takes $O(m \log n)$ time [26]). Totally all aggregations take $O(E \log V \log T)$ time. Separations require, as earlier, $O(\phi(V, E) \cdot \log T)$ time.

It remains to deal with *leaf* subproblems (those corresponding to $|T| \leq 3$). Fix a leaf subproblem with n nodes and m arcs. Computing a maximum multiflow $\overline{\mathcal{P}}$ in the underlying undirected network $\overline{N}$ takes $O(\phi(n, m))$ time; the resulting $\overline{\mathcal{P}}$ is then turned into a collection of three integer flows $\overline{f}_{ij}$, $1 \leq i < j \leq 3$, in $O(m \log(n^2/m))$ time with the help of the fast flow decomposition routine [1]. Residual edge capacities (those corresponding to slacks in (1)) are Eulerian (cf. Lemma 1). Similar to the unit-capacitated case one can compute (in linear time) a weighted collection of additional cycles (not necessarily simple) that exhaust the residual capacities and attach these cycles to an arbitrary component of $\overline{\mathcal{P}}$. Each $\overline{f}_{ij}$ is turned into a pair of symmetric integer t_i–t'_j and t_j–t'_i flows f_{ij} and

f_{ji} in $\widehat{G}$. The latter does not require explicit path-packing representations of $\overline{f}_{ij}$ and can be done in linear time. The algorithm from Subsection 2.3 is applied to get rid of flows on auxiliary arcs; this requires $O(\log(n \operatorname{val}(\overline{\mathcal{P}}))) = O(\log(nC))$ scaling phases. The resulting flows are finally decycled in $O(m \log n)$ time [26].

From the above estimates it follows that for a network $N = (G = (V, E), T, c)$ the new algorithm takes $O(\phi(V, E) \cdot \log T + E \log V \log T + E \log(V^2/E) \log(VC))$ time. This concludes the proof of Theorem 2. □

Acknowledgements. The authors are thankful to anonymous referees for helpful comments and suggestions.

References

1. Babenko, M.A., Karzanov, A.V.: Free multiflows in bidirected and skew-symmetric graphs. Discrete Applied Mathematics 155, 1715–1730 (2007)
2. Cherkassky, B.V.: A solution of a problem on multicommodity flows in a network. Ekonomika i Matematicheskie Metody 13(1), 143–151 (1977) (in Russian)
3. Dinic, E.A.: Algorithm for solution of a problem of maximum flow in networks with power estimation. Dokl. Akad. Nauk. SSSR 194, 754–757 (in Russian) (translated in Soviet Math. Dokl. 111, 277–279)
4. Edmonds, J., Johnson, E.L.: Matching: a well-solved class of integer linear programs. In: Guy, R., Hanani, H., Sauer, N., Schönhein, J. (eds.) Combinatorial Structures and Their Applications, pp. 89–92. Gordon and Breach, NY (1970)
5. Even, S., Tarjan, R.E.: Network Flow and Testing Graph Connectivity. SIAM Journal on Computing 4, 507–518 (1975)
6. Ford, L.R., Fulkerson, D.R.: Flows in Networks. Princeton Univ. Press, Princeton (1962)
7. Fortune, S., Hopcroft, J., Wyllie, J.: The directed subgraph homeomorphism problem. Theoretical Computer Sci. 10, 111–121 (1980)
8. Frank, A.: On connectivity properties of Eulerian digraphs. Ann. Discrete Math. 41, 179–194 (1989)
9. Goldberg, A.V., Karzanov, A.V.: Path problems in skew-symmetric graphs. Combinatorica 16, 129–174 (1996)
10. Goldberg, A.V., Karzanov, A.V.: Maximum skew-symmetric flows and matchings. Mathematical Programming 100(3), 537–568 (2004)
11. Goldberg, A.V., Rao, S.: Beyond the flow decomposition barrier. In: Proc. 38th IEEE Symposium Foundations of Computer Science (1997); adn Journal of the ACM 45, 783–797 (1998)
12. Goldberg, A.V., Tarjan, R.E.: A new approach to the maximum flow problem. J. ACM 35, 921–940 (1988)
13. Ibaraki, T., Karzanov, A.V., Nagamochi, H.: A fast algorithm for finding a maximum free multiflow in an inner Eulerian network and some generalizations. Combinatorica 18(1), 61–83 (1998)
14. Karger, D.R.: Random sampling in cut, flow, and network design problems. Mathematics of Operations Research (1998)
15. Karger, D.R.: Better random sampling algorithms for flows in undirected graphs. In: Proc. 9th Annual ACM+SIAM Symposium on Discrete Algorithms, pp. 490–499 (1998)

16. Karger, D.R., Levine, M.S.: Finding Maximum flows in undirected graphs seems easier than bipartite matching. In: Proc. 30th Annual ACM Symposium on Theory of Computing, pp. 69–78 (1997)
17. Karzanov, A.V.: O nakhozhdenii maksimalnogo potoka v setyakh spetsialnogo vida i nekotorykh prilozheniyakh. In: Matematicheskie Voprosy Upravleniya Proizvodstvom, vol. 5. University Press (1973) (in Russian)
18. Karzanov, A.V.: Combinatorial methods to solve cut-dependent problems on multiflows. In: Combinatorial Methods for Flow Problems, Inst. for System Studies, Moscow, vol. (3), pp. 6–69 (1979) (in Russian)
19. Karzanov, A.V.: Fast algorithms for solving two known problems on undirected multicommodity flows. In: Combinatorial Methods for Flow Problems, Inst. for System Studies, Moscow, vol. (3), pp. 96–103 (1979) (in Russian)
20. Kupershtokh, V.L.: A generalization of Ford-Fulkerson theorem to multiterminal networks. Kibernetika 7(3), 87–93 (1971) (in Russian) (Translated in Cybernetics 7(3) 494-502)
21. Lawler, E.L.: Combinatorial Optimization: Networks and Matroids. Holt, Reinhart, and Winston, NY (1976)
22. Lovász, L.: On some connectivity properties of Eulerian graphs. Acta Math. Akad. Sci. Hung. 28, 129–138 (1976)
23. Lovász, L.: Matroid matching and some applications. J. Combinatorial Theory, Ser. B 28, 208–236 (1980)
24. Mader, W.: Über die Maximalzahl kantendisjunkter A-Wege. Archiv der Mathematik (Basel) 30, 325–336 (1978)
25. Schrijver, A.: Combinatorial Optimization. Springer (2003)
26. Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. J. Comput. Syst. Sci. 26(3), 362–391 (1983)
27. Tutte, W.T.: Antisymmetrical digraphs. Canadian J. Math. 19, 1101–1117 (1967)

Towards Optimal and Expressive Kernelization for d-Hitting Set

René van Bevern*

Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany
rene.vanbevern@tu-berlin.de

Abstract. A sunflower in a hypergraph is a set of hyperedges pairwise intersecting in exactly the same vertex set. Sunflowers are a useful tool in polynomial-time data reduction for problems formalizable as d-Hitting Set, the problem of covering all hyperedges (of cardinality at most d) of a hypergraph by at most k vertices. Additionally, in fault diagnosis, sunflowers yield concise explanations for "highly defective structures". We provide a linear-time algorithm that, by finding sunflowers, transforms an instance of d-Hitting Set into an equivalent instance comprising at most $\mathrm{O}(k^d)$ hyperedges and vertices. In terms of parameterized complexity, we show a problem kernel with asymptotically optimal size (unless coNP $\subseteq$ NP/poly). We show that the number of vertices can be reduced to $\mathrm{O}(k^{d-1})$ with additional processing in $\mathrm{O}(k^{1.5d})$ time—nontrivially combining the sunflower technique with problem kernels due to Abu-Khzam and Moser.

1 Introduction

Many practically relevant problems like the examples given below boil down to solving the following NP-hard problem:

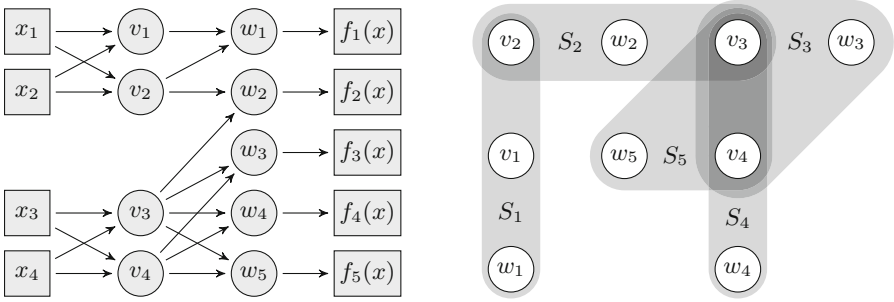
d-Hitting Set
Input: A hypergraph $H = (V, E)$ with hyperedges of cardinality at most d and a natural number k.
Question: Is there a *hitting set* $S \subseteq V$ with $|S| \leq k$ and $\forall e \in E\colon e \cap S \neq \emptyset$?

Examples for NP-hard problems encodeable into d-Hitting Set arise in the following fields.

1. Fault diagnosis: The task is to detect faulty components of a malfunctioning system. To this end, those sets of components are mapped to hyperedges of a hypergraph that are assumed to contain at least one broken component [1, 14, 21]. By the principle of Occam's Razor, a small hitting set is then a likely explanation of the malfunction.
2. Data clustering: all optimization problems in the complexity classes MIN $\mathrm{F}^{+}\Pi_1$ and MAX NP, including (s-Plex) Cluster Vertex Deletion [5, 13] and all problems of establishing by means of vertex deletion a

* Supported by the DFG, project DAPA, NI 369/12. The main part of the work was done under DFG project AREG, NI 369/9.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 121–132, 2012.



(a) A boolean circuit with circle nodes representing gates and square nodes representing input and output nodes.

(b) Sets containing at least one faulty gate, found by the analysis of the circuit.

Fig. 1. Illustrations for Example 1

graph property characterized by forbidden induced subgraphs with at most d vertices, can be formalized as d-Hitting Set [15].

All problems have in common that a large number of "conflicts" (the possibly $O(|V|^d)$ hyperedges in a d-Hitting Set instance) is caused by a small number of components (the hitting set S), whose removal or repair could fix a broken system or establish a useful property. However, often, not only a solution to a problem is sought, but also a concise explanation of why a solution should be of the given form. In this work, we contribute to this by combining concise explanations with data reduction, wherein our data reduction preserves the possibility of finding optimal solutions and gives a performance guarantee. This kind of data reduction or, more specifically, problem kernels, are a powerful tool in attacking NP-hard problems like d-Hitting Set [6, 11]. Our main ingredient and contribution are efficient algorithms to find *sunflowers*, a structure first observed by Erdős and Rado [9]:

Definition 1. *In a hypergraph* (V, E)*, a* sunflower *is a set of* petals $P \subseteq E$ *such that every pair of sets in* P *intersects in exactly the same set* $C \subseteq V$*, called the* core *(possibly,* $C = \emptyset$*). The* size *of the sunflower is* $|P|$*.*

A sunflower with $k+1$ petals in a d-Hitting Set instance yields a concise explanation of why one of the elements in its core should be removed or repaired: For every sunflower P with at least $k+1$ petals, every hitting set of size at most k contains one of P's core-elements, since it cannot contain an element of each of the $k+1$ petals. Analyzing the petals of this sunflower could guide the 'causal analysis' of a problem. We illustrate this using an example.

Example 1. Figure 1(a) shows a boolean circuit. It gets as input a 4-bit string $x = x_1 \dots x_4$ and outputs a 5-bit string $f(x) = f_1(x) \dots f_5(x)$. The nodes drawn as circles represent boolean gates, which output some bit depending on their two input bits. They might, for example, represent the logical operations "$\wedge$" or "$\vee$".

Assume that all output bits of $f(x)$ are observed to be the opposite of what would have been expected by the designer of the circuit. We want to identify broken gates. For each wrong output bit $f_i(x)$, we obtain a set S_i of gates for which we know that at least one is broken, because $f_i(x)$ is wrong. That is, S_i contains precisely those gates that have a directed path to $f_i(x)$ in the graph shown in Figure 1(a). We obtain the sets illustrated in Figure 1(b):

$$S_1 = \{v_1, v_2, w_1\}, \quad S_2 = \{v_2, v_3, w_2\}, \quad S_3 = \{v_3, v_4, w_3\},$$
$$S_4 = \{v_3, v_4, w_4\}, \quad S_5 = \{v_3, v_4, w_5\}.$$

The sets S_1 and S_5 are disjoint. Therefore, the wrong output is not explainable by only one broken gate. Therefore, we now assume that there are *two* broken gates and search for a hitting set of size $k = 2$ in the hypergraph with the vertices $v_1, \ldots, v_4, w_1, \ldots, w_5$ and hyperedges $S_1, \ldots, S_5$. The set $\{S_3, S_4, S_5\}$ is a sunflower of size $k + 1 = 3$ with core $\{v_3, v_4\}$. Therefore, the functionality of gate v_3 and v_4 must be checked. If, in contrast to our expectation, both gates v_3 and v_4 turn out to be working correctly, the usefulness of the sunflower becomes even more apparent: it is immediately clear not only that at least three gates are broken, but it is also clear which gates have to be checked for malfunctions next: w_3, w_4, and w_5. □

In addition to fault diagnosis, sunflowers also yield a good tool for data reduction preserving optimal solutions, so that we can remove hyperedges and vertices from the input hypergraph, until it is small enough to be analyzed as a whole. This can be seen as follows. For every sunflower P with at least $k + 1$ petals, every hitting set of size at most k contains one of P's core-elements. Therefore, we can repeatedly discard a petal of a sunflower of size $k + 2$ from the hypergraph, yielding a decision-equivalent d-Hitting Set instance whose largest sunflower has $k+1$ petals [15]. This, by the sunflower lemma of Erdős and Rado [9], implies that the resulting hypergraph has $O(k^d)$ hyperedges [10, Theorem 9.8], therefore showing that this form of data reduction yields a problem kernel [6, 11].

Previous Work. Downey and Fellows [8] showed that Hitting Set is W[2]-complete with respect to the parameter k when the cardinality of the hyperedges is unbounded. Hence, unless FPT = W[2], it has no problem kernel. Various problem kernels for d-Hitting Set have been developed [2, 10, 15, 16, 18, 19]. However, the problem kernels aiming for efficiency faced some problems: Niedermeier and Rossmanith [18] showed a problem kernel for 3-Hitting Set of size $O(k^3)$. They implicitly claimed that a polynomial-size problem kernel for d-Hitting Set is computable in linear time, not giving a proof for the running time. Nishimura et al. [19] claimed that a problem kernel with $O(k^{d-1})$ vertices is computable in $O(k(|V| + |E|) + k^d)$ time, which, however, does not always yield correct problem kernels [2]. The problem kernels of Flum and Grohe [10] and Kratsch [15] exploit the sunflower lemma by Erdős and Rado [9] and therefore yield concise explanations of why certain vertices should be part of optimal solutions. However, their running times are only analyzed to be polynomial in

the input size. Abu-Khzam [2] showed a problem kernel with $O(k^{d-1})$ vertices for d-Hitting Set, thus proving the previously claimed result of Nishimura et al. [19] on the number of vertices in the problem kernel. The problem kernel of Abu-Khzam [2] may still comprise $\Omega(k^{2d-2})$ hyperedges.[1] Dell and van Melkebeek [7] showed that the existence of a problem kernel with $O(k^{d-\varepsilon})$ hyperedges for any ε would imply coNP $\subseteq$ NP/poly and a collapse of the polynomial-time hierarchy to the third level. Therefore, a problem kernel with $O(k^{d-\varepsilon})$ hyperedges is presumed not to exist.

Our Results. We show that a problem kernel for d-Hitting Set with $O(k^d)$ hyperedges and vertices is computable in linear time. Thereby, we prove the previously claimed result by Niedermeier and Rossmanith [18] and complement recent results in improving the efficiency of kernelization algorithms [4, 12, 20]. In contrast to many other problem kernels [2, 16, 18, 19], our algorithm outputs sunflowers to guide fault diagnosis. Additionally, using ideas from Abu-Khzam [2] and Moser [16], we show that the number of vertices can be further reduced to $O(k^{d-1})$ with an additional amount of $O(k^{1.5d})$ time. Summarizing, by merging these techniques, we can compute in $O(|V| + |E| + k^{1.5d})$ time a problem kernel comprising $O(k^d)$ hyperedges and $O(k^{d-1})$ vertices.

Preliminaries. A *hypergraph* $H = (V, E)$ consists of a set of *vertices* V and a set of *(hyper)edges* E, where each hyperedge in E is a subset of V. In a *d-uniform hypergraph* every edge has cardinality exactly d. A 2-uniform hypergraph is a *graph*. A hypergraph $G = (V', E')$ is a *subgraph* of its *supergraph* H if $V' \subseteq V$ and $E' \subseteq E$. A set $S \subseteq V$ intersecting every set in E is a *hitting set*. A parameterized problem is a subset $L \subseteq \Sigma^* \times \mathbb{N}$ [8, 10, 17]. A problem kernel for a parameterized problem L is a polynomial-time algorithm that, given an instance (I, k), computes an instance (I', k') such that $|I'| + k' \leq f(k)$ and $(I', k') \in L \iff (I, k) \in L$. Herein, the function f is called the *size* of the problem kernel and depends only on k.

2 A Linear-Time Problem Kernel for d-Hitting Set

This section shows a linear-time problem kernel for d-Hitting Set comprising $O(k^d)$ edges. That is, we show that a hypergraph H can be transformed in linear time to a hypergraph G such that G has $O(k^d)$ edges and allows for a hitting set of size k if and only H does. In Section 3, we show how to shrink the number of vertices to $O(k^{d-1})$.

[1] Although not directly given in the work of Abu-Khzam [2] itself, this can be seen as follows: the kernel comprises vertices of each hyperedge in a set W of pairwise "weakly related" hyperedges and an independent set I. In the worst case, $|W| = k^{d-1}$ and $|I| = dk^{d-1}$ and each hyperedge in W has d subsets of size $d-1$. Then, each subset can constitute a hyperedge with each vertex in I and the kernel has $\Omega(k^{2d-2})$ hyperedges.

Algorithm 1. Linear-Time Kernelization for d-Hitting Set

Input: Hypergraph $H = (V, E)$, natural number k.
Output: Hypergraph $G = (V', E')$ with $|E'| \in O(k^d)$.

```
1  E' ← ∅;
2  foreach e ∈ E do                                   // Initialization for each edge
3      foreach C ⊆ e do                               // Initialization for all possible cores of sunflowers
4          petals[C] ← 0;                             // No petals found for sunflower with core C yet
5          foreach v ∈ e do                           // No vertex in a petal of a sunflower with core C yet
6              used[C][v] ← false
7  foreach e ∈ E do delete all e' ⊋ e from E;
8  foreach e ∈ E do
9      if ∀C ⊆ e: petals[C] ≤ k then
10         E' ← E' ∪ {e};
11         foreach C ⊆ e do                           // Consider all possible cores for the petal e
12             if ∀v ∈ e \ C: used[C][v] = false then
13                 petals[C] ← petals[C] + 1;
14                 foreach v ∈ e \ C do used[C][v] ← true;
15 V' := ⋃_{e∈E'} e;
16 return (V', E');
```

Theorem 1. *d-Hitting Set admits a linear-time computable problem kernel comprising $\mathrm{O}(k^d)$ hyperedges and vertices.*

Until now, problem kernels based on the sunflower lemma by Erdős and Rado [9] proceed in the following fashion [10, 15]: repeatedly (i) find a sunflower of size $k+1$ in the input graph and (ii) delete redundant petals until no more sunflowers of size $k+1$ exist. This approach has the drawback of finding only one sunflower at a time and restarting the process from the beginning.

In contrast, to prove Theorem 1, we construct a subgraph $G = (V', E')$ of a given hypergraph $H = (V, E)$ not by edge deletion; instead, we follow a bottom-up approach that allows us to "grow" many sunflowers in G simultaneously, stopping "growing sunflowers" if they become too large. Algorithm 1 repeatedly (after some initialization work in lines 1–7) in line 10 copies a hyperedge e from H to the initially empty G unless we find in line 9 that e contains the core C of a sunflower of size $k+1$ in G. To check this, the number of petals found for a core C is maintained in a data structure petals$[C]$. If we find that an edge e is suitable as petal of a sunflower with core C in line 12, then we mark the vertices in $e \setminus C$ as "used" for the core C in line 14. This information is maintained by setting "used$[C][v] \leftarrow$ true". In this way, vertices in $e \setminus C$ are not considered again for finding petals for the core C in line 12, therefore ensuring that additionally found petals intersect e only in C.

We now prove the correctness and running time of Algorithm 1, which will, together with the result that the output graph contains no large sunflowers, provide a proof of Theorem 1. Note that, by storing in petals$[C]$ a list of found

petals, they can serve as concise explanations of why a small hitting set contains vertices of C.

Lemma 1. *The hypergraph G returned by Algorithm 1 on input H admits a hitting set of size k if and only if H does.*

Proof. We first show that, if H admits a hitting set of size k, then so does G. For every hitting set S for $H = (V, E)$, the set $S' := S \cap V'$ is a hitting set for $G = (V', E')$ with $|S'| \leq |S|$: the set S contains an element of every edge in E and, since $E' \subseteq E$ and $V' = \bigcup_{e \in E'} e$, the set S' contains an element of every edge in E'. It remains to show that if G admits a hitting set of size k, then so does H. Assume that S is a hitting set of size k for G. Obviously, all edges that H and G have in common are hit in H by S. It remains to show that every edge e in H that is not in G is also hit.

First, consider the case where e was not deleted in line 7. Then, adding e to G in line 10 of Algorithm 1 has been skipped, because the condition in line 9 is false. That is, $\text{petals}[C] \geq k+1$ for some $C \subseteq e$. Consequently, for this particular C, a sunflower P with $k+1$ petals and core C exists in G, since we only increment $\text{petals}[C]$ in line 13 if we find e to be suitable as additional petal for a sunflower with core C in line 12. Note that $C \neq \emptyset$, because otherwise $k+1$ pairwise disjoint edges would exist in G, contradicting our assumption that S is a hitting set of size k for G. Since $|S| \leq k$, we have $S \cap C \neq \emptyset$ as discussed in the introduction. Therefore, since $C \subseteq e$ and $C \subseteq V$, the edge e is hit by S also in H.

Second, in the case where e was removed in line 7, e is also hit by S, because either G contains a sub-edge $e' \subsetneq e$ or e' is hit since its addition to G was skipped in application of the previous case. We conclude that S is a hitting set of size k also for H. □

Lemma 2. *Given a hypergraph $H = (V, E)$ and a natural number k, Algorithm 1 can be implemented to run in $\mathrm{O}(d|V| + 2^d d \cdot |E|)$ time.*

Proof. We first describe the data structure that is used to maintain $\text{petals}[C]$ and $\text{used}[C][v]$, then its initialization in lines 1–6, then the implementation of lines 8–15, and finally that of line 7.

We assume that every vertex is represented as an integer in $\{1, \ldots, |V|\}$ and that every edge is represented as a sorted array. We can initially sort all edges of H in $\mathrm{O}(|E| d \log d)$ total time. Then, the set subtraction operation needed in line 12 can be executed in $\mathrm{O}(d)$ time such that the resulting set is again sorted. Moreover, we can generate all subsets of a sorted set such that the resulting subsets are sorted. Hence, we can assume to always deal with sorted edges and thus obtain a canonical representation of an edge as a length-d character string over the alphabet V. This enables us to maintain $\text{petals}[C]$ and $\text{used}[C][v]$ in a *trie*: a trie is a tree-like data structure, in which, when each of its inner nodes is implemented as an array, a value associated with a character string X can be looked up and stored in $\mathrm{O}(|X|)$ time [3, Section 5.3]. Hence, we can look up and store values associated with a set C in $\mathrm{O}(d)$ time. We use such a trie to associate with some sets $C \subseteq V$, $|C| \leq d$, an integer $\text{petals}[C]$, and a pointer to a vector $\text{used}[C][]$ of length $|V|$.

For initial creation of the trie in lines 1–6, we do not initialize every cell of the array that implements an inner node of the trie, as this would take $O(|V|)$ time for each non-empty node. However, we have to initialize all cells that will be accessed: otherwise, it will be unknown if a cell contains a pointer to a another node or random data. We achieve this as follows: in lines 1–6, we obtain a length-$2^d|E|$ list L of all possible sets $C \subseteq e$ for all $e \in E$. We will only associate values with sets in L, and therefore initialize the inner nodes of the trie to only hold values associated with sets in L. This works in $O(d|V| + 2^d d \cdot |E|)$ time, since the representation of sets in L as length-d strings over the alphabet V enables us to sort L in $O(d(|V| + |L|)) = O(d|V| + 2^d d \cdot |E|)$ time using Radix Sort [3, Section 8.3]. We build the trie by iterating over L once: in each iteration, we check in $O(d)$ time in which positions the character string for a set C differs from the character string of its predecessor set in L. This tells us which array entries of the inner nodes of the trie have to be newly initialized, and which nodes in the trie on the path to the leaf corresponding to C have been previously initialized and may not be overwritten. Hence, we can implement lines 1–6 to run in $O(d|V| + 2^d d|E|)$ time, observing that line 5 can be implemented to run in $O(d)$-time, as only one look-up to used$[C][]$ is needed to obtain an array, in which then $O(d)$ necessary values are initialized.

The for-loop in line 8 iterates $|E|$ times. Its body works in $O(2^d d)$ time: obviously, this time bound holds for lines 9 and 10; it remains to show that the body of the for-loop in line 11 works in $O(d)$ time. This is easy to see if one considers that, in lines 12 and 14, one only has to do one look-up to used$[C][]$ to find an array that holds the values for the at most d vertices $v' \in e$. Also line 15 works in linear time by first initializing all entries of an array vertices$[]$ of size $|V|$ to "false". Then, for each edge $e \in E'$ and each vertex $v \in e$, set "vertices$[v] \leftarrow$ true" in $O(d)$ time. Afterward, let V' be the set of vertices v for which vertices$[v] =$ true. This takes $O(|V| + d|E|)$ time.

It remains to discuss the running time of line 7. Similarly as in lines 8–14, we iterate over all edges $e \in E$, and for all proper subsets $e' \subset e$ add a pointer to the position of e in E to the list supersets$[e']$ (associated with e' using a trie). It then remains to remove the edges in supersets$[e']$ from E for each edge $e \in E$. □

We now show that there is an upper bound on the size of the sunflowers in the graph output by Algorithm 1. This enables us to upper-bound the size of the output graph similarly to how the sunflower lemma of Erdős and Rado [9] is used in the d-Hitting Set kernel of Flum and Grohe [10, Theorem 9.8].

Lemma 3. *Given a hypergraph $H = (V, E)$ and a natural number k, Algorithm 1 outputs a hypergraph G whose largest sunflower has $d(k + 1)$ petals.*

Proof. Let P be a sunflower with core C in G. If $C \in P$, then $|P| = 1$ because of line 7 of Algorithm 1. If $C \notin P$, the following two observations yield $|P| \leq d(k + 1)$:

(i) Every petal $e \in P$ present in G is copied from H in line 10 of Algorithm 1. Consequently, every petal $e \in P$ contains a vertex v satisfying used$[C][v] =$ true:

if this condition would be violated in line 12, then line 14 applies "used$[C][v] \leftarrow$ true" to all vertices $v \in e \setminus C$.

(ii) Whenever petals$[C]$ is incremented by one in line 13, then, in line 14, "used$[C][v] \leftarrow$ true" is applied to the at most d vertices $v \in e$. Thus, since always petals$[C] \leq k+1$, at most $d(k+1)$ vertices v satisfy used$[C][v] =$ true. Moreover, since, by line 14, no $v \in C$ satisfies used$[C][v] =$ true and the petals in P pairwise intersect only in C, it follows that at most $d(k+1)$ petals in P contain vertices satisfying used$[C][v] =$ true. □

The last ingredient in the proof of Theorem 1 is the sunflower lemma by Erdős and Rado [9]. In a similar way as Flum and Grohe [10, Lemma 9.7], we can show the following refined version, which we need for Section 3. Note that, for $b = 1$, this is exactly the sunflower lemma [10].

Lemma 4. *Let $H = (V, E)$ be an ℓ-uniform hypergraph, $b, c \in \mathbb{N}$, and $b \leq \ell$ such that every pair of edges in H intersects in at most $\ell - b$ vertices. If H contains more than $\ell! c^{\ell+1-b}$ edges, then H contains a sunflower with more than c petals.*

We finally have all ingredients to show that d-Hitting Set admits a linear-time computable $\mathrm{O}(k^d)$-size problem kernel, thus proving Theorem 1.

Proof (Theorem 1). Lemma 1 and Lemma 2 show that Algorithm 1 executes linear-time data reduction such that the input and output graph are equivalent with respect to d-Hitting Set. It remains to show that the graph G output by Algorithm 1 comprises at most $d! \cdot d^{d+1} \cdot (k+1)^d \in \mathrm{O}(k^d)$ edges. This then also implies that G has $\mathrm{O}(k^d)$ vertices, as the vertex set of G is constructed as the union of its edges in line 15 of Algorithm 1.

To bound the number of edges, consider for $1 \leq \ell \leq d$ the ℓ-uniform hypergraph $G_\ell = (V_\ell, E_\ell)$ comprising only the edges of size ℓ of G. If G had more than $d! \cdot d^{d+1} \cdot (k+1)^d$ edges, then, for some $\ell \leq d$, G_ℓ would have more than $d! \cdot d^d \cdot (k+1)^d$ edges. This, however, leads to a contradiction with Lemma 3: Lemma 4 with $b = 1$ and $c = d(k+1)$ states that if G_ℓ had more than $\ell! \cdot d^\ell \cdot (k+1)^\ell$ edges, then G_ℓ would contain a sunflower with more than $d(k+1)$ petals. This sunflower would also exist in the supergraph G of G_ℓ. □

3 Reducing the Number of Vertices to $\mathrm{O}(k^{d-1})$

This section combines the problem kernel in Section 2 with techniques from Abu-Khzam [2] and Moser [16, Section 7.3], yielding a problem kernel for d-Hitting Set comprising $\mathrm{O}(k^d)$ edges and $\mathrm{O}(k^{d-1})$ vertices in $\mathrm{O}(|V|+|E|+k^{1.5d})$ time. To this end, we first briefly sketch the running-time bottleneck of the kernelization idea of Abu-Khzam [2], which is also a bottleneck in the algorithm of Moser [16].

The problem kernels of Abu-Khzam [2] and Moser [16, Section 7.3]. Given a hypergraph $H = (V, E)$ and a natural number k, Abu-Khzam [2] first computes a maximal *weakly related* set W, where data reduction ensures $|W| \leq k^{d-1}$:

Definition 2 ([2]). *A set $W \subseteq E$ is* weakly related *if every pair of edges in W intersects in at most $d-2$ vertices.*

Whether a given edge e can be added to a set W of weakly related edges can be checked in $O(d|W|)$ time. After adding e, data reduction on W is executed in $O(2^d|W|\log|W|)$ time. Hence, since always $|W| \leq k^{d-1}$, Abu-Khzam [2] can compute W in $O(2^d k^{d-1} \log k \cdot |E|)$ time.

Since $|W| \leq k^{d-1}$, it remains to bound the size of the set I of vertices not contained in edges of W. The set I is an *independent set*, that is, I contains no pair of vertices occurring in the same edge [2]. A bipartite graph $B = (I \uplus S, E')$ is constructed, where $S := \{e \subseteq V \mid \exists v \in I \colon \exists w \in W \colon e \subseteq w, \{v\} \cup e \in E\}$ and $E' := \{\{v, e\} \mid v \in I, e \in S, \{v\} \cup e \in E\}$. Whereas Abu-Khzam [2] shrinks the size of I using so-called *crown reductions*, Moser [16, Lemma 7.16] shows that it is sufficient to compute a maximum matching in B and to remove unmatched vertices in I from G together with the edges containing them. The bound of the number of vertices in the problem kernel is thus $O(k^{d-1})$, since $|W| \leq k^{d-1}$, and therefore $|I| \leq |S| \leq dk^{d-1}$.

Our improvements. Given a hypergraph $H = (V, E)$ and a natural number k, we can first compute our problem kernel in $O(|V| + |E|)$ time, leaving $O(k^d)$ edges in H. Afterward applying the problem kernel of Abu-Khzam [2] would reduce the number of vertices to $O(k^{d-1})$. However, the computation of the maximal weakly related set on our reduced instance already takes $O(2^d k^{d-1} \log k \cdot |E|) = O(k^{2d-1} \log k)$ additional time, as discussed above. We improve the running time of this step in order to show:

Theorem 2. *d-Hitting Set admits a problem kernel comprising $O(k^d)$ hyperedges and $O(k^{d-1})$ vertices computable in $O(|V| + |E| + k^{1.5d})$ time.*

To prove Theorem 2, we compute a maximal weakly related set W in linear time and show that our problem kernel already ensures $|W| \in O(k^{d-1})$. Further data reduction on W is therefore unnecessary. This makes finding a maximum matching the new bottleneck of the kernelization described by Moser [16, Section 7.3].

Lemma 5. *Given a hypergraph $H = (V, E)$, a maximal weakly related set is computable in $O(d|V| + d^2 \cdot |E|)$ time.*

To prove Lemma 5, we employ Algorithm 2.

Proof. First, observe that the set W returned in line 12 of Algorithm 2 is indeed weakly related: let $w_1 \neq w_2 \in E$ intersect in more than $d-2$ vertices and assume that w_1 is added to W in line 8. Let $C := w_1 \cap w_2$. Obviously, $|C| = d-1$. Hence, when w_1 is added to W, then we apply "intersection$[C] \leftarrow$ true" in line 10. Therefore, when $e = w_2$ is considered in line 6, the condition in line 7 does not hold, which implies that w_2 is not added to W in line 8. In the same way it follows that every edge is added to W that does not intersect any edge of W in more than $d-2$ vertices. Therefore, W is maximal.

Algorithm 2. Linear-time computation of a maximal weakly related set

Input: Hypergraph $H = (V, E)$, natural number k.
Output: Maximal weakly related set W.

```
1  W ← ∅;
2  foreach e ∈ E do                                   // Initialization for each edge
3      foreach C ⊆ e, |C| = d − 1 do
4          intersection[C] ← false;                   // No edges in W contain C yet.
5          intersection[e \ C] ← false; // The vertex in e \ C is not in W yet. We use
                                        // this later to compute an independent set.
6  foreach e ∈ E do
7      if ∀C ⊆ e, |C| = d − 1: intersection[C] = false then
8          W ← W ∪ {e};
9          foreach C ⊆ e, |C| = d − 1 do
10             intersection[C] ← true;
11             intersection[e \ C] ← true;
12 return W;
```

We first sort all edges of H in $O(|E|d \log d)$ time. Using the trie data structure and initialization method as used in Lemma 2, we can do each look-up of a value intersection$[C]$ in $O(d)$ time if C is the result of a set subtraction operation of two sorted sets. We initialize the trie to associate values with at most $2d \cdot |E|$ sets. Hence, as discussed in Lemma 2, the initialization in lines 1–5 can be done in $O(d|V| + d^2 \cdot |E|)$ time. Finally, for every edge, the body of the for-loop in line 6 can be executed in $O(d^2)$ time doing $O(d)$-time look-ups for each of the $2d \cdot |E|$ sets. □

We can now prove Theorem 2 by showing how to compute a problem kernel with $O(k^{d-1})$ vertices in $O(|V| + |E| + k^{1.5d})$ time.

Proof (of Theorem 2). We may assume that the hypergraph $H = (V, E)$ in an instance of d-Hitting Set satisfies $|V| + |E| \in O(k^d)$ and contains sunflowers with at most $d(k + 1)$ petals, since otherwise using Algorithm 1, we can transform H accordingly in linear time, as stated by Lemma 3 and Theorem 1. To reduce the number of vertices in H to $O(k^{d-1})$, we follow the approach of Moser [16, Lemma 7.16] as discussed in the beginning of this section.

First, compute a maximal weakly related set W in H in $O(|V| + |E|) = O(k^d)$ time using Algorithm 2. We show that $|W| \in O(k^{d-1})$. Because every pair of edges in W intersects in at most $d-2$ vertices, by Lemma 3 and Lemma 4 for $b = 2$ and $c = d(k+1)$, we know that the hypergraph (V, W_ℓ) for $\ell \geq 2$, where W_ℓ is the set of cardinality-ℓ edges in W, has at most $O(k^{d-1})$ edges. Moreover, W_1 contains at most $O(k)$ edges, as they form a sunflower with empty core. Therefore, $|W| \in O(k^{d-1})$. Next, we construct a bipartite graph $B = (I \uplus S, E')$, where

(i) I is the set of vertices in V not contained in any edge in W, forming an independent set [2, 16],

(ii) $S := \{e \subseteq V \mid \exists v \in I \colon \exists w \in W \colon e \subseteq w, \{v\} \cup e \in E\}$, and
(iii) $E' := \{\{v, e\} \mid v \in I, e \in S, \{v\} \cup e \in E\}$.

This can be done in $O(|E|) = O(k^d)$ time as follows: for each $e \in E$ with $|e| = d$ and each $v \in e$, add $\{v, e \setminus \{v\}\}$ to E' if and only if intersection$[e \setminus \{v\}] =$ true and intersection$[\{v\}] =$ false. In this case, it follows that e is separable into

(i) a subset $e \setminus \{v\}$ of an edge of W, since intersection$[e \setminus \{v\}] =$ true, and
(ii) the vertex v that is not contained in any edge in W and, hence, contained in I, since intersection$[\{v\}] =$ false.

Thus, e clearly satisfies the definition of E'. Finally, for each edge $\{v, C\}$ added to E', add v to I and C to S. Herein, checking that an element is not added to I or S multiple times can be done in $O(d)$ time per element: to this end, we use a trie data structure similarly as "petals[]" in Lemma 2 or "intersection[]" in Algorithm 2. Similarly to Lemma 2, the trie can be initialized in linear time, since we know the elements to be added to I and S in advance.

It remains to shrink I to $O(k^{d-1})$ vertices by computing a maximum matching in B and deleting from H the unmatched vertices in I and the edges containing them. However, note that by construction of B, for each edge in H, we add at most one edge and two vertices to B. Therefore, B has $O(k^d)$ edges and vertices. Hence, a maximum matching on B can be computed in $O(\sqrt{|I \uplus H|} \cdot |E'|) = O(k^{1.5d})$ time using the algorithm of Hopcroft and Karp [22, Theorem 16.4]. □

4 Conclusion

We have improved the running times of the $O(k^{d-1})$-vertex problem kernels for d-Hitting Set by Abu-Khzam [2] and Moser [16]. To this end, we showed, as claimed earlier by Niedermeier and Rossmanith [18], that a polynomial-size problem kernel for d-Hitting Set can be computed in linear time—more specifically, a problem kernel comprising $O(k^d)$ hyperedges and vertices. In contrast to these problem kernels, our algorithm maintains expressiveness by finding, in forms of sunflowers, concise explanations of potential problem solutions. However, the constant hidden in our $O(k^{d-1})$-bound on the number of vertices is $d!d^{d+2}$ and therefore higher than the constant $2d - 1$ obtained by Abu-Khzam [2]. This is due to the fact that our upper bound on the size of the weakly related set W comes from the sunflower lemma in Lemma 3, whereas Abu-Khzam [2] executes more effective data reduction on W. Regarding these constants, first experiments with an implementation of our algorithm show that the data reduction is indeed effective. It is interesting whether a problem kernel with $O(k^{d-1})$ vertices and $O(k^d)$ edges for d-Hitting Set can be computed in linear time. This would merge the best known results for problem kernels for d-Hitting Set. However, all known $O(k^{d-1})$-vertex problem kernels for d-Hitting Set, that is, the problem kernels by Abu-Khzam [2] and Moser [16, Section 7.3], involve the computation of maximum matchings. This seems to be a difficult to avoid bottleneck.

Acknowledgment. The author is very thankful to Rolf Niedermeier for many valuable comments.

References

[1] Abtreu, R., Zoeteweij, P., van Gemund, A.J.C.: A dynamic modeling approach to software multiple-fault localization. In: Proc. 19th DX, pp. 7–14. Blue Mountains, NSW, Australia (2008)

[2] Abu-Khzam, F.N.: A kernelization algorithm for d-hitting set. J. Comput. Syst. Sci. 76(7), 524–531 (2010)

[3] Aho, A.V., Hopcroft, J.E., Ullman, J.D.: Data Structures and Algorithms. Addison-Wesley (1983)

[4] van Bevern, R., Hartung, S., Kammer, F., Niedermeier, R., Weller, M.: Linear-Time Computation of a Linear Problem Kernel for Dominating Set on Planar Graphs. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 194–206. Springer, Heidelberg (2012)

[5] van Bevern, R., Moser, H., Niedermeier, R.: Approximation and tidying—a problem kernel for s-plex cluster vertex deletion. Algorithmica 62(3), 930–950 (2012)

[6] Bodlaender, H.L.: Kernelization: New Upper and Lower Bound Techniques. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 17–37. Springer, Heidelberg (2009)

[7] Dell, H., van Melkebeek, D.: Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In: Proc. 42nd STOC 2010, pp. 251–260. ACM (2010)

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

[9] Erdős, P., Rado, R.: Intersection theorems for systems of sets. J. London Math. Soc. 35, 85–90 (1960)

[10] Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer (2006)

[11] Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. SIGACT News 38(1), 31–45 (2007)

[12] Hagerup, T.: Linear-Time Kernelization for Planar Dominating Set. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 181–193. Springer, Heidelberg (2012)

[13] Hüffner, F., Komusiewicz, C., Moser, H., Niedermeier, R.: Fixed-parameter algorithms for cluster vertex deletion. Theory Comput. Syst. 47(1), 196–217 (2010)

[14] de Kleer, J., Williams, B.C.: Diagnosing multiple faults. Artif. Intell. 32(1), 97–130 (1987)

[15] Kratsch, S.: Polynomial kernelizations for MIN F^+ Π_1 and MAX NP. Algorithmica 63(1), 532–550 (2012), ISSN 0178-4617

[16] Moser, H.: Finding Optimal Solutions for Covering and Matching Problems. PhD thesis. Institut für Informatik, Friedrich-Schiller-Universität Jena (2010)

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

[18] Niedermeier, R., Rossmanith, P.: An efficient fixed-parameter algorithm for 3-hitting set. J. Discrete Algorithms 1(1), 89–102 (2003)

[19] Nishimura, N., Ragde, P., Thilikos, D.M.: Smaller Kernels for Hitting Set Problems of Constant Arity. In: Downey, R.G., Fellows, M.R., Dehne, F. (eds.) IWPEC 2004. LNCS, vol. 3162, pp. 121–126. Springer, Heidelberg (2004)

[20] Protti, F., Dantas da Silva, M., Szwarcfiter, J.: Applying modular decomposition to parameterized cluster editing problems. Theory Comput. Syst. 44, 91–104 (2009)

[21] Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. 32(1), 57–95 (1987)

[22] Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency, vol. A. Springer (2003)

Maximum Number of Minimal Feedback Vertex Sets in Chordal Graphs and Cographs*

Jean-François Couturier[1], Pinar Heggernes[2],
Pim van 't Hof[2], and Yngve Villanger[2]

[1] LITA, Université Paul Verlaine - Metz, 57045 Metz Cedex 01, France
couturier@univ-metz.fr
[2] Department of Informatics, University of Bergen, N-5020 Bergen, Norway
{pinar.heggernes,pim.vanthof,yngve.villanger}@ii.uib.no

Abstract. A feedback vertex set in a graph is a set of vertices whose removal leaves the remaining graph acyclic. Given the vast number of published results concerning feedback vertex sets, it is surprising that the related combinatorics appears to be so poorly understood. The maximum number of minimal feedback vertex sets in a graph on n vertices is known to be at most 1.864^n. However, no examples of graphs having 1.593^n or more minimal feedback vertex sets are known, which leaves a considerable gap between these upper and lower bounds on general graphs. In this paper, we close the gap completely for chordal graphs and cographs, two famous perfect graph classes that are not related to each other. We prove that for both of these graph classes, the maximum number of minimal feedback vertex sets is $10^{\frac{n}{5}} \approx 1.585^n$, and there is a matching lower bound.

1 Introduction

The study of maximum number of vertex subsets satisfying a given property in a graph has always attracted interest and found applications in combinatorics and computer science. Especially during the last decades there has been a tremendous increase of interest in exponential time algorithms, whose running times often rely on the maximum number of certain objects in graphs [12]. A classical example is the highly cited and widely used result of Moon and Moser [19], who showed that the maximum number of maximal cliques and maximal independent sets, respectively, in any graph on n vertices is $3^{\frac{n}{3}} \approx 1.442^n$. More recently, maximum numbers of minimal dominating sets, minimal feedback vertex sets, minimal subset feedback vertex sets, minimal separators, and potential maximal cliques in general graphs, and minimal feedback vertex sets in tournaments have been studied [9,10,11,13,14].

A *feedback vertex set (fvs)* in a graph is a set of vertices whose removal from the graph results in an acyclic graph. Computing a fvs of minimum cardinality or minimum weight is one of the most well-studied NP-hard problems in graph

* This work is supported by the Research Council of Norway.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 133–144, 2012.

algorithms, and a large number of papers have been published on the topic, especially during recent years. In 2002, Schwikowski and Speckenmeyer [22] showed that all minimal fvs of a given graph can be enumerated with polynomial delay. In 2008, Fomin et al. [9] showed that the maximum number of minimal fvs in a graph on n vertices is at most 1.864^n. This immediately implies an algorithm with running time $O(1.864^n)$ for listing all minimal fvs of a graph, and for computing a fvs of minimum weight. It also shows the impact of good upper bounds on the maximum number of important objects in graphs. Although for some objects, e.g., maximal independent sets [19], the known upper bound on their maximum number matches the known lower bound, this is unfortunately not the case for all objects, and in particular for minimal fvs. In fact, we do not have examples of graphs that have 1.593^n or more minimal fvs [9], and hence there is a considerable gap between the upper and lower bounds on the maximum number of minimal fvs in graphs.

Motivated by the results of Fomin et al. [9] and the mentioned gap, we turn our attention to graph classes with the objective of narrowing the gap on graphs with particular structure. In this paper we close the gap completely on chordal graphs and cographs. In particular, we show that the maximum number of minimal fvs in these graph classes is at most $10^{\frac{n}{5}} \approx 1.585^n$, and that this also matches the lower bound. Our results are obtained by purely combinatorial arguments, whereas the upper bound in [9] and most of the other cited upper bounds are obtained by algorithmic tools. One of the computational implications of our results is that all minimal fvs in chordal graphs and cographs can be listed in time $O(1.585^n)$, using the algorithm of Schwikowski and Speckenmeyer [22]. Furthermore, our result on chordal graphs implies that all minimal fvs in circular arc graphs can also be listed in time $O(1.585^n)$. Another implication is that the search for better lower bounds on the maximum number of minimal fvs in general graphs can discard all chordal graphs and cographs.

Although a lot of attention has been given to graph classes when it comes to tractability of optimization problems, they have been left largely unexplored when it comes to counting, enumerating, and determining the maximum number of objects, apart from a few recent results [7,16,20,21]. Chordal graphs and cographs are famous and well-studied subclasses of perfect graphs with many applications in real-life problems, like sparse matrix computations, perfect phylogeny, VLSI, and computer vision [2,15,23]. Many problems that are NP-hard in general can be solved in polynomial time on these graph classes, and this is also the case for the problem of computing a fvs of minimum weight [6,23]. Hence our motivation and results are not related to efficient computation of a fvs of minimum weight or cardinality in these graph classes (the same applies to several of the results in [7,16,20,21]). However, these results might have computational implications for other, seemingly unrelated, optimization problems on the studied graph classes. For example, using Moon and Moser's [19] upper bound on the maximum number of maximal independent sets, Lawler [17] gave an algorithm for graph coloring, which was the fastest for over two decades. A

faster algorithm for graph coloring was obtained by Eppstein [8] by improving the upper bound for maximal independent sets of small size.

2 Preliminaries

We work with simple undirected graphs. We denote such a graph by $G = (V, E)$, where V is the set of vertices and E is the set of edges of G. We adhere to the convention that $n = |V|$. The set of *neighbors* of a set of vertices $S \subset V$ is the set $N_G(S) = \{u \notin S \mid v \in S, uv \in E\}$. A vertex is *universal* if it is adjacent to every other vertex. The subgraph of G induced by a set $S \subset V$ is denoted by $G[S]$. A graph is *connected* if there is a path between every pair of its vertices. A maximal connected subgraph is called a *connected component*. A set $S \subseteq V$ is called an *independent set* if $uv \notin E$ for every pair of vertices $u, v \in S$, and S is called a *clique* if $uv \in E$ for every pair of vertices $u, v \in S$. An independent set or a clique is *maximal* if no proper superset of it is an independent set or a clique, respectively. A complete graph on n vertices is denoted by K_n.

Graph Classes. A *chord* of a cycle (or path) is an edge between two non-consecutive vertices of the cycle (or path). A graph is *chordal* if every cycle of length at least 4 has a chord. A chordal graph has at most n maximal cliques. A *clique tree* of a graph G is a tree T, whose set of nodes is the set of maximal cliques of G, that satisfies the following: for every vertex v of G, the nodes of T that correspond to maximal cliques of G containing v induce a connected subtree of T. A graph has a clique tree if and only if it is chordal [4]. Chordal graphs can be recognized and a clique tree can be constructed in linear time [1].

The *disjoint union* operation on graphs takes as input a collection of graphs and outputs the collection as one graph, without adding any edges. The *complete join* operation on graphs takes as input a collection of graphs and adds edges between every pair of vertices that belong to two different graphs in the collection. A graph is a *cograph* if it can be generated from single-vertex graphs with the use of disjoint union and complete join operations. Cographs are exactly the graphs that do not contain chordless paths of length at least 4 as induced subgraphs, and they can be recognized in linear time [5]. Chordal graphs and cographs are subclasses of perfect graphs, but they are not related to each other. They form two of the most well-studied graph classes.

A *circular arc graph* is the intersection graph of a set of arcs on a circle. Circular arc graphs have at most n maximal cliques, and they have a *clique cycle* representation, analogous to clique trees. Circular arc graphs do not form a subclass of perfect graphs or a superclass of chordal graphs or cographs. They can be recognized in linear time [18]. All three mentioned graph classes are closed under taking induced subgraphs. More details about these graph classes and perfect graphs, and references omitted due to page restrictions, can be found in the books by Brandstädt et al. [2] and Golumbic [15].

Minimal Feedback Vertex Sets and Our Lower Bound. A set $S \subseteq V$ is called a *feedback vertex set (fvs)* if $G[V \setminus S]$ is a forest. A fvs S is *minimal* if no

proper subset of S is a fvs. In that case, $G[V \setminus S]$ is a *maximal induced forest* of G. Hence there is a bijection between the minimal fvs and the maximal induced forests of a graph, and the number of minimal fvs is equal to the number of maximal induced forests. For a disconnected graph, the total number of minimal fvs is the product of the numbers of minimal fvs of its connected components. We will use these facts extensively in our arguments.

In particular, we obtain a lower bound on the number of minimal fvs of chordal graphs and cographs as follows: for any positive integer k, let G_k be a graph on $n = 5k$ vertices that is the disjoint union of k copies of K_5. G_k is both a chordal graph and a cograph. Observe that each edge in K_5 is a maximal induced forest of K_5, and K_5 has no other maximal induced forests. Hence K_5 has 10 maximal induced forests, or minimal fvs. Consequently, for every k, G_k has $10^k = 10^{\frac{n}{5}} \approx 1.585^n$ minimal fvs. This provides an infinite family of graphs that constitute our lower bound example. In the next two sections, we will give matching upper bounds for chordal graphs and cographs.

3 A Tight Bound for Chordal Graphs

In this section, we provide the upper bound on the number of minimal fvs in chordal graphs that matches the lower bound given in Section 2. For our arguments, it is more convenient to work with maximal induced forests instead of minimal feedback vertex sets.

We will in fact prove a slightly stronger statement than previously announced, answering the following question: Given a chordal graph $G = (V, E)$ and a set of vertices $F \subseteq V$, what is the maximum number of maximal induced forests in G that contain all vertices of F? Clearly, if $G[F]$ is not a forest the answer is 0, and if F is empty the answer is exactly the maximum number of minimal fvs of G.

Theorem 1. *Let $G = (V, E)$ be a chordal graph and let $F \subseteq V$ be such that $G[F]$ is a forest. Then G has at most $10^{\frac{n-|F|}{5}}$ maximal induced forests containing F.*

Proof. We assume that G is connected. If we can prove the bound on connected chordal graphs, then the bound trivially applies to disconnected chordal graphs, as explained in Section 2. Let T be a clique tree of G and let k be the number of nodes in T. We prove the statement of the theorem by induction on k. Recall that $n = |V|$. Let $n' = n - |F|$ and let $V' = V \setminus F$.

The base case is when $k = 1$, and thus G is a complete graph on $n \geq 1$ vertices. As G is complete, every maximal induced forest contains exactly two vertices if $n \geq 2$, and a single vertex if $n \leq 1$. For $n = 1$, there is a unique maximal induced forest, and since $1 \leq 10^0 \leq 10^{\frac{1-|F|}{5}}$, the claim holds in this case. For $n \geq 2$, since any maximal induced forest has 2 vertices, F can contain 0, 1, or 2 vertices of G. Let $i = 2 - |F|$ be the number of vertices we need to pick from $G[V']$ to obtain a maximal induced forest containing F. It is not hard to verify that $\binom{n'}{i} \leq 10^{\frac{n'}{5}}$ for every $n' \geq 0$ and $i \leq 2$, and hence there are at most

$10^{\frac{n-|F|}{5}}$ maximal induced forests in G that contain F. This completes the base case.

Now let $k \geq 2$, and assume that the statement of the theorem is true for all chordal graphs whose clique trees have at most $k-1$ nodes.

Consider a clique tree T of our graph G, which has $k \geq 2$ nodes. Let X_ℓ be a clique corresponding to a leaf of T, and let X_p be the clique corresponding to the parent of X_ℓ in T. Let $L = X_\ell \setminus X_p$ and let $C = X_\ell \cap X_p$. Observe that $C = N_G(L)$, and $X_\ell = L \cup C$. Hence, the vertices of L appear only in X_ℓ and in no other clique of G. In particular, removing L from G results in a chordal graph that has one less maximal clique, and hence has a clique tree with one less node. Also, by construction, $|L| \geq 1$ and $|C| \geq 1$.

Since C is a clique, each maximal induced forest A in G is covered by exactly one of the following three cases:

- $|A \cap C| = 2$. In this case, $A \cap L = \emptyset$, since selecting any vertex of L creates a cycle of length 3.
- $|A \cap C| = 1$. In this case, due to the maximality of A and by the same argument as in the previous case, $|A \cap L| = 1$.
- $|A \cap C| = 0$. In this case, by the same arguments as above, if $|L| \geq 2$ then $|A \cap L| = 2$, and if $|L| = 1$ then $|A \cap L| = 1$.

Recall the set F mentioned in the lemma, and let $\mathcal{A}_G$ be the set of all maximal induced forests in G containing F. Let $F' = F \setminus X_\ell = F \setminus (L \cup C)$. In order to obtain an upper bound on $|\mathcal{A}_G|$, we will first count all maximal induced forests of G containing F'. This number is clearly at least as large as the number of maximal induced forests of G containing F. We will argue later how we can take care of possible over-counting that may occur as a result of weakening the provided restriction from F to F'.

As a direct consequence of the three cases mentioned above, we obtain the following upper bound for the number of maximal induced forests of G containing F', and hence also for $|\mathcal{A}_G|$:

$$|\mathcal{A}_G| \leq \binom{|C|}{2}|\mathcal{A}_{G_2}| + |L||C||\mathcal{A}_{G_1}| + \max\left\{1, \binom{|L|}{2}\right\}|\mathcal{A}_{G_0}|, \tag{1}$$

where $\mathcal{A}_{G_0}$ is the set of maximal induced forests in $G_0 = G[V \setminus X_\ell]$ containing F', $\mathcal{A}_{G_1}$ is the set of maximal induced forests in $G_1 = G[V \setminus (X_\ell \setminus \{u\})]$ containing $F' \cup \{u\}$, where u is some vertex in C, and $\mathcal{A}_{G_2}$ is the set of maximal induced forests of $G_2 = G[V \setminus (X_\ell \setminus \{u, v\})]$ containing $F' \cup \{u, v\}$, where u, v is some pair of vertices in C.

Since none of the graphs G_0, G_1, G_2 contain vertices of L, each of them has a clique tree with $\leq k-1$ nodes, as argued above. By our induction assumption it follows that $|\mathcal{A}_{G_0}| \leq 10^{\frac{n-|X_\ell|-|F'|}{5}}$, $|\mathcal{A}_{G_1}| \leq 10^{\frac{n+1-|X_\ell|-(|F'|+1)}{5}}$, and $|\mathcal{A}_{G_2}| \leq 10^{\frac{n+2-|X_\ell|-(|F'|+2)}{5}}$. In other words, each of the three sets contains at most $10^{\frac{n-|X_\ell|-|F'|}{5}}$ maximal induced forests.

The formula $\binom{|C|}{2} + |L||C| + \binom{|L|}{2}$ gives the number of ways we can select two vertices from C, or one vertex from L and one from C, or two from L. This

number is equal to the number of ways we can select two vertices from $L \cup C$, i.e., $\binom{|C|}{2} + |L||C| + \binom{|L|}{2} = \binom{|L \cup C|}{2} = \binom{|X_\ell|}{2}$. Formula (1) therefore implies:

$$|\mathcal{A}_G| \leq 10^{\frac{n-|X_\ell|-|F'|}{5}} \binom{|X_\ell|}{2}. \tag{2}$$

We now use Formula (2) in order to find an upper bound for the number of maximal induced forests of G containing F instead of F'. Consider the set $F \cap X_\ell$. As $G[F]$ is a forest and X_ℓ is a clique, we know that $i = |F \cap X_\ell| \in \{0, 1, 2\}$. This means that i vertices of X_ℓ are preselected to be in a forest, which must also contain the vertices of F'. As $\binom{|X_\ell|}{2-i} \leq 10^{\frac{|X_\ell|-i}{5}}$ and $|F'| + i = |F|$, we can use Formula (2) to get:

$$|\mathcal{A}_G| \leq 10^{\frac{n-|X_\ell|-|F'|}{5}} \binom{|X_\ell|}{2-i} \leq 10^{\frac{n-|X_\ell|-|F|+i+|X_\ell|-i}{5}} \leq 10^{\frac{n-|F|}{5}}.$$

This concludes the proof of Theorem 1. □

Corollary 1. *Every chordal graph on n vertices has at most $10^{\frac{n}{5}}$ minimal feedback vertex sets.*

Although our main aim was to prove the statement of Corollary 1, note that the stronger statement of Theorem 1 makes it more useful than Corollary 1. Suppose we want to bound the number of minimal fvs in other graph classes or in general graphs using a branching algorithm or a combinatorial argument. If at some stage of the algorithm we end up with subgraphs of the input graph that are chordal, in which some vertices have been preselected to belong to the solution, then Theorem 1 can be applied directly to these subproblems, allowing their solutions to be combined into a solution for the whole graph.

4 A Tight Bound for Cographs

We will now prove an upper bound of $10^{\frac{n}{5}}$, matching the lower bound given in Section 2, also on the number of minimal fvs in cographs. As in the previous section, we will work with maximal induced forests. Even though the upper bound for cographs is the same as the upper bound for chordal graphs, the two proofs are different. In particular, the proof for cographs requires in addition the use of an upper bound on the number of maximal independent sets.

We start by defining a function f, which will ease the notation in the rest of this section:

$$f(\alpha, \beta, i, j) = \alpha^i + \alpha^j + i\beta^j + j\beta^i$$

We need a property of f described in the next lemma whose proof is omitted in this extended abstract.

Lemma 1. *For $\alpha = 10^{\frac{1}{5}}$, $\beta = 3^{\frac{1}{3}}$, $i \geq i' \geq 1$, and $j \geq j' \geq 1$, such that $(i', j') \in \{(1, 10), (2, 8), (3, 7), (4, 5), (5, 4), (7, 3), (8, 2), (10, 1)\}$, the following holds:*

$$f(\alpha, \beta, i, j) \leq \alpha^{i+j}.$$

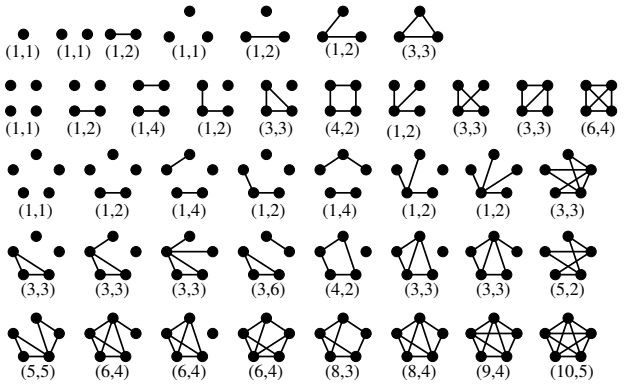


Fig. 1. All cographs on at most five vertices [24]. Each graph is labeled with a pair of numbers: the first number is the number of maximal induced forests, and the second number is the number of maximal independent sets in the graph.

We are now ready to prove our result on cographs.

Theorem 2. *Every cograph on n vertices has at most $10^{\frac{n}{5}}$ maximal induced forests.*

Proof. We will prove the theorem by induction on the number of vertices of a cograph. The base case concerns cographs on at most 5 vertices. All cographs on at most 5 vertices [24] are listed in Figure 1. Examining this figure, one can verify that none of these graphs has more than $10^{\frac{n}{5}}$ maximal induced forests.

Assume now that the statement of the theorem is true for all cographs on at most $n-1$ vertices, and let $G = (V, E)$ be a cograph on $n \geq 6$ vertices. By the definition of cographs, G is either disconnected or the complete join of two cographs. As explained in Section 2, if G is disconnected, then it suffices to prove the upper bound on each connected component. Hence we can assume that G is connected. Let $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$ be the two cographs such that G is the complete join of G_0 and G_1. Let $n_i = |V_i|$ for $i \in \{0, 1\}$. Observe that $n = n_0 + n_1$, that $n_0 \geq 1$ and $n_1 \geq 1$, and that every vertex in V_0 is adjacent to every vertex in V_1.

Let $A \subseteq V$ be such that $G[A]$ is a maximal induced forest of G. For $i, j \in \{0, 1\}$ and $i \neq j$, we can observe that

- if $|A \cap V_i| \geq 2$, then $|A \cap V_j| \leq 1$, since otherwise the forest would contain a cycle of length 4,
- if $|A \cap V_i| = 1$, then $A \cap V_j$ is a maximal independent set, since otherwise the forest would contain a cycle of length 3, and
- if $A \cap V_i = \emptyset$, then $A \subseteq V_j$ and $G[A]$ is a maximal induced forest in G_j.

Let $\mathcal{I}_n$ be the maximum number of maximal independent sets in a graph on n vertices. The three observations above show that a maximal induced forest in G

is either a maximal induced forest in G_i, or a single vertex in V_i and a maximal independent set in G_j, for $i, j \in \{0, 1\}$ with $i \neq j$. This gives us the following formula, where $\mathcal{A}_G$ is the set of all maximal induced forests in G:

$$|\mathcal{A}_G| \leq |\mathcal{A}_{G_0}| + |\mathcal{A}_{G_1}| + n_1\mathcal{I}_{n_0} + n_0\mathcal{I}_{n_1}. \tag{3}$$

Moon and Moser [19] have shown that $\mathcal{I}_n \leq 3^{\frac{n}{3}}$. Inserting this in (3) and using our induction assumption, we obtain the following:

$$|\mathcal{A}_G| \leq 10^{\frac{n_0}{5}} + 10^{\frac{n_1}{5}} + n_1 3^{\frac{n_0}{3}} + n_0 3^{\frac{n_1}{3}}. \tag{4}$$

Now, our aim is to apply Lemma 1 on Formula (4) for values of n_0 and n_1 that satisfy the premises of that lemma, which will immediately imply that $|\mathcal{A}_G| \leq 10^{\frac{n}{5}}$ for large enough values of n. However, before we can do that, for the soundness of our induction, we need to show that for all smaller values of n_0 and n_1, it holds that $|\mathcal{A}_G| \leq 10^{\frac{n_0+n_1}{5}} = 10^{\frac{n}{5}}$. We use Table 1 to help us keep track of the possible cases. For all values of n_0 and n_1 such that $n_0 + n_1 \leq 5$, we know by the induction base case that $|\mathcal{A}_G| \leq 10^{\frac{n}{5}}$. Each combination of pairs of such values is marked with "≤ 5" in Table 1. We now examine all pairs of values of n_0 and n_1 with $n_0 + n_1 > 5$, and $n_0 \leq i$ and $n_1 \leq j$ for some pair $(i, j) \in \mathcal{B} = \{(1, 10), (2, 8), (3, 7), (4, 5), (5, 4), (7, 3), (8, 2), (10, 1)\}$. Once we have shown that $|\mathcal{A}_G| \leq 10^{\frac{n}{5}}$ for all such values, we can apply Lemma 1 for all larger values. Pairs of small n_0, n_1 values for which Lemma 1 can be applied are marked with "OK" in Table 1. Clearly we can also apply the lemma on all larger values of n_0 and n_1.

We will now analyze, in the correct induction order, the values of n_0 and n_1 in the area between those entries that are marked "≤ 5" and those that are marked "OK" in Table 1. Let us start with $n_0 = 1$ and $n_1 = 5$. The single vertex of G_0 is either included in a maximal induced forest F of G, in which case F must be a maximal independent set in G_1, or it is not, in which case F is a maximal induced forest in G_1. Hence the number of maximal induced forests in G is at most the number of maximal induced forests in G_1 plus the number of maximal independent sets in G_1, i.e., $|\mathcal{A}_G| \leq |\mathcal{A}_{G_1}| + \mathcal{I}_{n_1}$. Using Figure 1, we can verify that the number of maximal induced forests plus the number of maximal independent sets is at most 15 for any cograph on 5 vertices. Since $15 < 10^{\frac{6}{5}}$, the statement of the theorem holds for this case. This case is marked with "$C1$" in Table 1.

Now consider the case where $n_0 = 2$ and $n_1 = 4$. Observe first that if a vertex of G_0 is universal, then it can be moved to G_1, and G is still a complete join of the modified graphs G_0 and G_1. By this operation n_0 becomes 1, n_1 becomes 5, and we are back in Case $C1$. Therefore, assume now that the two vertices of G_0 are not adjacent. Let F be a maximal induced forest in G. If F contains 0 vertices from G_0, then F is a maximal induced forest in G_1. If F contains exactly 1 vertex from G_0, then F is a maximal independent set in G_1. Finally, if F contains both vertices of G_0, then F contains exactly 1 vertex from G_1. This yields the formula $|\mathcal{A}_G| \leq |\mathcal{A}_{G_1}| + 2\mathcal{I}_{n_1} + n_1$. For any cograph G_1 on 4 vertices in which the number of maximal induced forests plus 2 times the number of

Table 1. This table aids in the analysis of $|\mathcal{A}_G|$ for a cograph G on $n_0 + n_1$ vertices

n_0 \ n_1	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	≤ 5	≤ 5	≤ 5	≤ 5	$C1$	$R1$	$R2$	$R3$	$R4$	OK
2	≤ 5	≤ 5	≤ 5	$C2$	$C4$	$C6$	$C7$	OK	OK	OK
3	≤ 5	≤ 5	$C3$	$C5$	$C8$	$C9$	OK	OK	OK	OK
4	≤ 5	$C2$	$C5$	$C10$	OK	OK	OK	OK	OK	OK
5	$C1$	$C4$	$C8$	OK	OK	OK	OK	OK	OK	OK
6	$R1$	$C6$	$C9$	OK	OK	OK	OK	OK	OK	OK
7	$R2$	$C7$	OK	OK	OK	OK	OK	OK	OK	OK
8	$R3$	OK	OK	OK	OK	OK	OK	OK	OK	OK
9	$R4$	OK	OK	OK	OK	OK	OK	OK	OK	OK
10	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK

maximal independent sets is at most 11 we are fine, as $11 + 4 < 10^{\frac{6}{5}}$. The only graph violating this condition is K_4 (see Figure 1). When G_1 is K_4, we notice that a maximal independent set of size 1 in G_1 will not be maximal induced forest when joined with only one vertex of G_0. Thus the term $2\mathcal{I}_{n_1}$ can be ignored, and we get $|\mathcal{A}_G| \leq |\mathcal{A}_{G_1}| + n_1 \leq 6 + 4 \leq 10$. This case is marked with "$C2$" in Table 1.

The next case is when $n_0 = 3$ and $n_1 = 3$. Universal vertices can again be moved between G_0 and G_1, and due to symmetry this argument holds in both directions. We see in Figure 1 that cographs on 3 vertices without a universal vertex have at most one maximal induced forest and at most two maximal independent sets. Thus we get that $|\mathcal{A}_G| \leq |\mathcal{A}_{G_0}| + |\mathcal{A}_{G_1}| + n_1\mathcal{I}_{n_0} + n_0\mathcal{I}_{n_1} \leq 1 + 1 + 3 \cdot 2 + 3 \cdot 2 = 14 < 10^{\frac{6}{5}}$. This case is marked with "$C3$" in Table 1.

Consider next a more general argument for the cases where $n_0 = 1$ and $n_1 = n - 1 \geq 6$. For the same reason as in Case $C1$, we have that $|\mathcal{A}_G| \leq |\mathcal{A}_{G_1}| + \mathcal{I}_{n_1}$. Using the induction assumption, $|\mathcal{A}_{G_1}| \leq 10^{\frac{n_1-1}{5}} \leq 10^{\frac{n-1}{5}}$, we can obtain the desired upper bound in this case by proving that $10^{\frac{n-1}{5}} + 3^{\frac{n-1}{3}} \leq 10^{\frac{n}{5}}$. We can rewrite this as $(3^{\frac{1}{3}}/10^{\frac{1}{5}})^{n-1} \leq 10^{\frac{1}{5}} - 1$, which is true for every $n \geq 7$. However, this can be used for each such n only after our proof has covered all (n_0, n_1) pairs such that $n_0 + n_1 \leq n - 1$. Thus at this point, it can only be used to cover $(1, 6)$ and $(6, 1)$. Accordingly, we mark cells $(1, 6)$ and $(6, 1)$ with "$R1$" in Table 1.

The next case is when $n_0 = 2$ and $n_1 = 5$. As $n_0 = 2$, we have the formula $|\mathcal{A}_G| \leq |\mathcal{A}_{G_1}| + 2\mathcal{I}_{n_1} + n_1$ as we saw in Case $C2$. When G_1 is the disjoint union of a triangle and an edge, we get that $|\mathcal{A}_G| \leq 3 + 2 \cdot 6 + 5 = 20 < 10^{\frac{7}{5}}$. By considering the numbers for the remaining graphs on 5 vertices in Figure 1, we get $|\mathcal{A}_G| \leq 10 + 2 \cdot 5 + 5 = 25 < 10^{\frac{7}{5}}$. This case is marked with "$C4$" in Table 1.

The next case is when $n_0 = 3$ and $n_1 = 4$. Note that G_0 does not contain universal vertices, as otherwise these can be moved to G_1 and we get case $(2, 5)$, which is already covered. Examining Figure 1, G_0 has at most one maximal induced forest and at most two maximal independent sets. We will use the formula $|\mathcal{A}_G| \leq |\mathcal{A}_{G_0}| + |\mathcal{A}_{G_1}| + n_1\mathcal{I}_{n_0} + n_0\mathcal{I}_{n_1}$. If G_1 is K_4, then no maximal induced

forest in G consists of only one vertex from G_0 and a maximal independent set of G_1, so the term $n_0\mathcal{I}_{n_1}$ can be ignored in this case. Consequently, we get $|\mathcal{A}_G| \leq |\mathcal{A}_{G_0}| + |\mathcal{A}_{G_1}| + n_1\mathcal{I}_{n_0} \leq 1 + 6 + 4 \cdot 2 = 15 < 10^{\frac{7}{5}}$. For any remaining cograph on 4 vertices, the maximum number of maximal induced forests is 4, and the maximum number of maximal independent sets is 4, and we get that $|\mathcal{A}_G| \leq |\mathcal{A}_{G_0}| + |\mathcal{A}_{G_1}| + n_1\mathcal{I}_{n_0} + n_0\mathcal{I}_{n_1} \leq 1 + 4 + 4 \cdot 2 + 3 \cdot 4 = 25 < 10^{\frac{7}{5}}$. In Table 1 this case is marked with "$C5$".

Recalling Case $R1$, by the arguments so far, the cells of Table 1 marked with "$R2$" have also been covered.

At this point, we have verified that the statement of the theorem holds for all cographs on at most 7 vertices. The two cases where $n_0 = 2$ and $n_1 \in \{6, 7\}$ are considered next. As the numbers of maximal induced forests and maximal independent sets are integers, using the induction assumption, we get the formula for $n_0 = 2$ that we saw in Case $C2$, i.e., $|\mathcal{A}_G| \leq |\mathcal{A}_{G_1}| + 2\mathcal{I}_{n_1} + n_1 \leq \lfloor 10^{\frac{n_1}{5}} \rfloor + 2 \cdot \lfloor 3^{\frac{n_1}{3}} \rfloor + n_1$. We want to argue that this is at most $10^{\frac{n_1+2}{5}}$. For $n_1 = 6$, we have $\lfloor 10^{\frac{6}{5}} \rfloor + 2 \cdot \lfloor 3^{\frac{6}{3}} \rfloor + 6 \leq 15 + 2 \cdot 9 + 6 \leq 39 < 10^{\frac{8}{5}}$. For $n_1 = 7$, we have $\lfloor 10^{\frac{7}{5}} \rfloor + 2 \cdot \lfloor 3^{\frac{7}{3}} \rfloor + 7 \leq 25 + 2 \cdot 12 + 7 \leq 56 < 10^{\frac{9}{5}}$. These cases are marked with "$C6$" and "$C7$" in Table 1.

We continue now with the case where $n_0 = 3$ and $n_1 \in \{5, 6\}$. Like for the case $n_0 = 2$, we can notice that any universal vertex in G_0 can be moved to G_1, yielding a situation covered by either Case $C6$ or $C7$. Hence we assume that G_0 does not contain a universal vertex. For $n_1 = 5$, we use the upper bounds for graphs on 5 vertices given in Figure 1 to get $|\mathcal{A}_G| \leq |\mathcal{A}_{G_0}| + |\mathcal{A}_{G_1}| + n_1\mathcal{I}_{n_0} + n_0\mathcal{I}_{n_1} \leq 1+10+5\cdot 2+3\cdot 6 = 39 < 10^{\frac{8}{5}}$. This case is marked with "$C8$" in Table 1. For $n_1 = 6$, the argument is quite similar: $|\mathcal{A}_G| \leq |\mathcal{A}_{G_0}|+|\mathcal{A}_{G_1}|+n_1\mathcal{I}_{n_0}+n_0\mathcal{I}_{n_1} \leq 1 + 10^{\frac{6}{5}} + 6 \cdot 2 + 3 \cdot 3^{\frac{6}{3}} < 55 < 10^{\frac{9}{5}}$. This case is marked with "$C9$" in Table 1.

The only remaining case is when $n_0 = 4$ and $n_1 = 4$. If either G_0 or G_1 has a universal vertex, then we can move one vertex and reach a case that is already covered. Examining Figure 1, G_0 has at most 4 maximal induced forests and at most 4 maximal independent sets, and G_1 is either a cycle of length 4 or has at most 3 maximal induced forests and 4 maximal independent sets. If G_1 is a cycle of length 4, we get that $|\mathcal{A}_G| \leq |\mathcal{A}_{G_0}| + |\mathcal{A}_{G_1}| + n_1\mathcal{I}_{n_0} + n_0\mathcal{I}_{n_1} \leq 4 + 4 + 4 \cdot 4 + 4 \cdot 2 = 32 < 10^{\frac{8}{5}}$. For the remaining case, we get that $|\mathcal{A}_G| \leq |\mathcal{A}_{G_0}| + |\mathcal{A}_{G_1}| + n_1\mathcal{I}_{n_0} + n_0\mathcal{I}_{n_1} \leq 4 + 3 + 4 \cdot 4 + 4 \cdot 4 = 39 < 10^{\frac{8}{5}}$. This last case is marked with "$C10$" in Table 1.

By the arguments so far, the cells of Table 1 marked with "$R3$" and "$R4$" have also been covered. We can now safely apply Lemma 1 on all other n_0, n_1 values, and the statement of the theorem follows. □

5 Circular Arc Graphs and Concluding Remarks

Fomin et al. [9] give an example of an infinite family of graphs having $105^{\frac{n}{10}} \approx 1.593^n$ minimal fvs, providing the best known lower bound for general graphs. Interestingly, their example is a disjoint union of copies of a particular circular

arc graph. This circular arch graph has 10 vertices and 105 minimal fvs [9]. An infinite family of graphs having $105^{\frac{n}{10}}$ minimal fvs is obtained by taking $\frac{n}{10}$ copies of this graph. Note, however, that disjoint unions of circular arc graphs are not necessarily circular arc graphs. In particular, this obtained family does not belong to the class of circular arc graphs. As a consequence, two interesting questions emerge. What is the upper bound for disjoint unions of circular arc graphs; can it be that it matches the lower bound? What is the maximum number of minimal fvs in circular arc graphs?

By our upper bound on chordal graphs, we immediately obtain that the maximum number of minimal fvs of a circular arc graph is $O(1.585^n)$, as follows. Let us call the intersection between two consecutive maximal cliques in a clique cycle of a circular arc graph a *breaker*. There are at most n breakers, and the removal of each breaker results in a chordal graph. It is not difficult to see that for every maximal induced forest F, there is a breaker that does not contain any vertex of F. Consequently, by the results of Section 3, we obtain that the maximum number of minimal fvs in a circular arc graph is at most $n \cdot 10^{\frac{n}{5}} = O(1.585^n)$. A combinatorial upper bound without the use of O-notation remains an open question.

We have shown that the maximum number of fvs in chordal graphs and cographs is at most 1.585^n, and that this bound is tight. As a consequence of our results and the result of Schwikowski and Speckenmeyer [22], all minimal fvs, or equivalently all maximal induced forests, of a chordal graph, cograph, or circular arc graph can be listed in time $O(1.585^n)$.

As a final question, we ask whether the exact number of minimal fvs of a given graph can be computed in polynomial time for these three graph classes. For cographs, arguments along the lines of the results by Bui-Xuan et al. [3] are likely to work for a polynomial-time algorithm, since cographs have bounded clique-width. We would not be surprised if also for chordal graphs the counting problem could be solved in polynomial time.

Acknowledgement. The authors are indebted to Dieter Kratsch for useful discussions on the topic.

References

1. Blair, J.R.S., Peyton, B.W.: An Introduction to Chordal Graphs and Clique Trees. In: Graph Theory and Sparse Matrix Computations. IMA Vol. in Math. Appl., vol. 56, pp. 1–27. Springer
2. Brandstädt, A., Le, V.B., Spinrad, J.: Graph Classes: A Survey. Monographs on Discrete Mathematics and Applications (1999)
3. Bui-Xuan, B.M., Telle, J.A., Vatshelle, M.: Feedback Vertex Set on Graphs of low Cliquewidth. In: Fiala, J., Kratochvíl, J., Miller, M. (eds.) IWOCA 2009. LNCS, vol. 5874, pp. 113–124. Springer, Heidelberg (2009)
4. Buneman, P.: A characterization of rigid circuit graphs. Disc. Math. 9, 205–212 (1974)

5. Corneil, D.G., Perl, Y., Stewart, L.: A linear recognition algorithm for cographs. SIAM J. Computing 14, 926–934 (1985)
6. Corneil, D.G., Fonlupt, J.: The complexity of generalized clique covering. Disc. Appl. Math. 22, 109–118 (1988/1989)
7. Couturier, J.-F., Heggernes, P., van 't Hof, P., Kratsch, D.: Minimal Dominating Sets in Graph Classes: Combinatorial Bounds and Enumeration. In: Bieliková, M., Friedrich, G., Gottlob, G., Katzenbeisser, S., Turán, G. (eds.) SOFSEM 2012. LNCS, vol. 7147, pp. 202–213. Springer, Heidelberg (2012)
8. Eppstein, D.: Small maximal independent sets and faster exact graph coloring. J. Graph Algor. Appl. 7(2), 131–140 (2003)
9. Fomin, F.V., Gaspers, S., Pyatkin, A.V., Razgon, I.: On the minimum feedback vertex set problem: Exact and enumeration algorithms. Algorithmica 52(2), 293–307 (2008)
10. Fomin, F.V., Grandoni, F., Pyatkin, A.V., Stepanov, A.A.: Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. ACM Trans. Algorithms 5(1) (2008)
11. Fomin, F.V., Heggernes, P., Kratsch, D., Papadopoulos, C., Villanger, Y.: Enumerating Minimal Subset Feedback Vertex Sets. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 399–410. Springer, Heidelberg (2011)
12. Fomin, F.V., Kratsch, D.: Exact Exponential Algorithms. Springer, Texts in Theoretical Computer Science (2010)
13. Fomin, F.V., Villanger, Y.: Finding induced subgraphs via minimal triangulations. In: Proceedings STACS 2010, pp. 383–394 (2010)
14. Gaspers, S., Mnich, M.: On Feedback Vertex Sets in Tournaments. In: de Berg, M., Meyer, U. (eds.) ESA 2010. LNCS, vol. 6346, pp. 267–277. Springer, Heidelberg (2010)
15. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Annals of Disc. Math. 57 (2004)
16. Kanté, M.M., Limouzy, V., Mary, A., Nourine, L.: Enumeration of Minimal Dominating Sets and Variants. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 298–309. Springer, Heidelberg (2011)
17. Lawler, E.L.: A note on the complexity of the chromatic number problem. Inform. Proc. Lett. 5, 66–67 (1976)
18. McConnell, R.: Linear-time recognition of circular-arc graphs. Algorithmica 37, 93–147 (2003)
19. Moon, J.W., Moser, L.: On cliques in graphs. Israel J. Math. 3, 23–28 (1965)
20. Okamoto, Y., Uehara, R., Uno, T.: Counting the Number of Matchings in Chordal and Chordal Bipartite Graph Classes. In: Paul, C., Habib, M. (eds.) WG 2009. LNCS, vol. 5911, pp. 296–307. Springer, Heidelberg (2010)
21. Okamoto, Y., Uno, T., Uehara, R.: Counting the number of independent sets in chordal graphs. J. Disc. Alg. 6, 229–242 (2008)
22. Schwikowski, B., Speckenmeyer, E.: On enumerating all minimal solutions of feedback problems. Disc. Appl. Math. 117, 253–265 (2002)
23. Spinrad, J.P.: Efficient graph representations. AMS, Fields Institute Monograph Series 19 (2003)
24. Weisstein, E.W.: Cograph. MathWorld, http://mathworld.wolfram.com/Cograph.html

A Local Algorithm for Finding Dense Bipartite-Like Subgraphs*

Pan Peng[1,2]

[1] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
[2] School of Information Science and Engineering, Graduate University of China Academy of Sciences, P.R. China
pengpan@ios.ac.cn

Abstract. We give a *local algorithm* to extract dense bipartite-like subgraphs which characterize cyber-communities in the Web [13]. We use the *bipartiteness ratio* of a set as the quality measure that was introduced by Trevisan [20]. Our algorithm, denoted as `FindDenseBipartite`(v, s, θ), takes as input a starting vertex v, a volume target s and a bipartiteness ratio parameter θ and outputs an induced subgraph of G. It is guaranteed to have the following approximation performance: for any subgraph S with bipartiteness ratio θ, there exists a subset $S_\theta \subseteq S$ such that $\mathrm{vol}(S_\theta) \geq \mathrm{vol}(S)/9$ and that if the starting vertex $v \in S_\theta$ and $s \geq \mathrm{vol}(S)$, the algorithm `FindDenseBipartite`(v, s, θ) outputs a subgraph (X, Y) with bipartiteness ratio $O(\sqrt{\theta})$. The running time of the algorithm is $O(s^2(\Delta + \log s))$, where Δ is the maximum degree of G, independent of the size of G.

1 Introduction

A local algorithm for massive graphs is one that explores only portion of the given graph and finds a solution with good approximation guarantee. Given a graph G as an oracle, from which the algorithm can request the degree of a vertex or the adjacency list of a vertex, and a numerical property P of subgraphs (such as diameter, conductance), a local algorithm is supposed to have the following form: it takes as input a starting vertex v (or a small set of vertices), only traverses the vertices that near v and outputs a subgraph S such that $P(S)$ is close to $P(S^*)$, where S^* is the subgraph containing v that has the optimal value for P. However, in the design of local algorithms, an approximation guarantee result as above is too strong, if possible, to obtain and it is usually relaxed as follows: if S is a subgraph, then there exists a large subset $S' \subseteq S$ such that for any starting vertex $v \in S'$, the algorithm will output a subgraph for which the P value is close to $P(S)$. This algorithmic paradigm was introduced by Spielman and Teng, who gave a local algorithm for finding subgraph with small conductance [18]. Building

* The author is partially supported by the Grand Project "Network Algorithms and Digital Information" of the Institute of Software, Chinese Academy of Sciences.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 145–156, 2012.

on their work, other local clustering algorithms with better approximation ratio and running time have been proposed by Anderson, Chung and Lang [3] and Anderson and Yepes [4]. Local algorithm for finding dense subgraphs have been studied by Anderson [2]. These algorithms have important applications in graph sparsificasion, solving linear equations [17], Laplacian algorithmic paradigm [19] and have also been used to handle real networks data (e.g., [14, 15]). However, to our knowledge, only a few number of problems are shown to have such local algorithms.

In this paper, we add one more problem to this list, and give a local algorithm for extracting *dense bipartite-like* subgraphs. Such a subgraph serves as a good channel for us to understand the link structures of the Web graph (that is, the nodes are the Web pages and a directed edge (i, j) represents a hyperlink from i to j). We are interested in extracting useful information from this huge graph, one of particular interest are the cyber-communities, which gives insights into the intellectual evolution of the Web and facilitates adverting at a more precise level [13]. As found by Kumar et al [13], the cyber-communities are characterized by dense bipartite subgraphs.

To measure the property of a group of Web pages being cyber-community like, that is, whether the group is close to a dense bipartite subgraph or not, we will adopt a concept *bipartiteness ratio* introduced by Trevisan [20]. Given a graph $G = (V, E)$, a subgraph S and one of its partitions $S = (L, R)$, the bipartiteness ratio $\beta(S_{L,R})$ of S under partition (L, R) is defined to be

$$\beta(S_{L,R}) = \frac{2e(L, L) + 2e(R, R) + e(S, V \backslash \bar{S})}{\text{vol}(S)}.$$

The bipartiteness ratio $\beta(S)$ of the subgraph S is the minimum value of $\beta(S_{L,R})$ over all its possible partitions (L, R). Intuitively speaking, if $\beta(S)$ is small, then there must exist a partition (L, R) such that the number of edges from S to the outside as well as the number of edges that lie entirely in L or R is relatively small compared with all the edges involved with S. Thus, (L, R) can be seen close to a dense bipartite subgraph and S can be seen as a good Web community. Using the bipartiteness ratio as the measure of a set being dense and bipartite-like has the advantage that it unifies both properties in a natural way and admits theoretical analysis, which is difficult for many other measures.

We give a local algorithm for finding a subgraph with small bipartiteness ratio around a starting vertex v. In particular, we show that for any subgraph S with bipartiteness ratio θ and volume at most s, there exists a subset $S_\theta \subseteq S$ of large volume such that for any $v \in S_\theta$, our algorithm finds a subgraph of bipartiteness ratio $O(\sqrt{\theta})$ and runs in time $O(s^2(\log s + \Delta))$, where Δ is the maximum degree of the graph.

Our algorithm is composed of two parts. In the first part, the algorithm simulates the power method for the largest eigenvalue by a truncated process, which has already been used in previous local algorithms (eg.,[18, 2]). Such a process iteratively multiply a vector by some matrix of the graph (for example, the normalized Laplacian matrix in this paper), and during each iterative step, it only

keeps a small fraction of non-zero elements of the vector by truncating elements of relatively small values. We will see that this process allows our algorithm to be "local" in that it will only traverse a small portion of the graph.

In the second part, the algorithm will sweep over all the vectors produced in the truncated process. Such a sweep operation is implied in a spectral algorithm for the bipartiteness ratio and is similar to the sweep operation that are widely used to find small conductance from the second smallest eigenvalue of the normalized Laplacian $\mathcal{L}$. The approximation guarantee of our algorithm is derived from this part and relies on the relation between the bipartiteness ratio and the largest eigenvalue of $\mathcal{L}$ given by Trevisan [20].

Other Related Works: Previous work on extracting dense bipartite subgraphs from the Web graph have used different measures and mainly focused on giving heuristic methods (e.g., [13, 1, 7, 6]). All of them did not give theoretical analysis on the performance of the corresponding algorithms on general graphs.

The definition of bipartiteness ratio is closely related to the notion of conductance and dense subgraphs. The conductance of a vertex subset S is defined as $\frac{e(S,V\setminus S)}{\min\{\mathrm{vol}(S),\mathrm{vol}(V\setminus S)\}}$. A set of small conductance can be thought of a good community as the connections crossing the set are relatively smaller than the total number of edges involved with the set. In particular, Kannan, Vempala and Veta gave a bicriteria measure of the quality of clustering based on the concept of conductance and analyzed a corresponding spectral algorithm [12]. For the literature on dense subgraphs, Kannan and Vinay defined $d(S,T) = \frac{e(S,T)}{\sqrt{|S||T|}}$ as the measure of the density of a subgraph induced on $S \cup T$ in a directed graph and gave a spectral algorithm for finding subgraphs with large density [11]. Other density measures are also extensively studied. For example, Goldberg [8] introduced the average degree as the density measure of a set S, that is, $d(S) = \frac{e(S,S)}{|S|}$. Though both conductance and the density provide us good measures to study the communities of the networks, they do not give us any information on the bipartiteness of these subgraphs, which is the main motivation of the paper.

As mentioned above, the measure we are using here was introduced by Trevisan, who found its deep connections with the Max Cut problem, the Cheeger inequality and the Geomans-Williamson Relaxation [20]. Soto [16] and Kale and Seshadhri [10] gave further analysis on the quantity that is related to the bipartiteness ratio. Both of their work are motivated by designing approximation algorithms for Max Cut.

In section 2, we give the basic definitions of the problem and some processes that will be used in our algorithm. Then we give our local algorithm and main theorem in Section 3. In section 4, we give the proof of our main theorem.

2 Preliminaries

Let $G = (V, E)$ be an undirected weighted graph. Let A denote the adjacency matrix of the graph such that $A_{u,v}$ is the weight of edge (u, v). We let d_v denote the (weighted) degree of vertex v. Let D denote the diagonal degree matrix of

G such that $D_{u,u} = d_u$ and $D_{u,v} = 0$ for $u \neq v$. Let $\mathcal{L} = I - D^{-1/2}AD^{-1/2}$ be the *normalized Laplacian* (or just Laplacian) of the graph. It is well known that $\mathcal{L}$ is a positive semi-definite (PSD, for short) matrix; that is, $\mathcal{L}$ is a real symmetric matrix and all its eigenvalues are non-negative. Let Δ denote the maximum degree of G. We define the volume $\mathrm{vol}(S)$ of a subset S to be the sum of degrees of the vertices in S, that is, $\mathrm{vol}(S) = \sum_{v \in S} d_v$. For any two vertex sets L and R, let $e(L, R)$ denote the number of edges between L and R. We define $\mathbf{1}_v$ to be the indicator vector of vertex v. In the following, we will let S denote subgraphs induced on the vertex set S and also let $S = (L, R)$ denote the subgraphs induces on the $S = L \cup R$. For a vector x, we let $\|x\|$ denote its Euclid norm and let $\mathrm{supp}(x)$ denote the support of it (the set of vertices on which x is non-zero).

Definition 1. *For any subgraph S and a partition (L, R) of S, that is, $L \cup R = S$ and $L \cap R = \emptyset$, we define the bipartiteness ratio $\beta(S_{L,R})$ of S under partition (L, R) by*

$$\beta(S_{L,R}) = \frac{2e(L, L) + 2e(R, R) + e(S, V \backslash S)}{vol(S)}.$$

We define the bipartiteness ratio $\beta(S)$ of the subgraph S to be the minimum value of $\beta(S_{L,R})$ over all its possible partitions (L, R), that is

$$\beta(S) = \min_{(L,R)\ partion\ of\ S} \beta(S_{L,R});$$

and define the bipartiteness ratio of the graph $\beta(G)$ to be the minimum value of $\beta(S)$ over all induced subgraphs in G, that is

$$\beta(G) = \min_{S} \beta(S).$$

When it is clear, we will use $\beta(L, R)$ to denote $\beta(S_{L,R})$. Our algorithm for finding subgraphs with small bipartiteness ratio is based on the power method for the largest eigenvector of a matrix. This method is also the base of many other local algorithms. We start from a vector x and iteratively multiply the Laplacian $\mathcal{L}$. We will then make use of these vectors to find the subgraphs with good properties. To guarantee that our algorithm is local, instead of doing the dense matrix vector multiplication, in each step, we will only keep track of the set of vertices u whose value is larger then a certain threshold. We will use the following truncated process as defined in [2].

Definition 2. *1. Given a vector x and a nonnegative real number ϵ, the truncated vector is*

$$[x]_\epsilon(u) = \begin{cases} x(u) & \text{if } |x(u)| \geq \epsilon \|x\| \\ 0 & \text{otherwise} \end{cases}$$

2. Given a vector $x = x_0$ and a set of real numbers $\epsilon_t \in [0, 1]$ for $t \leq T$, the truncated process with staring vector x_0 and parameters $\{\epsilon_t\}$ is defined to be the process that generates a sequence of vectors $x_0, \cdots, x_T$ such that $x_{t+1} = [x_t \mathcal{L}]_{\epsilon_{t+1}}$.

Note that for a given vector x, since the absolute value of $[x]_\epsilon$ is at least $\epsilon\|x\|$ whenever it is nonzero, and $\|x\|^2 \geq \|[x]_\epsilon\|^2$, we have that the number of nonzero entries in $[x]_\epsilon$ is at most $1/\epsilon^2$. That is, $|\text{supp}([x]_\epsilon)| \leq 1/\epsilon^2$.

After we get a set of vectors $x_0, \cdots, x_T$ of the truncated process, we will perform the following *sweep* process to produce subgraphs for each x_t.

Definition 3. *Given a vector $x \in \mathbb{R}^V$ such that $|supp(x)| = s$, the sweep process over vector x is defined to be the following process:*

1. *Order the vertices so that $\frac{|x(v_1)|}{\sqrt{d_{v_1}}} \geq \frac{|x(v_2)|}{\sqrt{d_{v_2}}} \geq \cdots \geq \frac{|x(v_s)|}{\sqrt{d_{v_s}}}$.*
2. *For each $i \leq s$, define $L_i = \{v_j : x(v_j) > 0 \, and \, j \leq i\}$ and $R_i = \{v_j : x(v_j) \leq 0 \, and \, j \leq i\}$ and compute the bipartiteness ratio of the subgraph $S_i = (L_i, R_i)$.*
3. *Output the subgraph $S_m = (L_m, R_m)$ that achieves the minimum bipartiteness ratio among all the s subgraphs. Let $\beta(x) = \beta(L_m, R_m)$.*

3 Description of the Algorithm and the Main Theorem

Now we describe our algorithm as follows.

`FindDenseBipartite`(v, s, θ)
Input: A vertex v, a target volume s and a target bipartiteness ratio $\theta < 1/4$.
Output: A subgraph (X, Y).

1. Let $x_0 = \frac{\mathbf{1}_v}{\sqrt{d_v}}$, $T = \log_{2-4\theta}(8s)$, and $\epsilon_t = (2-4\theta)^{t/2}/\sqrt{8s}$.
2. Compute $x_1, \cdots, x_T$ of the truncated process with starting vector x_0 and parameters $\epsilon_1, \cdots, \epsilon_T$.
3. For each time $t \leq T$, sweep over x_t and find the subgraph (X_t, Y_t) such that $\beta(X_t, Y_t) = \beta(x_t)$. Output the subgraph with the smallest bipartiteness ratio among all such pairs.

Our main theorem about the algorithm is the following.

Theorem 1. *If $S = (L, U)$ is a subgraph with bipartiteness ratio $\beta(S_{L,U}) \leq \theta$, then there exists a subset $S_\theta \subseteq S$ such that*

1. *$vol(S_\theta) \geq vol(S)/9$,*
2. *for any $v \in S_\theta$, and $s \geq vol(S)$, the algorithm* `FindDenseBipartite`(v, s, θ) *outputs a subset (X, Y) satisfying that $\beta(X, Y) \leq 2\sqrt{2\theta}$.*

Remark: we can make the bound condition on the bipartiteness ratio $\theta < 1/4$ be $\theta < 1 - \delta$, for any constant δ smaller than 1, just with a different (constant fraction) bound on vol(S_θ).

The proof of Theorem 1 is given in Section 4. Roughly speaking, we will first give the spectral algorithm of Trevisan [20] for the bipartiteness ratio of graph G. We give an alternative proof of the approximation performance of this algorithm and show that under certain conditions, a vector can be used to find subgraphs

with small bipartiteness ratio. Then we will show that there exists a large subset of "good" starting vertices so that the truncated process from a scaled indicator vector of such a vertex will produce a vector that satisfies the conditions of the former spectral algorithm, and thus finish the proof.

In the remaining of this section, we bound the running time of our algorithm.

Theorem 2. *The running time of* `FindDenseBipartite`(v, s, θ) *is* $O(s^2(\Delta + \log s))$.

Proof. We note that in each step t of the truncated process, the number of vertices in the support $\mathrm{supp}(x_t)$ of x_t is at most $1/\epsilon_t^2$. The running time of computing $x_t\mathcal{L}$ is bounded by the volume of the degrees of the vertices in $\mathrm{supp}(x_t)$, which is at most $O(\Delta/\epsilon_t^2) = O(\Delta s^2(2-4\theta)^{-t})$.

The running time of the whole truncated process is thus $\sum_{t=0}^{T} O(\Delta s^2(2-4\theta)^{-t}) = O(\Delta s^2)$.

Finally, the computation of the sweep process might require sorting the vectors in x_t, which could take time $O(|\mathrm{supp}(x_t)| \log |\mathrm{supp}(x_t)|) = O(s^2 \log s(2-4\theta)^{-t})$. Thus, the running time of the whole sweep process is $\sum_{t=0}^{T} O(s^2 \log s(2-4\theta)^{-t}) = O(s^2 \log s)$.

Thus, the running time of `FindDenseBipartite` is bounded by $O(s^2(\Delta + \log s))$.

4 Analysis of the Local Algorithm

4.1 A Spectral Algorithm for Finding Subgraphs with Small Bipartiteness Ratio

In this section, we show that under certain conditions on a vector x, the sweep over x will produce a good subgraph with low bipartiteness ratio, which is proved by Trevisan [20], and further analyzed by Soto [16] and Kale and Seshadhri [10]. The result is given in the following Lemma 1. Here, we give a self-contained proof that is somewhat different from the previous proofs. In fact, former proofs of the lemma all proceed by designing and analyzing a probabilistic algorithm. Instead, we prove the lemma by directly analyzing the deterministic version of the algorithm, which provides us more insight on the combinatorial property of the bipartiteness ratio and may be of independent interest.

Lemma 1. *For any graph G and $\theta < 1/4$, if there exists a vector $x \in \mathbb{R}^V$ such that $x\mathcal{L}x^T \geq (2-4\theta)\|x\|^2$, then the sweep over x produces a subgraph (X, Y) with bipartiteness ratio $\beta(X, Y) \leq 2\sqrt{2\theta}$.*

Proof. Let $z = xD^{-1/2}$. Let $u \sim v$ denote that $(u, v) \in E$ and let $\bar{S}$ denote $V \backslash S$. By the condition of the lemma, we have that $x(2I - \mathcal{L})x^T \leq 4\theta\|x\|^2$ and thus that

$$
\begin{aligned}
4\theta &\geq \frac{x(I + D^{-1/2}AD^{-1/2})x^T}{\|x\|^2} \\
&= \frac{z(D+A)z^T}{\langle z, zD\rangle} \\
&= \frac{\sum_{u\sim v}(z(u)+z(v))^2}{\sum_{v\in V} z^2(v)d_v} \\
&= \frac{\sum_{u\sim v}(z(u)+z(v))^2 \sum_{u\sim v}(|z(u)|+|z(v)|)^2}{\sum_{v\in V} z^2(v)d_v \sum_{u\sim v}(|z(u)|+|z(v)|)^2} \\
&\geq \frac{(\sum_{u\sim v}|z(u)+z(v)|(|z(u)|+|z(v)|))^2}{2(\sum_{v\in V} z^2(v)d_v)^2}, \qquad (1)
\end{aligned}
$$

where the last inequality follows from the Cauchy-Schwarz inequality.

Assume the support of x has size s. We perform a sweep over x so that $\frac{|x(v_1)|}{\sqrt{d_{v_1}}} \geq \frac{|x(v_2)|}{\sqrt{d_{v_2}}} \geq \cdots \geq \frac{|x(v_s)|}{\sqrt{d_{v_s}}}$. Equivalently, we have $|z(v_1)| \geq |z(v_2)| \geq \cdots \geq |z(v_s)|$.

Let $L_i = \{v_j : j \leq i, z(v_j) > 0\}, R_i = \{v_j : j \leq i, z(v_j) \leq 0\}$ and $S_i = L_i \cup R_i$. Recall that $\beta(x) = \min_i \beta(L_i, R_i)$. Then we have for any i, $\beta(x)\mathrm{vol}(S_i) \leq 2e(L_i, L_i) + 2e(R_i, R_i) + e(S_i, \bar{S}_i)$.

Now we consider the square root of the numerator of (1) to obtain

$$
\begin{aligned}
&\sum_{u\sim v} |z(u)+z(v)|(|z(u)|+|z(v)|) \\
&\geq \sum_{u\sim v, z(u)z(v)<0} |z^2(u) - z^2(v)| + \sum_{u\sim v, z(u)z(v)\geq 0} (z(u)+z(v))^2 \\
&\geq \sum_{\substack{i<j, v_i\sim v_j, \\ z(v_i)z(v_j)<0}} (z^2(v_i) - z^2(v_j)) + \sum_{\substack{i<j, v_i\sim v_j, \\ z(v_i)z(v_j)\geq 0}} (z^2(v_i) + z^2(v_j)) \qquad (2) \\
&= \sum_{i=1}^{s} (z^2(v_i) - z^2(v_{i+1}))(2e(L_i, L_i) + 2e(R_i, R_i) + e(S_i, \bar{S}_i)) \qquad (3) \\
&\geq \beta(x) \sum_{i=1}^{s} (z^2(v_i) - z^2(v_{i+1}))\mathrm{vol}(S_i) \\
&= \beta(x) \sum_{i=1}^{s} z^2(v_i) d_{v_i},
\end{aligned}
$$

where we define $z(v_{n+1})$ to be 0 if $s = n$. The main difficulty lies in the third equation, which can be obtained by comparing the coefficient of $z^2(v_k)$ on both sides for every $k \leq n$ and we defer the proof of it at the end. Now from the above calculations, we have that $4\theta \geq \frac{\beta(x)^2(\sum_{v\in V} z^2(v)d_v)^2}{2(\sum_{v\in V} z^2(v)d_v)^2} = \frac{\beta(x)^2}{2}$, and the lemma follows if we set $(X, Y) = (L_m, R_m)$ for which the bipartiteness ratio achieves $\beta(x)$.

Now we show that formula (2) is equivalent to formula (3). Let $\text{coef}_1(k)$ and $\text{coef}_2(k)$ be the coefficient of $z^2(v_k)$ in (2) and (3), respectively. We only need to show that for each $k \leq n$, $\text{coef}_1(k) = \text{coef}_2(k)$. Assume that $z(v_k) \leq 0$. The case when $z(v_k) > 0$ is similar.

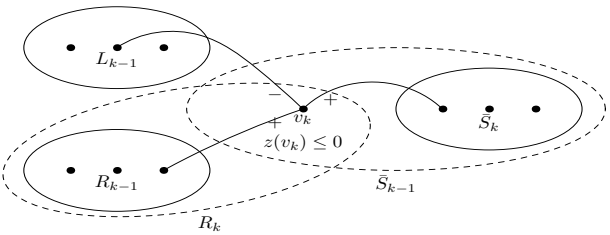


Fig. 1. The case when $z(v_k) \leq 0$. The sign on an edge denotes whether it contributes 1 or -1 to the coefficient $\text{coef}_1(k)$ of $z^2(v_k)$ in (2).

By definition and our assumption that $z(v_k) \leq 0$, we know that $L_{k-1} = L_k$ and $R_k = R_{k-1} \cup \{v_k\}$ (see Figure 1). It is easy to see that only edges incident to vertex v_k can contribute to $\text{coef}_1(k)$. More specifically, for each edge $u \sim v_k$, if $u \in R_{k-1} \cup \bar{S}_k$, it contributes 1 to $\text{coef}_1(k)$ and if $u \in L_{k-1}$, it contributes -1 to $\text{coef}_1(k)$. Totally, we have $\text{coef}_1(k) = e(\{v_k\}, R_{k-1}) + e(\{v_k\}, \bar{S}_k) - e(\{v_k\}, L_{k-1})$.

On the other hand, from (3), we can get that

$$\begin{aligned}
\text{coef}_2(k) &= (2e(L_k, L_k) + 2e(R_k, R_k) + e(S_k, \bar{S}_k)) \\
&\quad -(2e(L_{k-1}, L_{k-1}) + 2e(R_{k-1}, R_{k-1}) + e(S_{k-1}, \bar{S}_{k-1})) \\
&= 2e(\{v_k\}, R_{k-1}) + e(S_{k-1}, \bar{S}_k) + e(\{v_k\}, \bar{S}_k) \\
&\quad -e(S_{k-1}, \{v_k\}) - e(S_{k-1}, \bar{S}_k) \\
&= 2e(\{v_k\}, R_{k-1}) + e(\{v_k\}, \bar{S}_k) - e(S_{k-1}, \{v_k\}) \\
&= e(\{v_k\}, R_{k-1}) + e(\{v_k\}, \bar{S}_k) - e(\{v_k\}, L_{k-1}) \\
&= \text{coef}_1(k).
\end{aligned}$$

This completes the proof.

4.2 The Abundance of Good Starting Vertices

We now show that for any given subgraph $S = (L, R)$ with small bipartiteness ratio θ, there exists a large subset $S_\theta \subseteq S$ of "good" vertices, such that for any $v \in S_\theta$, the truncated process with starting vector $\mathbf{1}_v/\sqrt{d_v}$ produces a vector $x \in \mathbb{R}^V$ that satisfies the condition of Lemma 1.

We will consider the normalized Laplacian $\mathcal{L}_S$ of the subgraph S. Here, we extend the dimension of $\mathcal{L}_S$ to $|V|$ by adding the corresponding zero entries. Note that $\mathcal{L}_S$ is a submatrix of $\mathcal{L} = \mathcal{L}_G$ restricted on the vertex set S. In particular, $\mathcal{L} - \mathcal{L}_S$ is still positive semidefinite. So $x\mathcal{L}x^T \geq x\mathcal{L}_S x^T$ holds for all $x \in \mathbb{R}^V$. Let

$l = |S|$ and $2 = \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_l = \lambda_{l+1} = \cdots = \lambda_n = 0$ be the eigenvalues of $\mathcal{L}_S$ and let $\mu_1, \mu_2, \cdots, \mu_l, \mu_{l+1}, \cdots, \mu_n$ be the corresponding orthonormal left eigenvectors. By the definition of $\mathcal{L}_S$, we can assume that for all i such that $i \leq l$, the support of μ_i's are contained in S, and for all i such that $l < i \leq n$, the support of μ_i's are contained in $V - S$. It is easy to see that for $i \leq l$, the vectors obtained by restricting μ_i's on S form an orthonormal basis of $\mathbb{R}^S$. Any vector $x \in \mathbb{R}^V$ can be expressed in terms of the eigenvectors of $\mathcal{L}_S$ so that $x = \sum_{k=1}^{n} \alpha_k \mu_k$. For an integer m such that $1 \leq m \leq n$, define the m-norm $\|x\|_m$ of x to be the length of the projection of x onto the subspace spanned by the first m eigenvectors; that is, $\|x\|_m = \sqrt{\sum_{k \leq m} \alpha_k^2}$. It is well known that $\|x\|_m$ is a norm [5]. (Also note that $\|x\| = \|x\|_n$.) For any nonnegative number $\epsilon < 1$, let $h_\epsilon = \max\{k : \lambda_k \geq 2 - \epsilon\}$. Note that $h_\epsilon \leq l$. So, $\|x\|_{h_\epsilon} \leq \|x\|_l$.

To show there are many "good" vertices, we show in the following lemma that there is a large subset S_θ of vertices whose h_ϵ-norm is large, for any $\epsilon \geq \theta$. We will see in Section 4 that the algorithm starting from a vertex in S_θ is guaranteed to produce a vector satisfying the condition of Lemma 1 in $T = O(\log s)$ steps. In the following, when the value ϵ is clear, we will abbreviate h_ϵ as h.

Lemma 2. *Let $\epsilon = 4\theta$. If $S = (L, R)$ is a subgraph with bipartiteness ratio $\beta(L, R) \leq \theta$, then there exists a subset $S_\theta \subseteq S$ satisfying that*

1. *$vol(S_\theta) \geq vol(S)/9$, and*
2. *for any $v \in S_\theta$, $\|\mathbf{1}_v/\sqrt{d_v}\|_h \geq 1/\sqrt{8vol(S)}$.*

Proof. Define a vector ψ as follows: $\psi(v) = \sqrt{d_v}/\mathrm{vol}(S)$ if $v \in L$, $\psi(v) = -\sqrt{d_v}/\mathrm{vol}(S)$ if $v \in R$ and $\psi(v) = 0$ otherwise. Then, we have

$$\begin{aligned}\psi(2I - \mathcal{L})\psi^T = \psi(I + D^{-1/2}AD^{-1/2})\psi^T &= \sum_{u \sim v} (\psi(u)/\sqrt{d_u} + \psi(v)/\sqrt{d_v})^2 \\ &= \frac{4e(L,L) + 4e(R,R) + e(S, V \backslash S)}{\mathrm{vol}(S)^2} \\ &\leq 2\theta/\mathrm{vol}(S)\end{aligned}$$

Now let $\psi = \sum_{k=1}^{n} \alpha_i \mu_i$, then $\|\psi\|^2 = 1/\mathrm{vol}(S) = \sum_{k=1}^{n} \alpha_i^2$, and

$$\begin{aligned}\psi(2I - \mathcal{L})\psi^T = \frac{2}{\mathrm{vol}(S)} - \sum_{k=1}^{n} \lambda_k \alpha_k^2 &\geq \frac{2}{\mathrm{vol}(S)} - 2\sum_{i \leq h} \alpha_k^2 - (2 - \epsilon)\sum_{k > h} \alpha_k^2 \\ &= \frac{2}{\mathrm{vol}(S)} - 2\|\psi\|_h^2 - (2 - \epsilon)(\frac{1}{\mathrm{vol}(S)} - \|\psi\|_h^2).\end{aligned}$$

From the above bounds, we can get $\|\psi\|_h^2 \geq \frac{\epsilon - 2\theta}{\epsilon \mathrm{vol}(S)} = \frac{1}{2\mathrm{vol}(S)}$.

We now define $T = \{v \in S : \|\mathbf{1}_v/\sqrt{d_v}\|^2 < \frac{1}{8\mathrm{vol}(S)}\}$. Assume that $\mathrm{vol}(T) \geq 8\mathrm{vol}(S)/9$, we will derive a contradiction.

Define a vector η as follows: $\eta(v) = \sqrt{d_v}/\mathrm{vol}(T)$ if $v \in L \cap T$, $\eta(v) = -\sqrt{d_v}/\mathrm{vol}(T)$ if $v \in R \cap T$ and $\eta(v) = 0$ otherwise. Then, we have

$$\|\eta\|_h^2 = \|\sum_{v \in T} \frac{d_v}{\mathrm{vol}(T)} \cdot \frac{\mathbf{1}_v}{\sqrt{d_v}}\|_h^2 \leq \sum_{v \in T} \frac{d_v}{\mathrm{vol}(T)} \cdot \|\frac{\mathbf{1}_v}{\sqrt{d_v}}\|_h^2 < \frac{1}{8\mathrm{vol}(S)},$$

where the second inequality follows by the Jensen's inequality.

To get a lower bound of $\|\eta\|_h$, we first get an upper bound of $\|\eta - \psi\|_h$.

$$\|\psi - \eta\|_h^2 \leq \|\psi - \eta\|^2 = \sum_{v \in T} (\frac{\sqrt{d_v}}{\text{vol}(T)} - \frac{\sqrt{d_v}}{\text{vol}(S)})^2 + \sum_{v \in S \setminus T} (\frac{\sqrt{d_v}}{\text{vol}(S)})^2 = \frac{1}{\text{vol}(T)} - \frac{1}{\text{vol}(S)} \leq \frac{1}{8\text{vol}(S)},$$

where the last inequality follows from our assumption on $\text{vol}(T)$. By the triangle inequality, we have

$$\|\eta\|_h \geq \|\psi\|_h - \|\psi - \eta\|_h \geq \sqrt{\frac{1}{2\text{vol}(S)}} - \sqrt{\frac{1}{8\text{vol}(S)}} = \sqrt{\frac{1}{8\text{vol}(S)}},$$

which contradicts the upper bound we obtained for $\|\eta\|_h$. Therefore, we must have $\text{vol}(T) \leq 8\text{vol}(S)/9$. Let $S_\theta = S \setminus T$. We have $\text{vol}(S_\theta) \geq \text{vol}(S)/9$ and for any $v \in S_\theta$, $\|\mathbf{1}_v/\sqrt{d_v}\|_h^2 \geq \frac{1}{8\text{vol}(S)}$.

This complete the proof of the lemma.

4.3 Proof of Theorem 1

Proof. For any $S = (L, R)$ with bipartiteness ratio $\beta(L, R) \leq \theta < 1/4$, we will consider its extended Laplacian matrix $\mathcal{L}_S$ and the corresponding h-norm of vectors determined by $\mathcal{L}_S$, where h is as defined in Subsection 4.2. Let S_θ be the subset as described in Lemma 2. We show that for any s and a vertex $v \in S_\theta$, the algorithm `FindDenseBipartite`(v, s, θ) produces a subgraph (X, Y) such that $\beta(X, Y) \leq 2\sqrt{2\theta}$, where we have assumed that $s \geq \text{vol}(S)$.

Let $x_1, \cdots, x_T$ be the vectors produced by the pruned process with starting vector $\mathbf{1}_v/\sqrt{d_v}$ and parameters $\epsilon_1, \cdots, \epsilon_T$. If there exists a vector x_t such that $x_t \mathcal{L} x_t^T \geq (2 - 4\theta)\|x_t\|^2$, then by Lemma 1, we are done. Now assume that there is no such vector, that is, for all $t \leq T$, $x_t \mathcal{L} x_t^T < (2 - 4\theta)\|x_t\|^2$. We will derive a contradiction.

First, note that by the definition of Laplacian matrix $\mathcal{L}$, we have that for any $k \geq 0$, $\mathcal{L}^k - \mathcal{L}^{k+1}$ is PSD. This follows from the fact that $\mathcal{L}^k - \mathcal{L}^{k+1} = \mathcal{L}^k \cdot D^{-1/2} A D^{-1/2}$, that $D^{-1/2} A D^{-1/2}$ is a PSD matrix, and that the multiplication of two commutable PSD matrices is also PSD [9]. Therefore, by our assumption, for any $t \leq T$, we have

$$\begin{aligned} \|x_t\|^2 \leq \|x_{t-1}\mathcal{L}\|^2 = x_{t-1}\mathcal{L}^2 x_{t-1}^T \leq x_{t-1}\mathcal{L}x_{t-1}^T &\leq (2 - 4\theta)\|x_{t-1}\|^2 \\ &\leq (2 - 4\theta)^t \|x_0\|^2 \\ &\leq (2 - 4\theta)^t. \end{aligned} \tag{4}$$

Now we show that $\|x_t\|_h$ increase exponentially with t such that

$$\|x_t\|_h \geq \frac{1}{\sqrt{8\text{vol}(S)}}(2 - 4\theta)^t. \tag{5}$$

By $\|x_t\| \geq \|x_t\|_h$, this contradicts equation (4) when $t = \log_{2-4\theta}(8\mathrm{vol}(S)) \leq T$, and this completes the proof.

We will prove equation (5) by induction. When $t = 0$, it is true by the choice of v. Now let $r_t = x_{t-1}\mathcal{L} - x_t = x_{t-1}\mathcal{L} - [x_{t-1}\mathcal{L}]_{\epsilon_t}$ be the vector that is removed from the pruned process for any t and let r'_t be the vector that is equal to r_t on S, and zero elsewhere. Now recall that μ_i's are the eigenvectors of $\mathcal{L}_S$, that for all $i \leq l = |S|$, the support of μ_i is contained in S and the vectors obtained by restricting μ_i's on S form an orthonormal basis of $\mathbb{R}^S$. We have,

$$\begin{aligned}\|r_t\|_h \leq \|r_t\|_l = \sqrt{\sum_{i\leq l} |\langle r_t, \mu_i\rangle|^2} = \sqrt{\sum_{i\leq l} |\langle r'_t, \mu_i\rangle|^2} &= \|r'_t\| \\ &\leq \epsilon_t \|x_{t-1}\mathcal{L}\| \sqrt{|S|} \\ &\leq \epsilon_t \sqrt{\mathrm{vol}(S)}(2-4\theta)^{t/2},\end{aligned}$$

where the last inequality follows from inequality (4) and that $|S| \leq \mathrm{vol}(S)$.

Also note that for any $x = \sum_{k=1}^{n} \alpha_k \mu_k$, we have

$$\|x\mathcal{L}\|_h = \|x\mathcal{L}_S\|_h = \sqrt{\sum_{k\leq h} \lambda_k^2 \alpha_k^2} \geq (2-4\theta)\sqrt{\sum_{k\leq h} \alpha_k^2} = (2-4\theta)\|x\|_h,$$

where the first equation follows from the fact that $\|x\mathcal{L}\|_h = \sqrt{\sum_{i\leq h} |\langle x\mathcal{L}, \mu_i\rangle|^2}$; and that $\langle x\mathcal{L}, \mu_i\rangle = \langle x(\mathcal{L} - \mathcal{L}_S) + x\mathcal{L}_S, \mu_i\rangle = \langle x\mathcal{L}_S, \mu_i\rangle$, since the support of μ_i is contained in S and the corresponding elements in the matrix $\mathcal{L} - \mathcal{L}_S$ are all zero.

Now assume that the induction hypothesis holds for $t-1$, that is, $\|x_{t-1}\|_h \geq \frac{1}{\sqrt{8\mathrm{vol}(S)}}(2-4\theta)^{t-1}$. Then

$$\begin{aligned}\|x_t\|_h &= \|x_{t-1}\mathcal{L} - r_t\|_h \\ &\geq \|x_{t-1}\mathcal{L}\|_h - \|r_t\|_h \\ &\geq (2-4\theta)\|x_{t-1}\|_h - \epsilon_t\sqrt{\mathrm{vol}(S)}(2-4\theta)^{t/2} \\ &\geq (2-4\theta)\frac{1}{\sqrt{8\mathrm{vol}(S)}}(2-4\theta)^{t-1} - \epsilon_t\sqrt{\mathrm{vol}(S)}(2-4\theta)^{t/2} \\ &\geq \frac{1}{\sqrt{8\mathrm{vol}(S)}}(2-4\theta)^t,\end{aligned}$$

where the last inequality follows because $\epsilon_t = (2-4\theta)^{t/2}/\sqrt{8}s$ and that $s \geq \mathrm{vol}(S)$. This completes the proof.

References

[1] Abello, J., Resende, M., Sudarsky, S.: Massive Quasi-Clique Detection. In: Rajsbaum, S. (ed.) LATIN 2002. LNCS, vol. 2286, pp. 598–612. Springer, Heidelberg (2002)

[2] Andersen, R.: A local algorithm for finding dense subgraphs. ACM Trans. Algorithms 6, 60:1–60:12 (2010)
[3] Andersen, R., Chung, F., Lang, K.: Local graph partitioning using pagerank vectors. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, pp. 475–486. IEEE Computer Society, Washington, DC (2006)
[4] Andersen, R., Peres, Y.: Finding sparse cuts locally using evolving sets. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, pp. 235–244. ACM, New York (2009)
[5] Bhatia, R.: Matrix Analysis. Springer, New York (1997)
[6] Dourisboure, Y., Geraci, F., Pellegrini, M.: Extraction and classification of dense implicit communities in the web graph. ACM Transactions on the Web (TWEB) 3(2), 7 (2009)
[7] Gibson, D., Kumar, R., Tomkins, A.: Discovering large dense subgraphs in massive graphs. In: Proceedings of the 31st International Conference on Very Large Data Bases, pp. 721–732. VLDB Endowment (2005)
[8] Goldberg, A.V.: Finding a maximum density subgraph. Tech. rep., Berkeley, CA, USA (1984)
[9] Horn, R.A., Johnson, C.R.: Matrix Analysis. Cambridge University Press (1985)
[10] Kale, S., Seshadhri, C.: Combinatorial approximation algorithms for maxcut using random walks. In: 2nd Symposium on Innovations in Computer Science, ICS 2011 (2011)
[11] Kannan, R., Vinay, V.: Analyzing the structure of large graphs. Unpublished manuscript (1999), http://research.microsoft.com/en-us/um/people/kannan/Papers/webgraph.pdf
[12] Kannan, R., Vempala, S., Vetta, A.: On clusterings: Good, bad and spectral. J. ACM 51(3), 497–515 (2004)
[13] Kumar, R., Raghavan, P., Rajagopalan, S., Tomkins, A.: Trawling the web for emerging cyber-communities. In: Proceedings of the Eighth International Conference on World Wide Web, WWW 1999, pp. 1481–1493. Elsevier North-Holland, Inc., New York (1999)
[14] Leskovec, J., Lang, K.J., Dasgupta, A., Mahoney, M.W.: Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. Internet Mathematics 6(1), 29–123 (2009)
[15] Liao, C.S., Lu, K., Baym, M., Singh, R., Berger, B.: IsoRankN: spectral methods for global alignment of multiple protein networks. Bioinformatics 25(12), 253–258 (2009)
[16] Soto, J.: Improved analysis of a max cut algorithm based on spectral partitioning. CoRR abs/0910.0504 (2009)
[17] Spielman, D.A.: Algorithms, graph theory, and linear equations. In: Proceedings of the International Congress of Mathematicians 2010, pp. 2698–2722 (2010)
[18] Spielman, D.A., Teng, S.H.: Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In: Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing, STOC 2004, pp. 81–90. ACM, New York (2004)
[19] Teng, S.H.: The Laplacian Paradigm: Emerging Algorithms for Massive Graphs. In: Kratochvíl, J., Li, A., Fiala, J., Kolman, P. (eds.) TAMC 2010. LNCS, vol. 6108, pp. 2–14. Springer, Heidelberg (2010)
[20] Trevisan, L.: Max cut and the smallest eigenvalue. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, pp. 263–272. ACM, New York (2009)

Algorithms for the Strong Chromatic Index of Halin Graphs, Distance-Hereditary Graphs and Maximal Outerplanar Graphs

Ton Kloks[1], Sheung-Hung Poon[1], Chin-Ting Ung[1], and Yue-Li Wang[2]

[1] Department of Computer Science
National Tsing Hua University, Hsinchu, Taiwan
{kloks,spoon}@cs.nthu.edu.tw
[2] Department of Information Management
National Taiwan University of Science and Technology, Taipei, Taiwan
ylwang@cs.ntust.edu.tw

Abstract. We show that there exist linear-time algorithms that compute the strong chromatic index of Halin graphs, of maximal outerplanar graphs and of distance-hereditary graphs.

Keywords: Strong chromatic index, Halin graphs, Distance-hereditary graphs, Outerplanar graphs.

1 Introduction

Definition 1. *Let $G = (V, E)$ be a graph. A* strong edge coloring *of G is a proper edge coloring such that no edge is adjacent to two edges of the same color.*

Equivalently, a strong edge coloring of G is a vertex coloring of $L(G)^2$, the square of the linegraph of G. The strong chromatic index of G is the minimal integer k such that G has a strong edge coloring with k colors. We denote the strong chromatic index of G by $s\chi'(G)$.

Recently it was shown that the strong chromatic index is bounded by

$$(2 - \epsilon)\Delta^2$$

for some $\epsilon > 0$, where Δ is the maximal degree of the graph [18].[1] Earlier, Andersen showed that the strong chromatic index of a cubic graph is at most ten [1].

Let $\mathcal{G}$ be the class of chordal graphs, or the class of cocomparability graphs, or the class of weakly chordal graphs. If $G \in \mathcal{G}$ then also $L(G)^2 \in \mathcal{G}$ and it follows that the strong chromatic index can be computed in polynomial time for these classes [3,4,5]. Also for graphs of bounded treewidth there exists a polynomial time algorithm that computes the strong chromatic index [20].[2]

[1] In their paper Molloy and Reed state that $\epsilon \geq 0.002$ when Δ is sufficiently large.
[2] This algorithm checks in $O(n(s+1)^t)$ time whether a partial k-tree has a strong edge coloring that uses at most s colors. Here, the exponent $t = 2^{4(k+1)+1}$.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 157–168, 2012.

Definition 2. *Let T be a tree without vertices of degree two. Consider a plane embedding of T and connect the leaves of T by a cycle that crosses no edges of T. A graph that is constructed in this way is called a Halin graph.*

Halin graphs have treewidth at most three. Furthermore, if G is a Halin graph of bounded degree, then also $L(G)^2$ has bounded treewidth and thus the strong chromatic index of G can be computed in linear time. Recently, Lih and Liu proved that a cubic Halin graph other than one of the two 'necklaces' Ne_2 (the complement of C_6) and Ne_4, has strong chromatic index at most 7. The two exceptions have strong chromatic index 9 and 8, respectively [17]. If T is the underlying tree of the Halin graph, and if $G \neq Ne_2$ and G is not a wheel W_n with $n \neq 0 \bmod 3$, then Hsin-Hao Lai et al show that the strong chromatic index is bounded by $s\chi'(T) + 3$ [15]. (See [21,22] for earlier results.)

If G is a Halin graph then $L(G)^2$ has bounded rankwidth. In [10] it is shown that there exists a polynomial algorithm that computes the chromatic number of graphs with bounded rankwidth, thus the strong chromatic index of Halin graphs can be computed in polynomial time. In passing, let us mention the following result. A class of graphs $\mathcal{G}$ is χ-bounded if there exists a function f such that $\chi(G) \leq f(\omega(G))$ for $G \in \mathcal{G}$. Here $\chi(G)$ is the chromatic number of G and $\omega(G)$ is the clique number of G. Recently, Dvořák and Král showed that for every k, the class of graphs with rankwidth at most k is χ-bounded [8]. Obviously, the graphs $L(G)^2$, for G in the class of Halin graphs, have a uniform χ-bound.

In Section 2 we show that there exists a linear-time algorithm that computes the strong chromatic index of Halin graphs. In Section 3 we show that there exists a linear-time algorithm that computes the strong chromatic index of distance-hereditary graphs. In Section 4 we show that there exists a linear-time algorithm that computes the strong chromatic index of maximal outerplanar graphs.

2 The Strong Chromatic Index of Halin Graphs

The following lemma is easy to check.

Lemma 1 ([15]). *Let C_n be the cycle with n vertices and let W_n be the wheel with n vertices in the cycle. Then*

$$s\chi'(C_n) = \begin{cases} 3 & \text{if } n = 0 \bmod 3 \\ 5 & \text{if } n = 5 \\ 4 & \text{otherwise} \end{cases} \qquad s\chi'(W_n) = \begin{cases} n+3 & \text{if } n = 0 \bmod 3 \\ n+5 & \text{if } n = 5 \\ n+4 & \text{otherwise.} \end{cases}$$

A double wheel is a Halin graph in which the tree T has exactly two vertices that are not leaves.

Lemma 2 ([15]). *Let W be a double wheel where x and y are the vertices of T that are not leaves. Then $s\chi'(T) = d(x) + d(y) - 1$ where $d(x)$ and $d(y)$ are the degrees of x and y. Furthermore,*

$$s\chi'(W) = \begin{cases} s\chi'(T) + 4 = 9 & \text{if } d(x) = d(y) = 3, \text{ ie if } W = \bar{C}_6 \\ s\chi'(T) + 2 = d(y) + 4 & \text{if } d(y) > d(x) = 3 \\ s\chi'(T) + 1 = d(x) + d(y) & \text{if } d(y) \geq d(x) > 3. \end{cases}$$

Let G be a Halin graph with tree T and cycle C. Then obviously,

$$s\chi'(G) \leq s\chi'(T) + s\chi'(C). \tag{1}$$

The linegraph of a tree is a claw-free blockgraph. Since a sun S_r with $r > 3$ has a claw, $L(T)$ has no induced sun S_r with $r > 3$. It follows that $L(T)^2$ is a chordal graph [16] (see also [3]; in that paper Cameron proves that $L(G)^2$ is chordal for any chordal graph G). Notice that

$$s\chi'(T) = \chi(L(T)^2) = \omega(L(T)^2) \leq 2\Delta(G) - 1 \quad \Rightarrow \quad s\chi'(G) \leq 2\Delta(G) + 4. \tag{2}$$

2.1 Cubic Halin Graphs

In this subsection we outline a simple linear-time algorithm for the cubic Halin graphs.

Theorem 1. *There exists a linear-time algorithm that computes the strong chromatic index of cubic Halin graphs.*

Proof. Let G be a cubic Halin graph with plane tree T and cycle C. Let k be a natural number. We describe a linear-time algorithm that checks if G has a strong edge coloring with at most k colors. By (2) we may assume that k is at most 10. Thus the correctness of this algorithm proves the theorem.

Root the tree T at an arbitrary leaf r of T. Consider a vertex x in T. There is a unique path P in T from r to x in T. Define the subtree T_x at x as the maximal connected subtree of T that does not contain an edge of P. If $x = r$ then $T_x = T$.

Let $H(x)$ be the subgraph of G induced by the vertices of T_x. Notice that, if $x \neq r$ then the edges of $H(x)$ that are not in T form a path $Q(x)$ of edges in C.

For $x \neq r$ define the boundary $B(x)$ of $H(x)$ as the union of the following sets of edges.

(a) The unique edge of P that is incident with x.
(b) The two edges of C that connect the path $Q(x)$ of C with the rest of C.
(c) Consider the endpoints of the edges mentioned in (a) and (b) that are in T_x. Add the remaining two edges that are incident with each of these endpoints to $B(x)$.

Thus the boundary $B(x)$ consists of at most 9 edges. The following claim is easy to check. It proves the correctness of the algorithm described below. Let e be an

edge of $H(x)$. Let f be an edge of G that is not an edge of $H(x)$. If e and f are at distance at most 1 in G then e or f is in $B(x)$.[3]

Consider all possible colorings of the edges in $B(x)$. Since $B(x)$ contains at most 9 edges and since there are at most k different colors for each edge, there are at most

$$k^9 \leq 10^9$$

different colorings of the edges in $B(x)$.

The algorithm now fills a table which gives a boolean value for each coloring of the boundary $B(x)$. This boolean value is `TRUE` if and only if the coloring of the edges in $B(x)$ extends to a strong edge coloring of $H(x)$ with at most k colors. These boolean values are computed as follows. Below, we prove the correctness by induction on the size of the subtree at x.

First consider the case where the subtree at x consists of the single vertex x. Then $x \neq r$ and x is a leaf of T. In this case $B(x)$ consists of three edges, namely the three edges that are incident with x. These are two edges of C and one edge of T. If the colors of these three edges in B are different then the boolean value is set to `TRUE`. Otherwise it is set to `FALSE`. Obviously, this is a correct assignment.

Next consider the case where x is an internal vertex of T. Then x has two children in the subtree at x. Let y and z be the two children and consider the two subtrees rooted at y and z.

The algorithm that computes the tables for each vertex x processes the subtrees in the order of increasing number of vertices. (Thus the roots of the subtrees are visited in postorder). We now assume that the tables at y and z are computed correctly and show how the table for x is computed correctly and in constant time. That is, we prove that the algorithm described below computes the table at x such that it contains a coloring of $B(x)$ with a value `TRUE` if and only if there exists an extension of this coloring to the edges of $H(x)$ and $B(x)$ such that any two different edges e and f at distance at most one in G, each one in $H(x)$ or in $B(x)$, have different colors.

Consider a coloring of the edges in the boundary $B(x)$. The boolean value in the table of x for this coloring is computed as follows. Notice that

(i) $B(y) \cap B(z)$ consists of one edge and this edge is not in $B(x)$, and
(ii) $B(x) \cap B(y)$ consists of at most four edges, namely the edge (x, y) and the three edges of $B(y)$ that are incident with one vertex of $C \cap H(y)$. Likewise, $B(x) \cap B(z)$ consists of at most four edges.

The algorithm enumerates the possible colorings of the edge in $B(y) \cap B(z)$. Colorings of $B(x)$, $B(y)$ and $B(z)$ are consistent if the intersections have the same color and the pairs of edges in

$$B(x) \cup B(y) \cup B(z)$$

[3] Two edges $e = \{a, b\}$ and $f = \{u, v\}$ in G are at distance at most one if $\{a, b\} \cap \{u, v\} \neq \varnothing$ or when all four endpoints are distinct and at least one of $\{a, b\}$ is adjacent to at least one of $\{u, v\}$. We assume that it can be checked in constant time if two edges e and f are at distance at most one. This can be achieved by using a suitable data structure.

that are at distance at most one in G have different colors. A coloring of $B(x)$ is assigned the value `TRUE` if there exist colorings of $B(y)$ and $B(z)$ such that the three colorings are consistent and $B(y)$ and $B(z)$ are assigned the value `TRUE` in the tables at y and at z respectively. Notice that the table at x is built in constant time.

Consider a coloring of $B(x)$ that is assigned the value `TRUE`. Consider colorings of the edges of $B(y)$ and $B(z)$ that are consistent with $B(x)$ and that are assigned the value `TRUE` in the tables at y and z. By induction, there exist extensions of the colorings of $B(y)$ and $B(z)$ to the edges of $H(y)$ and $H(z)$. The union of these extensions provides a k-coloring of the edges in $H(x)$.

Consider two edges e and f in $B(x) \cup B(y) \cup B(z)$. If their distance is at most one then they have different colors since the coloring of $B(x) \cup B(y) \cup B(z)$ is consistent. Let e and f be a pair of edges in $H(x)$. If they are both in $H(y)$ or both in $H(z)$ then they have different colors. Assume that e is in $H(y)$ and assume that f is not in $H(y)$. If e and f are at distance at most one, then e or f is in $B(y)$. If they are both in $B(y)$, then they have different colors, due to the consistency. Otherwise, by the induction hypothesis, they have different colors. This proves the claim on the correctness.

Finally, consider the table for the vertex x which is the unique neighbor of r in T. By the induction hypothesis, and the fact that every edge in G is either in $B(x)$ or in $H(x)$, G has a strong edge coloring with at most k colors if and only if the table at x contains a coloring of $B(x)$ with three different colors for which the boolean is set to `TRUE`.

This proves the theorem. □

Remark 1. The involved constants in this algorithm are improved considerably by the recent results of Ko-Wei Lih, Ping-Ying Tsai, et al.

2.2 Halin Graphs of General Degree

In this section we show that the strong chromatic index of general Halin graphs can be computed in linear time. The result is somewhat surprising, since there is little hope for obtaining linear-time algorithms for graphs of bounded rankwidth in general. One reason is that there is no known linear-time algorithm to compute a decomposition tree. But even when a decomposition tree is a part of the input it remains unclear whether there is a linear-time algorithm for the strong chromatic index for graphs of bounded rankwidth [8,9,10].

Theorem 2. *There exists a linear-time algorithm that computes the strong chromatic index of Halin graphs.*

Proof. The algorithm is similar to the algorithm for the cubic case. Let G be a Halin graph, let T be the underlying plane tree, and let C be the cycle that connects the leaves of T. Since $L(T)^2$ is chordal the chromatic number of $L(T)^2$ is equal to the clique number of $L(T)^2$, which is

$$s\chi'(T) = \max \{\, d(u) + d(v) - 1 \mid \{u, v\} \in E(T) \,\},$$

where $d(u)$ is the degree of u in the tree T. By Formula (1) and Lemma 1 the strong chromatic index of G is one of the six possible values[4]

$$s\chi'(T), s\chi'(T)+1, \ldots, s\chi'(T)+5.$$

Root the tree at some leaf r and consider a subtree T_x at a node x of T. Let $H(x)$ be the subgraph of G induced by the vertices of T_x. Let y and z be the two boundary vertices of $H(x)$ in C.

We distinguish the following six types of edges corresponding to $H(x)$.

1. The set of edges in T_x that are adjacent to x.
2. The edge that connects x to its parent in T.
3. The edge that connects y to its neighbor in C that is not in T_x.
4. The set of edges in $H(x)$ that have endpoint y.
5. The edge that connects z to its neighbor in C that is not in T_x.
6. The set of edges in $H(x)$ that have endpoint z.

When x is adjacent to y then we make a separate type for the edge $\{x, y\}$ and similar in the case where x is adjacent to z.

Notice that the set of edges of every type has bounded cardinality, except the first type.

Consider a 0/1-matrix M with rows indexed by the six to eight types of edges and columns indexed by the colors. A matrix entry M_{ij} is 1 if there is an edge of the row-type i that is colored with the color j and otherwise this entry is 0. Since M has at most 8 rows, the rank over $GF[2]$ of M is at most 8.

Two colorings are equivalent if there is a permutation of the colors that maps one coloring to the other one. Let $S \subseteq \{1, \ldots, 8\}$ and let $W(S)$ be the set of colors that are used by edges of type i for all $i \in S$. A class of equivalent colorings is fixed by the set of cardinalities

$$\{\ |W(S)| \mid S \subseteq \{1, \ldots, 8\}\ \}.$$

We claim that the number of equivalence classes is bounded by an absolute constant. The number of ones in the first row of M, corresponding to the first type, is the degree of x in $H(x)$. Every other row of M has at most 3 ones. This proves the claim.

Consider the union of two subtrees, say at x and x'. The algorithm considers all equivalence classes of colorings of the union, and checks, by table look-up, whether it decomposes into valid colorings of $H(x)$ and $H(x')$. An easy way to do this is as follows. First double the number of types, by distinguishing the edges of $H(x)$ and $H(x')$. Then enumerate all equivalence classes of colorings. Each equivalence class is fixed by a sequence of 2^{16} numbers, as above. By table look-up, check if an equivalence class restricts to a valid coloring for each of $H(x)$ and $H(x')$. Since this takes constant time, the algorithm runs in linear time.

This proves the theorem. □

[4] Actually, according to the recent results of Hsin-Hao Lai et al, the strong chromatic index of G is at most $s\chi'(T)+3$ except when G is a wheel or the complement of C_6 [15].

3 Distance-Hereditary Graphs

Definition 3 ([13]). *A graph G is distance hereditary if any two nonadjacent vertices in a component of any induced subgraph H are at the same distance in H as they are in the graph G.*

In other words, any two chordless paths between two nonadjacent vertices is of the same length. Distance-hereditary graphs are exactly the graphs that have rankwidth one [6]. In this section we prove that there is a linear-time algorithm that computes the strong chromatic index of distance-hereditary graphs. Distance-hereditary graphs are perfect. They are the graphs without induced gem, house, hole or domino.

Cameron et al prove that, for $k \geq 4$, if G has no induced cycles on at least k vertices then also $L(G)^2$ has no such induced cycles [5]. It follows that, if G is distance hereditary then $L(G)^2$ is perfect. Therefore, to compute the chromatic number of $L(G)^2$ it suffices to compute the clique number.

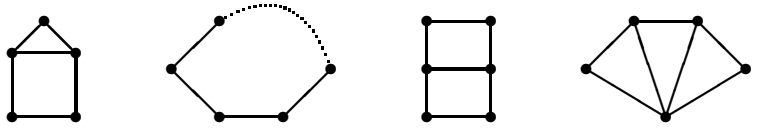


Fig. 1. A graph is distance hereditary if it has no induced house hole, domino or gem

A pendant vertex in a graph is a vertex of degree one. A twin is a pair of vertices x and y with the same open or the same closed neighborhood. When x and y are adjacent then the twin is called a true twin and otherwise it is called a false twin. A P_4 is a path with four vertices.

Theorem 3 ([2]). *A connected graph G is distance hereditary if and only if G is obtained from an edge by a sequence of the following operations.*

(a) *Creation of a pendant vertex.*
(b) *Creation of a twin.*

Lemma 3. *Let G be a graph and consider the graph G' obtained from G by creating a false twin x' of a vertex x in G. Then $L(G')^2$ is obtained from $L(G)^2$ by a series of true twin operations.*

Proof. Let $a_1, \ldots, a_s$ be the neighbors of x in G. By definition of $L(G)^2$, each edge $\{x', a_i\}$ is a true twin of the edge $\{x, a_i\}$ in $L(G')^2$. □

Definition 4. *A graph G is a* cograph *if G has no induced P_4.*

A cograph is obtained from a graph consisting of one vertex by a series of twin operations. Chordal cographs are the graphs without induced P_4 and C_4. These are often called 'trivially perfect.'

Lemma 4. *If G is a cograph then $L(G)^2$ is trivially perfect.*

Proof. A cograph with at least two vertices is either the join or the union of two cographs G_1 and G_2. Assume that G is the join of two cographs G_1 and G_2. The set of edges with one endpoint in G_1 and the other in G_2 form a clique in $L(G)^2$. Furthermore, every edge in this set is adjacent in $L(G)^2$ to every edge that is contained in G_i for $i \in \{1, 2\}$. In other words, every component of $L(G)^2$ has a universal vertex, *i.e.*, a vertex adjacent to all other vertices. The graphs that satisfy this property are exactly the graphs in which every component is the comparability graph of a tree and these are exactly the graphs without induced P_4 and C_4 [23]. □

Notice that Lemma 4 provides a linear-time algorithm for computing the strong chromatic index of cographs. A decomposition tree can be obtained in linear time [7]. Assume that G is the join of two cographs G_1 and G_2. Then every edge with both ends in G_1 is adjacent in $L(G)^2$ to every edge with both ends in G_2. Let X be the set of edges with one endpoint in G_1 and the other endpoint in G_2. By dynamic programming on the cotree, compute the clique numbers of $L(G_1)^2$ and $L(G_2)^2$. Add $|X|$ to the sum of both. If G is the union of G_1 and G_2 then the strong chromatic index of G is the maximum of the clique numbers of $L(G_1)^2$ and $L(G_2)^2$. This proves the following theorem.

Theorem 4. *There exists a linear-time algorithm that computes the strong chromatic index of cographs.*

Our linear-time algorithm for computing the clique number in $L(G)^2$ for graphs G that are distance hereditary is based on the following lemma.

Lemma 5. *If G is distance hereditary then every neighborhood of a vertex in $L(G)^2$ induces a trivially perfect graph.*

Proof. We prove the theorem by induction on the elimination ordering of G by pendant vertices and elements of twins.

First, assume that G' is obtained from G by creating a false twin x' of a vertex x in G. By Lemma 3 $L(G')^2$ is obtained from $L(G)^2$ by a series of true twin operations. In that case the claim follows easily via induction.

Secondly, consider the operation which adds a pendant vertex x', made adjacent to a vertex x in G. Let $a_1, \ldots, a_s$ be the neighbors of x in G. The adjacencies of the edge $\{x, x'\}$ in $L(G')^2$ are of the following types.

(a) All edges $\{x, a_i\}$, $i \in \{1, \ldots, s\}$. Call this set of edges X.
(b) The edges $\{a_i, a_j\} \in E(G)$, for $i, j \in \{1, \ldots, s\}$.
(c) Edges $\{a_i, u\}$, for $i \in \{1, \ldots, s\}$ and $u \in N_G(a_i) \setminus N_G[x]$.

Call two vertices in $N_G(x)$ equivalent if they have the same neighbors in $G - N_G[x]$. Since there is no house, hole, domino or gem every equivalence class is joined to or disjoint from every other equivalence class. Let H be the graph with vertex set the set of equivalence classes and edge set the pairs of equivalence

classes that are joined. Since G has no gem, the graph H has no induced P_4 and so it is a cograph. Furthermore, by Lemma 4, $L(H)^2$ is trivially perfect.

Consider the components of $G - N_G[x]$. For any two components C_1 and C_2 their neighborhoods $N_G(C_1)$ and $N_G(C_2)$ are either disjoint or ordered by inclusion. First consider the components that have a maximal neighborhood in $N_G(x)$ and remove all other components. Consider the equivalence classes defined by these components. The graph on these equivalence classes is a cograph and the square of the linegraph is a chordal cograph. Next, consider such an equivalence class Q with at least two vertices. Remove the components C of $G - N_G[x]$ with $N(C) = Q$ and consider the maximal neighborhoods that are properly contained in Q. As above, partition the vertices of Q into secondary equivalence classes. These are the equivalence classes with respect to the maximal neighborhoods that are properly contained in Q. If there are no more components with their neighborhood contained in Q then define the secondary equivalence classes as sets of single vertices. As above, for each equivalence class Q, the secondary equivalence classes form a cograph H_Q. Also, $L(H_Q)^2$ is trivially perfect. Continuation of this process defines a chordal cotree on the subgraph of $L(G)^2$ induced by the edges of types (b) and (c). Notice that the set X of edges is universal in the neighborhood of $\{x, x'\}$ in $L(G)^2$.

Finally, consider the case where G' is obtained from G by creating a true twin x' of a vertex x in G. Subdivide this operation into two steps. First create a false twin. Let G^* be the graph obtained in this manner. We proved above that every neighborhood in $L(G^*)^2$ is trivially perfect. Secondly, adding the edge $\{x, x'\}$ to G^* is similar to the operation of adding a pendant vertex. The set X of edges as described above, now consists of pairs of true twins in $L(G)^2$. The other types of adjacencies of $\{x, x'\}$, as described in (b) and (c), are the same as above.

This proves the lemma. □

Theorem 5. *There exists a linear-time algorithm that computes the strong chromatic index of distance-hereditary graphs.*

Proof. Let G be distance hereditary. Consider a rank decomposition of G of rankwidth one. This is a pair (T, f) where T is a rooted binary tree and where f is a bijection from the vertices in G to the leaves of T. Consider a subtree T_e of T rooted at some edge e of T. Define G_e as the subgraph of G induced by the vertices that are mapped to leaves in T_e. Let S_e be the set of vertices of G_e that have neighbors in $G - V(G_e)$. The set S_e is called the twinset of G_e [6]. All vertices of S_e have the same neighbors in $G - V(G_e)$.

Consider an edge e of T and let e_1 and e_2 be the two children of e in T. The graph G_e is obtained from G_{e_1} and G_{e_2} by a join or by a union of the twinsets S_{e_1} and S_{e_2}. The twinset S_e of G_e is either one of S_{e_1} and S_{e_2} or it is the union of the two [6].

Let e be an edge in T with children e_1 and e_2. Let S_1 and S_2 be the twinsets of G_{e_1} and of G_{e_2} and assume that there is a join between S_1 and S_2. Let X be the set of edges between S_1 and S_2. For $i \in \{1, 2\}$, choose a set of edges Ω_i in S_i which form a maximal clique in $L(S_i)^2$ and which maximizes the number of edges in Ω_i plus the number of edges in $G_{e_i} - S_i$ with end-vertices in $V(\Omega_i)$. Let ω_i be

the number of edges in Ω_i and let N_i be the number of edges with one endpoint in $V(\Omega_i)$. The algorithm keeps track of the maximal value of $|X|+\omega_1+\omega_2+N_1+N_2$. It is easy to see that this algorithm can be implemented to run in linear time. □

4 Maximal Outerplanar Graphs

A maximal outerplanar graph G is a ternary tree T (*i.e.*, every vertex in T has degree at most three) of triangles, where two triangles that are adjacent in the tree share an edge (see, eg [14]). When G is maximal outerplanar then G is a planar 2-tree and so $L(G)^2$ is chordal [3].

Definition 5. *An extended triangle in a maximal outerplanar graph consists of a triangle plus all the edges that are incident with some vertex of the triangle.*

The sets of edges of extended triangles form the maximal cliques of $L(G)^2$. Let ϕ be the maximal number of edges in an extended triangle. In the following theorem we prove that there exists a strong edges coloring that uses ϕ colors.

Theorem 6. *There exists a linear-time algorithm that computes the strong chromatic index of maximal outerplanar graphs.*

Proof. The algorithm colors the edges in a greedy manner as follows. First we mark one leaf node of T as the root. Then we traverse T in a breadth-first manner. When we reach a node v, we color the uncolored edges of its corresponding extended triangle τ so that the colors used for uncolored edges are different from the colors of those colored edges in τ. As the number of edges of τ is at most ϕ, ϕ colors are sufficient to do color the edges of τ. We proceed to color the edges in other extended triangles for other nodes in T via the breadth-first order. At the end of the traversal, we finish the coloring of all edges in G using only ϕ colors.

For the correctness of the algorithm observe that $\omega(L(G)^2) = \phi$ and that $L(G)^2$ is perfect. The algorithm above produces a strong edge coloring which uses ϕ colors. □

5 Concluding Remarks

For Halin graphs we tried to prove that there is an optimal strong edge-coloring such that the edges in the cycle can be colored with colors from a fixed set of constant size. If true, then this would probably improve the time bound for the strong chromatic index problem on Halin graphs.

Moser and Sikdar proved that the maximum induced matching problem on planar graphs is fixed-parameter tractable [19]. As far as we know the parameterized complexity of the strong chromatic index problem on planar graphs is open. Computing a maximum induced matching in planar graphs, or in bipartite graphs is NP-complete [4].

An example of a distance-hereditary graph G for which $L(G)^2$ is not chordal is depicted in Figure 2.

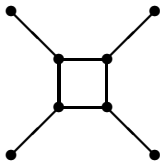


Fig. 2. A distance-hereditary graph G for which $L(G)^2$ is not chordal

Acknowledgements. Ton Kloks, Sheung-Hung Poon and Chin-Ting Ung thank the National Science Council of Taiwan for their support. Ton Kloks is supported under grant NSC 99–2218–E–007–016. Sheung-Hung Poon and Chin-Ting Ung are supported under grants NSC 99–2218–E–007–016, NSC 100–2218–E–007–007 and NSC 100–2628–E–007–020–MY3.

References

1. Andersen, L.: The Strong Chromatic Index of a Cubic Graph Is at Most 10. Discrete Math. 108, 231–252 (1992)
2. Bandelt, H., Mulder, H.: Distance-hereditary Graphs. J. Comb. Theory B 41, 182–208 (1986)
3. Cameron, K.: Induced Matchings. Discrete Appl. Math. 24, 97–102 (1989)
4. Cameron, K.: Induced Matchings in Intersection Graphs. Discrete Math. 278, 1–9 (2004)
5. Cameron, K., Sritharan, R., Tang, Y.: Finding a Maximum Induced Matching in Weakly Chordal Graphs. Discrete Math. 266, 133–142 (2003)
6. Chang, M., Hsieh, S., Chen, G.: Dynamic Programming on Distance-hereditary Graphs. In: Leong, H.-V., Jain, S., Imai, H. (eds.) ISAAC 1997. LNCS, vol. 1350, pp. 344–353. Springer, Heidelberg (1997)
7. Corneil, D., Perl, Y., Stewart, L.: A Linear Recognition Algorithm for Cographs. SIAM J. Comput. 14, 926–934 (1985)
8. Dvořák, Z., Král, D.: Classes of Graphs with Small Rank Decompositions Are χ-bounded. Manuscript on ArXiv: 1107.2161.v1 (2011).
9. Fomin, F., Golovach, P.A., Lokshtanov, D., Saurabh, S.: On the Price of Generality. In: Proceedings of the 20th Annual-SIAM Symposium on Discrete Algorithms, pp. 825–834 (2009)
10. Ganian, R., Hliněný, P.: Better Polynomial Algorithms on Graphs of Bounded Rank-Width. In: Fiala, J., Kratochvíl, J., Miller, M. (eds.) IWOCA 2009. LNCS, vol. 5874, pp. 266–277. Springer, Heidelberg (2009)
11. Halin, R.: Studies on Minimally n-connected Graphs. In: Welsh, D. (ed.) Combinatorial Mathematics and its Applications, pp. 129–136. Academic Press, London (1971)
12. Hayward, R., Spinrad, J., Sritharan, R.: Improved Algorithms for Weakly Chordal Graphs. ACM Trans. Alg. 3, 1549–6325 (2007)
13. Howorka, E.: A Characterization of Distance-hereditary Graphs. Q. J. Math. 28, 417–420 (1977)
14. Kloks, T.: Treewidth – Computations and Approximations. LNCS, vol. 842. Springer, Heidelberg (1994)

15. Lai, H., Lih, K., Tsai, P.: The Strong Chromatic Index of Halin Graphs. Discrete Math. 312, 1536–1541 (2012)
16. Laskar, R., Shier, D.: On Powers and Centers of Chordal Graphs. Discrete Appl. Math. 6, 139–147 (1983)
17. Lih, K., Liu, D.: On the Strong Chromatic Index of Cubic Halin Graphs. Appl. Math. Lett. 25, 898–901 (2012)
18. Molloy, M., Reed, B.: A Bound on the Strong Chromatic Index of a Graph. J. Comb. Theory B. 69, 103–109 (1997)
19. Moser, M., Sikdar, S.: The Parameterized Complexity of the Induced Matching Problem in Planar Graphs. Discrete Appl. Math. 157, 715–727 (2009)
20. Salavatipour, M.: A Polynomial Algorithm for Strong Edge Coloring of Partial k-trees. Discrete Appl. Math. 143, 285–291 (2004)
21. Shiu, W., Lam, P., Tam, W.: On Strong Chromatic Index of Halin Graphs. J. Comb. Math. Comb. Comput. 57, 211–222 (2006)
22. Shiu, W., Tam, W.: The Strong Chromatic Index of Complete Cubic Halin Graphs. Appl. Math. Lett. 22, 754–758 (2009)
23. Wolk, E.: A Note on "The Comparability Graph of a Tree". In: Proceedings of the American Mathematical Society, vol. 16, pp. 17–20 (1965)

On the Minimum Degree Hypergraph Problem with Subset Size Two and the Red-Blue Set Cover Problem with the Consecutive Ones Property

Biing-Feng Wang and Chih-Hsuan Li

Department of Computer Science, National Tsing Hua University
Hsinchu, Taiwan 30013, Republic of China
{bfwang,chsuan}@cs.nthu.edu.tw

Abstract. Let S be a set and let C_b (blue collection) and C_r (red collection) be two collections of subsets of S. The MDH problem is to find a subset $S' \subseteq S$ such that $S' \cap B \neq \emptyset$ for all $B \in C_b$ and $|S' \cap R| \leq k$ for all $R \in C_r$, where k is a given non-negative integer. The RBSC problem is to find a subset $S' \subseteq S$ with $S' \cap B \neq \emptyset$ for all $B \in C_b$ which minimizes $|\{R|R \in C_r, S' \cup R \neq \emptyset\}|$. In this paper, improved algorithms are proposed for the MDH problem with $k = 1$ and all sets in C_b having size two and the RBSC problem with $C_b \cup C_r$ having the consecutive ones property. For the first problem, we give an optimal $O(|S|+|C_b|+\sum_{R\in C_r}|R|)$-time algorithm, improving the previous $O(|S|+|C_b|+\sum_{R\in C_r}|R|^2)$ bound by Dom *et al.* Our improvement is obtained by presenting a new representation of a dense directed graph, which may be of independent interest. For the second problem, we give an $O(|C_b| + |C_r| \lg |S| + |S| \lg |S|)$-time algorithm, improving the previous $O(|C_b||S| + |C_r||S| + |S|^2)$ bound by Chang *et al.*

Keywords: set cover, minimum degree hypergraphs, 2-satisfiability, algorithms.

1 Introduction

Given a set S and a collection C of subsets of S, the well-known *set cover problem* is to find a minimum-size sub-collection $C' \subseteq C$ that covers S [3,6]. Due to practical considerations, numerous generalizations of the set cover problem have been defined and studied [3,4,7,8,12,14]. The minimum degree hypergraph (MDH) problem and the red-blue set cover (RBSC) problem are two examples. Let S be a set and let C_b (blue collection) and C_r (red collection) be two collections of subsets of S. The MDH problem is to find a subset $S' \subseteq S$ such that $S' \cap B \neq \emptyset$ for all $B \in C_b$ and $|S' \cap R| \leq k$ for all $R \in C_r$, where k is a given non-negative integer. The RBSC problem is to find a subset $S' \subseteq S$ with $S' \cap B \neq \emptyset$ for all $B \in C_b$ which minimizes $|R|R \in C_r, S' \cap R \neq \emptyset|$. Feder *et al.* [8] introduced the MDH problem and gave a polynomial-time approximation algorithm that has an approximation ratio of $O(\lg |S|)$. Motivated by applications

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 169–180, 2012.

in cellular networks, Kuhn *et al.* [9] studied a special case of the MDH problem, in which $C_r = C_b$, and showed that it has similar approximation properties as the classical set cover problem. The RBSC problem was introduced by Carr *et al.* [4]. They provided practical applications such as data mining and several positive and negative results concerning the polynomial-time approximability of the RBSC problem.

Since the set cover problem and the above two generalizations are NP-complete [7], one may only hope to find efficient algorithms for special cases of practical interest. A famous case is the set cover problem with the *consecutive ones property*(*C1P*), which means that the elements of S can be arranged in a linear order such that each set in C contains consecutive elements of S. This special case admits a polynomial-time solution, a fact which is utilized in many practical applications [11,12,14,16]. Recently, Dom *et al.* [7] studied the MDH and the RSBC problems with the C1P. Their study is motivated by geometric applications which may have the C1P or be "close" to the C1P [9,11,12]. They proved several NP-completeness results in case that at most one of C_b and C_r has the C1P. And, they gave efficient algorithms for the MDH and the RBSC problems with $C_b \cup C_r$ having the C1P. In addition, they gave an efficient algorithm for the MDH problem with $k = 1$ and all sets in C_b having size two.

In this paper, improved algorithms are proposed for the MDH problem with $k = 1$ and all sets in C_b having size two and the RBSC problem with $C_b \cup C_r$ having the C1P. For the first problem, Dom *et al.*'s algorithm requires $O(|S| + |C_b| + \sum_{R \in C_r} |R|^2)$ time. Their algorithm reduces the problem to an instance of the 2-SAT problem and then solves the instance by using Aspvall *et al.*'s 2-SAT algorithm [1]. We present an optimal algorithm whose running time and space are linear to the input size $|S| + |C_b| + \sum_{R \in C_r} |R|$. Our algorithm uses the same schema as Dom's algorithm. Aspvall *et al.*'s 2-SAT algorithm relies upon identifying the strongly connected components of a directed graph corresponding to the given Boolean formula. Our improvement is obtained by presenting a new representation of a directed graph. The new representation, called cluster-edge representation, may be of independent interest. It provides a more efficient way to perform depth-first search (and breadth-first search) on a directed graph with sub-graphs that are almost completely-connected. For the RBSC problem with $C_b \cup C_r$ having the C1P, Dom *et al.*'s algorithm requires $O(|C_b||C_r||S|^2)$ time. Later, Chang *et al.* [5] improved this result to $O(|C_b||S| + |C_r||S| + |S|^2)$. In this paper, we present an improved $O(|C_b| + |C_r| \lg |S| + |S| \lg |S|)$-time algorithm.

2 The MDH Problem with $k = 1$ and Subset Size Two

Assume that $k = 1$ and all sets in C_b having size two. Section 2.1 reviews Dom *et al.*'s algorithm for the MDH problem. Section 2.2 presents a new representation of a directed graph and shows how to perform depth-first search efficiently on a graph in the new representation. By using the new representation, Section 2.3 reduces the running time of Dom *et al.*'s algorithm to $O(|C_b| + |S| + \sum_{R \in C_r} |R|)$.

2.1 Dom *et al.*'s Algorithm

Let $S = \{s_1, s_2, \ldots, s_n\}$. Dom *et al.*'s algorithm reduces the problem to an instance F of the 2-SAT problem as follows.

(1) F contains a variable x_i for each element $s_i \in S$;
(2) F contains a clause $(x_i \vee x_j)$ for each $\{s_i, s_j\} \in C_b$, indicating that at least one of s_i and s_j must be selected; and
(3) F contains $d(d-1)/2$ clauses $(\neg x_{i_a} \vee \neg x_{i_b})$ for each $\{s_{i_1}, s_{i_2}, \ldots, s_{i_d}\} \in C_r$, where $1 \leq a < b \leq d$, indicating that at most one element in $\{s_{i_1}, s_{i_2}, \ldots, s_{i_d}\}$ can be selected.

Clearly, there is a one-to-one correspondence between the satisfying assignments of F and the feasible solutions to the MDH problem. Using Aspvall *et al.*'s linear-time 2-SAT algorithm [1], a satisfying assignment of F is found as follows.

Step 1. Construct a directed graph G_F as follows: for each variable x_i, add two vertices x_i and $\overline{x}_i$; and for each clause $(u \vee v)$ in F, add two edges $(\overline{u}, v)$ and $(\overline{v}, u)$. We call x_i and $\overline{x}_i$ *complements* of each other. (For example, if $\{s_1, s_2, s_5, s_9\} \in C_r$, then F contains $(\neg x_1 \vee \neg x_2)$, $(\neg x_1 \vee \neg x_5)$, $(\neg x_1 \vee \neg x_9)$, $(\neg x_2 \vee \neg x_5)$, $(\neg x_2 \vee \neg x_9)$, and $(\neg x_5 \vee \neg x_9)$. Figure 1 depicts the edges corresponding to these clauses.)
Step 2. Find strongly connected components of G_F. Then, arrange the strongly connected components in reverse topological order; that is, in an order such that if a component C_1 is before a component C_2, then no edge leads from a vertex in C_1 to a vertex in C_2. Every strong component C has a *dual component* $\overline{C}$ consisting of the complements of the vertices in C.
Step 3. Process the strongly connected components C of G_F in reverse topological order as follows: If C is marked, do nothing. Otherwise if C equals $\overline{C}$, then stop: F is unsatisfiable. Otherwise, mark C TRUE and FALSE.

After Step 3, the set of vertices in all TRUE components corresponds to a satisfying assignment of F. Let $V(G_F)$ and $E(G_F)$ be the vertex set and edge set of G_F. Topological sorting and finding strongly components can be done in $O(|V(G_F)| + |E(G_F)|)$ time [6,15]. Given a component C, its dual component can be determined in $O(1)$ time by simply finding the component containing the complement of any vertex in C. Thus, Step 3 takes $O(|S|)$ time. The construction of G_F in Step 1 is the bottleneck. The size of $V(G_F)$ is $O(|S|)$. The number of clauses in G_F is $O(|C_b| + \sum_{R \in C_r} |R|^2)$ and thus the size of $E(G_F)$ is $O(|C_b| + \sum_{R \in C_r} |R|^2)$. Consequently, Dom *et al.*'s algorithm requires $O(|S| + |C_b| + \sum_{R \in C_r} |R|^2)$ time.

2.2 Cluster-Edge Representation and Depth-First Search

Consider a directed graph $G = (V, E)$. Let X and Y be two subsets of V. Let δ be the set of vertex pairs $(x, y) \in X \times Y$ such that $(x, y) \notin E$. We represent the edges from the vertices in X to the vertices in Y by a *cluster-edge*

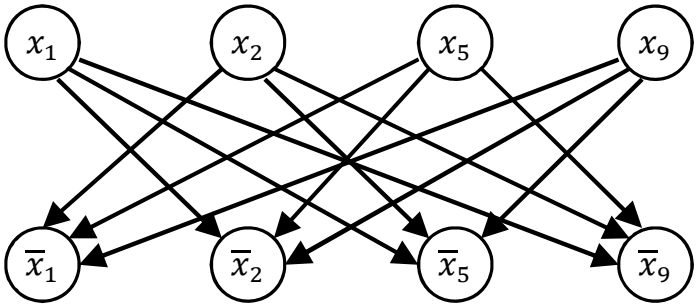


Fig. 1. Edges induced by a set $\{s_1, s_2, s_5, s_9\} \in C_r$

(X, Y, δ). For example, the edges in Figure 1 are represented by $(\{x_1, x_2, x_5, x_9\}$, $\{\overline{x}_1, \overline{x}_2, \overline{x}_5, \overline{x}_9\}$, $\{(x_1, \overline{x}_1), (x_2, \overline{x}_2), (x_5, \overline{x}_5), (x_9, \overline{x}_9)\})$. A cluster-edge (X, Y, δ) is an *almost fully-connected* cluster-edge if $|\delta| = O(|X|+|Y|)$. For ease of discussion, we use a cluster-edge only when it is almost fully-connected. A *cluster-edge representation* of a directed graph G is a pair (V, E_c), where V is the vertex set and E_c is a set of almost fully-connected cluster-edges such that the edge set of G equals the set of edges represented by all the cluster-edges in E_c. The *size* of a cluster-edge representation (V, E_c) is $O(|V| + \sum_{(X,Y,\delta)\in E_c}(|X| + |Y| + |\delta|)) = O(|V| + \sum_{(X,Y,\delta)\in E_c}(|X| + |Y|))$.

A cluster-edge representation provides a more space-efficient way to represent dense graphs. Depth-first search is a technique which has been widely used for finding solutions to problems in combinatorial theory and artificial intelligence. Let G be a directed graph given in a cluster-edge representation (V, E_c). In the following, we show that depth-first search on G can be done in time linear in the size of (V, E_c).

First, we describe the data structures used to store the graph G. Let $E_c = \{(X_1, Y_1, \delta_1), (X_2, Y_2, \delta_2), \ldots, (X_t, Y_t, \delta_t)\}$. In a depth-first search, we color the vertices to indicate their states. Each vertex is initially white, is grey when it is discovered in the search, and is black when it is finished, that is, when all vertices to which it has edges connected have been discovered completely. Consider a fixed cluster-edge (X_i, Y_i, δ_i). If the color of a vertex $y \in Y_i$ is checked by a vertex along an edge represented by (X_i, Y_i, δ_i), we say that y is *checked* via the cluster-edge (X_i, Y_i, δ_i). Our intent is to make each $y \in Y_i$ be checked via the cluster-edge (X_i, Y_i, δ_i) exactly once.

The data structures we use for (X_i, Y_i, δ_i) are described as follows. We label the vertices in Y_i by $1, 2, \ldots, |Y_i|$ and denote the vertex with label l by y_l^i. For convenience, we add two pseudo vertices y_0^i and $y_{|Y_i|+1}^i$ into Y_i. We maintain a doubly linked list L_i, which initially stores all the vertices of Y_i in order of increasing labels, and we maintain an array $Checked_i$, where $Checked_i[l] = 0$ for each $y_l^i \in Y_i$ at the beginning. In the course of depth-first search, once a vertex $y_l^i \in Y_i$ is checked via the cluster-edge (X_i, Y_i, δ_i), we set $Checked_i[l] = 1$ and delete y_l^i from L_i. For each $x \in X_i$, let $\delta_i(x)$ be the set of vertices y with

$(x, y) \in \delta_i$. Note that $y_0^i, y_{|Y_i|+1}^i \in \delta_i(x)$ for each $x \in X_i$. Define a subroutine as follows.

ExtractUnchecked(x, i): remove and return a vertex $y \notin \delta_i(x)$ in L_i for a given vertex $x \in X_i$. If all vertices in L_i are contained in $\delta_i(x)$, return $\emptyset$.

To implement ExtractUnchecked efficiently, more data structures are needed for the cluster-edge (X_i, Y_i, δ_i). For each vertex $x \in X_i$, we maintain a double linked list $D_i(x)$ and a pointer $p_i(x)$. Initially, $D_i(x)$ stores all the vertices of $\delta_i(x)$ in order of increasing labels and $p_i(x)$ points to the first node of $D_i(x)$, which stores y_0^i. We use $p_i(x).prev$ and $p_i(x).next$ to denote the predecessor and successor of the node pointed by $p_i(x)$. Let $p_i(x).label$ be the label of the vertex stored in the node pointed by $p_i(x)$. In the course of depth-first search, the following invariant is maintained.

Invariant (I). All vertices preceding y_l^i in the current list L_i belong to $\delta_i(x)$, where $l = p_i(x).label$.

We proceed to discuss how to find a vertex $y \notin \delta_i(x)$ in the current list L_i for a given vertex $x \in X_i$. Let $l_1 = p_i(x).label$ and $l_2 = p_i(x).next.label$. First, consider the case that $Checked_i[l_1] = Checked_i[l_2] = 0$, that is, both $y_{l_1}^i$ and $y_{l_2}^i$ occur in the current list L_i. According to Invariant (I), we do not need to examine all vertices preceding $y_{l_1}^i$ in L_i. Let l be the label of the vertex currently next to $y_{l_1}^i$ in L_i. Since L_i and $D_i(x)$ both store vertices in order of increasing labels, we can check whether $y_l^i \in \delta_i(x)$ by simply comparing l with l_2. If $l \neq l_2$, y_l^i is a vertex in L_i that is not contained in $\delta_i(x)$. Otherwise, in the current list L_i, there are no vertices between $y_{l_1}^i$ and $y_{l_2}^i$, and thus we move $p_i(x)$ to $p_i(x).next$ and continue to examine the vertices succeeding $y_{l_2}^i$. According to (I), all vertices in L_i are contained in $\delta_i(x)$ once $p_i(x).label = |Y_i| + 1$.

In the above discussion, we assume that $Checked_i[l_1] = Checked_i[l_2] = 0$. In case $Checked_i[l_1] = 1$, we do the following: repeatedly move $p_i(x)$ one step to the left and then delete from $D_i(x)$ the node pointed by $p_i(x).next$, until $Checked_i[l_1] = 0$. In case $Checked_i[l_2] = 1$, we do the following: repeatedly delete from $D_i](x)$ the node pointed by $p_i(x).next$, until $Checked_i[l_2] = 0$. An implementation of ExtractUnchecked is as follows.

Procedure ExtractUnchecked(x, i)
begin
1. $y_l^i \leftarrow$ FindUnchecked(x, i) /* find a vertex $y \notin \delta_i(x)$ in L_i
2. **if** $y_l^i \neq \emptyset$ **then** delete y_l^i from L_i and set $Checked_i[l] = 1$
3. **return** (y_l^i)

end

Procedure FindUnchecked(x, i)
begin
1. **while** $p_i(x).label \neq |Y_i| + 1$ **do**
2. **begin**

3. **while** $Checked_i[p_i(x).label] = 1$ **do**
4. move $p_i(x)$ to $p_i(x).prev$ and
then delete the node $p_i(x).next$ from $D_i(x)$
5. **while** $Checked_i[p_i(x).next.label] = 1$ **do**
6. delete the node $p_i(x).next$ from $D_i(x)$
7. $l_1 \leftarrow p_i(x).label; l_2 \leftarrow p_i(x).next.label$
8. $l \leftarrow$ label of the vertex currently next to $y^i_{l_1}$ in L_i
9. **if** $l \neq l_2$ **then return** (y^i_l) /* a vertex $y \notin \delta_i(x)$ in L_i is found
10. $p_i(x) \leftarrow p_i(x).next$ /* $l = l_2$, continue to examine vertices succeeding $y^i_{l_2}$
11. **end**
12. **return** $(\emptyset)$ /* all vertices in L_i are contained in $\delta_i(x)$
end

Similar to the adjacent-list representation, we maintain an array Adj of $|V|$ lists, one for each vertex in V. Consider a fixed $x \in V$. For each (X_i, Y_i, δ_i) with $x \in X_i$, there is a node of $Adj[x]$ that stores i, indicating that $x \in X_i$ and x is connected to each $y \in Y_i \setminus \delta_i(x)$.

The following procedure performs depth-first search on the graph G.

Procedure DFS(G)
begin
1. **for** each $x \in V$ **do** $color[x] \leftarrow$ WHITE
2. **for** each $x \in V$ **do**
3. **if** $color[x] =$ WHITE **then** DFS-VISIT(x)
end

Procedure DFS-VISIT(x)
begin
1. $color[x] \leftarrow$ GREY
2. **for** each cluster-edge (X_i, Y_i, δ_i) in $Adj(x)$ **do**
3. **repeat**
4. $y \leftarrow$ EXTRACTUNCHECKED(x, i)
5. **if** $y \neq \emptyset$ **then**
6. **if** $color(y) =$ WHITE **then** DFS-VISIT(y)
7. **until** $y = \emptyset$
8. $color[x] \leftarrow$ BLACK
end

The time complexity of DFS is analyzed as follows. Consider the procedure FINDUNCHECKED. Let $c(x, i)$ be the number of calls to FINDUNCHECKED(x, i) for a fixed vertex $x \in X_i$. Observe that Line 10 of FINDUNCHECKED moves $p_i(x)$ one step closer to the tail of $D_i(x)$ at each time. In addition, each iteration of the two while-loops in Lines 3 and 5 removes the node pointed by $p_i(x).next$ from $D_i(x)$. By combining these two observations, it is easy to conclude that FINDUNCHECKED spends a total of $O(|\delta_i(x)| + c(x, i))$ time on the vertex x. In case FINDUNCHECKED(x, i) returns $\emptyset$, the vertex x stops calling EXTRACTUNCHECKED(x, i). Thus, $c(x, i) - 1$ equals the number of vertices in $|Y_i|$

that are checked by x via the cluster-edge (X_i, Y_i, δ_i). And therefore, the overall running time of ExtractUnchecked spent on a fixed cluster-edge (X_i, Y_i, δ_i) is $O(\sum_{x \in X_i}(|\delta_i(x)| + c(x,i)) = O(|X_i| + |Y_i| + |\delta_i|) = O(|X_i| + |Y_i|)$. Consequently, the time complexity of DFS is $O(|V| + \sum_{(X,Y,\delta) \in E_c}(|X| + |Y|))$. We have the following.

Theorem 1. Depth-first search on a graph in a cluster-edge representation (V, E_c) can be done in time linear to the size of (V, E_c).

Note that Theorem 1 holds even when not all cluster-edges in E_c are almost fully-connected, since our analysis of DFS does not rely on the almost fully-connected assumption. When DFS-Visit(x) discovers a white vertex y (in Line 6), we call x the *predecessor* of y. In many applications, depth-first search needs to output for each vertex x its predecessor and two timestamps: the first records when x is grayed, and the second records when the search blackens x. It is easy to modify DFS-Visit to output the predecessor and timestamps of each vertex.

The strongly connected components of a directed graph G can be computed in reverse topological order by using two depth-first searches [6,15]: the first on G and the second on the transpose, G_T, of G. Let (V, E_c) be a cluster-edge representation of G. Then, (V, E_c^T) is a cluster-edge representation of G^T, where $E_c^T = \{(Y, X, \delta_T) | (X, Y, \delta) \in E_c,\ \delta^T = \{(y, x) | (x, y) \in \delta\}\}$. Consequently, we obtain the following immediately from Theorem 1.

Corollary 1. The strongly connected components of a directed graph in a cluster-edge representation can be output in reverse topological order in linear time.

2.3 A Linear Time Algorithm for the MDH Problem with $k = 1$ and Subset Size Two

Let F and G_F be defined as in Section 2.1. A set E_c of almost fully-connected cluster-edges that represents the edges of G_F is constructed as follows: for each $\{s_i, s_j\} \in C_b$, we add two cluster-edges $(\{\overline{x}_i\}, \{x_j\}, \emptyset)$ and $(\{\overline{x}_j\}, \{x_i\}, \emptyset)$ to represent the edges induced by the clause $(x_i \vee x_j)$; and for each $\{s_{i_1}, s_{i_2}, \ldots, s_{i_d}\} \in C_r$, we add a cluster-edge $(\{x_{i_a} | 1 \le a \le d\}, \{\overline{x}_{i_b} | 1 \le b \le d\}, \{(x_{i_a}, \overline{x}_{i_a}) | 1 \le a \le d\})$ to represent the edges induced by the $d(d-1)/2$ clauses $(\neg x_{i_a} \vee \neg x_{i_b})$, where $1 \le a < b \le d$. The size of (V, E_c) is $O(|S| + |C_b| + \sum_{R \in C_r} |R|)$.

Consider the algorithm in Section 2.1. By using (V, E_c) to represent the graph G_F, Steps 1 and 2 require $O(|S| + |C_b| + \sum_{R \in C_r} |R|)$ time. Step 3 takes $O(|S|)$ time. Therefore, we have the following.

Theorem 2. The MDH problem with $k = 1$ and all sets in C_b having size two can be solved optimally in linear time and space.

We remark that the set E_c we constructed for G_F satisfies the following property: for each cluster-edge $(X, Y, \delta) \in E_c$, $|\delta(x)| = 1$ for all $x \in X$. Given a cluster-edge set E_c with this wonderful property, it is easy to implement

FindUnchecked(x, i) in $O(1)$ time by simply examining the first two vertices in L_i. Our implementation in Section 2.2 can be applied to any cluster-edge representation, which may be of independent interest.

3 The RBSC Problem with the C1P

A collection C of S is said to have the *consecutive ones property* (*C1P*) if there exists a linear order $\prec$ on S such that for every set $A \in C$ and $s_i \prec s_k \prec s_j$, it holds that if $s_i, s_j \in A$, then $s_k \in A$. It is well-known that if a collection C has the C1P, the above linear order $\prec$ can be found in time linear to $O(|S| + \sum_{A \in C} |A|)$[2].

Assume that $C_b \cup C_r$ has the C1P. This section presents an improved algorithm for the RBSC problem. Without loss of generality, assume that the elements of S are already sorted in a linear order that makes $C_b \cup C_r$ possess the C1P. In addition, assume that each set $\{s_i, s_{i+1}, ...s_j\} \in C_b \cup C_r$ is given by an interval $[i, j]$. Section 3.1 reviews Chang *et al.*'s algorithm in [5]. Section 3.2 presents an improved algorithm.

3.1 Chang *et al.*'s Algorithm

Chang *et al.*'s algorithm solves the problem by a reduction to the single-source shortest path problem. An interval $[i, j] \in C_b \cup C_r$ is *hit* by an element s_k if $i \leq k \leq j$. A subset X of S is a *feasible solution* if each interval in C_b is hit by at least one element in X. Define the cost of a subset X of S, denoted by $W(X)$, to be the number of intervals in C_r that are hit by the elements in X. Then, our problem is to find a feasible solution X that minimizes the cost $W(X)$.

For ease of description, we assume that there are two dummy elements s_0 and s_{n+1} in S. We say that an interval $[i, j] \in C_b$ is *fully contained* between two elements s_p and s_q if $0 \leq p < i \leq j < q \leq n + 1$. For any two indices p and q, $0 \leq p < q \leq n + 1$, the ordered pair (p, q) is called a *feasible pair* if no interval in C_b is fully contained between s_p and s_q.

Define a directed acyclic graph H as follows:

(1) the vertex set is $V = \{v_0, v_1, \ldots, v_{n+1}\}$;
(2) the edge set is $E = \{(v_p, v_q) | 0 \leq p < q \leq n+1, (p, q)$ is a feasible pair$\}$; and
(3) each edge $(v_p, v_q) \in E$ has a length $w(p, q) = |[i, j]|[i, j] \in C_r, p < i \leq q \leq j|$, which is the number of intervals in C_r hit by s_q but not by s_p.

According to the definition of H, it is easy to see that a subset $(s_{i_1}, s_{i_2}, s_{i_3}, \ldots, s_{i_k})$ of S is a feasible solution of cost c if and only if $P = (v_0, v_{i_1}, v_{i_2}, v_{i_3}, \ldots, v_{i_k}, v_{n+1})$ is a path in H of length c, where $1 \leq i_1 < i_2 < \ldots < i_k \leq n$. Therefore, a solution to the RBSC problem can be found by computing the shortest path from v_0 to v_{n+1} in H.

Chang *et al.* had an efficient algorithm that constructs H in $O(|C_b||S| + |C_r||S| + |S|^2)$ time. A shortest path from v_0 to v_{n+1} can be easily computed in $O(|S|^2)$ time by using Dijkstra's shortest-path algorithm [6]. Therefore, we have the following.

Theorem 3. [5] The RBSC problem with $C_b \cup C_r$ having the C1P can be solved in $O(|C_b||S| + |C_r||S| + |S|^2)$ time using $O(|S|^2)$ space.

3.2 An Improved Algorithm

Essentially, our algorithm is a modified version of Chang *et al.*'s. It also solves the problem by finding a shortest path from v_0 to v_{n+1} in H. If the construction of H is necessary, since $O(|V|+|E|) = O(|S|^2)$, Chang *et al.*'s algorithm is optimal in both space and time when $|C_b|$ and $|C_r|$ are not larger than $|S|$. Our idea is to do the finding of a shortest path from v_0 to v_{n+1} without the construction of H. For each q, $1 \le q \le n+1$, let $l(q)$ be the smallest nonnegative integer p such that (p, q) is a feasible pair, and let $d(q)$ be the length of a shortest path from v_0 to v_q. Then, since H is a directed acyclic graph, a recursive formula of $d(q)$ can be given as follows: When $q = 0$, $d(q) = 0$; and, if $q \ge 1$, we have

$$\text{(R1)} \qquad d(q) = \min_{l(q) \le p \le q-1} \{d(p) + w(p, q)\}.$$

According to (R1), we need an efficient way to compute $\min_{l(q) \le p \le q-1} \{d(p) + w(p, q)\}$ for each q, $1 \le q \le n+1$.

Lemma 1. We can compute $l(q)$ in $O(|C_b|)$ time for each q, $1 \le q \le n+1$.

Proof. An interval $[i, j] \in C_b$ is fully contained between two elements s_p and s_q if and only if $i > p$ and $j < q$. Thus, each interval $[i, j] \in C_b$ sets a constraint $l(q) \ge i$ for all $q \ge j + 1$. And therefore, we can compute each $l(q)$ as follows. An auxiliary array A is used. Initially, we set $A[q] = 0$ for each q, $1 \le q \le n+1$. First, for each interval $[i, j] \in C_b$, if $A[j + 1] < i$, we set $A[j + 1] = i$, indicating that $l(q) \ge i$ for all $q \ge j + 1$. Then, we compute $l[1], l[2], \ldots, l[n + 1]$ as the prefix maximums of the numbers in A, where $l[q] = \max\{A[1], A[2], \ldots, A[q]\}$. Clearly, the above computation takes $O(|C_b|)$ time. Thus, the lemma holds. □

Let $\Delta(p, q) = \{R |\ R \in C_r,\ R$ is hit by s_q but not by $s_p\}$. By definition, $|\Delta(p, q)| = w(p, q)$. Let $\Delta^+(p, q) = \Delta(p, q) \setminus \Delta(p, q-1)$, which is the set of intervals in C_r that are contained in $\Delta(p, q)$ but not in $\Delta(p, q-1)$, and let $\Delta^-(p, q) = \Delta(p, q-1) \setminus \Delta(p, q)$, which is the set of intervals in C_r that are contained in $\Delta(p, q-1)$ but not in $\Delta(p, q)$. By definition, we have $\Delta(p, q) = (\Delta(p, q-1) \cup \Delta^+(p, q)) \setminus \Delta^-(p, q)$. Moreover, since $\Delta^+(p, q)$ is disjoint with $\Delta(p, q-1)$ and $\Delta^-(p, q)$ is a subset of $\Delta(p, q-1)$, we have $|\Delta(p, q)| = |\Delta(p, q-1)| + |\Delta^+(p, q)| - |\Delta^-(p, q)|$. Equivalently, we have $w(p, q) = w(p, q-1) + |\Delta^+(p, q)| - |\Delta^-(p, q)|$, which shows that $w(p, q)$ can be obtained by using the values of $w(p, q-1)$, $|\Delta^+(p, q)|$, and $|\Delta^-(p, q)|$.

An interval in C_r is contained in $\Delta^+(p, q)$ if and only if it is hit by s_q but not by s_p and s_{q-1}. Thus, it is easy to obtain the following.

Lemma 2. An interval $[i, j] \in C_r$ is contained in $\Delta^+(p, q)$ if and only if $p < i$ and $i = q$.

Lemma 3. An interval $[i, j] \in C_r$ is contained in $\Delta^-(p, q)$ if and only if $p < i$ and $j = q - 1$.

Proof. Consider an interval $[i, j] \in C_r$. It is contained in $\Delta(p, q-1)$ if and only if $p < i$ and $j \geq q-1$. Moreover, it is hit by s_{q-1} but not by s_q if and only if $j = q-1$. By combining these two statements, we conclude that an interval $[i, j] \in C_r$ is contained in $\Delta^-(p, q)$ if and only if $p < i$ and $j = q-1$. Thus, the lemma holds. □

For example, if $[4, 7] \in C_r$, then $[4, 7]$ is contained in $\Delta^+(p, 4)$ for any $p < 4$ by Lemma 2, and is contained in $\Delta^-(p, 8)$ for any $p < 4$ by Lemma 3.

Let $x(p, q) = d(p) + w(p, q)$, where $0 \leq p < q \leq n+1$. For each q, $0 \leq q \leq n+1$, define X_q to be the sequence $(x(0, q), x(1, q), ..., x(q-1, q))$. According to recurrence (R1), we have $d(q) = \min_{l(q) \leq p \leq q-1}\{X_q[p]\}$, where $X_q[p]$ denotes $x(p, q)$. For each q, $0 \leq q \leq n+1$, let $B(q)$ be the set of intervals in C_r starting at q and $E(q)$ be the set of intervals in C_r ending at q. Clearly, all sets $E(q)$ and $B(q)$ can be computed in $O(|C_r|)$ time. The following lemma shows how to obtain X_q from X_{q-1} by using $E(q-1)$ and $B(q)$.

Lemma 4. For each q, $1 \leq q \leq n+1$, we have

(1) $x(q-1, q) = d(q-1) + |B(q)|$, and
(2) $(x(0, q), x(1, q), \ldots, x(q-2, q))$ can be obtained from $X_{q-1} = (x(0, q-1), x(1, q-1), \ldots, x(q-2, q-1))$ as follows: add $|B(q)|$ to $x(p, q-1)$ for each $p \leq q-2$; and for each interval $[i, q-1] \in E(q-1)$, decrease all $x(p, q-1)$ with $p < i$ by one.

Due to the page limit, the proof of Lemma 4 is omitted. Our algorithm for the RBSC problem with the C1P is as follows.

Algorithm 1. The-RBSC-Problem
begin
1. compute $l(q)$ for $1 \leq q \leq n+1$ and
   compute $|B(q)|$ and $E(q)$ for $0 \leq q \leq n+1$
2. $X_0 \leftarrow$ an empty sequence
3. **for** $q \leftarrow 1$ **to** $n+1$ **do**
4. **begin**
5. update X_{q-1} into X_q
6. $d(q) \leftarrow \min_{l(q) \leq p \leq q-1}\{X_q[p]\}$
7. **end**
8. **return** $(d(n+1))$

end

According to Lemma 4, the update of X_{q-1} into X_q in Line 5 of Algorithm 1 is done by performing the following procedure.

Procedure Update(X, q)
Input: $X = X_{q-1}$
Output: X_q
begin
1. **if** $|B(q)| > 0$ **then**

2. add $|B(q)|$ to $X[p]$ for all $p \leq q-2$
3. **for** each interval $[i, q-1] \in E(q-1)$ **do**
4. subtract 1 from $X[p]$ for all $p < i$
5. $X[q] \leftarrow d(q-1) + |B(q)|$ /* append $d(q-1) + |B(q)|$ to the tail of X
6. **return** (X)

end

To implement the above procedure and the computation of $\min_{l(q) \leq p \leq q-1}\{X_q[p]\}$ in Line 6 of Algorithm 1 efficiently, we need an efficient data structure that initially represents an empty sequence X and supports a sequence of the following operations:

Append(X, y): append a number y to the tail of X;
RangeMin(X, a, b): report the minimum of $X[a], X[a+1], \ldots, X[b]$, where $0 \leq a \leq b \leq |X| - 1$ and $X[i]$ denotes the $(i+1)$th element in X; and
PrefixAdd(X, j, y): add a value y to each of $X[0], X[1], \ldots, X[j]$.

With such a data structure, Lines 2, 4, and 5 of Update are implemented, respectively, by performing PrefixAdd$(X, q-2, |B(q)|)$, PrefixAdd$(X, i-1, -1)$, and Append$(X, d(q-1) + |B(q)|)$. In addition, the computation of $d(q) = \min_{l(q) \leq p \leq q-1}\{X_q[p]\}$ in Line 6 of Algorithm 1 is implemented by performing RangeMin$(X, l(q), q-1)$. Using a segment tree [13], it is easy to support each Append, RangeMin, and PrefixAdd operation in $O(\lg n)$ time.

The time complexity of Algorithm 1 is analyzed as follows. Lines 1 and 2 take $O(|S| + |C_r| + |C_b|)$ time. Line 5 is done by a call to Update, which needs at most $|E(q-1)| + 1$ PrefixAdd and one Append operations. Line 6 is done by performing a RangeMin operation. Therefore, each iteration of the for-loop in Lines 3-7 takes $O((|E(q-1)| + 3) \lg n)$ time. Consequently, time complexity of Algorithm 1 is $O(|S| + |C_r| + |C_b| + \sum(|E(q-1)| + 3) \lg n) = O(|C_b| + |C_r| \lg |S| + |S| \lg |S|)$. We have the following.

Theorem 4. The RBSC problem with $C_b \cup C_r$ having the C1P can be solved in $O(|C_b| + |C_r| \lg |S| + |S| \lg |S|)$ time using $O(|S|)$ space.

4 Concluding Remarks

For the MDH problem with $C_r \cup C_b$ having the C1P, Dom *et al.* had an $O(|S|^3)$-time algorithm. An improved linear time algorithm is given in [10]. One direction for further study on the set cover problem and its generalizations is to investigate more special cases that are of practical interest and can be solved in polynomial time. Another direction is to investigate problems that can be solved more efficiently (in space or time) by using the cluster-edge representation.

References

1. Aspvall, B., Plass, M.F., Tarjan, R.E.: A linear-time algorithm for testing the truth of certain quantified boolean formulas. Information Processing Letters 8(3), 121–123 (1979)

2. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. Journal of Computer and System Sciences 13(3), 335–379 (1976)
3. Caprara, A., Toth, P., Fischetti, M.: Algorithms for the set covering problem. Annals of Operations Research 98, 353–371 (2000)
4. Carr, R.D., Doddi, S., Konjevod, G., Marathe, M.: On the red-blue set cover problem. In: Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 345–353 (2000)
5. Chang, M.S., Chung, H.H., Lin, C.C.: An improved algorithm for the redvblue hitting set problem with the consecutive ones property. Information Processing Letters 110(20), 845–848 (2010)
6. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. McGraw-Will (2001)
7. Dom, M., Guo, J., Niedermeier, R., Wernicke, S.: Red-blue covering problems and the consecutive ones property. Journal of Discrete Algorithms 6(3), 393–407 (2008)
8. Feder, T., Motwani, R., Zhu, A.: k-connected spanning subgraphs of low degree. Tech. Rep. TR06-041. Electronic Colloquium on Computational Complexity (2006)
9. Kuhn, F., von Rickenbach, P., Wattenhofer, R., Welzl, E., Zollinger, A.: Interference in Cellular Networks: The Minimum Membership Set Cover Problem. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 188–198. Springer, Heidelberg (2005)
10. Li, C.H., Ye, J.H., Wang, B.F.: A linear-time algorithm for the minimum degree hypergraph problem with the consecutive ones property (2012) (unpublished manuscript)
11. Mecke, S., Schöbel, A., Wagner, D.: Station location - complexity and approximation. In: 5th Workshop on Algorithmic Methods and Models for Optimization of Railways (2006)
12. Mecke, S., Wagner, D.: Solving Geometric Covering Problems by Data Reduction. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 760–771. Springer, Heidelberg (2004)
13. Preparata, F.P., Shamos, M.I.: Computational Geometry: An Introduction. Springer-Verlag New York, Inc. (1985)
14. Ruf, N., Schobel, A.: Set covering with almost consecutive ones property. Discrete Optimization 1(2), 215–228 (2004)
15. Tarjan, R.: Depth-first search and linear graph algorithms. SIAM Journal on Computing 1(2), 146–160 (1972)
16. Veinott, A.F., Wagner, H.M.: Optimal capacity scheduling. Operations Research 10(4), 518–532 (1962)

Rainbow Colouring of Split and Threshold Graphs

L. Sunil Chandran and Deepak Rajendraprasad

Department of Computer Science and Automation,
Indian Institute of Science, Bangalore -560012, India
{sunil,deepakr}@csa.iisc.ernet.in

Abstract. A *rainbow colouring* of a connected graph is a colouring of the edges of the graph, such that every pair of vertices is connected by at least one path in which no two edges are coloured the same. Such a colouring using minimum possible number of colours is called an *optimal rainbow colouring*, and the minimum number of colours required is called the *rainbow connection number* of the graph. A *Chordal Graph* is a graph in which every cycle of length more than 3 has a chord. A *Split Graph* is a chordal graph whose vertices can be partitioned into a clique and an independent set. A *threshold graph* is a split graph in which the neighbourhoods of the independent set vertices form a linear order under set inclusion. In this article, we show the following:

1. The problem of deciding whether a graph can be rainbow coloured using 3 colours remains NP-complete even when restricted to the class of split graphs. However, any split graph can be rainbow coloured in linear time using at most one more colour than the optimum.
2. For every integer $k \geq 3$, the problem of deciding whether a graph can be rainbow coloured using k colours remains NP-complete even when restricted to the class of chordal graphs.
3. For every positive integer k, threshold graphs with rainbow connection number k can be characterised based on their degree sequence alone. Further, we can optimally rainbow colour a threshold graph in linear time.

Keywords: rainbow connectivity, rainbow colouring, threshold graphs, split graphs, chordal graphs, degree sequence, approximation, complexity.

1 Introduction

Connectivity is one of the basic concepts of graph theory. It plays a fundamental role both in theoretical studies and in applications. When a network (transport, communication, social, etc) is modelled as a graph, connectivity gives a way of quantifying its robustness. This may be the reason why connectivity is possibly the problem that has been studied on the largest variety of computational models [26]. Due to the diverse application requirements and manifold

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 181–192, 2012.

theoretical interests, many variants of the connectivity problem have been studied. One typical case is when there are different possible types of connections (edges) between nodes and additional restrictions on connectivity based on the types of edges that can used in a path. In this case we can model the network as an edge-coloured graph. One natural restriction to impose on connectivity is that any two nodes should be connected by a path in which no edge of the same type (colour) occurs more than once. This is precisely the property called *rainbow connectivity*. Such a restriction for the paths can arise, for instance, in routing packets in a cellular network with transceivers that can operate in multiple frequency bands or in routing secret messages between security agencies using different handshaking passwords in different links [20] [5]. The problem was formalised in graph theoretic terms by Chartrand et al. [9] in 2008.

An *edge colouring* of a graph is a function from its edge set to the set of natural numbers. A path in an edge coloured graph with no two edges sharing the same colour is called a *rainbow path*. An edge coloured graph is said to be *rainbow connected* if every pair of vertices is connected by at least one rainbow path. Such a colouring is called a *rainbow colouring* of the graph. A rainbow colouring using minimum possible number of colours is called *optimal*. The minimum number of colours required to rainbow colour a connected graph is called its *rainbow connection number*, denoted by $rc(G)$. For example, the rainbow connection number of a complete graph is 1, that of a path is its length, that of an even cycle is its diameter, that of an odd cycle of length at least 5 is one more than its diameter, and that of a tree is its number of edges. Note that disconnected graphs cannot be rainbow coloured and hence the rainbow connection number for them is left undefined. Any connected graph can be rainbow coloured by giving distinct colours to the edges of a spanning tree of the graph. Hence the rainbow connection number of any connected graph is less than its number of vertices.

While formalising the concept of rainbow colouring, Chartrand et al. also determined the precise values of rainbow connection number for some special graphs [9]. Subsequently, there have been various investigations towards finding good upper bounds for rainbow connection number in terms of other graph parameters [4] [23] [17] [6] [2] and for many special graph classes [21] [6] [2] [3]. Behaviour of rainbow connection number in random graphs is also well studied [4] [13] [25] [11]. A basic introduction to the topic can be found in Chapter 11 of the book *Chromatic Graph Theory* by Chartrand and Zhang [8] and a survey of most of the recent results in the area can be found in the article by Li and Sun [20] and also in their forthcoming book *Rainbow Connection of Graphs* [19].

On the computational side, the problem has received relatively less attention. It was shown by Chakraborty et al. that computing the rainbow connection number of an arbitrary graph is NP-Hard [5]. In particular, it was shown that the problem of deciding whether a graph can be rainbow coloured using 2 colours is NP-complete. Later, Ananth et al. [1] complemented the result of Chakraborty et al., and now we know that for every integer $k \geq 2$, it is NP-complete to decide whether a given graph can be rainbow coloured using k

colours. Chakraborty et al., in the same article, also showed that deciding whether a given edge coloured graph is rainbow connected is NP-complete. It was then shown by Li and Li that this problem remains NP-complete even when restricted to the class of bipartite graphs [18].

On the positive side, Basavaraju et al. have demonstrated an $O(nm)$-time $(r + 3)$-factor approximation algorithm for rainbow colouring any graph with radius r [2]. Constant factor approximation algorithms for rainbow colouring Cartesian, strong and lexicographic products of non-trivial graphs are reported in [3]. Constant factor approximation algorithms for bridgeless chordal graphs, and additive approximation algorithms for interval, AT-free, threshold and circular arc graphs without pendant vertices will follow from the proofs of their upper bounds [6]. To the best of our knowledge, no efficient optimal rainbow colouring algorithm has been reported for any non-trivial subclass of graphs.

1.1 Our Results

In this article we consider the problem of rainbow colouring split graphs and a particular subclass of split graphs called threshold graphs (Definition 3). We show the following results.

1. The problem of deciding whether a graph can be rainbow coloured using 3 colours remains NP-complete even when restricted to the class of split graphs (Corollary 1). Any split graph can be rainbow coloured in linear time using at most one more colour than the optimum (Algorithm 1).

This is similar to the problem of finding the chromatic index of a graph. Though every graph with maximum degree Δ can be properly edge-coloured in $O(nm)$ time using $\Delta + 1$ colours using a constructive proof of Vizing's Theorem [22], it is NP-hard to decide whether the graph can be coloured using Δ colours [14].

No two pendant edges (Definition 2) can share the same colour in any rainbow colouring of a graph (Observation 2). The $+1$-approximation algorithm above is obtained by carefully reusing the same colours on most of the remaining edges of the graph. The hardness result is obtained by demonstrating a reduction from the problem of 3-colourability of 3-uniform hypergraphs. In fact, the technique in the reduction can be extended to show the following result for chordal graphs.

2. For every integer $k \geq 3$, the problem of deciding whether a graph can be rainbow coloured using k colours remains NP-complete even when restricted to the class of chordal graphs (Theorem 4).

Though a similar hardness result is known for deciding the rainbow connection number of general graphs, the above strengthening to chordal graphs is interesting since, unlike for general graphs, a constant factor approximation algorithm is already known for rainbow colouring chordal graphs. Chandran et al. [6] have shown that any bridgeless chordal graph can be rainbow coloured using at most $3r$ colours, where r is the radius of the graph. The proof given there is constructive and can be easily extended to a polynomial-time algorithm which will colour

any chordal graph G with b bridges and radius r using at most $3r + b$ colours. Since $\max\{r, b\}$ is easily seen to be a lower bound for $rc(G)$, this immediately gives us a 4-factor approximation algorithm.

3. For every positive integer k, threshold graphs with rainbow connection number exactly k can be characterised based on their degree sequence (Definition 2) alone (Corollary 3). Further, we can optimally rainbow colour a threshold graph in linear time (Algorithm 4).

In particular we show that if $d_1 \geq \cdots \geq d_n$ is the degree sequence of an n-vertex threshold graph G, then

$$rc(G) = \begin{cases} 1, & d_n = n-1 \\ 2, & d_n < n-1 \text{ and } \sum_{i=k}^{n} 2^{-d_i} \leq 1 \\ \max\{3, p\}, & \text{otherwise} \end{cases} \tag{1}$$

where $k = \min\{i : 1 \leq i \leq n,\ d_i \leq i-1\}$ and $p = |\{i : 1 \leq i \leq n,\ d_i = 1\}|$.

Both the characterisation and the algorithm are obtained by connecting the problem of rainbow colouring a threshold graph to that of generating a prefix-free binary code.

1.2 Preliminaries

All graphs considered in this article are finite, simple and undirected. For a graph G, we use $V(G)$ and $E(G)$ to denote its vertex set and edge set respectively. Unless mentioned otherwise, n and m will respectively denote the number of vertices and edges of the graph in consideration. The shorthand $[n]$ denotes the set $\{1, \ldots, n\}$. The cardinality of a set S is denoted by $|S|$.

Definition 1. *Let G be a connected graph. The* length *of a path is its number of edges. The* distance *between two vertices u and v in G, denoted by $d(u, v)$ is the length of a shortest path between them in G. The* eccentricity *of a vertex v is $ecc(v) := \max_{x \in V(G)} d(v, x)$. The* diameter *of G is $diam(G) := \max_{x \in V(G)} ecc(x)$ and* radius *of G is $radius(G) := \min_{x \in V(G)} ecc(x)$.*

Definition 2. *The* neighbourhood *$N(v)$ of a vertex v is the set of vertices adjacent to v but not including v. The* degree *of a vertex v is $d_v := |N(v)|$. The* degree sequence *of a graph is the non-increasing sequence of its vertex degrees. A vertex is called* pendant *if its degree is 1. An edge incident on a pendant vertex is called a* pendant edge*.*

Definition 3. *A graph G is called* chordal*, if there is no induced cycle of length greater than 3. A graph G is a* split graph*, if $V(G)$ can be partitioned into a clique and an independent set. A graph G is a* threshold graph*, if there exists a weight function $w : V(G) \to \mathbb{R}$ and a real constant t such that two vertices $u, v \in V(G)$ are adjacent if and only if $w(u) + w(v) \geq t$.*

Before getting into the main results, we note two elementary and well known observations on rainbow colouring whose proofs we omit.

Observation 1. *For every connected graph G, we have $rc(G) \geq diam(G)$.*

Observation 2. *If u and v are two pendant vertices in a connected graph G, then their incident edges get different colours in any rainbow colouring of G. In particular, if G has p pendant vertices, then $rc(G) \geq p$.*

Two of the proofs are not included in this short version. They are available in the full version uploaded to arXiv [7].

2 Split Graphs: Hardness and Approximation Algorithm

We first show that determining the rainbow connection number of a split graph is NP-hard, by demonstrating a reduction to it from the 3-colouring problem on 3-uniform hypergraphs.

Definition 4. *A* hypergraph *H is a tuple (V, E), where V is a finite set and $E \subseteq 2^V$. Elements of V and E are called vertices and (hyper-)edges respectively. The hypergraph H is called* r-uniform *if $|e| = r$ for every $e \in E$. An r-uniform hypergraph is called* complete *if $E = \{e \subset V : |e| = r\}$.*

Definition 5. *Given a hypergraph $H(V, E)$ and a colouring $C_H : V \to \mathbb{N}$, an edge is called k-*coloured *if the edge contains vertices of k different colours. An edge is called* monochromatic *if it is 1-coloured. The colouring C_H is called* proper *if no edge in E is monochromatic under C_H. The minimum number of colours required to properly colour H is called its* chromatic number *and is denoted by $\chi(H)$.*

It follows from Theorem 1.1 in [15] that it is NP-hard to decide whether an n-vertex 3-uniform hypergraph can be properly coloured using 3 colours. A reduction from this problem to a problem of computing the rainbow connection number of a split graph is illustrated in the proofs of Theorem 3 and Theorem 4.

Theorem 3. *The first problem below* (P1) *is polynomial-time reducible to the second* (P2).

P1. *Given a 3-uniform hypergraph H', decide whether $\chi(H') \leq 3$.*
P2. *Given a split graph G, decide whether $rc(G) \leq 3$.*

Proof of Theorem 3 is given in the full version of the paper [7].

Since Problem P1 is known to be NP-hard, so is Problem P2. Further, it is easy to see that the problem P2 is in NP. Hence the following corollary.

Corollary 1. *Deciding whether $rc(G) \leq 3$ remains NP-complete even when G is restricted to be in the class of split graphs.*

The reduction used in the proof of Theorem 3 can be extended to show that for every $k \geq 3$, it is NP-complete to decide whether a chordal graph can be rainbow coloured using k colours.

Theorem 4. *For any integer $k \geq 3$, the first problem below* (P1) *is polynomial-time reducible to the second* (P2).

P1. *Given a 3-uniform hypergraph H', decide whether $\chi(H') \leq 3$.*
P2. *Given a chordal graph G, decide whether $rc(G) \leq k$.*

In particular, for every integer $k \geq 3$, the problem of deciding whether $rc(G) \leq k$ remains NP-complete even when G is restricted to be in the class of chordal graphs.

Proof of Theorem 4 is also given in the full version of the paper [7].

In the wake of Corollary 1, it is unlikely that there exists a polynomial-time algorithm to optimally rainbow colour split graphs in general. In Section 3, we show that the problem is efficiently solvable when restricted to threshold graphs, which are a subclass of split graphs. Before that, we describe a linear-time (approximation) algorithm which rainbow colours any split graph using at most one colour more than the optimum (Theorem 5). First we note that it is easy to find a maximum clique in a split graph, as follows.

The vertices of a graph can be sorted according to their degrees in $O(n)$ time using a counting sort [24]. If $G([n], E)$ is a split graph with the vertices labelled so that $d_1 \geq \cdots \geq d_n$, where d_i is degree of vertex i, then $\{i \in V(G) : d_i \geq i-1\}$ is a maximum clique in G and $\{i \in V(G) : d_i \leq i-1\}$ is a maximum independent set in G [12]. Hence we can assume, if needed, that a maximum clique or a maximum independent set or an ordering of the vertices according to their degrees is given as input to our algorithms.

Algorithm 1. ColourSplitGraph

Input: $G([n], E)$, a connected split graph with a maximum clique C.
Output: A rainbow colouring $C_G : E(G) \to \{0, \ldots, \max\{p, 2\}\}$, where p is the number of pendant vertices in $V(G) \setminus C$.

1: $I \leftarrow V(G) \setminus C$ // I is an independent set in G
2: $P \leftarrow \{i \in I : d_i = 1\}$, $p \leftarrow |P|$ // P is the set of pendant vertices in I
3: $C_G(e) \leftarrow 0$, for all edges e with both end points in C.
4: $C_G(e_i) \leftarrow i$ for each pendant edge $e_1, \ldots, e_p$
5: **for** $i \in I \setminus P$ **do**
6: Let $\{e_1, \ldots, e_{d_i}\}$ be the edges incident on i
7: $C_G(e_1) \leftarrow 1$
8: $C_G(e) \leftarrow 2$ for every other edge e incident on i
9: **end for** // Now every vertex in $I \setminus P$ has a 1-coloured and a 2-coloured edge to C
10: **return** C_G

Theorem 5. *For every connected split graph G, Algorithm 7* (ColourSplitGraph) *rainbow colours G using at most $rc(G)+1$ colours. Further, the time-complexity of Algorithm 7 is $O(m)$.*

Proof. If G is a clique, then $C = V(G)$ and Algorithm 1 colours every edge of G with colour 0. This is an optimal rainbow colouring for G. Hence we can assume that G is not a clique in the following discussions. So $d := diam(G) \geq 2$. It is easy to check, by considering all pairs of non-adjacent vertices, that Algorithm 1 indeed produces a rainbow colouring of G. For example, between two vertices $v, v' \in I \setminus P$, we get a rainbow path $v \overset{1}{\text{---}} C \overset{0}{\text{---}} C \overset{2}{\text{---}} v'$. It is also evident that the algorithm uses at most $k := \max\{p+1, 3\}$ colours. By Observation 1 and Observation 2, $rc(G) \geq \max\{p, d\} \geq \max\{p, 2\} = k-1$. Hence the rainbow colouring produced by Algorithm 1 uses at most $rc(G)+1$ colours.

Further, the algorithm visits each edge exactly once and hence the time-complexity is $O(m)$.

The following bounds follow directly from Observation 1, Observation 2, and Theorem 5.

Corollary 2. *For every connected split graph G with p pendant vertices and diameter d,*

$$\max\{p, d\} \leq rc(G) \leq \max\{p+1, 3\}.$$

3 Threshold Graphs: Characterisation and Exact Algorithm

Threshold graphs form a subclass of split graphs (Observation 6b). The neighbourhoods of vertices in a maximum independent set of a threshold graph form a linear order under set inclusion (Observation 6c). We exploit this structure to give a full characterisation of rainbow connection number of threshold graphs based on degree sequences (Corollary 3). We use this characterisation to design a linear-time algorithm to optimally rainbow colour any threshold graph (Algorithm 4).

The following observations are easy to make from the definition of a threshold graph (Definition 3).

Observation 6. *Let $G([n], E)$ be a threshold graph with a weight function $w : V(G) \to \mathbb{R}$. Let the vertices be labelled so that $w(1) \geq \cdots \geq w(n)$. Then*

(a) *$d_1 \geq \cdots \geq d_n$, where d_i is the degree of vertex i.*
(b) *$I = \{i \in V(G) : d_i \leq i-1\}$ is a maximum independent set G and $V(G) \setminus I$ is a clique in G. In particular, every threshold graph is a split graph.*
(c) *$N(i) = \{1, \ldots, d_i\}$, for every $i \in I$. Thus the neighbourhoods of vertices in I form a linear order under set inclusion. Further, if G is connected, then every vertex in G is adjacent to 1.*

Definition 6. *A* binary codeword *is a finite string over the alphabet* $\{0, 1\}$ *(bits). The* length *of a codeword* b*, denoted by* $length(b)$*, is the number of bits in the string* b*. We denote the* i*-th bit of* b *by* $b(i)$*. A codeword* b_1 *is said to be a* prefix *of a codeword* b_2 *if* $length(b_1) \leq length(b_2)$ *and* $b_1(i) = b_2(i)$ *for all* $i \in \{1, \ldots, length(b_1)\}$*. A* binary code *is a set of binary codewords. A binary code* B *is called* prefix-free *if no codeword in* B *is a prefix of another codeword in* B*.*

The Kraft's Inequality [16] gives a necessary and sufficient condition for the existence of a prefix-free code for a given set of codeword lengths.

Theorem 7 (Kraft 1949 [16]). *For every prefix-free binary code* $B = \{b_1, \ldots, b_n\}$*,*

$$\sum_{i=1}^{n} 2^{-l_i} \leq 1$$

where $l_i = length(b_i)$*, and conversely, for any sequence of lengths* $l_1, \ldots, l_n$ *satisfying the above inequality, there exists a prefix-free binary code* $B = \{b_1, \ldots, b_n\}$*, with* $length(b_i) = l_i$*,* $i = 1, \ldots, n$*.*

Observation 8. *Given any sequence of lengths* $l_1 \leq \cdots \leq l_n$ *satisfying the Kraft Inequality, we can construct a prefix-free binary code* $B = \{b_1, \ldots, b_n\}$*, with* $length(b_i) = l_i$*,* $i = 1, \ldots, n$ *in time* $O\big(\sum_{i=1}^{n} l_i\big)$*. Further, we can ensure that every bit in* b_1 *is* 0*.*

Proof. A *binary tree* is a rooted tree in which every node has at most two child nodes. A node with only one child node is said to be *unsaturated*. The *level* of a node is its distance from the root. We assume that every edge from a parent to its first (second) child, if it exists, is labelled 0 (1). We can represent a prefix-free binary code by a binary tree such that (i) every codeword b_i corresponds to a leaf t_i of the binary tree at level $length(b_i)$ and (ii) the labels on the unique path from the root to a leaf will be the codeword associated with that leaf [10]. We construct a prefix-free binary code with the given length sequence by constructing the corresponding binary tree as explained below.

Create the root, and for every new node created, create its first child till we hit a node t_1 at depth l_1 for the first time. Declare t_1 as a leaf. Once we have created a leaf t_i, $i < n$, we proceed to create the next leaf as follows. Backtrack from t_i along the tree created so far towards the root till we hit the first unsaturated node. Create its second child. If the second child is at level l_{i+1}, then declare it as the leaf t_{i+1}. Else, recursively create first child till we create a node at level l_{i+1} and declare it as leaf t_{i+1}. Terminate this process once we create the leaf t_n.

The process will continue till we create all the n leaves. Otherwise, it has to be the case that every internal node in the tree got saturated by the time we created some leaf t_i, $i < n$. If we have a binary tree T with every internal node saturated, it is easy to see by an inductive argument that $\sum_{t \in L} 2^{-d_t} = 1$, where L is the set of leaves of T and d_t denotes the level of leaf t. Hence $\sum_{j=1}^{n} 2^{-l_j} >$

Algorithm 2. ColourThresholdGraph-Case1

Input: $G([n], E)$, a connected threshold graph, with $d_1 \geq \cdots \geq d_n$ and $\sum_{i=k}^{n} 2^{-d_i} \leq 1$, where d_i is the degree of vertex i and $k = \min\{i : 1 \leq i \leq n,\ d_i \leq i-1\}$.
Output: A rainbow colouring $C_G : E(G) \to \{0, 1\}$ of G.

```
I = {k, ..., n}                                   // I is a maximal independent set in G
Let B = {b_k, ..., b_n} be a prefix-free code with length(b_i) = d_i (constructed as
    mentioned in Observation 8)
for i ∈ I do
    C_G({i, j}) = b_i(j), ∀j ∈ {1, ..., d_i}
end for
for i ∈ V(G) \ I do                                                          // i < k
    C_G({i, j}) = b_k(j), ∀j ∈ {1, ..., i − 1}     // Note that length(b_k) = d_k = k − 1
end for
return C_G
```

$\sum_{j=1}^{i} 2^{-l_j} = 1$, contradicting the hypothesis that the lengths $l_1, \ldots, l_n$ satisfy the Kraft Inequality.

It follows from the construction that every bit of b_1 is 0. Since every edge in the tree constructed corresponds to a bit in at least one of the codewords returned, the total number of edges in the tree constructed is at most $\sum_{i=1}^{n} l_i$. Since each edge of the tree is traversed at most twice, the construction will be completed in time $O\big(\sum_{i=1}^{n} l_i\big)$.

Now we give a necessary and sufficient condition for 2-rainbow-colourability of a threshold graph.

Theorem 9. *For every connected threshold graph $G([n], E)$ with $d_1 \geq \cdots \geq d_n$, $rc(G) \leq 2$ if and only if*

$$\sum_{i=k}^{n} 2^{-d_i} \leq 1, \tag{2}$$

where d_i is the degree of vertex i and $k = \min\{i : 1 \leq i \leq n,\ d_i \leq i-1\}$. Further, if G satisfies Inequality (2), then Algorithm 2 (ColourThresholdGraph-Case1) gives an optimal rainbow colouring of G in $O(m)$ time.

Proof. Note that $I := \{k, \ldots, n\}$ is a maximal independent set in G (Observation 6b) and that the summation on the left hand side of Inequality (2) is over all the vertices in I. Hence $C := \{1, \ldots, k-1\}$ is a clique in G.

First we show that if $rc(G) \leq 2$, then the inequality is satisfied. Let $C_G : E(G) \to \{0, 1\}$ be a rainbow colouring of G. We can associate a codeword with each vertex $i \in I$ by reading the colours assigned by C_G to edges $\{i, c\}, c = 1, \ldots, d_i$. Since every pair $i, j \in I, d_i \leq d_j$ are non-adjacent, they need a 2-length rainbow path between them through a common neighbour $c \in \{1, \ldots, d_i\}$ (Observation 6c). This ensures that the codewords corresponding to i and j are complementary in at least one bit position. Hence the binary code formed by codewords corresponding to all the vertices in I form a prefix-free code. Hence the inequality is satisfied (by Theorem 7).

Algorithm 3. ColourThresholdGraph-Case2

Input: $G([n], E)$, a connected threshold graph, with $d_1 \geq \cdots \geq d_n$, where d_i is the degree of vertex i.

Output: A rainbow colouring $C_G : E(G) \to \{0, \ldots, \max\{p, 3\} - 1\}$ of G, where p is the number of pendant vertices in G.

1: $P \leftarrow \{i \in V(G) : d_i = 1\}$, $p \leftarrow |P|$ // P is the set of pendant vertices in G
2: $C_G(\{p_i, 1\}) \leftarrow i - 1$ for each pendant vertex $p_1, \ldots, p_p$
3: **if** $p = n - 1$ **then**
4: **return** C_G // G is a star
5: **end if**
6: $C_G(\{1, 2\}) = 0$
7: **for** $i = 3$ **to** $i = n - p$ **do**
8: $C_G(\{i, 1\}) = 1$
9: $C_G(\{i, 2\}) = 2$ // Every $v \in \{3, \ldots, n - p\}$ is adjacent to vertices 1 and 2.
10: **end for**
11: $C_G(e) = 0$ for each edge e of G not coloured so far.
12: **return** C_G

Conversely, if the inequality is satisfied, then Algorithm 2 gives a colouring C_G of $E(G)$ using at most 2 colours. We show that C_G is indeed a rainbow colouring of G. Consider any two non-adjacent vertices $i, j \in V(G)$, $i < j$. Since they are non-adjacent, either both of them are in I or otherwise j is in I and i is from the clique C such that $i > d_j$ (Since $N(j) = \{1, \ldots, d_j\}$). In the former case, $length(b_i) \geq length(b_j)$ and there exists a $v \in \{1, \ldots, d_j \leq d_i\}$ such that $b_j(v) \neq b_i(v)$ since b_j is not a prefix of b_i (They both belong to a prefix-free code B). Hence i–v–j is a rainbow path. Similarly in the latter case, $length(b_k) \geq length(b_j)$ and there exists a $v \in \{1, \ldots, d_j < i\}$ such that $b_j(v) \neq b_k(v)$ since b_j is not a prefix of b_k. Hence $C_G(\{v, j\}) \neq C_G(\{v, i\})$ and i–v–j is a rainbow path. Hence C_G is a rainbow colouring of G.

If G is not a clique, then $rc(G) \geq 2$ (Observation 1), and hence the above rainbow colouring is optimal. If G is a clique then $k = n$ and $|B| = |I| = 1$. So the single codeword b_n constructed as mentioned in Observation 8 has all the bits 0. So every edge of G is coloured using the single colour 0, which is optimal for G.

Since $\sum_{i=1}^{n} l_i = \sum_{i=1}^{n} d_i = 2m$, the prefix-free code B can be constructed in $O(m)$ time (Observation 8). Moreover, Algorithm 2 visits each edge only once. Hence the total time complexity is $O(m)$.

Now we consider the case of threshold graphs which violate Inequality (2).

Theorem 10. *For every connected threshold graph G which does not satisfy Inequality (2),*

$$rc(G) = \max\{p, 3\},$$

where p is the number of pendant vertices in G.

Further, Algorithm 3 (ColourThresholdGraph-Case2) *gives an optimal rainbow colouring of G in $O(m)$ time*

Proof. It is easy to check, by considering all pairs of non-adjacent vertices, that Algorithm 3 indeed produces a rainbow colouring of G. It is also evident that it uses at most $\max\{p, 3\}$ colours. By Observation 2 and Theorem 9, it follows hat $rc(G) \geq \max\{p, 3\}$. Hence $rc(G) = \max\{p, 3\}$ and hence the rainbow colouring produced by Algorithm 3 is optimal. Further, since Algorithm 3 visits each edge only once, its time complexity is $O(m)$.

Algorithm 4. ColourThresholdGraph

Input: $G([n], E)$, a connected threshold graph with $d_1 \geq \cdots \geq d_n$, where d_i is the degree of vertex i.
Output: An optimal rainbow colouring $C_G : E(G) \to \{0, \ldots, rc(G) - 1\}$ of G.

```
k = min{i : 1 ≤ i ≤ n, d_i ≤ i − 1}
if Σ_{i=k}^{n} 2^{−d_i} ≤ 1 then
    C_G = ColourThresholdGraph-Case1(G)
else
    C_G = ColourThresholdGraph-Case2(G)
end if
return C_G
```

Combining Theorem 9 and Theorem 10, we get a complete characterisation for threshold graphs whose rainbow connection number is k, based on its degree sequence alone. Further we can find the optimally rainbow colour every threshold graph in linear-time.

Corollary 3. *Let $G([n], E)$, be a connected threshold graph with $d_1 \geq \cdots \geq d_n$, where d_i is the degree of vertex i. Then,*

$$rc(G) = \begin{cases} 1, & \text{if } G \text{ is a clique} \\ 2, & \text{if } G \text{ is not a clique and } \sum_{i=k}^{n} 2^{-d_i} \leq 1 \\ \max\{3, p\}, & \text{otherwise,} \end{cases} \tag{3}$$

where $k = \min\{i : 1 \leq i \leq n,\ d_i \leq i - 1\}$ and $p = |\{i : 1 \leq i \leq n,\ d_i = 1\}|$.

Further, Algorithm 4 (ColourThresholdGraph) gives an optimal rainbow colouring of G in $O(m)$ time.

References

1. Ananth, P., Nasre, M., Sarpatwar, K.K.: Rainbow Connectivity: Hardness and Tractability. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011), vol. 13, pp. 241–251 (2011)
2. Basavaraju, M., Chandran, L.S., Rajendraprasad, D., Ramaswamy, A.: Rainbow connection number and radius. Preprint arXiv:1011.0620v1 (math.CO) (2010)
3. Basavaraju, M., Chandran, L.S., Rajendraprasad, D., Ramaswamy, A.: Rainbow connection number of graph power and graph products. Preprint arXiv:1104.4190v2 (math.CO) (2011)

4. Caro, Y., Lev, A., Roditty, Y., Tuza, Z., Yuster, R.: On rainbow connection. Electron. J. Combin. 15(1), Research paper 57, 13 (2008)
5. Chakraborty, S., Fischer, E., Matsliah, A., Yuster, R.: Hardness and algorithms for rainbow connection. J. Comb. Optim. 21(3), 330–347 (2011)
6. Chandran, L.S., Das, A., Rajendraprasad, D., Varma, N.M.: Rainbow connection number and connected dominating sets. Journal of Graph Theory (2011)
7. Chandran, L.S., Rajendraprasad, D.: Rainbow colouring of split and threshold graphs. Preprint arXiv:1205.1670v1 (cs.DM) (2012)
8. Chartrand, G., Zhang, P.: Chromatic Graph Theory. Chapman & Hall (2008)
9. Chartrand, G., Johns, G.L., McKeon, K.A., Zhang, P.: Rainbow connection in graphs. Math. Bohem. 133(1), 85–98 (2008)
10. Cover, T.M., Thomas, J.A.: Data Compression, pp. 103–158. John Wiley & Sons, Inc. (2005)
11. Frieze, A., Tsourakakis, C.: Rainbow connectivity of $g(n, p)$ at the connectivity threshold. Preprint arXiv:1201.4603 (2012)
12. Hammer, P.L., Simeone, B.: The splittance of a graph. Combinatorica 1(3), 275–284 (1981)
13. He, J., Liang, H.: On rainbow k-connectivity of random graphs. Preprint arXiv:1012.1942v1 (math.CO) (2010)
14. Holyer, I.: The NP-completeness of edge-coloring. SIAM Journal on Computing 10(4), 718–720 (1981)
15. Khot, S.: Hardness results for coloring 3-colorable 3-uniform hypergraphs. In: Proceedings of The 43rd Annual IEEE Symposium on Foundations of Computer Science, pp. 23–32. IEEE (2002)
16. Kraft, L.: A device for quanitizing, grouping and coding amplitude modulated pulses. Master's thesis. Electrical Engineering Department, Massachusetts Institute of Technology (1949)
17. Krivelevich, M., Yuster, R.: The rainbow connection of a graph is (at most) reciprocal to its minimum degree. J. Graph Theory 63(3), 185–191 (2010)
18. Li, S., Li, X.: Note on the complexity of determining the rainbow connectedness for bipartite graphs. Preprint arXiv:1109.5534 (2011)
19. Li, X., Sun, Y.: Rainbow Connections of Graphs. Springerbriefs in Mathematics. Springer (2012)
20. Li, X., Sun, Y.: Rainbow connections of graphs – a survey. Preprint arXiv:1101.5747v2 (math.CO) (2011)
21. Li, X., Sun, Y.: Upper bounds for the rainbow connection numbers of line graphs. Graphs and Combinatorics, 1–13 (2011), doi:10.1007/s00373-011-1034-1
22. Misra, J., Gries, D.: A constructive proof of vizing's theorem. Information Processing Letters 41(3), 131–133 (1992)
23. Schiermeyer, I.: Rainbow Connection in Graphs with Minimum Degree Three. In: Fiala, J., Kratochvíl, J., Miller, M. (eds.) IWOCA 2009. LNCS, vol. 5874, pp. 432–437. Springer, Heidelberg (2009)
24. Seward, H.H.: Information sorting in the application of electronic digital computers to business operations. Master's thesis, Digital Computer Laboratory, Massachusetts Institute of Technology (1954)
25. Shang, Y.: A sharp threshold for rainbow connection of random bipartite graphs. Int. J. Appl. Math. 24(1), 149–153 (2011)
26. Wigderson, A.: The complexity of graph connectivity. In: Mathematical Foundations of Computer Science, pp. 112–132 (1992)

Approximating the Rainbow – Better Lower and Upper Bounds

Alexandru Popa

Department of Communications & Networking,
Aalto University School of Electrical Engineering, Aalto, Finland
alexandru.popa@aalto.fi

Abstract. In this paper we study the *minimum rainbow subgraph problem*, motivated by applications in bioinformatics. The input of the problem consists of an undirected graph where each edge is coloured with one of the p possible colors. The goal is to find a subgraph of minimum order (i.e. minimum number of vertices) which has precisely one edge from each color class.

In this paper we show a $\max(\sqrt{2p}, \min_q(q + \frac{\Delta}{e^{pq^2/\Delta n}}))$-approximation algorithm using LP rounding, where Δ is the maximum degree in the input graph. In particular, this is a $\max(\sqrt{2n}, \sqrt{2\Delta \ln \Delta})$-approximation algorithm. On the other hand we prove that there exists a constant c such that the minimum rainbow subgraph problem does not have a $c \ln \Delta$-approximation, unless $\mathbf{NP} \subseteq \mathbf{TIME}(n^{O(\log \log n)})$.

1 Introduction

Motivation. An important problem in computational biology is the *pure parsimony haplotyping problem (PPH)*, introduced by Gusfield in 2003 [8]. The problem input consists in a set $\mathcal{G}$ of p genotypes (i.e. vectors with entries in $\{0, 1, 2\}$) corresponding to individuals in a population. A genotype g is explained by two haplotypes (i.e. vectors with entries in $\{0, 1\}$) h_1 and h_2 if for each entry i, either $g[i] = h_1[i] = h_2[i]$ or $g[i] = 2$ and $h_1[i] \neq h_2[i]$. For example, the genotype $g = 012$ is explained by the haplotypes $h_1 = 010$ and $h_2 = 011$ as $h_1[1] = h_2[1] = g[1] = 0$, $h_1[2] = h_2[2] = g[2] = 1$, $h_1[3] \neq h_2[3]$ and $g[3] = 2$. The goal is to find a set of haplotypes of minimum cardinality which explains the set $\mathcal{G}$ of genotypes. The positions where $g[i] = 2$ are named ambiguous positions. If the number of ambiguous positions is at most k, then the problem is termed $PPH(k)$.

Camacho et al. [4] show that the $PPH(k)$ problem for $k \leq O(\log p)$ can be reduced in polynomial time to the *minimum rainbow subgraph (MRS)* problem which we describe next. The input consists of an undirected graph G where each edge is coloured with one of the p possible colors. A rainbow subgraph $F \subseteq G$ contains precisely one edge from each color class. The goal of the problem is to find a rainbow subgraph of G which has a minimum number of vertices.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 193–203, 2012.

Previous Work

Pure Parsimony Haplotyping. The pure parsimony haplotyping problem was introduced by Gusfield [8]. Hubbell shows that the PPH problem is **NP**-hard [12]. Lancia et al. [16] show that the PPH(k) problem is **APX**-hard for $k \geq 3$ and present a 2^{k-1}-approximation algorithm. In the same paper [16] they also show a $\sqrt{p}$-approximation for the PPH problem. PPH(k) is fixed parameter tractable and is solvable in polynomial time for $k \leq 2$ [17]. In [11] the PPH problem is called the *optimal haplotype inference*. Huang et al. [11] present an approximation algorithm based on semidefinite programming which, with high probability, stops after $O(\log p)$ iterations and is a $O(\log p)$-approximation. The PPH problem is extensively studied in literature and several heuristics and approaches based on integer programming were proposed (see [5] for a survey).

Minimum Rainbow Subgraph. Rainbow subgraphs are fundamental in combiatorics and have been extensively studied (e.g. [6,18,1,9,19]). In general, combinatorists study the existence of a rainbow subgraph under various conditions. However, from the algorithmic perspective, the problem did not receive much attention until recently. Camacho et al. [4] give an approximation algorithm with a ratio of $\frac{5}{6}\Delta$. This was later improved to $\frac{1}{2} + (\frac{1}{2} + \epsilon)\Delta$ for arbitrary small ϵ [14]. Katrenič and Schiermeyer [14] also prove that the MRS problem is **APX**-hard on graphs with maximum degree 2 (notice that the **APX**-hardness of MRS in the general case follows from the **APX**-hardness of PPH(k)) and present an exact algorithm with time complexity $O(2^{(p+2p\log\Delta)}n^{O(1)})$. Koch et al [15] show that a natural greedy algorithm achieves a ratio of $\frac{\Delta}{2} + \frac{\ln\Delta+1}{2}$ (if the average degree of the minimum rainbow subgraph is d, then the greedy algorithm achieves a ratio of $\frac{d}{2} + \frac{\ln d+1}{2}$). Notice that the best approximation ratio is still $O(n)$ in the worst case.

Other Related Problems. If we do not consider the colouring of the edges, the MRS problem is known as the $k - f(k)$ dense subgraph problem introduced by Asahiro et al. [2]. The $k - f(k)$ dense subgraph problem is **NP**-hard as it is a special case of the maximum clique problem when $f(k) = k(k-1)/2$.

The MRS problem is a special case of the minimum k-coloured subgraph problem (MkCSP) introduced by Hajiaghayi et al. [10]. MkCSP is defined as follows: given an undirected graph G, a colour function that assigns to each edge one or more of p given colours, and an integer $k \leq p$, find a minimum set of vertices of G inducing edges of at least k colours. As shown in [10], this problem has a surprising connection to the $(k, f(k))$ dense subgraph problem and it is a generalization of the PPH problem. An important case of MkCSP occurs when $k = p$.

Our Results. In this paper we decrease the gap between the approximation lower and upper bounds of the minimum rainbow subgraph problem. First, we show a $\max(\sqrt{2p}, \min_q(q + \frac{\Delta}{e^{pq^2/\Delta n}}))$-approximation algorithm. In particular, this is a $\max(\sqrt{2n}, \sqrt{2\Delta\ln\Delta})$-approximation algorithm.

The algorithm is based on randomized linear programming (LP) rounding. The first step of the algorithm is to solve the LP relaxation of an integer program for the MRS problem. Then, we add each vertex in the solution with a probability proportional to the corresponding variable of the LP (multiplied by a certain factor). We show that the subgraph constructed in this way contains "most" of the p colors. Thus, we can apply the following naive algorithm for the remaining colors w: pick an arbitrary edge colored with w and add both its endpoints in the subgraph.

On the inapproximability side we show that the MRS is hard to approximate within a factor of $c \ln \Delta$, for some $c > 0$, unless $\mathbf{NP} \subseteq \mathbf{TIME}(n^{O(\log \log n)})$. The hardness result is obtained via a gap-preserving reduction from the set cover problem. Given a set cover instance with n elements we create an instance of the MRS problem such that $OPT_{MRS} = n(OPT_{SC} + 1)$, where OPT_{MRS} and OPT_{SC} are the values of the optimal solutions of the MRS problem and, respectively, the set cover. Feige shows [7] that it is not possible, assuming $\mathbf{NP} \not\subseteq \mathbf{TIME}(n^{O(\log \log n)})$, to decide in polynomial time if a set cover instance has a solution using k sets or the number of sets in the optimal solution is greater than $\ln n$. Combining Feige's result with our reduction, we obtain the claimed hardness result.

The rest of the paper is organized as follows. In Section 2 we give preliminary definitions. In Section 3 we present the approximation algorithm and in Section 4 we show the hardness result. Section 5 is reserved for conclusions and open problems.

2 Preliminaries

In this section we introduce notation and give preliminary definitions. We start with the definition of the minimum rainbow subgraph problem.

Problem 1 (Minimum rainbow subgraph). The input of the problem consists of an undirected graph $G = (V, E)$, and a function $col : E \to \{1, 2, \ldots, p\}$. A rainbow subgraph of G is a graph $G' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$ such that for any $i \in \{1, 2, \ldots, p\}$ there is exactly one edge $e \in E'$ with $col(e) = i$. The goal is to find a rainbow subgraph of minimum order (i.e. $|V'|$ is minimized).

In the rest of the paper we use the following notation. Let $\{1, 2, \ldots, n\}$ be the vertex set of the input graph, m be the number of edges in G and Δ be the maximum degree in G. We say that a color w is *covered* by a subgraph $G' = (V', E')$ of G if there exists an edge $e \in E'$ such that $col(e) = w$. For a color w, we denote by f_w the number of edges in G coloured with w. We use ln for the natural logarithm. All the other logarithms are in base 2, unless otherwise mentioned.

Next, we give the definition of a c-approximation algorithm.

Definition 1. *An algorithm $\mathcal{A}$ is a c-approximation algorithm for an optimization problem $\mathcal{P}$ if on any instance x we have:*

- $\mathcal{A}(x) \leq c \cdot OPT(x)$, *if* $\mathcal{P}$ *is a minimization problem.*
- $\mathcal{A}(x) \geq \frac{OPT(x)}{c}$, *if* $\mathcal{P}$ *is a maximization problem.*

where $\mathcal{A}(x)$ *is the cost of the solution returned by the algorithm on instance* x *and* $OPT(x)$ *is the cost of the optimal solution.*

To prove the hardness result for the MRS problem we use a gap-preserving reduction from the set cover problem defined below.

Definition 2 (Set-Cover). *The input consists of a collection of sets* $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ *over a universe of elements* $\mathcal{U} = \{1, 2, \ldots, n\}$. *A set cover* $\mathcal{S}' \subseteq \mathcal{S}$ *has the property that for each element* $i \in \mathcal{U}$ *there exists a set* $S_k \in \mathcal{S}'$ *such that* $i \in S_k$. *The goal is to find a set cover of minimum cardinality.*

Feige [7] proved that it cannot be decided in polynomial time, assuming $\mathbf{NP} \nsubseteq \mathbf{TIME}(n^{O(\log \log n)})$, if a set cover with k sets (where k is a large constant) exists or the smallest set cover has cardinality greater than $k \ln n$. This is stated in the following theorem (rephrased from [7]).

Theorem 1. *There exists a constant* k *such that it cannot be decided in polynomial time if a set cover instance has a cover with* k *subsets or the smallest set cover has roughly* d *subsets where* $(1 - 1/k)^d \approx 1/n$. *As* k *grows,* d *tends to* $k \ln n$.

To design approximation algorithms it is necessary to have a lower bound of the optimal solution (to be able to analyse the approximation guarantee). For the MRS problem, a trivial lower bound is the minimum order of a graph with p edges and maximum degree Δ:

$$\frac{2p}{\Delta} \tag{1}$$

Also notice that the naive algorithm which returns $2p$ vertices is a $\sqrt{2p}$-approximation algorithm (as each graph with p edges must have at least $\sqrt{2p}$ vertices). Thus, if $p \leq n$ then we have a $\sqrt{2n}$-approximation algorithm.

However, this lower bounds do not suffice for our purposes. To achieve our approximation ratio we express the MRS problem as an integer program. Integer programs are **NP**-hard to solve [13], but we can use their linear programming relaxations as lower bounds for the optimal solution. Linear programs can be solved in polynomial time using the ellipsoid method [3].

Camacho et al. [4] show that the $PPH(k)$ problem for $k \leq O(\log p)$ can be reduced in polynomial time to the *minimum rainbow subgraph (MRS)*. We mention that it is not known how to construct a polynomial time reduction from the MRS to the PPH(k) problem. Thus, a $f(\Delta)$-approximation algorithm for the MRS implies a $f(p)$-approximation for the PPH(k), but not vice-versa. Conversely, only the hardness of approximation results for PPH(k) can be applied to MRS.

3 Approximation Algorithm

In this section we present a $\max(\sqrt{2p}, \min_q(q + \frac{\Delta}{e^{pq^2/\Delta n}}))$-approximation algorithm for the minimum rainbow subgraph problem. Our approximation algorithm is based on randomized LP rounding. The integer programming formulation of the MRS problem is presented in Figure 1.

The integer program has a variable v_i corresponding to each vertex $i \in V$ and a variable x_e corresponding to each edge $e \in E$. A solution of the integer program has the property that: $v_i = 1$ if and only if the vertex i is part of the solution (i.e. i is a vertex of the rainbow subgraph) and $v_i = 0$, otherwise. Similarly, $x_e = 1$ if the edge e is part of the solution and $x_e = 0$, otherwise.

The first set of constraints ensures that the rainbow subgraph contains each one of the p colors. If an edge $e \in E$ is part of the subgraph, then, clearly, both its endpoints have to be as well in the subgraph. This requirement is expressed by the second set of constraints.

$$\text{minimize} \sum_{i=1}^{n} v_i$$

$$\text{subject to}$$

$$\sum_{e \in E:\ col(e)=i} x_e \geq 1 \qquad 1 \leq i \leq p \qquad (2)$$

$$v_i \geq x_e \qquad \forall e \in E \text{ incident to } i \qquad (3)$$

$$x_e \in \{0,1\} \qquad \forall\ e \in E$$

$$v_i \in \{0,1\} \qquad \forall\ 1 \leq i \leq n$$

Fig. 1. An integer programming formulation for the minimum rainbow subgraph problem

The LP relaxation (Figure 2) is obtained by allowing the variables v_i and x_e to have any positive value. Notice that the optimal solution of the LP is less than or equal to the optimal solution of the IP.

Now we describe the approximation algorithm. First, we solve the linear program from Figure 2. Then, we add each vertex i into the solution with probability $q \cdot v_i$ (q is a parameter that is specified later). Finally, for each color w which is not covered at the previous step, we choose an arbitrary edge colored with w and add both its endpoints to the solution. We mention that a more clever algorithm can be applied in practice to cover the remaining colors, but this suffices for our analysis. Algorithm 1 presents formally the approximation algorithm.

In the following theorem we state the approximation guarantee of Algorithm 1.

Theorem 2. *Algorithm 1 is a randomized polynomial time* $q + \frac{\Delta}{e^{pq^2/\Delta n}}$*- approximation for the minimum rainbow subgraph problem.*

$$\text{minimize} \sum_{i=1}^{n} v_i$$

subject to

$$\sum_{e \in E:\ col(e)=i} x_e \geq 1 \qquad 1 \leq i \leq p \tag{4}$$

$$v_i \geq x_e \qquad \forall e \in E \text{ incident to } i \tag{5}$$

$$x_e \geq 0 \qquad \forall\ e \in E$$

$$v_i \geq 0 \qquad \forall\ 1 \leq i \leq n$$

Fig. 2. The LP relaxation of the integer program from Figure 1

Input: Graph $G = (V, E)$ coloured with p colors

1. $V' \leftarrow \emptyset$
2. Solve the LP from Figure 2
3. For $i = 1$ to n add vertex i to V' with probability $q \cdot v_i$
4. For each uncovered color w:
   (a) Select arbitrarily an edge (i, j) coloured with w
   (b) $V' \leftarrow V' \cup \{i, j\}$
5. Let $E' \leftarrow (V' \times V') \cap E$. If there are more edges coloured with the same color in E', keep only one of them.

Output: $G' = (V', E')$

Algorithm 1. A $q + \frac{\Delta}{e^{pq^2/\Delta n}}$-approximation for the MRS problem

Proof. Clearly, Algorithm 1 runs in polynomial time as all the steps can be performed in polynomial time. The expected number of vertices in the solution is:

$$\begin{aligned} E(|V'|) &= E(\#\text{ vertices added at step 3}) + E(\#\text{ vertices added at step 4}) \\ &= E(\#\text{ vertices added at step 3}) + 2 \cdot E(\#\text{ uncovered colors}) \\ &= \sum_{i=1}^{n} P(i \text{ is covered at step 3}) + 2 \cdot E(\#\text{ uncovered colors}) \\ &= \sum_{i=1}^{n} q \cdot v_i + 2 \cdot E(\#\text{ uncovered colors}) \\ &= q \cdot OPT + 2 \cdot E(\#\text{ uncovered colors}) \end{aligned}$$

We bound the number of colors which are not covered by the subgraph after step 3.

$$
\begin{aligned}
E(\#\text{ uncovered colors}) &= \sum_{w=1}^{p} P(w \text{ is not covered}) \\
&= \sum_{w=1}^{p} \prod_{e:col(e)=w} P(e \text{ is not covered}) \\
&= \sum_{w=1}^{p} \prod_{e:col(e)=w} (1 - P(e \text{ is covered})) \\
&= \sum_{w=1}^{p} \prod_{(i,j):col((i,j))=w} (1 - P(i \text{ is covered})P(j \text{ is covered})) \\
&= \sum_{w=1}^{p} \prod_{(i,j):col((i,j))=w} (1 - q^2 v_i v_j)
\end{aligned}
$$

From the second set of constraints of the LP we know that $v_i \geq x_e$ and $v_j \geq x_e$. Thus:

$$1 - q^2 v_i v_j \leq 1 - q^2 x_e^2$$

The value of :

$$\prod_{e:col(e)=w} (1 - q^2 x_e^2)$$

is maximized when all the variables x_e are equal. If all the variables x_e are equal, then, from the first set of constraints we have that:

$$x_e = \frac{1}{f_w}$$

where f_w is the number of edges that have color w. Thus:

$$
\begin{aligned}
E(\#\text{ uncovered colors}) &\leq \sum_{w=1}^{p} \prod_{e:col(e)=w} (1 - \frac{q^2}{f_w^2}) && (6) \\
&= \sum_{w=1}^{p} (1 - \frac{q^2}{f_w^2})^{f_w} && (7)
\end{aligned}
$$

We know that:

$$e^x \geq (1 + \frac{x}{n})^n$$

where e is the base of the natural logarithm. Therefore:

$$(1 - \frac{q^2}{f_w^2})^{f_w} = (1 - \frac{q^2}{f_w^2})^{f_w^2/f_w} \leq e^{-q^2/f_w} = \frac{1}{e^{q^2/f_w}}$$

We substitute the above inequality in (7):

$$E(\#\text{ uncovered colors}) \leq \sum_{w=1}^{p} \frac{1}{e^{q^2/f_w}} \tag{8}$$

The expected number of uncovered colors is maximized when all f_w are equal and since the number of edges is at most Δn, we have:

$$E(\#\text{ uncovered colors}) \leq \frac{p}{e^{pq^2/\Delta n}} \tag{9}$$

The expected order of a subgraph returned by Algorithm 1 is:

$$E(|V'|) \leq q \cdot OPT + \frac{2p}{e^{pq^2/\Delta n}} \tag{10}$$

We now use the trivial lower bound (1) in order to bound the expected number of vertices in V':

$$\frac{2p}{e^{pq^2/\Delta n}} \cdot \frac{1}{OPT} \leq \frac{2p}{e^{pq^2/\Delta n}} \cdot \frac{\Delta}{2p}$$

We have that:

$$\frac{2p}{e^{pq^2/\Delta n}} \leq \frac{\Delta}{e^{pq^2/\Delta n}} \cdot OPT$$

$$E(|V'|) \leq q \cdot OPT + \frac{\Delta}{e^{pq^2/\Delta n}} OPT \tag{11}$$

Algorithm 1 returns a $q + \frac{\Delta}{e^{pq^2/\Delta n}}$-approximation and the theorem follows. □

Notice that if $p \leq n$, then we can use the naive algorithm to obtain a $\sqrt{2n}$-approximation algorithm. Otherwise, we apply Algorithm 1 and get Corollary 1.

Corollary 1. *Algorithm 2 is a* $\max(\sqrt{2n}, q + \frac{\Delta}{e^{q^2/\Delta}})$*-approximation for the MRS problem.*

Input: A graph $G = (V, E)$ coloured with p colors

1. If $p \leq n$, then apply the naive algorithm.
2. If $p > n$, then apply Algorithm 1.

Algorithm 2. A $\max(\sqrt{2n}, q + \frac{\Delta}{e^{q^2/\Delta}})$-approximation algorithm for the MRS problem

We fix $q = \sqrt{\frac{\Delta \ln \Delta}{2}}$ and obtain the following corollary.

Corollary 2. *Algorithm 1 is a* $\max(\sqrt{2n}, \sqrt{2\Delta \ln \Delta})$*-approximation for the MRS problem.*

4 Hardness of Approximation

In this section we show that the minimum rainbow subgraph problem does not have a $c \ln \Delta$-approximation algorithm, for some constant $c > 0$, unless **NP** has $n^{O(\log \log n)}$-time deterministic algorithms. This hardness result is stated in the following theorem.

Theorem 3. *There exists a constant $c > 0$ such that the minimum rainbow subgraph problem does not have a $c \ln \Delta$-approximation algorithm, unless* $\mathbf{NP} \subset \mathbf{TIME}(n^{O(\log \log n)})$.

Proof. We prove our result via a reduction from the set-cover problem. Given a set cover instance $\mathcal{S}$ with n elements $\{1, \ldots, n\}$ and m sets $S_1, \ldots, S_m$ we construct an instance $G = (V, E)$ of the minimum rainbow subgraph as follows:

- The vertex set V has one vertex a_i corresponding to each element of i the set cover universe and n vertices $b_1^j \ldots, b_n^j$ for each set S_j.
- For each element $i \in S_j$, we add the edges $(a_i, b_1^j), \ldots, (a_i, b_n^j)$. The edges $(a_i, b_k^j), \forall 1 \leq j \leq m$ have color c_i^k.

Let OPT_{MRS} be the minimum order of a rainbow subgraph on the input instance G and let OPT_{SC} be the number of sets in the minimum set cover instance $\mathcal{S}$. We show that:

$$OPT_{MRS} = n \cdot (OPT_{SC} + 1)$$

First we prove that given a set cover with q sets, we can find a rainbow subgraph of order $n(q+1)$. The rainbow subgraph contains all the qn vertices corresponding to sets in the cover and all the n vertices corresponding to the elements. We show now that all the colors are covered in this subgraph. Assume by contradiction that there is a color c_i^k which is not covered. Thus, all the vertices b_k^j corresponding to the sets S_j for which $i \in S_j$ are not part of the subgraph (this contradicts the definition of a set cover).

We now prove the converse implication. Given a rainbow subgraph of order $n(q+1)$, we show how to recover a set cover with q sets. First, observe that all the a_i vertices have to be part of the solution (as the graph G is bipartite and for each a_i, the color c_i^k does not touch any other vertex a_z). Now, let $B_i = \{b_i^1, \ldots, b_i^m\}$. There has to be a set B_w which has q or less vertices in the rainbow subgraph (otherwise the subgraph has order greater than $n(q+1)$). All these vertices correspond to sets in a set cover as for all vertices a_i the color c_i^w has to be covered. Thus, there is a vertex b_w^j in the rainbow subgraph such that $(a_i, b_w^j) \in E$ and element i is covered by S_j.

Following Therorem 1 we obtain that there exists a c such that the minimum rainbow subgraph does not have a $c \ln n$-approximation, unless $\mathbf{NP} \subseteq \mathbf{TIME}(n^{O(\log \log n)})$. Notice that n here is the number of elements in the set cover instance. However, Theorem 1 holds even if the number of sets m is polynomial in n. Since the maximum degree Δ in our construction is $\max(m, n)$, the theorem follows. □

5 Conclusions and Open Problems

In this paper, we present a $\max(\sqrt{2p}, \min_q(q + \frac{\Delta}{e^{pq^2/\Delta n}}))$ - approximation algorithm for the MRS problem. In particular, this is a $\max(\sqrt{2n}, \sqrt{2\Delta \ln \Delta})$ - approximation algorithm. On the other hand we show that the minimum rainbow subgraph problem is hard to approximate within a factor of a $c \ln n$, for some constant c, unless $\mathbf{NP} \subseteq \mathbf{TIME}(n^{O(\log \log n)})$.

We mention that our approximation algorithm has a better ratio when the number of colors p is large. Thus, we believe that an interesting research direction is to improve the approximation ratio of the MRS for small values of p.

Acknowledgements. The author would like to thank the anonymous reviewers for their helpful comments.

References

1. Alon, N., Jiang, T., Miller, Z., Pritikin, D.: Properly colored subgraphs and rainbow subgraphs in edge-colorings with local constraints. Random Structures & Algorithms 23(4), 409–433 (2003)
2. Asahiro, Y., Hassin, R., Iwama, K.: Complexity of finding dense subgraphs. Discrete Applied Mathematics 121(1-3), 15–26 (2002)
3. Bland, R.G., Goldfarb, D., Todd, M.J.: The ellipsoid method: A survey. Operations Research 29(6), 1039–1091 (1981)
4. Camacho, S.M., Schiermeyer, I., Tuza, Z.: Approximation algorithms for the minimum rainbow subgraph problem. Discrete Mathematics 310(20), 2666–2670 (2010)
5. Catanzaro, D., Labb, M.: The pure parsimony haplotyping problem: overview and computational advances. International Transactions in Operational Research 16(5), 561–584 (2009)
6. Erdős, P., Tuza, Z.: Rainbow subgraphs in edge-colorings of complete graphs. In: Kennedy, J.W., Gimbel, J., Quintas, L.V. (eds.) Quo Vadis, Graph Theory? A Source Book for Challenges and Directions. Annals of Discrete Mathematics, vol. 55, pp. 81–88. Elsevier (1993)
7. Feige, U.: A threshold of ln n for approximating set cover. Journal of the ACM 45(4), 634–652 (1998)
8. Gusfield, D.: Haplotype Inference by Pure Parsimony. In: Baeza-Yates, R., Chávez, E., Crochemore, M. (eds.) CPM 2003. LNCS, vol. 2676, pp. 144–155. Springer, Heidelberg (2003)
9. Hahn, G., Thomassen, C.: Path and cycle sub-Ramsey numbers and an edge-colouring conjecture. Discrete Mathematics
10. Hajiaghayi, M.T., Jain, K., Lau, L.C., Mandoiu, I.I., Russell, A., Vazirani, V.V.: Minimum multicolored subgraph problem in multiplex pcr primer set selection and population haplotyping. In: International Conference on Computational Science, vol. (2), pp. 758–766 (2006)
11. Huang, Y.-T., Chao, K.-M., Chen, T.: An approximation algorithm for haplotype inference by maximum parsimony. In: SAC 2005, pp. 146–150 (2005)
12. Hubbell, E.: Unpublished manuscript (2002)
13. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of Computer Computations, pp. 85–103 (1972)

14. Katrenič, J., Schiermeyer, I.: Improved approximation bounds for the minimum rainbow subgraph problem. Inf. Process. Lett. 111(3), 110–114 (2011)
15. Koch, M., Camacho, S.M., Schiermeyer, I.: Algorithmic approaches for the minimum rainbow subgraph problem. Electronic Notes in Discrete Mathematics 38(0), 765–770 (2011)
16. Lancia, G., Pinotti, M.C., Rizzi, R.: Haplotyping populations by pure parsimony: Complexity of exact and approximation algorithms. INFORMS Journal on Computing 16(4), 348–359 (2004)
17. Lancia, G., Rizzi, R.: A polynomial case of the parsimony haplotyping problem. Oper. Res. Lett. 34(3), 289–295 (2006)
18. Rődl, V., Tuza, Z.: Rainbow subgraphs in properly edge-colored graphs. Random Structures & Algorithms 3(2), 175–182 (1992)
19. Simonovits, M., Sós, V.T.: On restricted colourings of K_n. Combinatorica 4(1), 101–110 (1984)

Ramsey Numbers for Line Graphs and Perfect Graphs*

Rémy Belmonte, Pinar Heggernes, Pim van 't Hof, and Reza Saei

Department of Informatics, University of Bergen, Norway
{remy.belmonte,pinar.heggernes,pim.vanthof,reza.saeidinvar}@ii.uib.no

Abstract. For any graph class $\mathcal{G}$ and any two positive integers i and j, the Ramsey number $R_{\mathcal{G}}(i,j)$ is the smallest integer such that every graph in $\mathcal{G}$ on at least $R_{\mathcal{G}}(i,j)$ vertices has a clique of size i or an independent set of size j. For the class of all graphs Ramsey numbers are notoriously hard to determine, and the exact values are known only for very small integers i and j. For planar graphs all Ramsey numbers can be determined by an exact formula, whereas for claw-free graphs there exist Ramsey numbers that are as difficult to determine as for arbitrary graphs. No further graph classes seem to have been studied for this purpose. Here, we give exact formulas for determining all Ramsey numbers for various classes of graphs. Our main result is an exact formula for all Ramsey numbers for line graphs, which form a large subclass of claw-free graphs and are not perfect. We obtain this by proving a general result of independent interest: an upper bound on the number of edges any graph can have if it has bounded degree and bounded matching size. As complementary results, we determine all Ramsey numbers for perfect graphs and for several subclasses of perfect graphs.

1 Introduction

Ramsey Theory is a large and important subfield of combinatorics that studies how large a system must be in order to ensure that it contains some particular structure. Since the start of the field in 1930 [11], there has been a tremendous interest in Ramsey Theory, leading to many results as well as several surveys and books (see e.g., [7] and [10]). The fundamental theorem of Ramsey Theory, in its graph theoretic version, states that for every pair of positive integers i and j, there exists a finite integer $R(i,j)$ such that every graph on at least $R(i,j)$ vertices contains either a clique of size i or an independent set of size j [11].

It is already somewhat surprising that such values $R(i,j)$ exist, as discussed by Diestel [5]. Even more surprising is how difficult it is to determine these values exactly. Despite the vast amount of results that have been produced on Ramsey Theory during the past 80 years, we still do not know the exact value of $R(4,6)$ or $R(3,10)$ [10]. This is most adequately addressed by the following quote, attributed to Paul Erdős [12]. *"Imagine an alien force, vastly more powerful than us, landing on Earth and demanding the value of* $R(5,5)$ *or they will*

* This work is supported by the Research Council of Norway.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 204–215, 2012.

destroy our planet. In that case, we should marshal all our computers and all our mathematicians and attempt to find the value. But suppose, instead, that they ask for $R(6,6)$*. In that case, we should attempt to destroy the aliens."* During the last two decades, with the use of computers, lower and upper bounds have been established for more and more Ramsey numbers. However, the exact values are known only for very small integers.

Confronted with such difficulty, it is natural to restrict the set of considered graphs. Such a restriction is also motivated from a computational perspective, since cliques and independent sets are well studied and highly useful structures in computer science. Many applications and algorithms are tested on randomly generated graphs belonging to a particular graph class, and Ramsey numbers for these graph classes might be useful for generating more meaningful test sets. In this paper, we study Ramsey numbers for graph classes. In particular, given a graph class $\mathcal{G}$, for every pair of positive integers i and j, what is the smallest integer $R_{\mathcal{G}}(i,j)$ such that every graph in $\mathcal{G}$ on at least $R_{\mathcal{G}}(i,j)$ vertices contains either a clique of size i or an independent set of size j? To the best of our knowledge, this has been studied previously only when $\mathcal{G}$ is the class of planar graphs and when $\mathcal{G}$ is the class of claw-free graphs. For planar graphs, the exact values for all i,j were provided independently by Walker [15] and by Steinberg and Tovey [13]. For claw-free graphs, there exist Ramsey numbers that are as difficult to determine as for arbitrary graphs [9]. Hence it is highly unlikely that an exact formula for all Ramsey numbers for claw-free graphs will ever be found.

The study of graph classes has flourished during the past few decades, both within graph theory and within theoretical computer science, where perfect graphs and claw-free graphs represent some of the most well-studied graph classes (see e.g., [1,2,3,6]). Our main contribution here is an exact formula for the Ramsey numbers for line graphs, which are not perfect and form an important subclass of claw-free graphs. In addition, we provide exact formulas for the Ramsey numbers for the class of perfect graphs and several of its subclasses. We also observe that determining all Ramsey numbers is as difficult for AT-free graphs and for P_5-free graphs as it is for arbitrary graphs. Hence our results narrow the gap between known graph classes whose Ramsey numbers can be determined by exact formulas, and known graph classes whose Ramsey numbers are as hard to determine as in the general case.

To obtain our results on line graphs, we prove a result of independent interest on the number of edges of a graph that has bounded degree and bounded maximum matching size. A result regarding the number of vertices of such a graph was given by Cockayne and Lorimer [4].

2 Definitions and Notation

All graphs considered here are undirected, finite and simple. For a graph G, we use $V(G)$ and $E(G)$ to denote the set of vertices and set of edges of G, respectively, and we let $n = |V(G)|$. For a vertex v in G, the set $N_G(v) = \{w \in V \mid vw \in E(G)\}$, consisting of all the neighbors of v in G, is the *neighborhood*

of v. The set $N_G[v] = N_G(v) \cup \{v\}$ is the *closed neighborhood* of v. We omit subscripts when there is no ambiguity. The *degree* of a vertex v is $d(v) = |N(v)|$. The maximum degree of a vertex in G is denoted $\Delta(G) = \max_{v \in V(G)} d(v)$. A vertex u and an edge e are called *incident* if u is an endpoint of e. Two edges incident with a common vertex are also called incident. The *edge neighborhood* of a vertex u is the set of edges incident with u. If $uv \notin E(G)$ then uv is called a *non-edge* of G. The complement of a graph G, denoted $\overline{G}$, is the graph that has vertex set $V(G)$, whose edge set is equal to the set of non-edges of G. A vertex is *isolated* if its degree is 0, and *universal* if its degree is $n - 1$.

A graph is *connected* if there is a path between every pair of vertices. A maximal connected subgraph of a graph is called a *connected component*. For any set $S \subseteq V(G)$, we write $G[S]$ to denote the subgraph of G *induced* by S. We write $G - v$ to denote $G[V(G) \setminus \{v\}]$. A subset $S \subseteq V(G)$ is a *clique* of G if all vertices in S are pairwise adjacent in G, and S is an *independent set* of G if no two vertices of S are adjacent in G. A complete graph on ℓ vertices is denoted K_ℓ and an edgeless graph on ℓ vertices $\overline{K_\ell}$. P_ℓ and C_ℓ denote the graphs that are isomorphic to a chordless path and a cycle, respectively, on ℓ vertices. The *disjoint union* of graphs $G_1 = (V_1, E_1), \ldots, G_k = (V_k, E_k)$ is the graph $(V_1 \cup \ldots \cup V_k,\ E_1 \cup \ldots \cup E_k)$, assuming that $V_i \cap V_j = \emptyset$ and $E_i \cap E_j = \emptyset$, for $1 \leq i \neq j \leq k$.

A *matching* M of a graph G is a set of edges of G such that no two edges in M are incident. The *size* of a matching M, denoted $|M|$, is the number of edges in M. A vertex u is said to be *saturated* by a matching M if there exists an edge $e \in M$ such that e is incident with u; otherwise u is *unsaturated* by M. An *M-alternating path* is a path whose edges belong alternatively to M and not to M. An *M-augmenting path* is an M-alternating path whose end-vertices are both M-unsaturated.

A *proper vertex coloring* of G is an assignment of colors to the vertices of G such that pairs of vertices that are adjacent receive different colors. A *proper edge coloring* of G is an assignment of colors to the edges of G such that pairs of edges that are incident receive different colors.

For any two positive integers i and j, the *Ramsey number* $R(i, j)$ is the smallest integer such that every graph on at least $R(i, j)$ vertices contains a clique of size i or an independent set of size j. For a graph class $\mathcal{G}$, we define $R_{\mathcal{G}}(i, j)$ to be the smallest number such that every graph in $\mathcal{G}$ on at least $R_{\mathcal{G}}(i, j)$ vertices contains a clique of size i or an independent set of size j. Clearly, $R_{\mathcal{G}}(i, j) \leq R(i, j)$ for any graph class $\mathcal{G}$.

For every fixed graph H, the class of *H-free* graphs is the class of graphs that do not contain an induced subgraph isomorphic to H. A *claw* is another name for $K_{1,3}$, i.e., the graph formed by a vertex adjacent to three pairwise non-adjacent vertices. For every graph G, the *line graph* of G, denoted $L(G)$, is the graph with vertex set $E(G)$, where there is an edge between two vertices $e, e' \in E(G)$ if and only if the edges e and e' are incident in G. G is called the *preimage graph* of $L(G)$. A graph is a *line graph* if it is the line graph of some graph. Line graphs form a subclass of claw-free graphs. An *asteroidal triple (AT)* is a set of three

pairwise non-adjacent vertices such that between every two of these vertices, there is a path that does not contain a neighbor of the third. A graph is *AT-free* if it does not contain an AT.

Let $\omega(G)$ and $\chi(G)$ denote the the size of a largest clique in G and the smallest number of colors in a proper vertex coloring of G, respectively. A graph is *perfect* if $\omega(G') = \chi(G')$ for every induced subgraph G' of G. Perfect graphs are characterized as graphs that do not contain a chordless cycle of odd length at least 5 or the complement of such a cycle as an induced subgraph, as conjectured by Berge more than 40 years ago and recently proved by Chudnovsky et al. [2]. Perfect graphs contain many well studied graph classes, like chordal graphs, interval graphs, permutation graphs, split graphs, and cographs. A graph is *chordal* if it does not contain a chordless cycle of length greater than 3 as an induced subgraph. An *interval graph* is the intersection graph of intervals on the real line. Interval graphs are chordal. A *proper interval graph* is an interval graph where the intervals of the real line have unit length. Proper interval graphs are claw-free. A *permutation graph* is the intersection graph of straight lines between two parallels. A *cocomparability* graph is the intersection graph of curves from a line to a parallel line. Interval graphs and permutation graphs are unrelated to each other but they are both cocomparability graphs. A *cograph* is a graph that can be generated from single-vertex graphs by repeated application of the complete join and disjoint union operations. Cographs are permutation. A *split graph* is a graph whose vertices can be partitioned into a clique and an independent set. Split graphs are chordal. A *threshold graph* is a graph that can be constructed from a one-vertex graph by repeated applications of the following two operations: addition of a single isolated vertex and addition of a single universal vertex. The class of threshold graphs form a subclass of both split graphs and cographs. A graph is *bipartite* if its vertex set can be partitioned into two independent sets. The complement of a bipartite graph is a *co-bipartite* graph.

More information on the graph classes mentioned in this paper, including a wealth of information on applications of these classes, can be found in the excellent monographs by Brandstädt et al. [1] and by Golumbic [6].

3 Ramsey Numbers for Line Graphs

In this section we give an exact formula for all Ramsey numbers for line graphs. Throughout this section, let $\mathcal{L}$ denote the class of line graphs.

Observation 1. *For every integer $j \geq 0$, $R_{\mathcal{L}}(1, j) = 1$ and $R_{\mathcal{L}}(2, j) = j$.*

The case $i = 3$ is the first non-trivial case for the class of line graphs. The proof of the theorem below is omitted in this extended abstract.

Theorem 1. *For every integer $j \geq 1$,*

$$R_{\mathcal{L}}(3, j) = \begin{cases} \frac{5(j-1)-1}{2} + 1 & \text{if } j \text{ is even,} \\ \frac{5(j-1)}{2} + 1 & \text{if } j \text{ is odd.} \end{cases}$$

Theorem 2, together with Observation 1 and Theorem 1, provides the exact values of all Ramsey numbers for line graphs.

Theorem 2 (Main theorem). *For every pair of integers $i \geq 4$ and $j \geq 1$,*

$$R_{\mathcal{L}}(i,j) = \begin{cases} i(j-1) - (t+r) + 2 & i = 2k \\ i(j-1) - r + 2 & i = 2k+1, \end{cases}$$

where $j = tk + r$, $t \geq 0$ and $1 \leq r \leq k$.

Note that for every $i \geq 2$ and $j \geq 1$, the values of t, k and r are uniquely defined. For convenience, we define the following function β:

$$\beta(i,j) = \begin{cases} i(j-1) - (t+r) + 1 & i = 2k \\ i(j-1) - r + 1 & i = 2k+1, \end{cases}$$

where $j = tk + r$, $t \geq 0$ and $1 \leq r \leq k$. Hence Theorem 2 can alternatively be stated as follows: $R_{\mathcal{L}}(i,j) = \beta(i,j) + 1$ for all $i \geq 4$ and $j \geq 1$.

We will prove Theorem 2 by considering the number of edges in the preimage graph of a line graph. It is well-known that every connected line graph, except K_3, has a unique preimage graph (see e.g., [8]). Consequently, for every integer $i \neq 3$, if a line graph contains a clique of size i, then its preimage graph has a vertex of degree i. Similarly, for every integer $i \geq 0$, if a line graph contains an independent set of size j, then its preimage graph H has a matching of size j. Based on these observations, instead of proving Theorem 2 directly, we prove Theorems 3 and 4 below, that will immediately imply Theorem 2.

Cockayne and Lorimer [4] determined the maximum number of vertices a graph can have if both its maximum degree and its maximum matching size are bounded. In the next theorem, we show how many edges such a graph can have. We find Theorem 3 to be of combinatorial and computational interest, independent of the rest of our results.

Theorem 3. *Let $i \geq 4$ and $j \geq 1$ be two integers, and let H be an arbitrary graph such that $\Delta(H) \leq i - 1$ and H has a maximum matching of size at most $j - 1$. Then H has at most $\beta(i,j)$ edges.*

Proof. We prove the theorem by induction on j.

Base Case: We consider the case $j = 1$. Since $k \geq 2$ then $t = 0$ and $r = 1$, and therefore $\beta(i,j) = \beta(i,1) = 0$. Any graph with a maximum matching of size $j - 1 = 0$ does not contain any edge. Hence, the number of edges of H is at most $\beta(i,j)$ in this case.

Induction Hypothesis: For every $i \geq 4$ and $1 \leq \ell < j$, any graph H with $\Delta(H) \leq i - 1$ and a maximum matching of size at most $\ell - 1$ contains at most $\beta(i,l)$ edges.

Now we prove that for each $i \geq 4$ and $j \geq 2$, every graph H with $\Delta(H) \leq i - 1$ and a maximum matching of size at most $j - 1$ contains at most $\beta(i,j)$ edges. Let H be such a graph and M a maximum matching of H. Notice that in H every edge

should have a vertex in common with an edge in M, since otherwise M would not be maximum. Hence we can partition the edges of H into three classes:

M: Edges of the maximum matching M,
T: Edges that are not in M but both of their endpoints are endpoints of edges in M,
R: Edges of $E(H)\backslash(M \cup T)$.

We define $V(M)$ to be the set of endpoints of edges of M. In R, we distinguish three types of edges:

`Type I`: Edges ux such that there exist an edge vx with $uv \in M$ and $x \notin V(M)$. We call ux and vx a *pair* of edges of `type I`.
`Type II`: Edges ux such that there exist an edge vx with $u, v \in V(M), x \notin V(M)$ and $uv \notin M$. We call ux and vx a *pair* of edges of type `type II`.
`Type III`: Edges ux such that $u \in V(M), x \notin V(M)$ and $d(x) = 1$.

If an edge of R is of `type I` and also of `type II`, then we consider it as an edge of `type I`. For each vertex $v \in V(M)$, we write $d_T(v)$ and $d_R(v)$ to denote the number of edges of T and R, respectively, that are incident to v.

We claim that each edge in M has common endpoints with only one pair of edges of `type I`. Assume on the contrary that there exists and edge uv in M that has common endpoints with at least two pairs of edges of `type I`. Let ux, vx and uy, vy be two of these pairs. Now we can remove the edge uv from M and add ux and vy to it to find a matching in H that is larger than M, which contradicts the fact that M is maximum. We also claim that when an edge of M has common endpoints with a pair of edges of `type I`, then none of its endpoints can have any edge of other types of R in its edge neighborhood. Assume for contradiction that this happens, and let uv be an edge in M such that ux and vx are a pair of edges of `type I`, and uy is an edge in R of `type II` or `type III`. In this case, removing uv from M and adding uy and vx to it yields once again a matching larger than M, which is a contradiction. Finally, for each edge uv in M, either u or v can be incident to edges of `type II` or `type III`. Otherwise, removing uv from M and adding two edges of R to M, one from the edge neighborhood of u and the other one from the edge neighborhood of v, we can obtain a matching in H that is larger than M, again yielding a contradiction.

We will now determine an upper bound on the maximum number of edges H can have. We have three cases, depending on which type of edges exist in H.

Case 1: H has at least one edge of R of `type III`.

Let $ux \in R$ be an edge of `type III` with $u \in V(M)$. We remove u and x from H and call the remaining graph H'. Every matching M' of H' has at most $j-2$ edges, since otherwise $M' \cup \{ux\}$ would be a matching of size j in H. Hence, every matching in H' has at most $j-2$ edges and every vertex in H' has degree at most $i-1$. Let $j' = j-1 = t'k+r'$. Observe that $t'+r' \geq t+r-1$, because if $2 \leq r \leq k$ then $r' = r-1$ and $t' = t$, and if $r = 1$ then $r' = k \geq r$ and $t' = t-1$. Moreover, note that $j' < j$, which means we can apply the induction hypothesis and get:

$$|E(H')| \leq \begin{cases} i(j'-1)-(t'+r')+1 & i=2k \\ i(j'-1)-r'+1 & i=2k+1 \end{cases}$$

where $j' = t'k + r'$, $t' \geq 0$ and $1 \leq r' \leq k$. For $i = 2k$, the value $i(j'-1)-(t'+r')+1$ is equal to $i(j-2)-(t'+r')+1$, which is at most $i(j-2)-(t+r)+2$. For $i = 2k+1$, the value $i(j'-1)-r'+1$ is equal to $i(j-2)-r'+1$, which is equal to $i(j-2)-r+2$ for $r \geq 2$, and is at most $i(j-2)-r+1$ for $r = 1$. Therefore, for $i = 2k+1$ we have $i(j'-1)-r'+1 \leq i(j-2)-r+2$. Finally, recall that H contains at most . $d_H(u)$ edges more than H' and $\Delta(H) \leq i-1$. Therefore, we have:

$$\begin{aligned} |E(H)| &\leq |E(H')| + i - 1 \\ &\leq \begin{cases} i(j-2)-(t+r)+2+i-1 & i=2k \\ i(j-2)-r+2+i-1 & i=2k+1 \end{cases} \\ &\leq \beta(i,j). \end{aligned}$$

As a consequence, from here on we assume that H does not contain any edge of R of `type III`.

Case 2: There is a vertex $u \in V(M)$ such that $d_R(u) \geq 3$.

Let us call v the vertex such that $uv \in M$, and let H' denote the graph obtained by removing u from H. We prove that every matching in H' has at most $j-2$ edges and then we get the upper bound by the same argument we had in case 1. For contradiction, suppose there is a matching M' in H' that has $j-1$ edges. We define the following three induced subgraphs of H: X, Y and Z.

- X is the subgraph of H induced by the endpoints of the edges in $M \cap M'$;
- Y is the subgraph of H induced by the endpoints of the edges of all M'-alternating paths in H starting at some edge of R incident to u and not using edges of $T \setminus M'$;
- Z is the subgraph of H induced by the vertices of $V(M) \setminus (V(X) \cup V(Y))$.

Note that u belongs to $V(Y)$ and v can belong either to Y or to Z. If $v \in V(Y)$, then $uv \in E(Y)$. If $v \in V(Z)$, then uv belongs to none of X, Y or Z.

Claim 1. Every vertex in $Y \setminus \{u\}$ is M'-saturated by an edge both endpoints of which belong to Y.

Proof of Claim 1. Assume for contradiction that there exists a vertex $y \in V(Y)$ such that y is not M'-saturated. Then there exists an M'-alternating path P starting at some edge $uw \in R$ and ending at y. This path starts and ends with M'-unsaturated vertices (u and y), and therefore P is an M'-augmenting path. This means that there is a matching in H with at least j edges, which is a contradiction. Now we need to prove that the edge e of M' that saturates y belongs to $E(Y)$. If $e \in E(P)$, then $e \in E(Y)$ by definition of Y. Otherwise, since $e \in M'$ and the edge of P that is incident to y is not in M', we can add e to P to obtain a larger M'-alternating path. Hence $e \in E(Y)$ also in this case. $\diamond$

Claim 2. For every edge $yz \in M \setminus \{uv\}$, if $y \in Y$ then $yz \in E(Y) \setminus M'$.

Proof of Claim 2. Since $y \in Y$, there exists an M'-alternating path $P = uww' \cdots xy$ from u to y. We first show that $xy \in M'$.

Assume for contradiction that $xy \notin M'$. Then, by the definition of Y, xy does not belong to T. Therefore we have $xy \in R \setminus M'$. Hence we know that there is a vertex $x' \in V(M) \cap V(P)$, x' is the vertex on P preceding x, such that $xx' \in R \cap M'$. Such an edge xx' always exists, since $x \in V(H) \setminus V(M)$, x is on the path P from u to y, and no edge in H has both endpoints in $V(H) \setminus V(M)$. Note that $x' \neq u$, since the edge $xx' \in M'$, and no edge of M' is incident with u.

For any two vertices a and b of $V(P)$, let P_{ab} denote the subpath of P going from a to b. Consider the path P_{ux} and recall that it starts with uww'. Let ss' denote the edge of $E(P_{ux}) \cap R \cap M'$ closest to x and such that $s' \in V(M)$, $s \in V(H) \setminus V(M)$ and s' is closer to x than s. Such an edge always exists, since we can have $s = w$ and $s' = w'$. Now, consider the subpath P_{sx} and let ll' denote the edge of $E(P_{sx}) \cap R$ closest to s' and such that $l' \in V(M)$, $l \in V(H) \setminus V(M)$, $l \neq s$ and l' is closer to s' than l. Note that we cannot have $s' = l' = x'$, since ss' and xx' both belong to M'. If $l' = s' \neq x'$, then $ll' \notin M'$ and there is an edge $ll'' \in R \cap M' \cap V(P_{ux})$, which contradicts the choice of ss'. Therefore we have $l' \neq s'$ and $ll' \in M'$, since $ll' \notin M'$ would yield the existence of the edge $ll'' \in R \cap M' \cap E(P_{ux})$. Notice that $P_{s'l'}$ contains no edge of R by construction. Moreover, by the definition of Y, $P_{s'l'}$ also does not contain any edge of $T \setminus M'$. Therefore $P_{s'l'}$ is an M-alternating path. Moreover, note that $ss', ll' \in M'$, and hence $P_{s'l'}$ starts and ends with edges of M. Thus P_{sl} is an M-augmenting path, contradicting the assumption that M is a maximum matching in H. This means that $xy \in M'$, and consequently $yz \notin M'$.

To conclude, we need to show that $yz \in E(Y)$. This is easily seen from the fact that if $z \in V(P)$, then we have $yz \in E(Y)$ by definition of Y. If $z \notin V(P)$, then we can extend P to an M'-alternating path $P' = Pz$, since $xy \in M'$ and $yz \notin M'$. Thus $z \in V(Y)$ and $yz \in E(Y)$. ◇

Claim 3. $\{M \cap E(X), M \cap E(Y), M \cap E(Z)\}$ is a partition of the edges in $M \setminus \{uv\}$.

Proof of Claim 3. Note first that Claim 2 implies that every edge of $M \setminus \{uv\}$ belongs to $E(X) \cup E(Y) \cup E(Z)$, since uv is the only edge of M whose endpoints may belong to different sets. Moreover, we know that $E(X) \cap E(Z) = E(Y) \cap E(Z) = \emptyset$, since $E(Z)$ contains precisely those edges of M that belong to neither $E(X)$ nor $E(Y)$. Finally, it also follows from Claim 2 that $E(X) \cap E(Y) = \emptyset$. ◇

If v is not saturated by M', then adding uv to M' yields a matching of size j in H. Therefore v is saturated by M', and it can be either in Y or in Z. Also we should notice that v does not have any edge neighbor in R, because otherwise this edge neighbor, uv itself, and an edge neighbor of u in R make an M-augmenting path in H. This would yield a bigger matching than M, contradicting the maximality of M. Therefore, the edge of M' that saturates v is in T.

First assume $v \in Y$. We claim that the number of edges of M' that saturate vertices of Z is at most the number of edges of M in Z. Suppose this is not the case. We know from Claim 1 that all of the vertices of $V(X)$ and $V(Y) \setminus \{u\}$ are already M'-saturated by edges both endpoints of which belong to $V(X)$ and $V(Y)$ respectively, and from Claim 3 that Z is disjoint from X and Y. Also the vertex u is M' unsaturated since it is not in H' and M' is a matching in H'. Therefore there is no edge $xz \in M'$ with $x \in X \cup Y$ and $z \in Z$. Thus every edge of M' that saturates a vertex of Z does not saturate vertices of $V(M)$ in $V(X) \cup V(Y)$. Note in particular that the argument holds for v since $v \in V(Y)$. Recall that each edge of M is in X, Y or Z as a result of Claim 3. Therefore, we can remove the edges of $M \cap Z$ from M, and add those edges of M' that saturate vertices of Z, to obtain a matching strictly larger than M in H, giving a contradiction. Consequently, the number of edges of M' that saturate vertices of Z is at most the number of edges of M in Z. Now, removing the edges of M' which saturate the vertices of Z from M', and adding the edges of $M \cap E(Z)$ instead, yields a new matching in H', say M'', with $|M''| \geq |M'|$. We prove that this new matching M'' has at least j edges.

All endpoints of edges of $M \setminus \{uv\}$ and v are M''-saturated, which are $2j-3$ vertices. Recall that every vertex inside Y is saturated by M' and therefore also by M''. Let l denote the number of edges of M'' that are in X and Z. Then these edges saturate $2l$ vertices of those $2j-3$ vertices. Let d denote the number of edges of $R \cap M''$. Then these edges saturate d of those $2j-3$ vertices. Now we have $2j-3-2l-d$ vertices left to be saturated by edges of $M'' \cap T \cap E(Y)$, in particular vertex v, since v is saturated by an edge in $M'' \cap T$. Hence at least $j-\frac{3}{2}-l-\frac{d}{2}$ edges of T belong to M'', which means that $|M''| \geq j-\frac{3}{2}+\frac{d}{2}$. Recall that u has at least three edges of R of `type II` in its edge neighborhood. Each of these edges is the first edge of an M'-alternating path inside Y, and each of these paths has at least two edges, and hence for each edge of R incident with u, there is an edge of R in M', and therefore also in M''. Hence $d \geq 3$. Therefore, in this case the number of edges of M'' is at least j, which gives a contradiction.

Now assume $v \in Z$. In this case, the number of edges of M' that saturate vertices of Z is at most one more than the number of edges of M in Z. Suppose this is not the case. Then we can remove the edges of $M \cap E(Z)$ from M and replace them by edges of M' that saturate vertices of Z, except the one that saturates v. For the same reason as in the previous case, this yields a matching strictly larger than M in H; a contradiction. Then in this case, the number of edges of M' that saturate vertices of Z is at most the number of edges of M in Z plus one. Now, removing the edges of M' which saturate vertices of Z, and adding the edges of $M \cap E(Z)$ and uv instead, yields a new matching in H, say M'', with $|M''| \geq |M'|$. We prove that this new matching M'' has at least j edges. All vertices of $V(M)$ are M''-saturated, which are $2j-2$ vertices. Let l denote the number of edges of M'' that are in X and Z plus one for uv, then these edges saturate $2l$ vertices of those $2j-2$ vertices. Define d like in the previous case. Then we have $2j-2-2l-d$ remaining vertices to be saturated by edges of $M'' \cap T \cap E(Y)$, as before. Hence at least $j-1-l-\frac{d}{2}$ edges of T

belong to M'', which means that $|M''| \geq j - 1 + \frac{d}{2}$. Since we already saw that $d \geq 3$, the number of edges of M'' is at least j, which again gives a contradiction.

Case 3: For each vertex $v \in V(M)$, $d_R(v) \leq 2$.

The analysis of this case is based on the number of vertices in $V(H) \setminus V(M)$. When $|V(H) \setminus V(M)| = 1$, we can directly determine the number of edges in H. When $|V(H) \setminus V(M)| \geq 2$, to show the upper bound we need to count also non-edges of H which have both endpoints in $V(M)$. The full proof of this case must be omitted in this extended abstract. There are no remaining cases. □

Since the number of edges in a graph is equal to the number of vertices in its line graph, Theorem 3 implies that for every $i \geq 4$ and $j \geq 1$, any line graph on at least $\beta(i, j) + 1$ vertices contains either K_i or $\overline{K_j}$ as an induced subgraph.

The following theorem, whose proof is omitted in this extended abstract, shows that the upper bound of Theorem 3 is actually tight, completing the proof of Theorem 2.

Theorem 4. *For every pair of integers $i \geq 4$ and $j \geq 1$, there exists a graph H such that $\Delta(H) \leq i - 1$, H has a maximum matching of size at most $j - 1$, and H has $\beta(i, j)$ edges.*

4 Ramsey Numbers for Perfect Graph Classes

In this section, we give exact values for the Ramsey numbers of perfect graphs and several of their subclasses.

A graph class $\mathcal{G}$ is *χ-bounded* if there exists a non-decreasing function $f : \mathbb{N} \to \mathbb{N}$ such that for every $G \in \mathcal{G}$, we have $\chi(G') \leq f(\omega(G'))$ for every induced subgraph G' of G. Such a function f is called a *χ-bounding function* for $\mathcal{G}$, and we say that $\mathcal{G}$ is *χ-bounded* if there exists a χ-bounding function for $\mathcal{G}$. Both Walker [15] and Steinberg and Tovey [13] observed the close relationship between the chromatic number and Ramsey number of a graph when they studied Ramsey numbers for planar graphs. Their key observation can be applied to any χ-bounded graph class as follows.

Lemma 1. *Let $\mathcal{G}$ be a χ-bounded graph class with χ-bounding function f. Then $R_{\mathcal{G}}(i, j) \leq f(i - 1)(j - 1) + 1$ for all $i, j \geq 1$.*

Proof. Let G be a graph in $\mathcal{G}$ with $f(i - 1)(j - 1) + 1$ vertices. Suppose that G contains no K_i. Since G has no K_i, we have $\omega(G) \leq i - 1$. By the definition of a χ-bounding function, $\chi(G) \leq f(\omega(G)) \leq f(i - 1)$. Let ϕ be any proper vertex coloring of G. Since ϕ uses at most $f(i-1)$ colors and G has $f(i-1)(j-1) + 1$ vertices, there must be a color class that contains at least j vertices. Consequently, G contains an independent set of size j. □

Theorem 5. *Let $\mathcal{G}$ be a graph class that is perfect and that contains disjoint unions of complete graphs. Then $R_{\mathcal{G}}(i, j) = (i - 1)(j - 1) + 1$.*

Proof. Observe that the identity function is a χ-bounding function for the class of perfect graphs. Hence from Lemma 1 we have $R_{\mathcal{G}}(i,j) \leq (i-1)(j-1)+1$ when $\mathcal{G}$ is the class of perfect graphs or any of its subclasses. The matching lower bound is obtained by the disjoint union of $j-1$ copies of K_{i-1}. □

As a consequence of the above theorem, $R_{\mathcal{G}}(i,j) = (i-1)(j-1)+1$ in particular when $\mathcal{G}$ is the class of perfect graphs, chordal graphs, interval graphs, proper interval graphs, permutation graphs, cocomparability graphs, or cographs.

We now give the formulas for the Ramsey numbers for some perfect graph classes that do not contain disjoint unions of complete graphs, and hence have lower Ramsey numbers. The proofs of Theorems 7 and 8 are omitted in this extended abstract.

Theorem 6. *Let $\mathcal{G}$ be the class of split graphs. Then $R_{\mathcal{G}}(i,j) = i+j-1$.*

Proof. Let G be a split graph, where the vertices are partitioned into a clique of size $i-1$ and an independent set of size $j-1$. Then $|V(G)| = i+j-2$, and the addition of a vertex to either set creates either a larger clique or a larger independent set. Hence $R_{\mathcal{G}}(i,j) \leq i+j-1$. For the lower bound, consider a split graph G, where the vertices are partitioned into a clique C and an independent set I, such that $|C| = i-1$, $|I| = j-1$, C is a maximal clique in G, and for every vertex v in C, v has at least one neighbor in I. This graph G has $i+j-2$ vertices, no clique of size i and no independent set of size j. □

Theorem 7. *Let $\mathcal{G}$ be the class of threshold graphs. Then $R_{\mathcal{G}}(i,j) = i+j-2$.*

Theorem 8. *For every integer $i \geq 3$, $R_{\mathcal{G}}(i,j) = 2j-1$ when $\mathcal{G}$ is the class of bipartite graphs, forests, or paths.*

Corollary 1. *Let $\mathcal{G}$ be the class of co-bipartite graphs. For every integer $j \geq 3$, $R_{\mathcal{G}}(i,j) = 2i-1$.*

5 Conclusions and other Graph Classes

A question that emerges from our results is the following. For which superclasses of line graphs or perfect graphs can we hope to find exact formulas for their Ramsey numbers? Line graphs form a subclass of claw-free graphs, and they are not perfect. Matthews [9] showed that when $\mathcal{G}$ is the class of claw-free graphs, $R_{\mathcal{G}}(i,3) = R(i,3)$ for every positive integer i, which implies that determining Ramsey numbers for claw-free graphs can be as difficult as it is for arbitrary graphs. We show that his argument can be applied to other graphs classes as well.

Theorem 9. *Let j be a positive integer and let $\mathcal{G}(j)$ be the class of $\overline{K_j}$-free graphs. Then $R_{\mathcal{G}(j)}(i,j) = R(i,j)$ for every positive integer i.*

Proof. By definition, $R_{\mathcal{G}(j)}(i,j) \leq R(i,j)$ for all positive integers i,j. For all positive integers i and j, there is a graph G on $R(i,j)-1$ vertices that contains neither K_i nor $\overline{K_j}$ as an induced subgraph. Since G is $\overline{K_j}$-free, we have $R_{\mathcal{G}(j)}(i,j) \geq |V(G)|+1 = R(i,j)$. □

Note that for any two classes of graphs $\mathcal{G}$ and $\mathcal{G}'$, if $\mathcal{G} \subseteq \mathcal{G}'$ then $R_{\mathcal{G}(j)}(i,j) \leq R_{\mathcal{G}'(j)}(i,j)$ for all positive integers i,j. Consequently, for any class $\mathcal{G}'$ that contains the class of $\overline{K_j}$-free graphs, we also have $R_{\mathcal{G}'}(i,j) = R(i,j)$ for every positive integers i,j. In particular, the case $j=3$ implies that finding a general formula for all Ramsey numbers for the classes of claw-free, AT-free and P_ℓ-free ($\ell \geq 5$) graphs is as hard as finding a formula for general graphs. Is it possible to find a formula for $R_{\mathcal{G}}(i,j)$ when $\mathcal{G}$ is one of these graph classes and $j \geq 4$?

Further positive results could be achieved for graph classes that are not perfect and that are not superclasses of claw-free, AT-free or P_5-free graphs. We raise the question of computing the exact values of Ramsey numbers for such classes of graphs, e.g., circle graphs, unit disk graphs, and quasi-line graphs.

References

1. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph classes: a survey. SIAM (1999)
2. Chudnovsky, M., Robertson, N., Seymour, P., Thomas, R.: The strong perfect graph theorem. Ann. Math. 164, 51–229 (2006)
3. Chudnovsky, M., Seymour, P.: The structure of claw-free graphs. In: Surveys in Combinatorics 2005. London Math. Soc. Lecture Note Ser., p. 327 (2005)
4. Cockayne, E.J., Lorimer, P.J.: On Ramsey graph numbers for stars and stripes. Canad. Math. Bull. 18, 31–34 (1975)
5. Diestel, R.: Graph Theory, Electronic edition. Springer (2005)
6. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Annals of Disc. Math. 57 (2004)
7. Graham, R.L., Rothschild, B.L., Spencer, J.H.: Ramsey Theory, 2nd edn. Wiley (1990)
8. Harary, F.: Graph Theory. Addison-Wesley (1969)
9. Matthews, M.M.: Longest paths and cycles in $K_{1,3}$-free graphs. Journal of Graph Theory 9, 269–277 (1985)
10. Radziszowski, S.P.: Small Ramsey numbers. Electronic Journal of Combinatorics, Dynamic Surveys (2011)
11. Ramsey, F.P.: On a problem of formal logic. Proc. London Math. Soc. Series 2, vol. 30, pp. 264–286 (1930)
12. Spencer, J.H.: Ten Lectures on the Probabilistic Method. SIAM (1994)
13. Steinberg, R., Tovey, C.A.: Planar Ramsey numbers. J. Combinatorial Theory Series B 59, 288–296 (1993)
14. Vizing, V.G.: On an estimate of the chromatic class of a p-graph. Diskret. Analiz. 3, 25–30 (1964) (in Russian)
15. Walker, K.: The analog of Ramsey numbers for planar graphs. Bull. London Math. Soc. 1, 187–190 (1969)

Geodesic Order Types*

Oswin Aichholzer[1], Matias Korman[2], Alexander Pilz[1], and Birgit Vogtenhuber[1]

[1] Institute for Software Technology, Graz University of Technology, Austria
{oaich,apilz,bvogt}@ist.tugraz.at
[2] Universitat Politècnica de Catalunya (UPC), Barcelona
matias.korman@upc.edu

Abstract. The geodesic between two points a and b in the interior of a simple polygon P is the shortest polygonal path inside P that connects a to b. It is thus the natural generalization of straight line segments on unconstrained point sets to polygonal environments. In this paper we use this extension to generalize the concept of the order type of a set of points in the Euclidean plane to geodesic order types. In particular, we show that, for any set S of points and an ordered subset $\mathcal{B} \subseteq S$ of at least four points, one can always construct a polygon P such that the points of $\mathcal{B}$ define the geodesic hull of S w.r.t. P, in the specified order. Moreover, we show that an abstract order type derived from the dual of the Pappus arrangement can be realized as a geodesic order type.

1 Introduction

Order types are one of the most fundamental combinatorial descriptions of sets of points in the plane. For each triple of points the order type encodes its orientation and thus reflects most of the combinatorial properties of the given set. We are interested in how much the order type of a point set changes when the points lie inside a simple polygon, and the orientation of point triples is given with respect to the geodesic paths connecting them. As depicted in Fig. 1, this orientation can change depending on the polygon. In this paper we develop a generalization of point set order types to the concept of geodesic order types.

In set theory, order types impose an equivalence relation between sets of points in the Euclidean plane. Two sets have the same *order type* if there is a bijection between them that is order preserving [11, pp. 50–51]. Goodman and Pollack [6] extend this concept to finite, multidimensional sets. They define that two d-dimensional point sets S_1 and S_2 have the same *point set order type* when there exists a bijection σ between the sets such that each $(d+1)$-tuple in S_1 has the same orientation (i.e., the side of the hyperplane defined by $p_1 \dots p_d$ on which the point p_{d+1} lies) as its corresponding tuple in S_2. It is also common to consider

* Research supported by the ESF EUROCORES programme EuroGIGA - ComPoSe, Austrian Science Fund (FWF): I 648-N18 and grant EUI-EURC-2011-4306. M. K. received support of the Secretary for Universities and Research of the Ministry of Economy and Knowledge of the Government of Catalonia and the European Union. A.P. is recipient of a DOC-fellowship of the Austrian Academy of Sciences at the Institute for Software Technology, Graz University of Technology, Austria.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 216–227, 2012.

two point sets to be of the same order type if all orientations are inverted in the second set. In the plane, this means that for two sets of the same order type, the ordered point triple u, v and w has the same orientation (clockwise or counterclockwise) as $\sigma(u)$, $\sigma(v)$, $\sigma(w)$. The infinitely many different point sets of a given cardinality can therefore be partitioned into a finite collection of order types. The orientations of all triples of the point set determine for any two given line segments whether they cross. Therefore, the order type defines most of the combinatorial properties of a point set.[1] For example, its convex hull, planarity of a given geometric graph (e.g., a triangulation), its rectilinear crossing number, etc. only depend on the order type. One might wonder whether every (consistent) assignment of orientations to triples of an abstract set allows a realization as a point set in the Euclidean plane. This is in general not true, not even if the assignment fulfills axiomatic requirements. See Knuth's monograph [12] for a detailed and self-contained discussion of this topic.

Generalizing classic geometric results to geodesic environments is a well-studied topic. For example, Toussaint [16] generalized the concept of convex hulls of point sets to geodesic environments. Other topics like Voronoi Diagrams [2], Ham-sandwich Cuts [3], Linear Programming [4], etc. have also been covered. However, to the best of our knowledge, the concept of geodesic order types has not been studied in the literature. Hence, it constitutes a natural and general extension to the above results.

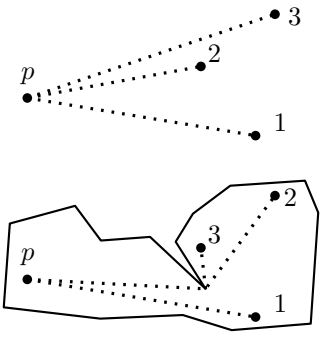


Fig. 1. The radial order of shortest paths to points around a point p can be different in unconstrained and geodesic settings

The classic order type is often used to identify extremal settings for combinatorial problems on point sets. For example, finding sets which minimize the number of crossings in a complete geometric graph, or maximize the number of elements of a certain class of graphs (spanning trees, matchings, etc.) are typical applications. In a similar spirit, the geodesic order type might be used to investigate extremal properties in geodesic environments. Examples might be problems on pseudo-triangulations (the side chains of a pseudo-triangle are geodesics), guarding problems inside polygonal boundaries (there, shortest paths are geodesics), and related problems; see, e.g., [15] for a recent survey on pseudo-triangulations.

[1] It is common to regard the properties defined by orientations of triples as the combinatorial ones. There are further settings on point sets that can be seen as being combinatorial as well, e.g., asking whether the fourth point of a quadruple lies inside the circle defined by the first three ones (see [12]). Also, the circular sequence of a point set is a richer way of describing the combinatorics of point sets, totally implying the order type [9].

1.1 Preliminaries

A closed polygonal path P is called a *simple* polygon if no point of the plane belongs to more than two edges of P, and the only points that belong to exactly two edges are the vertices of P. A closed polygonal path Q is a *weakly simple* polygon if every pair of points on its boundary separates Q into two polygonal chains that have no proper crossings, and if the angles of a complete traversal of the boundary of Q sum up to 2π [16]. Observe that a simple polygon is a weakly simple polygon, but the reverse is not true. Unless stated otherwise, all polygons are considered to be simple herein. We will follow the convention of including both, the interior and the boundary of a polygon, when referring to it. The boundary of polygon P will be denoted by ∂P.

The *geodesic* $\pi(s,t,P)$ between two points $s,t \in P$ in a simple polygon P is defined as the shortest path that connects s to t, among all the paths that stay within P. If P is clear from the context, we simply write $\pi(s,t)$. It is well known from earlier work that there always exists a unique geodesic between any two points [13], even if P is weakly simple. Moreover, this geodesic is either a straight line segment or a polygonal chain whose vertices (other than its endpoints) are reflex vertices of P. Thus, we sometimes denote the geodesic as the sequence of these reflex vertices traversed in the geodesic (i.e., $\pi(s,t) = \langle s = v_0, v_1, \ldots, v_k = t \rangle$). When the geodesic $\pi(s,t)$ is a segment, we say that s *sees* t (and vice versa).

For any fixed polygon P, a region $C \subseteq P$ is *geodesically convex* (also called *relative convex*) if for any two points $p,q \in C$, we have $\pi(p,q,P) \subseteq C$. The *geodesic hull* (*relative convex hull*) $\mathrm{CH}_P(U)$ of a set U is defined as the smallest (in terms of inclusion) geodesically convex region C that contains U. We will denote by $\mathrm{CH}(U)$ the standard Euclidean convex hull. Whenever a point $p \in U$ is in the boundary of $\mathrm{CH}_P(U)$, we say that p is an *extreme* point of U (with respect to P). The set of all such extreme points is called the *extreme* set of U, and is denoted by $E_P(U)$.

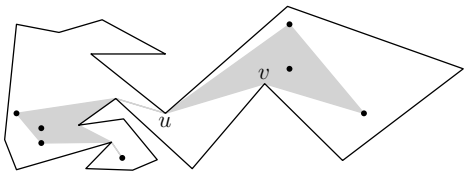


Fig. 2. Seven points inside a polygon P and their geodesic hull (marked in gray). Observe that the boundary of the geodesic hull consists of the concatenation of the shortest paths connecting the extreme vertices of S, in circular order. Further note that a vertex of the geodesic hull that stems from P can be a convex and a reflex vertex of the geodesic hull at the same time (like vertex u) or only a reflex vertex (like v).

Although these definitions are valid for any subset U of P, in this paper we will only use them for a finite set of points $S = \{p_1, \ldots, p_n\}$. Further note that the geodesic hull is a weakly simple polygon (see Fig. 2). From now on we assume that the points in the union of S with the set V of vertices of P are in *strong general* position. That is, there are no three collinear points, and, for any four distinct points $p_1, p_2, p_3, p_4 \in S \cup V$, the line passing through p_1 and p_2 is not parallel to the line passing through p_3 and p_4.

1.2 Orientations and Geodesics

The concept of *clockwise order* of a triple of points (p, q, r) naturally extends to geodesic environments. Let $\pi(p, q) = \langle p = v_0, \ldots, v_k = q \rangle$ and $\pi(p, r) = \langle p = u_0, \ldots, u_{k'} = r \rangle$ be the geodesics connecting p with q and r, respectively. Also, let $i > 0$ be the smallest index such that $v_i \neq u_i$. We say that (p, q, r) are in *geodesic clockwise order* if (v_{i-1}, v_i, u_i) are in (Euclidean) clockwise order. It is easy to see that, due to the strong general-position assumption, any triple is oriented either clockwise or counterclockwise in the geodesic environment. We adopt the common phrasing, and say that r is to the right of q (with respect to p) whenever (p, q, r) are in geodesic clockwise order (or that r is to the left, otherwise). By definition, if (p, q, r) are in geodesic clockwise order, then for any $a < i \leq b, c$, the triple (v_a, v_b, u_c) must also be in geodesic clockwise order. Hence, this definition also accounts for the intuitive perception of "left" and "right" when traversing the geodesics.

Note that "left" and "right" differ between the geodesic and the unconstrained setting, since we can use reflex vertices of the surrounding polygon to "reorder" unconstrained point triples. An illustration is shown in Fig. 3; in this example, the polygonal chain crosses two edges of the triangle and the supporting line of the third one. In general, this operation is not local, and might alter the order type of other triples (more details of this operation will be given in Section 3).

The orientation predicate can also be defined in terms of the geodesic hull $\mathrm{CH}_P(\{p, q, r\})$. When traversing this hull counterclockwise, the points appear in that order if and only if their geodesic orientation is counterclockwise.

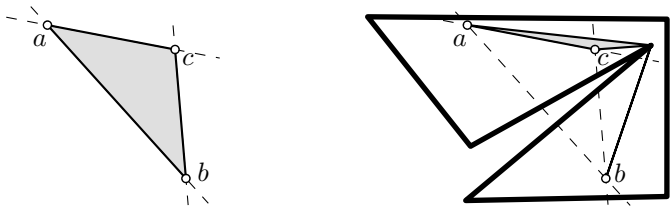


Fig. 3. Reordering a triangle using a polygonal chain. The triple (a, b, c) is in (Euclidean) counterclockwise order. However, upon introducing the polygon (right figure) the same triple is now in (geodesic) clockwise order.

1.3 Contribution

The triple orientation in geodesic environments extends the one in Euclidean environments. Since the latter defines the order type of a point set, we obtain a generalization of point set order types to *geodesic order types*. It is easy to see that the order type of a fixed point set S can change with different enclosing polygons. In particular, some points that appear in the (Euclidean) convex hull

may not be present in the geodesic hull and, vice versa, some non-extreme points of S may appear on the geodesic hull.

In this paper, we study the ways in which the set of extreme points of a given set S can change with the shape of the polygon. We show that any subset $\mathcal{B}$ of four or more points of S can become the extreme set of S (i.e., there exists a polygon P such that $E_P(S) = \mathcal{B}$). Moreover, we can make them appear in any predefined order along the boundary of the geodesic hull. We also characterize when this property is fulfilled for sets of size 3. Finally, we show in Section 3 that the abstract order types that can be realized as geodesic order types are a proper superset of the abstract order types realizable as Euclidean order types. Specifically, we show that the non-realizable abstract order type derived from Pappus' Theorem via duality can be realized as a point set inside a polygon. This can also be seen as the class of inverse problems to the classic questions for geodesic environments, where the polygon is usually part of the input.

2 Geodesic Hull versus Convex Hull

In this section, we study how much the geodesic hull of a given point set can alter from the Euclidean convex hull. We partition S into two sets of blue and red points ($\mathcal{B}$ and $\mathcal{R}$, respectively). A set $\mathcal{B}$ is said to be *separable* from $\mathcal{R}$ if there exists a polygon with at most $|\mathcal{B}|$ convex vertices (i.e., a pseudo-$|\mathcal{B}|$-gon) that contains all points of $\mathcal{R}$ and no point of $\mathcal{B}$ in its interior. From now on, we assume that the set S is fixed. Thus we omit writing "from $\mathcal{R}$" and simply refer to $\mathcal{B}$ as a separable point set. The following theorem draws a nice connection between the separability of point sets and their geodesic hull.

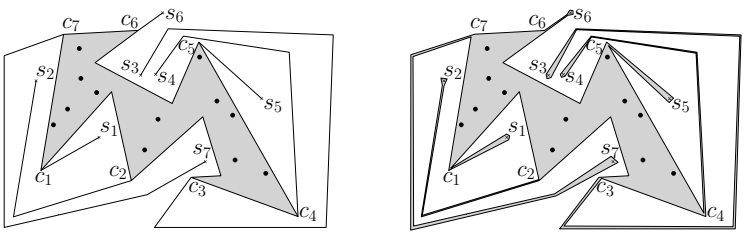


Fig. 4. Illustration of the proof of Theorem 1: with a one-to-one correspondence between the convex vertices $c_1, \ldots, c_7$ of P and the points $s_1, \ldots, s_7$ of $\mathcal{B}$, we can obtain a weakly simple polygon P' such that $E_{P'}(S) = \{s_1, \ldots, s_7\}$ (left), which then can be transformed to a polygon (right)

Theorem 1. *For any separable point set $\mathcal{B}$ and any permutation σ of $\mathcal{B}$, there exists a polygon P such that $E_P(S) = \mathcal{B}$ and the clockwise ordering of $\mathcal{B}$ on the boundary of $\mathrm{CH}_P(S)$ is exactly σ.*

Proof. Let $k = |\mathcal{B}|$ and P be a separating polygon of $\mathcal{B}$. If P has strictly less than k convex vertices, we introduce more by replacing any edge e by two edges, adding a convex vertex arbitrary close to the center point of e. Thus, we assume that P has k convex vertices $c_1, \ldots, c_k$.

Let $s_1, \ldots, s_k$ be an arbitrary ordering of the vertices of $\mathcal{B}$. For all $i \leq k$, we connect point $s_i \in \mathcal{B}$ to c_i by a polygonal chain. Observe that we can always do this in a way that no two chains cross. Now let P' be the union of P and the polygonal chains, see Fig. 4 (left). The union of geodesics connecting s_i with s_{i+1} (and s_k with s_1) exactly corresponds to the boundary of P'. Moreover, all points of $\mathcal{R}$ are in the interior of P'. Notice that P' is not a polygon, but a weakly simple polygon. As illustrated in Fig. 4 (right), we obtain a polygon from P' by transforming polygonal paths into narrow passages of width at most ε (for a sufficiently small ε, and such that no blue point sees any other blue point). □

We now study the separability of a point set as a function of its size. Surprisingly, the separability of the set $\mathcal{B}$ does not strongly depend on the set $\mathcal{R}$.

Theorem 2. *Any set $\mathcal{B}$ with cardinality $|\mathcal{B}| \geq 5$ is separable with a polygon with at most $2|\mathcal{B}| - 2$ vertices.*

In order to prove the above theorem, we first consider some simpler cases and then show how to deal with larger point sets. The proofs for the following statements are by construction of according polygons and can be found in the full version of this paper.

Lemma 1. *Any set $\mathcal{B}$ of five points is separable.*

The main idea of the proof is sketched in Fig. 5.

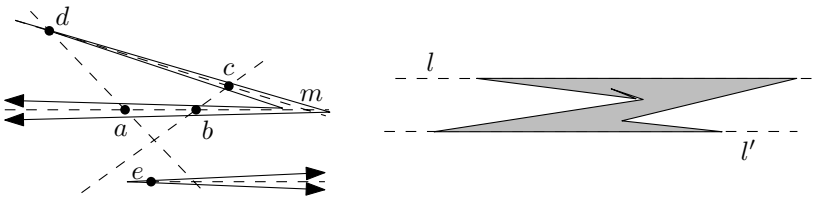


Fig. 5. A set of five points is always separable. A narrow, bent spike can be built around the empty convex quadrilateral of the set. A second spike is chosen parallel to the first one and in opposite direction. Sufficiently far away, the spike end points span a quadrilateral around the whole set.

Corollary 1. *Any subset $\mathcal{B}$ of four points in convex position is separable.*

Lemma 2. *For any set $\mathcal{B}$ separable by a polygon P and a point $q \notin S$, the set $\mathcal{B} \cup \{q\}$ is separable by a polygon P' having at most two more vertices than P.*

The class of polygons constructed in the proof of Lemma 1 will never have more than 8 vertices. Moreover, by Lemma 2, each additional point of $\mathcal{B}$ will add at most 2 additional vertices to the separating polygon. In particular, we will always have a separating polygon P whose number of edges is at most $2|\mathcal{B}| - 2$, which completes the proof of Theorem 2.

By definition, any point set of size 1 or 2 cannot be separated (since we cannot construct a simple polygon with one or two convex vertices). Hence, it remains to consider the cases in which $|\mathcal{B}| \in \{3, 4\}$. To this end, we say that a set $\mathcal{R}$ *ε-densely* covers $\mathcal{B}$ (for any $\varepsilon > 0$) if any wedge emanating from $p \in \mathcal{B}$ and not containing any point of $\mathcal{R}$ has an opening angle of at most ε. Observe that, if $\mathcal{R}$ ε-densely covers $\mathcal{B}$, no point of $\mathcal{B}$ can appear on the boundary of $\mathrm{CH}(S)$. Moreover, if $\varepsilon \leq \pi/3$, any convex region that contains three or more blue points must contain a red point. Showing that for any set $\mathcal{R}$ that ε-densely covers $\mathcal{B}$ (for some sufficiently small ε), $\mathcal{B}$ cannot be separated from $\mathcal{R}$, we obtain the following result (the proof is deferred to the full version of this paper).

Theorem 3. *For any set $\mathcal{B}$ of three points or four points in non-convex position, there exists a set $\mathcal{R}$ such that $\mathcal{B}$ is not separable from $\mathcal{R}$.*

The example in Fig. 6 shows a point set where the four blue points lie on the geodesic hull but are not separable. This implies, in contrast to sets of larger cardinality, that for $|\mathcal{B}| = 4$, the concepts of separability and geodesic hull are not equivalent. Thus, we switch back to the geodesic setting and consider the remaining cases $|\mathcal{B}| \in \{3, 4\}$.

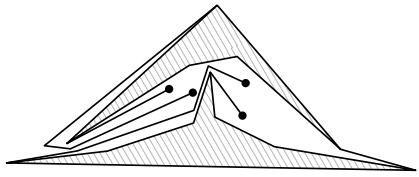


Fig. 6. Two pseudo-triangles containing many red points such that the four blue points are not separable. However, they are the extreme vertices w.r.t. some polygon.

Lemma 3. *For any set S, any set $\mathcal{B} \subset S$ of four points, and any permutation σ of $\mathcal{B}$, there exists a polygon P such that $E_P(S) = \mathcal{B}$ and the clockwise ordering of $\mathcal{B}$ on the boundary of $\mathrm{CH}_P(S)$ is exactly σ.*

Proof. If the points of $\mathcal{B}$ are in convex position, then the statement follows directly from Corollary 1 and Theorem 1. Thus, assume that $\mathcal{B}$ is not in convex position. Consider a line l_1 spanned by two of the extreme points of $\mathcal{B}$, and a line l_2 that is parallel to l_1 and passes through the third extreme point of $\mathcal{B}$ (see Fig. 7). We construct two pseudo-triangles P_1 and P_2, each with four edges, with the following properties: (1) P_1 has a convex and a reflex vertex on l_1, such that the reflex vertex is between the convex vertex and both blue points on l_1. (2) Accordingly, P_2 has a convex and a reflex vertex on l_2, such that the reflex vertex is between the convex vertex and the blue vertex on l_2. (3) Both, P_1 and P_2, have a vertex between l_1 and l_2, and the edges connecting the convex point on l_1 (l_2) to these vertices are parallel. (4) The non-extreme point of $\mathcal{B}$ lies between P_1 and P_2. (5) All red points lie inside P_1 or P_2. Note that these properties can always be fulfilled, as the convex points of the pseudo-triangles

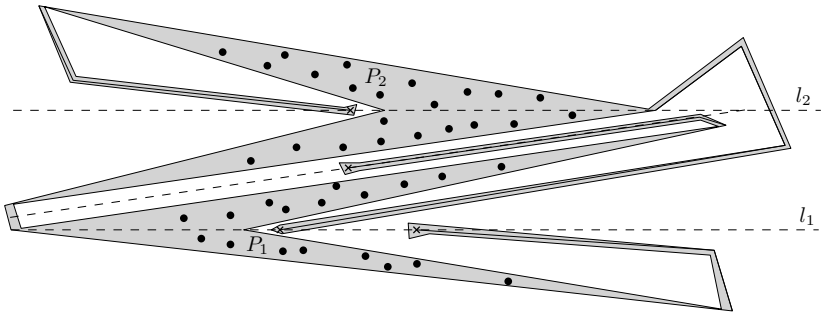


Fig. 7. Construction for a polygon P with $E_P(S) = \mathcal{B}$ based on two pseudo-triangles that contain all red points (depicted with dots) and none of the blue points (drawn as crosses)

can be far away, and thus the reflex angles can be made arbitrarily small and the area covered by the pseudo-triangles can be arbitrarily "thick".

As indicated in Fig. 7, we can merge the two pseudo-triangles to form a polygon by adding a narrow passage from a convex vertex of P_1 to a convex vertex of P_2. To obtain our final polygon P with $E_P(S) = \mathcal{B}$ in the desired order, we proceed like in the proof of Theorem 1, connecting the blue points to the four convex vertices of P_1 and P_2 that were not used for the passage between P_1 and P_2. □

If we combine this result with Theorems 1 and 2 we obtain the following statement.

Theorem 4. *For any set S, any set $\mathcal{B} \subset S$ of at least four points, and any permutation σ of $\mathcal{B}$, there exists a polygon P such that $E_P(S) = \mathcal{B}$ and the clockwise ordering of $\mathcal{B}$ on $E_P(S)$ is exactly σ.*

We conclude this section by studying what happens when the set $\mathcal{B}$ has cardinality three.

Theorem 5. *Let $\mathcal{B} \subset S$ be a set with $|\mathcal{B}| = 3$ such that $\mathcal{B}$ is the geodesic hull of S for some polygon P. Then $\mathcal{B}$ is separable.*

Proof. Recall that the geodesic hull of S is a weakly simple polygon which has all points of $\mathcal{B}$ on its boundary, and contains all points of $S \setminus \mathcal{B}$ in its interior. Moreover, a vertex v of the geodesic hull can only be convex if (1) $v \in \mathcal{B}$, or (2) v is part of some weakly simple polygonal chain and thus coincides with a reflex vertex of the geodesic hull. Thus, as $|\mathcal{B}| = 3$, the geodesic hull must consist of a pseudo-triangle Δ, possibly with polygonal chains attached to the convex vertices of Δ, where each blue vertex corresponds to one convex vertex of Δ (see Fig. 8). By slightly shrinking Δ, we obtain a pseudo-triangle Δ' still having all points of $S \setminus \mathcal{B}$ in its interior that leaves all points of $\mathcal{B}$ outside. Thus Δ' is a separating polygon for $\mathcal{B}$. □

Table 1. Overview of results and relationship between pushable and separable

$\lvert\mathcal{B}\rvert$	Pushable		Separable
≤ 2	never (Def.)	$\Leftrightarrow$	never (Def.)
3	not always	$\Leftrightarrow$ (Thm. 1 and 4)	not always (Thm. 3)
4	always (Thm. 3)	$\Leftarrow$ (Thm. 1)	convex position: always (Cor. 1) non-convex: not always (Thm. 3)
≥ 5	always (Thm. 4)	$\Leftrightarrow$	always (Thm. 2)

Together with Theorem 3 the above result implies that there exist point sets S with $|\mathcal{B}| = 3$ such that $\mathcal{B}$ can not be used to define the geodesic hull of S. This is in contrast to the fact that for any set with $|\mathcal{B}| \geq 4$ this is always possible. Table 1 gives an overview of the obtained results and also shows the relation between a set being 'pushable' (meaning that there is a polygon such that $\mathcal{B}$ is on the geodesic hull) and 'separable' for different cardinalities of $\mathcal{B}$.

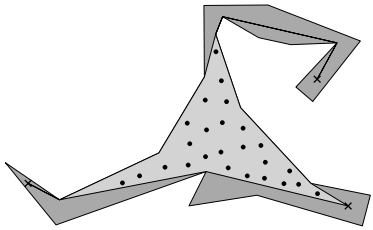
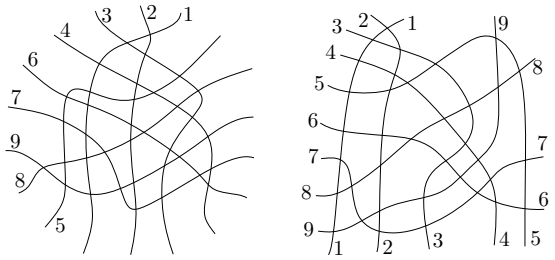


Fig. 8. A set $\mathcal{B} \subset S$ with $|\mathcal{B}| = 3$, and a polygon P (dark shaded) with $E_P(S) = \mathcal{B}$ (depicted $\times$). The geodesic hull is drawn light shaded.

3 Realizing the Pappus Arrangement

By duality, every set of points in the d-dimensional Euclidean space corresponds to an arrangement of hyperplanes in the same space (see e.g. [5] for details on this mapping). This dual is incidence and order preserving. When traversing a line u^* in the plane, the order in which the lines v^* and w^* are crossed gives the orientation of the corresponding point triple u, v, w in the primal setting [8]. Hence, the crossings in the line arrangement determine the order type of the corresponding point set. An *arrangement of pseudo-lines* is a set of simple curves such that each pair has exactly one point in common, and at this point the pair crosses. The crossings in the pseudo-line arrangement define an *abstract order type*. Obviously, if we can stretch the curves to straight lines without changing the order of all crossings, we obtain a realization of the order type defined by the crossings. This has been used in the exhaustive enumeration of point set order types [1]. However, for sets of size 9 or more, it is known that there exist non-realizable abstract order types (i.e., pseudo-line arrangements that are non-stretchable). The example for 9 pseudo-lines is based on the well-known Pappus' Theorem [10,14].

Using the axiomatic system of [12, p. 4], one can show that all geodesic order types are in fact a subset of abstract order types, i.e., of those that are defined by pseudo-line arrangements (details are given in the full paper).



278	345	467	567	678
279	348	468	568	679
	368	469	569	
	378	478	578	
	379	479	579	
			589	

Fig. 9. A non-stretchable pseudo-line arrangement derived from Pappus' Theorem, adapted from [7, Fig. 5.3.2] (left). The transformed arrangement, having all lines crossing line 1 first (middle). All ascending counterclockwise point triples derived from the arrangement (right).

3.1 The Arrangement

The non-stretchable arrangement whose abstract order type we realize in the geodesic setting is an adaption from the one shown in [7, p. 107], see Fig. 9 (left). It is well-known that this pseudo-line arrangement cannot be stretched and thus the corresponding abstract order type cannot be realized by a point set. From the correspondence between a straight line in the Euclidean plane to a great circle in the sphere model of the projective plane, it is easy to see that an arrangement is stretchable in the real plane if and only if it is stretchable in the projective plane. We can therefore apply projective transformations to the arrangement without affecting its realizability. In this way we transform the arrangement of [7] to the *standard labeling*, see Fig. 9 (middle) for the resulting drawing. Roughly speaking, the crossings of a pseudo-line that happen before the crossing with l_1 are "moved" to the other side. Namely, these are the crossing of l_9 with l_8 and the crossings of l_5 with l_9, l_8, l_7, and l_6, in the given order. We do so in order to make all pseudo-lines cross pseudo-line l_1 before any other. In the primal, this corresponds to p_1 being on the convex hull boundary and points $p_2, \ldots, p_9$ being sorted clockwise around it. Note that this kind of projective transformation actually preserves the order type. Fig. 9 (right) shows all triples with ascending indices that have counterclockwise orientation (which easily allows obtaining the orientation of all triples). For example, the entry "278" indicates that pseudo-line l_2 crosses l_8 before l_7, inducing counterclockwise orientation of the point triple $p_2p_7p_8$ in the primal.

3.2 The Realization

Consider the point set $S = \{p_1, \ldots, p_9\}$ shown in Fig. 10 (left). The only triples whose orientations do not match those indicated by Fig. 9 are the permutations of p_2, p_7 and p_9. Equivalently, one can say that the triangle defined by the three points is the only one that has the wrong orientation among all triangular subgraphs of the complete graph of S. This triangle is shown with thick (blue) edges.

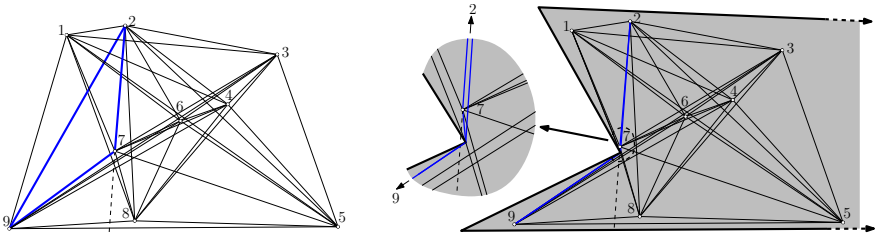


Fig. 10. A point set that "almost" realizes the unrealizable arrangement (left). The point triple spanning the thick blue triangle $\Delta p_2 p_7 p_9$ is the one for which the orientation is wrong. A geodesic realization of the arrangement (right). The shortest paths between the points are geodesics in the interior of the polygon (gray). The region of interest is shown in detail in the middle. The polygon closes with a convex vertex far on the right side, as indicated.

We already discussed how reflex vertices of a surrounding polygon can change the orientation of a triple. The problem with this tool is that the polygonal chain is likely to reorder other triangles as well. In the point set shown in Fig. 10 (left), this tool can, however, be applied. We create a polygon P that contains S. The result of the construction is shown in Fig. 10 (right). We cross four edges during this operation. Note that the geodesics $\pi(p_1, p_9, P)$ and $\pi(p_1, p_8, P)$ are now no longer line segments, still the order defined by their end vertices has not changed. The triple p_2, p_7, p_9, however, is now oriented counterclockwise, as demanded by the abstract order type. By checking all the point triples, the reader can verify that this geodesic order type indeed realizes the abstract order type of the Pappus arrangement.

Theorem 6. *There exists a point set S and a polygon whose geodesic order type realizes an abstract order type that is not realizable as a point set in the plane.*

We note that our construction is minimal; that is, there cannot exist a point set of nine points and a polygon of fewer vertices (than the one given in Fig. 10, right) that realize the Pappus arrangement.

4 Conclusion

In this paper, we made a first step into generalizing the concept of point set order types to geodesic order types. For a selection of four or more points out of a set S, we showed how to construct a polygon such that exactly these vertices are on the geodesic hull of S, in any order desired. To the contrary, this is not always possible for three points. We further showed an example of an abstract order type that is not realizable in the Euclidean plane, but is realizable in geodesic environments.

Several interesting questions rise from our investigations. Which bounds on the number of vertices in the polygon that forces the desired geodesic hull can

we derive? What is the complexity of minimizing the number of vertices? Even though we showed the realizability of the abstract order type derived from Pappus' Theorem, we have no general tools to realize order types inside polygons. Can every abstract order type (which is non-realizable in the Euclidean plane) be realized as a geodesic order type? And which of them can be realized in a given polygon?

References

1. Aichholzer, O., Krasser, H.: Abstract order type extension and new results on the rectilinear crossing number. Comput. Geom. 36(1), 2–15 (2007)
2. Aronov, B.: On the geodesic voronoi diagram of point sites in a simple polygon. In: SCG 1987: Proceedings of the Third Annual Symposium on Computational Geometry, pp. 39–49. ACM, New York (1987)
3. Bose, P., Demaine, E.D., Hurtado, F., Iacono, J., Langerman, S., Morin, P.: Geodesic ham-sandwich cuts. In: SCG 2004: Proceedings of the Twentieth Annual Symposium on Computational Geometry, pp. 1–9. ACM, New York (2004)
4. Demaine, E.D., Erickson, J., Hurtado, F., Iacono, J., Langerman, S., Meijer, H., Overmars, M.H., Whitesides, S.: Separating point sets in polygonal environments. Int. J. Comput. Geometry Appl. 15(4), 403–420 (2005)
5. Edelsbrunner, H.: Algorithms in combinatorial geometry. Springer-Verlag New York, Inc., New York (1987)
6. Goodman, J.E., Pollack, R.: Multidimensional sorting. SIAM Journal on Computing 12(3), 484–507 (1983)
7. Goodman, J.E.: Pseudoline arrangements. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, pp. 83–109. CRC Press, Inc., Boca Raton (1997)
8. Goodman, J.E., Pollack, R.: A theorem of ordered duality. Geometriae Dedicata 12(1), 63–74 (1982)
9. Goodman, J., Pollack, R.: On the combinatorial classification of nondegenerate configurations in the plane. J. Combin. Theory Ser. A 29, 220–235 (1980)
10. Grünbaum, B.: Arrangements and spreads. In: Regional Conference series in mathematics, Published for the Conference Board of the Mathematical Sciences by the American Mathematical Society (1972)
11. Hausdorff, F.: Set theory. AMS Chelsea Publishing Series. American Mathematical Society (1957)
12. Knuth, D.E.: Axioms and Hulls. LNCS, vol. 606. Springer, Heidelberg (1992)
13. Mitchell, J.S.B.: Shortest paths among obstacles in the plane. Int. J. Comput. Geometry Appl. 6(3), 309–332 (1996)
14. Richter, J.: Kombinatorische Realisierbarkeitskriterien für orientierte Matroide. Mitteilungen Math. Seminar Gießen, Selbstverlag des Mathematischen Seminars (1989) (in German)
15. Rote, G., Santos, F., Streinu, I.: Pseudo-triangulations — a survey. In: Goodman, E., Pach, J., Pollack, R. (eds.) Surveys on Discrete and Computational Geometry — Twenty Years Later, Contemporary Mathematics, vol. 453, pp. 343–411. American Mathematical Society, Providence (2008)
16. Toussaint, G.T.: Computing geodesic properties inside a simple polygon. Revue D'Intelligence Artificielle 3(2), 9–42 (1989)

Computing Partitions of Rectilinear Polygons with Minimum Stabbing Number*

Stephane Durocher and Saeed Mehrabi

Department of Computer Science, University of Manitoba, Winnipeg, Canada
{durocher,mehrabi}@cs.umanitoba.ca

Abstract. The stabbing number of a partition of a rectilinear polygon P into rectangles is the maximum number of rectangles stabbed by any axis-parallel line segment contained in P. We consider the problem of finding a rectangular partition with minimum stabbing number for a given rectilinear polygon P. First, we impose a *conforming* constraint on partitions: every vertex of every rectangle in the partition must lie on the polygon's boundary. We show that finding a conforming rectangular partition of minimum stabbing number is NP-hard for rectilinear polygons with holes. We present a rounding method based on a linear programming relaxation resulting in a polynomial-time 2-approximation algorithm. We give an $O(n \log n)$-time algorithm to solve the problem exactly when P is a histogram (some edge in P can see every point in P) with n vertices. Next we relax the conforming constraint and show how to extend the first linear program to achieve a polynomial-time 2-approximation algorithm for the general problem, improving the approximation factor achieved by Abam, Aronov, de Berg, and Khosravi (ACM SoCG 2011).

1 Introduction

A polygon P is rectilinear if all of its edges are axis-parallel. A *rectangular partition* of a rectilinear polygon P is a decomposition of P into rectangles whose interiors are disjoint. Rectilinear polygon decomposition has several applications, including VLSI layout design [10] and image processing [6].

Let P be a rectilinear polygon and let R be a rectangular partition of P. Given a line segment ℓ inside P, we say that ℓ *stabs* a rectangle of R if ℓ passes through the interior of the rectangle. The (*rectilinear*) *stabbing number* of R is the maximum number of rectangles of R stabbed by any axis-parallel line segment inside P. Moreover, the *vertical* (resp., *horizontal*) stabbing number of R is defined as the maximum number of rectangles stabbed by any vertical (resp., horizontal) line segment inside P. We say an edge of a rectangle in a rectangular partition of P is *fully anchored* if both of its endpoints are on the boundary of P. Consequently, a rectangular partition of P is called *conforming*,

* Work supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 228–239, 2012.

if all edges of its rectangles are fully anchored. A conforming rectangular (CR) partition of P is *optimal* if its stabbing number is minimum over of all such partitions of P.

De Berg and van Kreveld [3] prove that every n-vertex rectilinear polygon has a rectangular partition with stabbing number $O(\log n)$. They show that this bound is asymptotically tight, as the stabbing number of any rectangular partition of a staircase polygon with n vertices is $\Omega(\log n)$. De Berg and van Kreveld [3] and Hershberger and Suri [7] give polynomial-time algorithms that compute partitions with stabbing number $O(\log n)$. Recently, Abam et al. [1] consider the problem of computing an optimal rectangular partition of a simple rectilinear polygon P, that is, a rectangular partition whose stabbing number is minimum over all such partitions of P. By finding an optimal partition for histogram polygons in polynomial time (see Section Section 3), they obtain an $O(n^7 \log n \log\log n)$-time 3-approximation algorithm for this problem. As Abam et al. note, however, the computational complexity of the general problem is unknown.

De Berg et al. [2] studied a related problem in which the objective is to partition a given set of n points in $\mathbb{R}^d$ into sets of cardinality between $n/2r$ and $2n/r$ for a given r, where each set is represented by its bounding box, such that the stabbing number, defined as the maximum number of bounding boxes intersected by any axis-parallel hyperplane, is minimized. They show the problem is NP-hard in $\mathbb{R}^2$. They also give an exact $O(n^{4dr+3/2} \log^2 n)$-time algorithm in $\mathbb{R}^d$ as well as an $O(n^{3/2} \log^2 n)$-time 2-approximation algorithm in $\mathbb{R}^2$ when r is constant. Fekete et al. [5] prove that the problem of finding a perfect matching with minimum stabbing number for a given point set is NP-hard, where the (rectilinear) stabbing number of a matching is the maximum number of edges of the matching intersected by any (axis-parallel) line. They also show that, for a given point set, the problems of finding a spanning tree or a triangulation with minimum stabbing number are NP-hard.

This paper examines the problem of finding an optimal rectangular partition of a given rectilinear polygon, both in the unrestricted version of the problem (partitions need not be conforming) considered by Abam et al. [1] and in the case of conforming rectangular (CR) partitions. For CR partitions, we give an $O(n \log n)$-time algorithm for computing an optimal partition when the input polygon is a histogram with n vertices in Section 3, a polynomial-time 2-approximation algorithm for arbitrary rectilinear polygons (possibly with holes) in Section 4, and we show NP-hardness for finding an optimal partition on rectilinear polygons with holes in Section 5. To the authors' knowledge, this is the first complexity result related to determining the minimum stabbing number of a rectangular partition of a rectilinear polygon, partially answering an open problem posed by Abam et al. [1]. For general (not necessarily conforming) rectangular partitions we give a polynomial-time 2-approximation algorithm in Section 6 that improves on the 3-approximation algorithm of Abam et al. [1]. Complete proofs for the results of Sections 5 and 6 are omitted due to space constraints.

2 Preliminaries

Let P be a simple rectilinear polygon and let R be a CR partition of P. We refer to any maximal line segment whose interior lies in the interior of P and on the boundary of some rectangle in R as a *partition edge*. That is, the partition edges of R correspond to the "cuts" that divide P into rectangles. A vertex u of P is a *reflex* vertex if the angle at u interior to P is $3\pi/2$. We denote the set of reflex vertices of P by $V_R(P)$. For each reflex vertex $u \in V_R(P)$, we denote the maximal horizontal (resp., vertical) line segment contained in the interior of P with one endpoint at u by H_u (resp., V_u) and refer to it as the *horizontal line segment* (resp., *vertical line segment*) of u. Observe that for every reflex vertex u of P, at least one of H_u and V_u must be present in R. The following observation allows us to consider only a discrete subset of the set of all possible rectangular partitions of P to find an optimal partition:

Observation 1. *Any rectilinear polygon P has an optimal rectangular partition in which every partition edge has at least one reflex vertex of P as an endpoint.*

Consequently, every partition edge is either H_u or V_u for some $u \in V_R(P)$.

Given an integer $k \geq 1$, a k-Sum Linear Program (KLP)[1] [11] consists of an $m \times n$ matrix A, an m-vector b, and an n-vector $X = (x_1, x_2, \ldots, x_n)$ for which the objective is to

$$\begin{aligned} \text{minimize} \quad & \max_{S \subseteq N : |S| = k} \sum_{j \in S} c_j x_j \\ \text{subject to} \quad & AX \geq b \\ & X \geq 0, \end{aligned} \tag{1}$$

where $N = \{1, 2, \ldots, n\}$. Observe that when $k = n$, the KLP is equivalent to a classical linear program (LP).

3 Finding an Optimal CR Partition of a Histogram

In this section, we present an algorithm for computing an optimal CR partition of a histogram. A *histogram* (polygon) H is a simple rectilinear polygon that has one edge e that can see every point in P. Equivalently, as defined by Katz and Morgenstern [8], a simple orthogonal polygon P is a vertical (resp., horizontal) histogram if it is monotone with respect to some horizontal (resp., vertical) edge e that spans P; we call e the *base* of H.

Abam et al. [1] give a polynomial-time algorithm for computing an optimal rectangular partition of a histogram. A rectangular partition of a histogram is not necessarily a CR partition. Figure 1(a) shows a histogram whose optimal rectangular partition has stabbing number 2. However, any CR partition of this histogram has stabbing number at least 3; see Figure 1(b). Without loss of generality, suppose each histogram is a vertical histogram.

[1] Throughout the paper, we use KLP to abbreviate either k-Sum Linear Program or k-Sum Linear Programming. Similarly, we use LP to denote either Linear Program or Linear Programming.

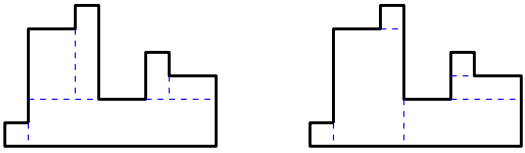


Fig. 1. A vertical histogram H. (a) An optimal rectangular partition of H with stabbing number 2. (b) Any CR partition of H has stabbing number at least 3.

Let H be a histogram with n vertices and let H^- denote the set of horizontal edges of H. Recall that every CR partition of H must include at least one of the edges H_u or V_u for every reflex vertex u in H. The algorithm begins with an initial partition of H, consisting exclusively of horizontal partition edges, that will be modified to produce an optimal CR partition of H by greedily replacing horizontal edges with vertical edges. The initial partition of H is obtained by adding the edge H_u for each reflex vertex u.

Observation 2. *For any* CR *partition of any vertical histogram H and any reflex vertex u in H, the vertical partition edge V_u may be included at u if and only if no horizontal partition edge is included directly below u (otherwise it would intersect V_u).*

Observation 2 suggests a hierarchical tree structure that determines a partial order in which each horizontal partition edge can be removed and replaced by a vertical partition edge, provided it does not intersect any horizontal partition edge below it. Thus, we construct a forest (initially a single tree denoted T_0) associated with the partition; the algorithm proceeds to update the forest and, in doing so, modifies the associated partition as horizontal partition edges are replaced by vertical ones. Define a tree node for each edge in $H^- \cup S$, where $S = \{H_u \mid u \in V_R(H)\}$. Add an edge between two vertices u and v if some vertical line segment intersects both edges associated with u and v, but no other edge of $H^- \cup S$. When the polygon H is a histogram, the resulting graph, T_0, is a tree. See the example in Figure 2(a). We now describe how to construct T_0 in $O(n \log n)$ time. Note that the set S need not be known before construction.

Each edge in H^- is adjacent to two vertical edges on the boundary of H, which we call its left and right neighbours, respectively. Sort the edges of H^- lexicographically, first by y-coordinates and then by x-coordinates. The algorithm sweeps a horizontal line ℓ across H from bottom to top. Initially, ℓ coincides with the base of H; root the tree T_0 at a node u that corresponds to the base of H. The construction refers to a separate balanced search tree that archives the set of vertical edges of H on or below the sweepline, indexed by x-coordinates. Initially, only the leftmost and rightmost vertical edges of H are in the search tree, i.e., the base's neighbours. The construction of the tree T_0 proceeds recursively on u as follows.

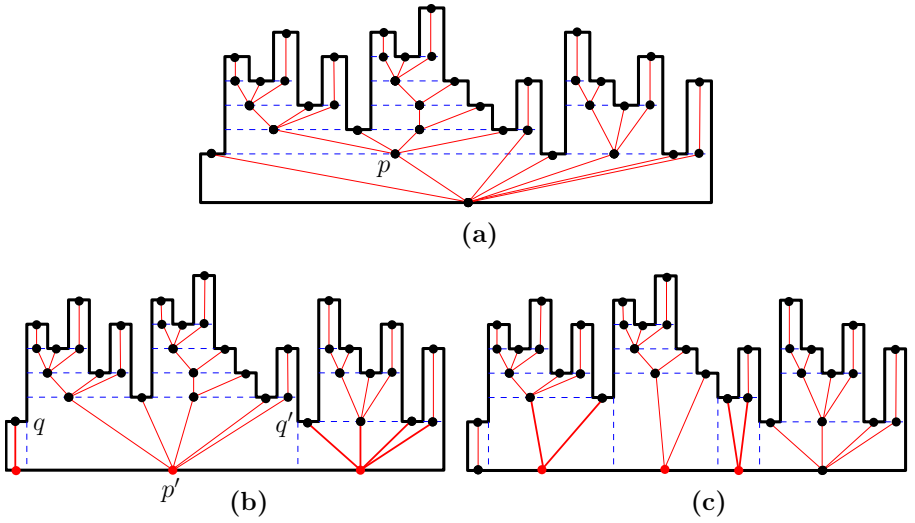


Fig. 2. (a) A histogram H and the tree T_0 that corresponds to the initial partition of H. (b) The edge associated with node p is removed from the partition and is replaced by two vertical edges anchored at the reflex vertices q and q'. The red vertices denote the roots of the three new resulting trees. (c) The algorithm terminates after one more iteration, giving an optimal CR partition of H (with stabbing number 5) along with the corresponding forest.

Suppose the next edges of H^- encountered by the sweepline ℓ are $e_1, \ldots, e_k$, each of which has equal y-coordinate. Add the respective left and right neighbours of $e_1, \ldots, e_k$ to the search tree. Let l_1 and r_1 denote the x-coordinates of the respective left and right endpoints of edge e_1. Add a node representing e_1 to T_0 as a child of u. Check whether the left neighbour of e_1 (indexed by l_1) lies below ℓ. If not, then find the predecessor of l_1 in the search tree and let y denote its x-coordinate. Let u' denote the line segment on line ℓ with respective endpoints at the x-coordinates y and l_1. Add a node representing u' to T_0 as a child of u. Recursively construct the subtree of u'. Apply an analogous procedure to the right neighbour of e_1 (indexed by r_1). Repeat for each edge $e_i \in \{e_2, \ldots, e_k\}$. Upon completion, the tree T_0 is constructed storing a representation of the initial horizontal partition (see Figure 2(a)). Finally, each tree node stores its height and links to its children in order of x-coordinates; the tree can be updated accordingly after construction. The running time for constructing T_0 is bounded by sorting $O(n)$ edges and a sequence of $O(n)$ searches and insertion on the search tree, resulting in $O(n \log n)$ time to construct T_0.

We now describe a greedy algorithm to construct an optimal CR partition of H using T_0. Observe that the horizontal stabbing number of the initial partition is initially one, whereas its vertical stabbing number corresponds to the height of T_0. The algorithm stores the forest's trees in a priority queue indexed by height. While the vertical stabbing number of H remains greater than its horizontal

stabbing number, split the tree of maximum height, say T. To do this, remove the horizontal partition edge stored in a tree node p, where p is a child of the root of T on a longest root-to-leaf path in T. The choice of T and p is not necessarily unique; it suffices to select any tallest tree T and any longest path in T. Observe that p has at least one and possibly two reflex vertices as endpoints, denoted a and b. Remove the horizontal partition edge associated with p and add a vertical partition edge (V_a or V_b) for each neighbour of p that lies above p on the boundary of H. The tree T is then divided into up to three new trees: a) the subtrees of the root of T to the left of p, b) the subtree rooted at p, and c) the subtrees of the root of T to the right of p. The root of each new tree corresponds to the base edge of H. See Figure 2(b). The following observation is straightforward:

Observation 3. *The horizontal stabbing number of the partition associated with the forest corresponds to the number of trees in the forest, whereas its vertical stabbing number corresponds to the height of the tallest tree in the forest.*

Once the height of the tallest tree becomes less than or equal to the number of trees in the forest, we return either the current partition or the previous partition, whichever has lower stabbing number. The number of steps is $O(n)$, where each step requires $O(\log n)$ time to determine the tree with maximum height using the priority queue.

The algorithm's correctness follows from Observations 2 and 3, and the fact that reducing the vertical stabbing number requires reducing the height of the tallest tree, which is exactly how the algorithm proceeds, decreasing the height of a tallest tree by one on each step. Therefore, we have the following theorem:

Theorem 1. *Given a histogram H, an optimal* CR *partition of H can be found in $O(n \log n)$ time, where n is the number of vertices of H.*

4 An Approximation Algorithm for Rectilinear Polygons

In this section, we present an LP relaxation for the problem of finding an optimal CR partition of a rectilinear polygon, possibly with holes. We show that a simple rounding of the LP relaxation leads to a 2-approximation algorithm for this problem. Our algorithm works even when the input polygon has holes.

Let P be a rectilinear polygon. We define two binary variables u_h and u_v for every reflex vertex $u \in V_R(P)$ that correspond to H_u and V_u, respectively. Each variable's value (1 = present, 0 = absent) determines whether its associated partition edge is included in the partition. If two reflex vertices align, then they share a common variable. For each reflex vertex u in $V_R(P)$, let ℓ_u^- and $\ell_u^|$ be respective maximal horizontal and vertical line segments that pass through $f_\epsilon(u)$ and are completely contained in P, where $f_\epsilon(u)$ denotes an ϵ translation of the point u along the bisector of the interior angle determined by the boundary of P

locally at u, for some ϵ less than the minimum distance between any two vertices of P. This perturbation ensures that ℓ_u^- and $\ell_u^|$ lie in the interior of P, as in the definition of stabbing number. See Figure 3. Let S_u^- (resp., $S_u^|$) be the set of reflex vertices in $V_R(P)$, like v, such that V_v (resp., H_v) intersects ℓ_u^- (resp., $\ell_u^|$). For each reflex vertex $u \in V_R(P)$, let

$$u_{\Sigma^-} = 1 + \sum_{p \in S_u^-} p_v, \quad \text{and} \quad u_{\Sigma^|} = 1 + \sum_{p \in S_u^|} p_h.$$

Thus, u_{Σ^-} and $u_{\Sigma^|}$ denote the number of rectangles stabbed by ℓ_u^- and $\ell_u^|$, respectively, and their maximum values among all reflex vertices u in P correspond to one less than the respective horizontal and vertical stabbing numbers of P. Consequently, the stabbing number of the partition of P determined by the binary variables is

$$1 + \max_{u \in V_R(P)} \{\max\{u_{\Sigma^-}, u_{\Sigma^|}\}\}. \tag{2}$$

A partition divides the polygon into convex regions (more specifically, rectangles) if and only if at least one partition edge is rooted at every reflex vertex. Thus, a CR partition of P corresponds to an assignment of truth values to the set of binary variables such that (i) no two edges of the partition cross, and (ii) for every reflex vertex u, at least one of V_u and H_u is present in the partition.

Therefore, the problem of finding an optimal CR partition can be formulated as a *k-sum integer linear program* as follows:

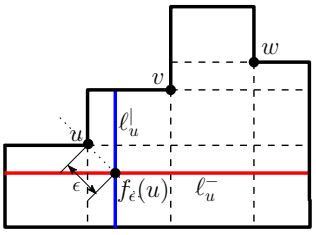


Fig. 3. The maximal line segments ℓ_u^- and $\ell_u^|$ that pass through the point $f_\epsilon(u)$ are shown in red and blue, respectively. In this example, $u_{\Sigma^-} = 1 + u_v + v_v + w_v$ and $u_{\Sigma^|} = 1 + u_h$.

$$\begin{aligned} &\text{minimize } (2) && && (3)\\ &\text{subject to } u_h + u_v \geq 1, && \forall u \in V_R(P),\\ &\qquad v_h + u_v \leq 1, && \text{if } H_v \text{ intersects } V_u,\\ &\qquad u_h, u_v \in \{0,1\}, && \forall u \in V_R(P). \end{aligned}$$

To obtain an integer linear program, we introduce an additional variable y. The following integer linear program is equivalent to the above KLP (see Section 2):

$$\begin{aligned} &\text{minimize } y && && (4)\\ &\text{subject to } y - u_{\Sigma^-} \geq 0 && \forall u \in V_R(P),\\ &\qquad y - u_{\Sigma^|} \geq 0 && \forall u \in V_R(P),\\ &\qquad u_h + u_v \geq 1, && \forall u \in V_R(P),\\ &\qquad -v_h - u_v \geq -1, && \text{if } H_v \text{ intersects } V_u,\\ &\qquad u_h, u_v \in \{0,1\}, && \forall u \in V_R(P). && (5) \end{aligned}$$

Since the number of sums in (2) is $O(n^2)$, the size of the integer linear program above is polynomial in n. Next, we relax the above program by replacing (5) with $u_h, u_v \in [0,1], \forall u \in V_R(P)$ and obtain the following LP:

$$\begin{aligned} &\text{minimize } y && && (6)\\ &\text{subject to } y - u_{\Sigma^-} \geq 0 && \forall u \in V_R(P),\\ &\qquad y - u_{\Sigma^|} \geq 0 && \forall u \in V_R(P),\\ &\qquad u_h + u_v \geq 1, && \forall u \in V_R(P),\\ &\qquad -v_h - u_v \geq -1, && \text{if } H_v \text{ intersects } V_u,\\ &\qquad u_h, u_v \geq 0, && \forall u \in V_R(P). \end{aligned}$$

We observe that the constraints $u_h, u_v \leq 1$ are redundant since we can reduce any $u_h > 1$ (resp., $u_v > 1$) to u_h=1 (resp., u_v=1) without increasing the value of the objective function for any feasible solution. Let s^* be a solution to the above LP. We round s^* to a feasible solution for our problem as follows. For each vertex $u \in V_R(P)$, let

$$u_h = \begin{cases} 0, & \text{if } s^*(u_h) \leq 1/2,\\ 1, & \text{if } s^*(u_h) > 1/2, \end{cases} \quad \text{and} \quad u_v = \begin{cases} 0, & \text{if } s^*(u_v) < 1/2,\\ 1, & \text{if } s^*(u_v) \geq 1/2. \end{cases} \tag{7}$$

We first show that, for every reflex vertex u, at least one of V_u and H_u is present in the partition.

Lemma 1. *For each vertex $u \in V_R(P)$, at least one of u_h and u_v is equal to 1 after rounding a solution to (6).*

Proof. We give a proof by contradiction. Suppose that after rounding a solution to (6), $u_h = u_v = 0$ for some $u \in V_R(P)$. Since $u_h = 0$ by (7) we have $s^*(u_h) \leq 1/2$ and, similarly, since $u_v = 0$ we have $s^*(u_v) < 1/2$. Therefore, $s^*(u_h) + s^*(u_v) < 1$, which contradicts the constraint $u_h + u_v \geq 1$ of (6). □

The next lemma proves that no two edges of the partition obtained by the LP cross each other.

Lemma 2. *If H_v intersects V_u, for two vertices $u, v \in V_R(P)$, then at most one of the variables v_h and u_v is 1 after rounding a solution to the LP.*

Proof. We give a proof by contradiction. Suppose that for two vertices $u, v \in V_R(P)$: (i) H_v intersects V_u, and, (ii) both v_h and u_v are 1 after rounding. Since after rounding v_h=1 by (7) we have $s^*(v_h) > 1/2$. Similarly, since after rounding u_v=1 we have $s^*(u_v) \geq 1/2$. Therefore, $s^*(v_h) + s^*(u_v) > 1$, which contradicts the constraint $v_h + u_v \leq 1$ (or equivalently $-v_h - u_v \geq -1$) of the LP. □

By combining Lemmas 1 and 2, we get the following result:

Corollary 1. *The partition determined by a feasible solution to the LP after rounding is a* CR *partition.*

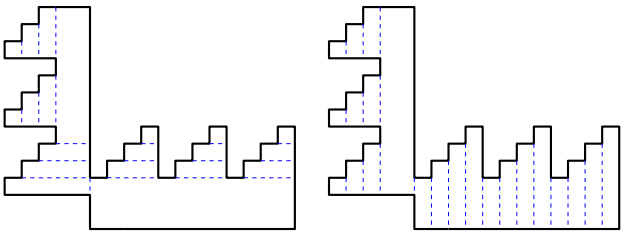


Fig. 4. A simple rectilinear polygon P for which (a) the optimal partition has stabbing number 4 while (b) assigning V_u (or H_u) to every reflex vertex u of P results in a partition with stabbing number at least 10

By (7), the value of each variable after rounding is at most twice the value of the corresponding variable in the LP solution. Moreover, it is easy to see that the number of constraints in (6) is polynomial in $V_R(P)$, allowing a 2-approximate solution to be found in polynomial time. Therefore, we have the following theorem:

Theorem 2. *There exists a polynomial-time algorithm that constructs a* CR *partition of any given rectilinear polygon P with stabbing number at most twice that of any* CR *partition of P.*

Remark. A preliminary attempt at obtaining a 2-approximation might be to assign to each reflex vertex u its vertical partition edge, V_u (or, equivalently, assigning the horizontal partition edge H_u to each u). Unfortunately, this is not the case: Figure 4 shows a rectilinear polygon for which the optimal CR partition has stabbing number 4. However, the partition obtained by assigning V_u (or H_u) consistently to every vertex $u \in V_R(P)$ has stabbing number at least 10. In fact, the polygon in this example can be extended to show that this heuristic does not provide any constant-factor approximation.

5 Hardness for Rectilinear Polygons with Holes

In this section we present an overview of a reduction showing that the following problem is NP-hard; the complete details of the reduction are omitted due to space constraints.

Optimal CR partition
Input: A rectilinear polygon P possibly with holes
Output: An optimal CR partition of P

We show that Optimal CR Partition is NP-hard by a reduction from Planar Variable Restricted 3SAT (Planar VR3SAT). The Planar VR3SAT problem is a constrained version of 3SAT in which each variable can appear

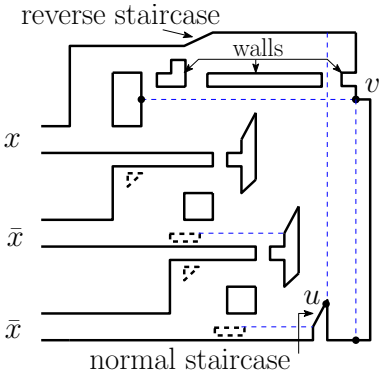


Fig. 5. An example of a variable gadget X linked by three respective corridors to its occurrences (x, $\overline{x}$ and $\overline{x}$) in clauses. Each pair of dashed triangular and rectangular holes form a negation gadget that negates the truth value of x in the associated clause linked by the adjacent corridor. Each staircase consists of c steps. Full details of the variable gadget appear in the complete version of this paper.

in at most three clauses and the corresponding *variable-clause graph* must be planar. Efrat et al. [4] show that PLANAR VR3SAT is NP-hard.

Let $I = \{C_1, C_2, \ldots, C_k\}$ be an instance of PLANAR VR3SAT with k clauses and n variables, $X_1, X_2, \ldots, X_n$. We construct a polygon P with holes such that P has a CR partition with stabbing number at most $5c$ if and only if I is satisfiable, where c is a constant that does not depend[2] on I. Given I, we first construct the variable-clause graph of I in the non-crossing comb-shape form of Knuth and Raghunathan [9]. Without loss of generality, we assume that the variable vertices lie on a vertical line and the clause vertices are connected from left or right of that line. Then, we replace each variable vertex X_i with a polygonal variable gadget to which three connecting corridors are attached from its left. The corridors are then connected to the clause gadgets whose associated clauses contain that variable. Figure 5 shows an example of a variable gadget; note the vertex v. Due to the structure of the variable gadget, any CR partition must contain exactly one of the edges V_v or H_v; including V_v (resp., H_v) in the partition corresponds to a truth assignment of *true* (resp., *false*) for the variable x. Moreover, choosing V_v or H_v imposes constraints on how the rest of the variable gadget and its associated clause gadgets can be partitioned. Due to space constraints, we omit detailed descriptions of variable gadgets and clause gadgets. The overall construction implies the following lemma whose proof appears in the complete version of this paper:

Lemma 3. *P has a* CR *partition with stabbing number at most* $5c$*, for some constant c, if and only if I is satisfiable.*

By Lemma 3, we obtain the following theorem:

Theorem 3. OPTIMAL CR PARTITION *is* NP*-hard.*

[2] The definition of the precise value of c refers to specific details of the reduction that have been omitted due to space constraints. Note, however, that the value of c can be specified in polynomial time.

6 Generalizing the Approximation Algorithm

In this section we relax the conforming constraint and consider the problem originally examined by Abam et al. [1], of finding an optimal rectangular partition that is not necessarily conforming. That is, partition edges need not be fully anchored; equivalently, vertices of partition rectangles may lie in the polygon's interior. We extend the LP relaxation presented in Section 4 to achieve a 2-approximation algorithm for this generalized problem for rectilinear polygons. In addition to improving the 3-approximation algorithm of Abam et al. [1], we present a simple algorithmic solution that also works when the input rectilinear polygon has holes. We present a brief overview of the LP relaxation in this section; the details of the algorithm are omitted due to space constraints.

The idea is to consider the arrangement of line segments induced by the intersection of vertical line segments (e.g., V_u for some reflex vertex u) with horizontal line segments (e.g., H_v for some reflex vertex v) inside the polygon P. We refer to these shorter line segments as *fragments* and associate a binary variable with each fragment (as opposed to associating a binary variable with each potential partition edge as in Section 4). By Observation 1 it suffices to consider rectangular partitions for which each partition edge is anchored at some reflex vertex. Thus, a rectangular partition corresponds to an assignment of binary values that observes the following constraints:

1. Every reflex vertex is adjacent to at least one fragment included in the partition.
2. A fragment may be included in a partition if and only if each of its endpoint meets either a) the polygon boundary, b) the continuation of a partition edge along an adjacent fragment, or c) two fragments that form a perpendicular partition edge.
3. At most three of four fragments with a common endpoint can be included in a partition.

A partition's stabbing number is represented as before by the maximum sum of binary variables crossed by any line segment ℓ_u^- or $\ell_u^|$, where u is a reflex vertex in P. The constraints 1–3 can be expressed as a linear program; details are omitted due to space restrictions. Rounding a solution to the linear program as in Section 4 gives the following theorem:

Theorem 4. *There exists a polynomial-time algorithm that constructs a rectangular partition of any given rectilinear polygon P with stabbing number at most twice that of any rectangular partition of P.*

7 Conclusion

This paper considers the problem of finding an optimal partition of a rectilinear polygon P (i.e., a partition with minimum stabbing number over all such partitions of P) for two different types of partitions.

For the first type, CR partitions (in which no partition edge can end in the interior of another edge of the partition) we first described an $O(n \log n)$-time algorithm when P is a histogram polygon with n vertices. Next we presented a LP relaxation of the problem to achieve a polynomial-time 2-approximation algorithm for any given rectilinear polygon with n vertices (possibly with holes). We also proved that the problem is NP-hard for rectilinear polygons with holes. The complexity of the problem for simple rectilinear polygons (without holes) remains open.

For the second type, in which endpoints of partition edges may lie in the interior of P, we gave a polynomial-time 2-approximation algorithm for rectilinear polygons. Our algorithm, based on the extension of our LP relaxation for CR partitions, not only improves the 3-approximation algorithm of Abam et al. [1], but also provides a simple solution that works when polygons have holes. The complexity of the general problem remains open for both simple rectilinear polygons and rectilinear polygons with holes, providing another interesting direction for future research.

References

1. Abam, M.A., Aronov, B., de Berg, M., Khosravi, A.: Approximation algorithms for computing partitions with minimum stabbing number of rectilinear and simple polygons. In: Proc. ACM SoCG, pp. 407–416 (2011)
2. de Berg, M., Khosravi, A., Verdonschot, S., van der Weele, V.: On Rectilinear Partitions with Minimum Stabbing Number. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 302–313. Springer, Heidelberg (2011)
3. de Berg, M., van Kreveld, M.: Rectilinear decompositions with low stabbing number. Inf. Proc. Let. 52(4), 215–221 (1994)
4. Efrat, A., Erten, C., Kobourov, S.: Fixed-location circular arc drawing of planar graphs. J. Graph Alg. & Applications 11(1), 165–193 (2007)
5. Fekete, S., Lübbecke, M., Meijer, H.: Minimizing the stabbing number of matchings, trees, and triangulations. Disc. Comp. Geom. 40, 595–621 (2008)
6. Gourley, K., Green, D.: A polygon-to-rectangle conversion algorithm. IEEE Comp. Graphics & App. 3(1), 31–36 (1983)
7. Hershberger, J., Suri, S.: A pedestrian approach to ray shooting: shoot a ray, take a walk. J. Alg. 18(3), 403–431 (1995)
8. Katz, M.J., Morgenstern, G.: Guarding orthogonal art galleries with sliding cameras. Inter. J. Comp. Geom. & App. 21(2), 241–250 (2011)
9. Knuth, D., Raghunathan, A.: The problem of compatible representatives. SIAM J. Disc. Math. 5(3), 422–427 (1992)
10. Lopez, M., Mehta, D.: Efficient decomposition of polygons into L-shapes with application to VLSI layouts. ACM Trans. Design Automation Elec. Sys. 1(3), 371–395 (1996)
11. Punnen, A.: K-sum linear programming. J. Oper. Res. Soc. 43(4), 359–363 (1992)

Monotone Paths in Planar Convex Subdivisions*

Adrian Dumitrescu[1], Günter Rote[2], and Csaba D. Tóth[3]

[1] Department of Computer Science, University of Wisconsin–Milwaukee, USA
[2] Institut für Informatik, Freie Universität Berlin, Germany
[3] Department of Mathematics, University of Calgary, Canada
dumitres@uwm.edu, rote@inf.fu-berlin.de, cdtoth@ucalgary.ca

Abstract. Consider a connected subdivision of the plane into n convex faces where every vertex is incident to at most Δ edges. Then, starting from every vertex there is a path with at least $\Omega(\log_\Delta n)$ edges that is monotone in some direction. This bound is the best possible. Consider now a connected subdivision of the plane into n convex faces where exactly k faces are unbounded. Then, there is a path with at least $\Omega(\log(n/k)/\log\log(n/k))$ edges that is monotone in some direction. This bound is also the best possible.

In 3-space, we show that for every $n \geq 4$, there exists a polytope P with n vertices, bounded vertex degrees, and triangular faces such that every monotone path on the 1-skeleton of P has at most $O(\log^2 n)$ edges. We also construct a polytope Q with n vertices, and triangular faces, (with unbounded degree however), such that every monotone path on the 1-skeleton of Q has at most $O(\log n)$ edges.

1 Introduction

A *geometric graph* $G = (V, E)$ in Euclidean d-space is a set V of distinct points (vertices) in Euclidean d-space $\mathbb{R}^d$, and a set E of line segments (edges) between vertices such that no vertex lies in the relative interior of any edge. For our investigation, it is convenient to define an *extended geometric graph* $G = (V, E)$, where E may also contain rays, each emitted by a vertex, and lines (disjoint from vertices). A directed path p in an extended geometric graph G is *monotone* (resp., *weakly monotone*) if there exists a unit vector $\boldsymbol{u}$ such that the inner product $\boldsymbol{e} \cdot \boldsymbol{u}$ is positive (resp., non-negative) for every directed edge $\boldsymbol{e}$ of p. In $\mathbb{R}^2$, in particular, the direction of a unit vector $\boldsymbol{u} = (\cos\theta, \sin\theta)$ is determined by the angle $\theta \in (-\pi, \pi]$. A directed path p is x-monotone (resp., y-monotone) if it is monotone in direction 0 (resp., $\frac{\pi}{2}$). The *size* (or *length*) of a path is the number of edges in the path, or equivalently, one plus the number of vertices

* Preliminary results were reported by the authors in [3,6]. Dumitrescu was supported in part by the NSF grant DMS-1001667; Rote was supported in part by the Centre Interfacultaire Bernoulli in Lausanne and by the National Science Foundation; Tóth was supported in part by the NSERC grant RGPIN 35586. Research by Tóth was conducted at the Fields Institute in Toronto and at the Centre Interfacultaire Bernoulli in Lausanne.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 240–251, 2012.

on the path. Notice that any path (monotone or not) in an extended geometric graph contains at most two rays.

We are looking for long monotone paths in the 1-skeletons of polytopes and convex subdivisions of the plane. The *1-skeleton* $G(P)$ of a polytope P in $\mathbb{R}^d$ is the (extended) geometric graph formed by the vertices and edges of P. A *convex subdivision* (for short, *subdivision*) of $\mathbb{R}^d$ is a set Π of (bounded or unbounded) convex polytopes (called *faces*) that tile $\mathbb{R}^d$. The 1-skeleton of a subdivision Π of the plane $\mathbb{R}^2$ is the extended geometric graph $G(\Pi)$ whose vertices are the points incident to 3 or more edges, and whose edges are the line segments, rays, and lines lying on the common boundary of two faces. To exclude some trivial cases, we always consider convex subdivisions whose 1-skeleton is connected, referred to as *connected subdivisions* for short. Our results are the following.

Theorem 1. *Let Π be a connected subdivision of the plane into n convex cells in which every vertex is incident to at most Δ edges. Then, for every vertex v, there is a weakly monotone path with at least $c \log_\Delta n$ edges starting from v, where $c > 0$ is an absolute constant. Apart from the constant c, this bound is the best possible.*

Theorem 2. *Let Π be a connected subdivision of the plane into n convex cells, k of which are unbounded with $n > k \geq 3$. Then $G(\Pi)$ contains a monotone path with at least $c \log \frac{n}{k} / \log\log \frac{n}{k}$ edges, where $c > 0$ is an absolute constant. Apart from the constant c, this bound is the best possible.*

We also consider long monotone paths in the 1-skeleton of a convex polytope in 3-space. We make two constructions, one with bounded vertex degrees and one with arbitrary vertex degrees.

Theorem 3. *For every $n \geq 4$, there is a polytope P in $\mathbb{R}^3$ with n vertices, bounded vertex degrees, and triangular faces such that every monotone path in $G(P)$ has $O(\log^2 n)$ edges.*

Theorem 4. *For every $n \geq 4$, there is a polytope Q in $\mathbb{R}^3$ with n vertices and triangular faces such that every monotone path in $G(Q)$ has $O(\log n)$ edges.*

We do not know whether the bounds in Theorems 3 and 4 are asymptotically tight. Since the diameter of a bounded degree graph on n vertices is $\Omega(\log n)$, every monotone path connecting a diametral pair of vertices of a polytope with n vertices of bounded degree has $\Omega(\log n)$ edges. If the maximum vertex degree of the polytope is not bounded, then a lower bound of $\Omega(\log n / \log\log n)$ follows from a result of Chazelle et al. [2] (see below), applied to the dual graph of a plane projection, using reciprocal diagrams and the Maxwell-Cremona correspondence.

Related Work. It is well known that the classical simplex algorithm in linear programming produces a monotone path on the 1-skeleton of a d-dimensional polytope of feasible solutions; it is called a *parametric simplex path*. According to the old *monotone Hirsch conjecture* [10], for any vector $\boldsymbol{u}$, the 1-skeleton of every d-dimensional polytope with n facets contains a $\boldsymbol{u}$-monotone path of

at most $n-d$ edges from any vertex to a $\boldsymbol{u}$-maximal vertex. For the weakly monotone version, counterexamples have already been found by Todd [8] in the 1980s. Recent counterexamples for this conjecture found by Santos [7] show that the monotone variant is also false. It is not known whether the Hirsch conjecture can be relaxed so that it holds when the length $n-d$ is replaced by a suitable polynomial in d and n.

Balogh et al. [1] showed that there is a convex subdivision Π_n generated by n lines in the plane with $O(n^2)$ faces such that $G(\Pi_n)$ contains a monotone path of length $\Omega(n^{2-c/\sqrt{\log n}})$, where $c>0$ is a constant.

A *monotone face sequence* in a convex subdivision Π is a sequence of faces such that there is a direction $\boldsymbol{u}$ such that any two consecutive faces, f_1 and f_2, are adjacent and a vector of direction $\boldsymbol{u}$ crosses their common boundary from f_1 to f_2. Chazelle et al. [2] showed that in a subdivision of the plane into n convex faces in which every face is adjacent to at most d other faces, there is a monotone face sequence of length $\Omega(\log_d n + \log n/\log\log n)$, and this bound is tight. Moreover, a monotone face sequence of this length can even be achieved by faces stabbed by a line. The latter result was generalized to d dimensions by Tóth [9]: for every subdivision of $\mathbb{R}^d$ into n convex faces, there is a line that stabs $\Omega((\log n/\log\log n)^{\frac{1}{d-1}})$ faces, and this bound is the best possible.

Outline. We start with the proof of Theorem 1 in Sec. 2. We study convex subdivisions of simple polygons in Sec. 3. The tools developed there are instrumental in the proof of Theorem 2 in Sec. 4. The polytopes P and Q in Theorem 3 and 4 are presented in the full version of the paper.

2 Proof of Theorem 1

Lower bound. The lower bound in Theorem 1 follows from the following lemma in a straightforward way (by counting, or inductively).

Lemma 1. *Let v be a vertex in a connected convex planar subdivision Π. Then $G(\Pi)$ contains a spanning tree rooted at v such that all paths starting at v are weakly monotone.*

Proof. For a generic direction $\boldsymbol{u}$, we define the *rightmost path* $R(\boldsymbol{u})$ starting at v as follows; see Fig. 1(left): start at v and always follow the rightmost outgoing edge that is weakly monotone in direction $\boldsymbol{u}$ until we arrive at an unbounded ray.

Now we start rotating $\boldsymbol{u}$ clockwise. At some direction $\boldsymbol{u}'$, $R(\boldsymbol{u}')$ will be different from $R(\boldsymbol{u})$. At that point, $R(\boldsymbol{u})$ is still weakly monotone in direction $\boldsymbol{u}'$. Now, any vertex w (and any edge) in the region between $R(\boldsymbol{u})$ and $R(\boldsymbol{u}')$ can be reached by a weakly monotone path in direction $\boldsymbol{u}'$. Indeed, simply start at w and go monotonically in the direction $-\boldsymbol{u}'$ until reaching $R(\boldsymbol{u})$ or $R(\boldsymbol{u}')$. From there, follow $R(\boldsymbol{u})$ or $R(\boldsymbol{u}')$ to v. In this way, we can form a spanning tree of all vertices between $R(\boldsymbol{u})$ and $R(\boldsymbol{u}')$ with the desired properties.

Continuing the rotation in this way, we eventually reach all vertices and all infinite rays. □

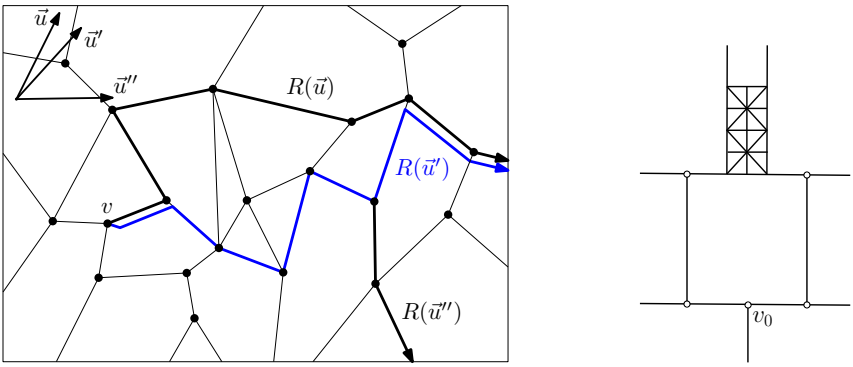


Fig. 1. Left: The rightmost path $R(\boldsymbol{u})$ starting from vertex v in direction $\boldsymbol{u}$. Right: A convex subdivision where only the five vertices marked with empty dots can be reached from v_0 along (strictly) monotone paths.

The subdivision in Fig. 1(right) shows that the lemma does not hold with (strictly) monotone paths. However, if there are no angles of 180°, the statement extends to strictly monotone paths.

Let v be a vertex in a connected convex planar subdivision Π. By Lemma 1, $G(\Pi)$ contains a spanning tree rooted a v such that all paths starting at v are weakly monotone. The maximum degree in $G(\Pi)$ is at most Δ. Hence the spanning tree contains a path of size $\Omega(\log_\Delta n)$ from v to some vertex of $G(\Pi)$.

Upper Bound. If the maximum degree Δ is n, Theorem 1 gives only a trivial statement. Dividing the plane into n convex sectors by n rays starting from the origin shows that, indeed, there is no non-constant lower bound on the length of monotone paths in this case. This construction can be generalized: for every $3 \leq \Delta \leq n$, there is a convex subdivision Π such that $G(\Pi)$ is a tree with maximum degree Δ and diameter $O(\log_\Delta n)$.

3 Monotone Paths in Simple Polygons

Monotone Polygons. We start by introducing some notation for simple polygons in $\mathbb{R}^2$. A *polygonal domain* (for short, *polygon*) P is a closed set in the plane bounded by a piecewise linear simple curve. A polygon P is monotone if its boundary is the union of two paths, which are both monotone with respect to a vector $\boldsymbol{u}$. In particular, P is y-monotone if it is bounded by two y-monotone paths. A *convex subdivision* of a polygon P is a set $\Pi = \Pi(P)$ of pairwise disjoint open convex sets (called *faces*) such that the union of their closures is P. The faces in Π together with the complement $\bar{P} = \mathbb{R}^2 \setminus P$ (the *outer face*) form a (nonconvex) subdivision of the plane $\Pi \cup \{\bar{P}\}$. We also define a geometric graph $G(\Pi) = G(\Pi(P))$, where the *vertices* are the union of all vertices of P and the set of points incident to 3 or more faces in $\Pi \cup \{\bar{P}\}$; and the *edges* are the line segments lying on the common boundaries of two faces in $\Pi \cup \{\bar{P}\}$.

A simple but crucial observation is that for every vertex v of $G(\Pi)$ lying in the interior of P and every direction $\boldsymbol{u}$, there is an edge vw such that $\boldsymbol{u} \cdot \overrightarrow{vw} \geq 0$, otherwise the face incident to v in direction $\boldsymbol{u}$ would not be convex. This implies the following.

Observation 1. *Let P be a simple polygon with a convex subdivision $\Pi = \Pi(P)$, let v be a vertex of $G(\Pi)$ lying in the interior of P, and $\boldsymbol{u}$ be a unit vector. Then*

(i) *there is a weakly $\boldsymbol{u}$-monotone directed path in $G(\Pi)$ from v to some vertex on the boundary of P;*
(ii) *if $\boldsymbol{u}$ is not orthogonal to any edge of $G(\Pi)$, then this path is $\boldsymbol{u}$-monotone.*

Recall that a y-monotone polygon is bounded from the left and from the right by two y-monotone directed paths. The common start (resp., end) point of the two boundary paths is called the *bottom* (resp., *top*) vertex of P.

Observation 2. *Let $\Pi = \Pi(P)$ be a convex subdivision of a y-monotone polygon P with no horizontal edges. For every vertex v of $G(\Pi)$, there is a y-monotone path from the bottom vertex of P to the top vertex of P which is incident to v.*

Criterion for y-monotone Polygons. To prove the lower bound in Theorem 2, we constructively build a monotone path of the required length for any given convex partition. In our recursive construction, we successively subdivide a y-monotone polygon P into smaller polygons along certain y-monotone paths in $G = G(\Pi)$. Lemma 2 below provides a criterion for producing y-monotone pieces. A vertex v in a geometric graph G is called *y-maximal* (resp., *y-minimal*) if all edges of G incident to v lie in a closed halfplane below (resp., above) the horizontal line passing through v. A vertex v is *y-extremal* if it is y-maximal or y-minimal. It is clear that the boundary of a y-monotone polygon has exactly two y-extremal vertices, namely its top vertex and its bottom vertex.

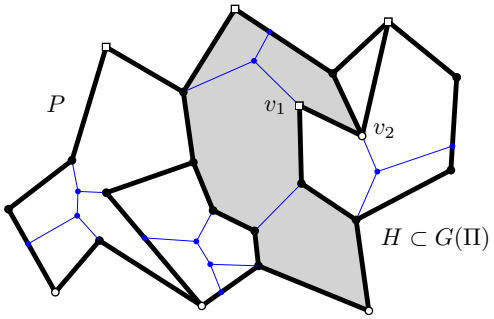


Fig. 2. The graph $G(\Pi)$ of a convex partition Π of a simple polygon P. A subgraph H in bold contains all edges of P. The y-maximal (resp., y-minimal) vertices of H are marked with empty squares (resp., empty circles). Vertices v_1 and v_2 are y-extremal in H, but not convex vertices of P (here they are not even vertices of P).

Lemma 2. *Let P be a simple polygon with a convex subdivision Π such that no edge in $G = G(\Pi)$ is horizontal. Let H be a subgraph of G that contains all edges and vertices of P. Then all bounded faces of H are y-monotone polygons if and only if all y-extremal vertices of H are convex vertices of P.*

Proof. Assume that all bounded faces of H are y-monotone polygons. Suppose that H has a y-extremal vertex v. We may assume without loss of generality that all edges of H incident to v are in the halfplane *above* v. Let f_v be the face of H incident to v that lies directly *below* v. Face f_v has a reflex interior angle at v, and v is neither the top nor the bottom vertex of f_v. Hence f_v is not y-monotone, and so it has to be an unbounded face of H. Since H contains all boundary edges of P, the face f_v is the unbounded face of G, as well. It follows that v is a convex vertex of P.

Assume that all y-extremal vertices of H are convex vertices of P. Consider a bounded face f of H. Let p_1 and p_2 be edge-disjoint directed paths on the boundary of f from a bottom (lowest) vertex of f to a top (highest) bottom vertex of f (ties are broken arbitrarily), such that f lies on the right side of p_1 and on the left side of p_2. Suppose for contradiction that f is not y-monotone. We may assume without loss of generality that p_2 is not y-monotone, as in Fig. 2 (where f is the shaded face). The first (resp., last) edge of p_2 has a positive inner product with $(0, 1)$ by construction. There are two consecutive edges $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ in p_2 such that $\boldsymbol{e}_1 \cdot (0, 1) > 0 > \boldsymbol{e}_2 \cdot (0, 1)$, since otherwise p_2 would be y-monotone. Let v be the common vertex of these two edges. Since f is on the left side of both $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$, these two edges are consecutive in the counterclockwise rotation order of the edges of H incident to v. Hence, H has no edge incident to v in the halfplane above v, and so v is y-extremal (y-maximal) in H. However, v is a reflex vertex of face f, hence it cannot be a convex vertex of P (specifically, v it is either interior to P, or a reflex vertex of P). This contradicts our initial assumption and completes the proof. □

Subdividing a Polygon into Monotone Pieces. Our upper bound relies on the following two lemmas. In Lemma 3, we partition the bounded faces of a convex subdivision of the plane into monotone polygons. In Lemma 4, we subdivide a y-monotone polygon P into smaller y-monotone polygons which are not incident to both the top and the bottom vertex of P.

Lemma 3. *Let Π be a subdivision of the plane into n convex faces, $k \geq 3$ of which are unbounded. Then there is a subset $\Pi' \subset \Pi$ of at least $(n - k)/(k - 2)$ faces such that Π' is the convex subdivision of a monotone polygon.*

Proof. We proceed by induction on the number of 2-connected components of G. In the base case, G is 2-connected. Suppose that no edge of G is horizontal. Let P be the simple polygon formed by the union of the closures of all bounded faces in Π. Let V_0 be the set of y-extremal reflex vertices of P. If V_0 is empty, then P is a y-monotone polygon, and the $n-k$ bounded faces form a convex subdivision of P. If V_0 is nonempty, then we construct a y-monotone path $\gamma(v)$ for each $v \in V_0$, in an arbitrary order, as follows. If $v \in V_0$ is y-maximal (resp., y-minimal), then

construct $\gamma(v)$ starting from v by successively appending edges in direction $\pi/2$ (resp., $-\pi/2$) until the path reaches another vertex on the boundary of P or a previously constructed path $\gamma(v')$, $v' \in V_0$. The paths $\gamma(v)$ subdivide P into $|V_0|+1$ simple polygons, each of which is y-monotone by Lemma 2.

It remains to show that $|V_0| \leq k-3$ after an appropriate rigid motion. Notice that every y-extremal reflex vertex of P is a y-extremal vertex of some unbounded face. An unbounded face cannot have both a top and a bottom vertex. Two of the unbounded faces, namely those containing rays in directions $(1,0)$ and $(-1,0)$, have neither a top nor a bottom vertex. This already implies $|V_0| \leq k-2$. Let $\overrightarrow{e}_0$ be a ray edge of G emitted by an extremal vertex of the convex hull of P. Assume, by applying a reflection if necessary, that the unbounded face on the left of $\overrightarrow{e}_0$ is not a halfplane. Rotate the subdivision such that no edge in G is horizontal and edge $\overrightarrow{e}_0$ has the smallest positive slope. Now the unbounded face above $\overrightarrow{e}_0$ has a bottom vertex on the convex hull of P, which is not a reflex vertex of P. Therefore, we have $|V_0| \leq k-3$.

Assume now that G has several 2-connected components. We distinguish two cases. Case 1: G is disconnected, and it is the disjoint union of G_1 and G_2, which are incident to k_1 and k_2 unbounded faces, respectively. Then there is at most one face (a parallel strip) incident to both G_1 and G_2. Hence $k_1+k_2 \leq k+1$, and so $(k_1-2)+(k_2-2) < k-2$. Induction completes the proof. Case 2: G is connected but has a cut vertex v. Then G decomposes into subgraphs $G_1,\ldots,G_\ell$, for some $\ell \geq 2$, whose only common vertex is v. Denote by k_i the number of unbounded faces in Π incident to some edges of G_i for $i=1,\ldots,\ell$. Any two consecutive subgraphs around v are incident to a common unbounded face. Hence $k=\sum_{i=1}^{\ell} k_i - \ell$, and so $\sum_{i=1}^{\ell}(k_i-2) \leq k-2$. Induction completes the proof. □

Lemma 4. *Let P be a y-monotone polygon with bottom vertex s, top vertex t, and a convex subdivision Π. Let m be the maximum size of a monotone path in $G=G(\Pi)$. Then at most two faces in Π are incident to both s and t; the remaining faces can be partitioned into at most $m+1$ sets, each of which is the convex subdivision of a y-monotone polygon whose top or bottom vertex is not in $\{s,t\}$.*

Proof. If a face $f \in \Pi$ is incident to both s and t, then by convexity the closure of f contains the line segment st. Thus, at most two faces in Π are incident to both s and t. We first partition the remaining faces into two subsets, lying on opposite sides of a monotone path α such that the faces in each subset are incident to at most one of s and t. Then we further partition each of the two subsets to form y-monotone polygons. Let H_{st}^- and H_{st}^+ denote the closed halfplanes on the left and right of st, respectively. We distinguish two cases.

Case 1: No Face in Π Is Incident to Both s and t. We define two points, v_1 and v_2, in the relative interior of the segment st. Let $f_1 \in \Pi$ be a face incident to s whose closure contains a maximal portion of st, and let the segment sv_1 be the intersection of the closure of f_1 with st. Refer to Fig. 3(left, middle). We may assume, by applying a reflection with respect to the y-axis if necessary, that

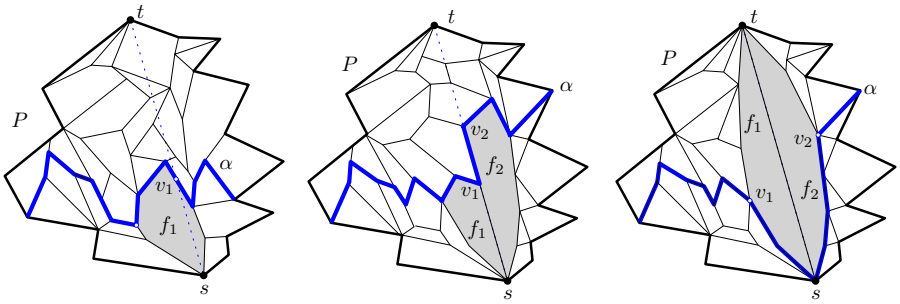


Fig. 3. A y-monotone polygon P with a bottom vertex s, a top vertex t, and a convex subdivision. Left: faces f_1 and f_2 are the same, but neither is incident to t. Middle: faces f_1 and f_2 are not is incident to t. Right: faces f_1 and f_2 are both incident to t.

the interior of f_1 intersects H_{st}^-. Since f_1 is not incident to t, there is an edge in G that contains v_1 and enters the interior of H_{st}^-. If there is some edge in G that contains v_1 and enters the interior of H_{st}^+ or if v_1 is on the boundary of polygon P, then let $v_2 = v_1$. Otherwise, all edges of G incident to v_1 lie in H_{st}^-, hence two consecutive edges are contained in st. Then v_1 is incident to a unique face $f_2 \in \Pi$ on the right of st. Since f_2 is not incident to both s and t, it has a vertex along st which is different from both s and t; let v_2 be an arbitrary such vertex. In summary: the segment $v_1v_2 \subset st$ is covered by edges of G; an edge of G contains v_1 and enters the interior of H_{st}^-; and v_2 is either on the boundary of P or an edge of G contains v_2 and enters the interior of H_{st}^+.

Let $\boldsymbol{u}$ be a unit vector orthogonal to st. Slightly rotate $\boldsymbol{u}$, if necessary, such that $\boldsymbol{u}$ is not parallel or orthogonal to any edge of G and $\boldsymbol{u} \cdot \overrightarrow{v_1v_2} > 0$ if $v_1 \neq v_2$. By Observation 1, there is a $-\boldsymbol{u}$-monotone path from v_1 to the boundary of P, and a $\boldsymbol{u}$-monotone path from v_2 to the boundary of P. The union of these two paths and the segment v_1v_2 forms a $\boldsymbol{u}$-monotone path, denoted α, between two boundary points of P and passing through v_1 and v_2. Direct α such that its starting point is in H_{st}^-. Since the endpoints of α are on the boundary of P, every face in Π is either on left or on right side of α. By construction, every face incident to s is on the right of α, and every face incident to t is on the left of α.

By our assumption, α has at most m edges. Construct a path $\gamma(v)$ for every y-extremal interior vertex v of α, in an arbitrary order, as follows. If v is a y-maximal (resp., y-minimal) vertex in α, then construct y-monotone path $\gamma(v)$ starting from v by successively appending edges in direction $\frac{\pi}{2}$ (resp., $-\frac{\pi}{2}$) until the path reaches a vertex on α, the boundary of P or a previously constructed path $\gamma(v')$, $v' \neq v$. These paths $\gamma(v)$ together with α subdivide P into at most $(m-1)+2 = m+1$ simple polygons, each of which is y-monotone by Lemma 2.

Case 2: Some Face in Π Is Incident to Both s and t. Let $f_1 \in \Pi$ be a face incident to both s and t. Refer to Fig. 3(right). We may assume, by applying a reflection with respect to the y-axis if necessary, that the interior of f_1 intersects H_{st}^-. Let $\boldsymbol{u}$ be a unit vector orthogonal to st. Slightly rotate $\boldsymbol{u}$, if necessary, such that $\boldsymbol{u}$ is not parallel or orthogonal to any edge of G and $\boldsymbol{u} \cdot \overrightarrow{st} > 0$. Let v_1 be

the $\boldsymbol{u}$-minimal vertex of f_1. We need to be more careful when defining v_2. If st is an edge of P, then let $v_2 = s$. If there is a face f_2 (possibly $f_2 = f_1$) incident to both s and t whose interior intersects H^+_{st}, then let v_2 be the $\boldsymbol{u}$-maximal vertex of f_2. Otherwise, if st is an edge of the face f_1 and G has vertices in the relative interior of st, then let v_2 be an arbitrary such vertex. Otherwise, st is on the boundary of P, and then let $v_2 = s$.

Similarly to Case 1, construct a $-\boldsymbol{u}$-monotone path from v_1 to the boundary of P, and a $\boldsymbol{u}$-monotone path from v_2 to the boundary of P. A $\boldsymbol{u}$-monotone path α is formed by the union of these two paths, a path from v_1 to s along the boundary of f_1, and a path from s to v_2 along the boundary of f_1 or f_2. Direct α such that its starting point is in H^-_{st}. Since the endpoints of α are on the boundary of P, every face in Π is either on the left or on the right side of α. By construction, every face incident to s other than f_1 and f_2 (if it exists) is on the right of α, and every face incident to t is on the left of α.

By our assumption, α has at most m edges. Construct a path $\gamma(v)$ for every y-extremal interior vertex of α that precedes v_1 or follows v_2 as in Case 1. If v_1 is an interior vertex of α, then let $\gamma(v_1)$ be the y-monotone path from v_1 to t along the boundary of f_1. Similarly, if v_2 is an interior vertex of α, then let $\gamma(v_2)$ be the y-monotone path from v_2 to t along the boundary of f_1 or f_2. Note that some of the interior vertices of α between v_1 and v_2 may be on the boundary of P. If exactly k interior vertices of α are on the boundary of P, then α subdivides P into at most $k+2$ simple polygons. The paths γ (which are not defined for vertices on the boundary of P) further subdivide these polygons into at most $(m-1)+2 = m+1$ simple polygons, each of which is y-monotone by Lemma 2.

By construction, one of these polygons is formed by the faces in Π incident to both s and t. Thus, the faces not incident to both s and t are partitioned into at most m sets, each of which is the convex subdivision of a y-monotone polygon whose top or bottom vertex is not in $\{s, t\}$. □

In Lemma 4, we have partitioned almost all faces of Π into subsets, where each subset forms a y-monotone polygon. In the proof of the lower bound in Theorem 2, we will only recurse on one of these polygons.

Corollary 1. *Let P be a y-monotone polygon with bottom vertex s and top vertex t. Let Π be a subdivision of P into n convex faces. Let m be the maximum size of a monotone path in $G = G(\Pi)$. If $n \geq 3$, then there is a subset $\Pi' \subset \Pi$ of at least $n/(m+3)$ faces such that Π' is the convex subdivision of a y-monotone sub-polygon of P whose top or bottom vertex is not in $\{s, t\}$.*

4 Proof of Theorem 2

Lower Bound Proof. Let $3 \leq k \leq n$ and Π be a subdivision of the plane into n convex faces, where k faces are unbounded. By Lemma 3 there exists a subset $\Pi_0 \subset \Pi$ of $n_0 \geq (n-k)/(k-2)$ faces that form a convex subdivision of a monotone polygon P_0. We may assume, by applying a rotation if necessary, that no edge of $G(\Pi)$ is horizontal and P_0 is y-monotone. We can assume that

$n_0 \geq 4$. It is enough to show that $G(\Pi_0)$ contains a monotone path of size at least $c_0 \log n_0 / \log\log n_0$, where $c_0 > 0$ is an absolute constant.

Let m denote the maximum size of a monotone path in $G(\Pi_0)$. We use Corollary 1 to construct a nested sequence $\Pi_0 \supset \Pi_1 \supset \ldots \supset \Pi_t$ such that $|\Pi_i| = n_i \geq n_0/(m+3)^i$ and Π_i is a convex subdivision of a y-monotone polygon P_i for $i = 1, 2, \ldots, t$. Moreover, the bottom or top vertex of P_i is different from that of P_{i-1} for $i = 1, 2 \ldots, t$. If $|\Pi_i| > 2$, then Π_{i+1} can be constructed from Π_i by Corollary 1. We may therefore assume that $n_t \in \{1, 2\}$.

Let β_t be an arbitrary y-monotone path in $G(\Pi_t)$ between the top and bottom vertex of P_t. For $i = t, t-1, \ldots, 1$, the path β_i can be extended to a y-monotone path β_{i-1} between the top and bottom vertex of $G(\Pi_{i-1})$ by Observation 1. Note that β_{i-1} is strictly longer than β_i, since at least one of two endpoints of β_i is not the top or bottom vertex of P_{i-1}. Therefore, β_0 is a y-monotone path with at least $t+1$ edges, and $t \leq m-1$ by the definition of m. We have

$$2 \geq n_t \geq \frac{n_0}{(m+3)^t} \geq \frac{n_0}{(m+3)^{m-1}}.$$

hence $n_0 \leq 2(m+3)^{m-1} = 2^{1+(m-1)\log(m+3)}$. This gives $m \geq c_0 \log n_0 / \log\log n_0$, for some absolute constant $c_0 > 0$, as required.

Upper Bound Construction. For every pair of integers $k, n \in \mathbb{R}$, where $3 \leq k < n$, we subdivide the plane into a set Π of $\Omega(n)$ convex cells, exactly k of which are unbounded, such that every monotone path in $G(\Pi)$ has $O(\log \frac{n}{k} / \log\log \frac{n}{k})$ edges. We first construct the unbounded faces. If $k = 3$, then let Q_1 be a triangle, and subdivide the exterior of Q_1 into 3 convex faces by 3 rays emitted from the vertices of Q_1. If $k \geq 4$, then subdivide the plane into k unbounded faces by a star graph with $\lfloor k/2 \rfloor$ leaves, $q_1, \ldots, q_{\lfloor k/2 \rfloor}$, and 2 or 3 rays emitted from each leaf. Then replace each vertex q_i, $1 \leq i \leq \lfloor k/2 \rfloor$ by a small triangle Q_i. Now it is enough to subdivide each triangle Q_i into at least $n_0 = 5n/k$ bounded faces such that every monotone path restricted to Q_i has $O(\log n_0 / \log\log n_0)$ edges. Since a monotone path can visit at most two triangles Q_i, it has $O(\log n_0 / \log\log n_0)$ edges.

Let $m = \lceil \log n_0 / \log\log n_0 \rceil$. The basic building block of our construction is a plane geometric graph R shown in Fig. 4(left). The outer face of R is a rhombus symmetric with respect to both the x- and the y-axes. Two opposite corners of the rhombus are connected by an x-monotone zig-zag path ξ of $2m$ edges. The edges of ξ have alternately negative and positive slopes, and so the $2m-1$ interior vertices are alternately y-minimal and y-maximal. Denote by Z the y-minimal interior vertices of ξ, so $|Z| = m$. The y-minimal interior vertices of ξ are joined to the bottom vertex s of the rhombus, and the y-maximal ones to the top vertex t. The vertices s and t are sufficiently far below and respectively far above ξ such that all bounded faces of R are convex. The graph R contains a monotone path of size $2m$ in directions close to horizontal, but every monotone path has at most 3 edges in directions close to vertical. After an appropriate affine transformation, every monotone path has at most 3 edges in all directions except for those in a prescribed interval of length $\frac{\pi}{2m}$ in $[0, 2\pi)$.

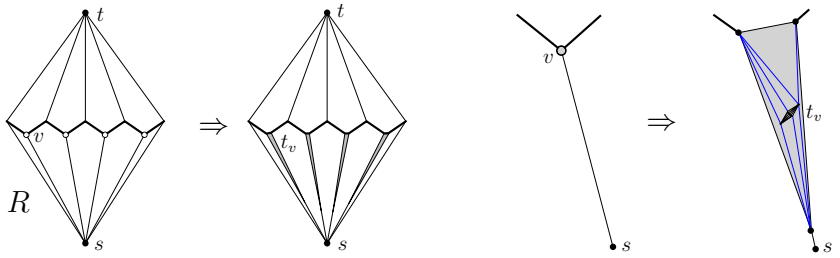


Fig. 4. Left: The subgraph R with $m = 4$. The vertices in Z are marked with empty circles. In one phase of our construction, the vertices in Z are replaced by long and skinny triangles. Right: A vertex $v \in Z$ is replaced by a skinny triangle that contains an affine copy of R, and the space between the triangle and R is triangulated.

We construct a convex subdivision Π of Q in m phases. For $i = 1, \ldots, m$, we maintain a convex subdivision Π_i, a set Z_i of special vertices in $G(\Pi_i)$, and a special edge incident to each vertex in Z_i. For constructing Π_1, consider an affine copy of R, such that any monotone path has at most 3 edges except for directions in the interval $(\frac{\pi}{2m}, \frac{2\pi}{2m})$. This graph, together with four axis-parallel rays from the four corners of the rhombus to the bounding box induce the convex subdivision Π_1 of the plane. Note that $G(\Pi_1) = R$. For each $v \in Z_1$, let the special edge of v be vs, the edge joining v to the bottom vertex of R.

In phase $i = 2, \ldots, m$, we construct Π_i from Π_{i-1} as follows. Replace each vertex $v \in Z_{i-1}$ by a long skinny triangle t_v along the special edge incident to v as in Fig. 4(right). In the interior of t_v, place a small affine copy of R near the midpoint of t_v, such that any monotone path has at most 3 edges in R except for directions in the interval $\left(\frac{(2i-1)\pi}{2m}, \frac{2i\pi}{2m}\right)$. Denote by r_v the outer boundary of this copy of R. Triangulate the space between r_v and t_v arbitrarily by using $O(1)$ edges; the edges of the triangulation are almost parallel to the special edge sv if t_v is sufficiently skinny and r_v is sufficiently small. Let Z_i be the union of the vertex sets Z from all affine copies of R created in phase i, and let the special edge of each vertex in Z_i be the edge connecting that vertex to the vertex s of the corresponding copy of R.

We show that $\Pi = \Pi_m$ has $\Omega(n_0)$ faces, and the longest monotone path in $G = G(\Pi)$ has size $O(m) = O(\log n_0 / \log\log n_0)$. Initially, we have $|Z_1| = m$ special vertices. Since $|Z_i| = m|Z_{i-1}|$ for $i = 2, \ldots, m$, it follows that $|Z_m| = m^m$. Note that for each special vertex in Z_m, there is an incident quadrilateral face in a copy of R which is not incident to any other vertex in Z_m. Hence Π has at least $|Z_m| = m^m = \Omega(n_0)$ faces.

For any $v \in Z_i$, $i = 1, 2 \ldots, m$, our recursive construction did not modify the edges of the triangle t_v and the rhombus r_v (only the interior edges of a copy of R inside r_v are modified in subsequent phases). Let $\mathcal{T}_i$ and $\mathcal{R}_i$ denote the set of triangles t_v and rhombi r_v, respectively, for all $v \in Z_i$. Note that a monotone path enters and exits the interior of a triangle or a rhombus in G at most once. Let γ be a path in $G(\Pi)$ that is monotone in some direction θ.

Assume that $\theta \in \left[\frac{(j-1)\pi}{m}, \frac{j\pi}{m}\right)$, for some $j \in \{1, 2, \ldots, m\}$. For $i < j$, the path γ enters at most one triangle of $\mathcal{T}_i$. For $j = i$, it can visit m triangles of $\mathcal{T}_i$, that all lie in a common triangle $t \in \mathcal{T}_{i-1}$. However, the edges between a triangle t_v and the rhombus $r_v \subset t_v$ are almost parallel to the special edge incident to v in $G(\Pi_{i-1})$, and all special edges within a copy of R are incident to a common vertex. It follows that γ can reach the rhombus $r_v \subset t_v$ in at most three triangles $t_v \in \mathcal{T}_i$: in at most two triangles $t_v \in \mathcal{T}_i$ that contain an endpoint of γ, and in at most one triangle t_v that γ traverses. For $i > j$, the path γ enters at most three triangles of $\mathcal{T}_i$, at most one inside each rhombus in $\mathcal{R}_{i-1}$.

It follows that γ traverses $O(m)$ edges in at most one zig-zag path created in phase j, and it traverses $O(1)$ edges created in any of the other $m-1$ phases. Consequently, every monotone path in $G(\Pi)$ has at most $O(m)$ edges, as required. □

Open Problems. The proofs of Lemmas 1, 2, and 3 crucially depend on the planarity of the subdivisions. Extending Theorems 1 and 2 for convex subdivisions of $\mathbb{R}^d$, $d \geq 3$, remain as open problems.

We have shown (Theorem 2) that in any connected subdivision with n faces, $k \geq 3$ of which are unbounded, there is a monotone path with $\Omega(\log \frac{n}{k} / \log\log \frac{n}{k})$ edges. Fig. 1(right) shows that this lower bound does not hold for (strictly) monotone paths starting from an arbitrary vertex. Deciding whether there exists a *weakly* monotone path of length $\Omega(\log \frac{n}{k} / \log\log \frac{n}{k})$ starting from *every* vertex of a convex subdivision with k unbounded faces remains an open problem.

Acknowledgment. The authors thank János Pach for insistently asking the question to which Theorem 1 gives the answer.

References

1. Balogh, J., Regev, O., Smyth, C., Steiger, W., Szegedy, M.: Long monotone paths in line arrangements. Discrete & Comput. Geom. 32, 167–176 (2004)
2. Chazelle, B., Edelsbrunner, H., Guibas, L.J.: The complexity of cutting complexes. Discrete & Comput. Geom. 4, 139–181 (1989)
3. Dumitrescu, A., Tóth, C.D.: Monotone paths in planar convex subdivisions. In: Abstracts of the 21st Fall Workshop on Comput. Geom., NY (2011)
4. McMullen, P.: The maximum numbers of faces of a convex polytope. Mathematika 17, 179–184 (1971)
5. Radoičić, R., Tóth, G.: Monotone paths in line arrangements. Comput. Geom. 24, 129–134 (2003)
6. Rote, G.: Long monotone paths in convex subdivisions In: Abstracts of the 27th European Workshop on Comput. Geom., Morschach, pp. 183–184 (2011)
7. Santos, F.: A counterexample to the Hirsch conjecture. Annals of Mathematics 176, 383–412 (2012)
8. Todd, M.J.: The monotonic bounded Hirsch conjecture is false for dimension at least 4. Math. Oper. Res. 5(4), 599–601 (1980)
9. Tóth, C.D.: Stabbing numbers of convex subdivisions. Period. Math. Hung. 57(2), 217–225 (2008)
10. Ziegler, G.M.: Lectures on Polytopes. GTM, vol. 152, pp. 83–93. Springer (1994)

The Cost of Bounded Curvature*

Hyo-Sil Kim[1] and Otfried Cheong[2]

[1] Department of Computer Science & Engineering, POSTECH, Pohang, Korea
hyosil.kim@gmail.com
[2] Department of Computer Science, KAIST, Daejeon, Korea
otfried@kaist.edu

Abstract. We study the motion-planning problem for a car-like robot whose turning radius is bounded from below by one and which is allowed to move in the forward direction only (Dubins car). For two robot configurations σ, σ', let $\ell(\sigma, \sigma')$ be the length of a shortest bounded-curvature path from σ to σ' without obstacles. For $d \geq 0$, let $\ell(d)$ be the supremum of $\ell(\sigma, \sigma')$, over all pairs (σ, σ') that are at Euclidean distance d. We study the function $\mathrm{dub}(d) = \ell(d) - d$, which expresses the difference between the bounded-curvature path length and the Euclidean distance of its endpoints. We show that $\mathrm{dub}(d)$ decreases monotonically from $\mathrm{dub}(0) = 7\pi/3$ to $\mathrm{dub}(d^*) = 2\pi$, and is constant for $d \geq d^*$. Here $d^* \approx 1.5874$. We describe pairs of configurations that exhibit the worst-case of $\mathrm{dub}(d)$ for every distance d.

1 Introduction

Motion planning or *path planning* involves computing a feasible path, possibly optimal for some criterion such as time or length, of a robot moving among obstacles; see the book by Lavalle [5]. A robot generally comes with physical limitations, such as bounds on its velocity, acceleration or curvature. Such differential constraints restrict the geometry of the paths the robot can follow. In this setting, the goal of motion planning is to find a feasible (or optimal) path satisfying both global (obstacles) and local (differential) constraints if such a path exists.

In this paper, we study the *bounded-curvature* motion planning problem which models a car-like robot. More precisely, the robot is considered a rigid body that moves in the plane. A *configuration* of the robot is specified by both its location, a point in $\mathbb{R}^2$ (typically, the midpoint of the rear axle), and its orientation, or direction of travel. The robot is constrained to move in the forward direction, and its turning radius is bounded from below by a positive constant, which can be assumed to be equal to one by scaling the space. In this context, the robot follows a bounded-curvature path, that is, a differentiable curve whose curvature is constrained to be at most one almost everywhere.

* This research is supported in part by Mid-career Researcher Program through NRF grant (No. R01-2008-000-11607-0) funded by the MEST and in part by SRC-GAIA through NRF grant (No. 2011-0030044) funded by the Government of Korea.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 252–263, 2012.

Planning the motion of a car-like robot has received considerable attention in the literature. In this paper, we consider the cost of this restriction: How much longer is the shortest path made by such a robot compared to the Euclidean distance travelled?

Formally, consider two configurations σ and σ'. Let $\ell(\sigma, \sigma')$ denote the length of a shortest curvature-constrained path from σ to σ' without obstacles, and let $d(\sigma, \sigma')$ denote the Euclidean distance between σ and σ'. We define

$$\mathrm{dub}(d) = \sup\{\ell(\sigma, \sigma') - d \mid \sigma,\ \sigma' \text{ configurations with } d(\sigma, \sigma') = d\}. \quad (1)$$

Note that the supremum here is not a maximum, as the path length is not a continuous function of the orientations at the two endpoints. Our goal is to understand the function $\mathrm{dub} : \mathbb{R} \mapsto \mathbb{R}$ in detail. While this is a natural and fundamental question related to motion planning with bounded curvature, it is also a relevant question that has repeatedly appeared in the literature, with only partial answers so far.

At least two interesting problems have been studied where not configurations but only *locations* for the robot are given. The first problem considers a *sequence* of points in the plane, and asks for the shortest curvature-constrained path that visits the points in this sequence. In the second problem, the *Dubins traveling salesman problem*, the input is a *set* of points in the plane, and asks to find a shortest curvature-constrained path visiting all points. Both problems have been studied by researchers in the robotics community, giving heuristics and experimental results [7,8]. From a theoretical perspective, Lee et al. [6] gave a linear-time, constant-factor approximation algorithm for the first problem. No general approximation algorithms are known for the Dubins traveling salesman problem (the approximation factor of the known algorithms depends on the smallest distance between points).

All this work depends on some knowledge of the function dub. Lee et al. [6], for instance, prove that the approximation ratio of their algorithms is $\max(\mathcal{A}, \pi/2 + \mathcal{B}/\pi)$, where $\mathcal{A} = 1 + \sup\{\mathrm{dub}(d)/d \mid d \geq 2\}$ and $\mathcal{B} = \sup\{\mathrm{dub}(d) + d \mid d \leq 2\}$. They claim without proof that $\mathrm{dub}(d) \leq 2\pi$ for $d \geq 2$ and derive from this that $\mathcal{A} = 1 + \pi$ and $\mathcal{B} \leq 5\pi/2 + 3$, leading to an approximation ratio of about 5.03. We give the first proof of $\mathcal{A} = 1 + \pi$, and improve the second bound to $\mathcal{B} = 2 + 2\pi$, improving the approximation ratio of their algorithm to $2 + 2/\pi + \pi/2 \approx 4.21$.

Savla et al. [9] prove that $\mathrm{dub}(d) \leq \kappa\pi$, where $\kappa \in [2.657, 2.658]$, and conjecture based on numerical experiments that the true bound is $7\pi/3$. We show that this is indeed true.

Results. We show that $d \mapsto \mathrm{dub}(d)$ is a decreasing function with two breakpoints, at $\sqrt{2}$ and at $d^* \approx 1.5874$ (see Fig. 1). More precisely, we have $\mathrm{dub}(0) = 7\pi/3$ and the two breakpoint

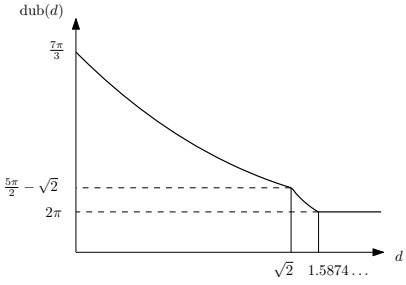


Fig. 1. The graph of $\mathrm{dub}(d)$

values are $\mathrm{dub}(\sqrt{2}) = 5\pi/2 - \sqrt{2}$, and $\mathrm{dub}(d^*) = 2\pi$. The function $\mathrm{dub}(d)$ is constant and equal to 2π for $d \geq d^*$.

For $0 \leq d < \sqrt{2}$ and for $d \geq d^*$, the supremum in (1) is in fact a maximum, and we give configurations σ, σ' at distance d such that $\ell(\sigma, \sigma') = \mathrm{dub}(d) + d$. Perhaps surprisingly, for $\sqrt{2} \leq d < d^*$, there are no such configurations—the supremum is not a maximum.

Because of the limited space, we omit most of the proofs. The reader can find a full version of this paper on the arXiv [3].

2 Preliminaries

Notations. For two points P and Q, we denote by $\overline{PQ}$ the line segment with endpoints P and Q, and by $\ddot{PQ}$ an arc of unit radius with endpoints P and Q. (If the length of $\overline{PQ}$ is less than two then there are four such arcs, so unless it is clear from the context, we will specify the supporting circle and the orientation of the arc.) We denote the length of the segment $\overline{PQ}$ as $|\overline{PQ}|$ or simply as $|PQ|$, and the length of the arc $\ddot{PQ}$ as $|\ddot{PQ}|$.

Without loss of generality, we assume that the starting configuration is $(0, 0, \alpha)$ and the final configuration is $(d, 0, \beta)$. Here, α and β express the orientation of the robot as an angle with the positive x-axis, and $d \geq 0$ is the Euclidean distance of the two configurations.

The *open* unit (radius) disks tangent to the starting and final configurations are denoted L_S, R_S, L_F, R_F, where the letters L or R depend on whether the disk is located on the left or right side of the direction vector. Let ℓ_S, r_S, ℓ_F, r_F denote the centers of L_S, R_S, L_F, R_F, respectively; their coordinates are $\ell_S = (-\sin\alpha, \cos\alpha)$, $r_S = (\sin\alpha, -\cos\alpha)$, $\ell_F = (d - \sin\beta, \cos\beta)$ and $r_F = (d + \sin\beta, -\cos\beta)$. Let the distance between two centers be denoted by: $d_{\mathrm{L}} = |\ell_S\ell_F|$, $d_{\mathrm{R}} = |r_S r_F|$, $d_{\mathrm{LR}} = |\ell_S r_F|$, and $d_{\mathrm{RL}} = |r_S\ell_F|$.

Dubins Paths. Dubins [1] showed that for two given configurations in the plane, shortest bounded-curvature paths consist of arcs of unit radius circles (C-segments) and straight line segments (S-segments); moreover, such shortest paths are of type CCC or CSC, or a substring thereof. These types of paths are referred to as *Dubins paths*.

For given $d \geq 0$, $\alpha, \beta \in [0, 2\pi]$, there are up to six types of Dubins paths. The two path types LSL and RSR use outer tangents—these path types exist for any choice of d, α, β. The two path types LSR and RSL use inner tangents, and exist only when the corresponding disks are disjoint. In particular, LSR exists if and only if $d_{\mathrm{LR}} \geq 2$, and RSL exists if and only if $d_{\mathrm{RL}} \geq 2$. The remaining two path types LRL and RLR exist whenever there is a disk tangent to the two disks, and so LRL exists if and only if $d_{\mathrm{L}} \leq 4$, and RLR exists if and only if $d_{\mathrm{R}} \leq 4$.

Dubins showed that in LRL- and RLR-paths the middle circular arc has length larger than π. This implies that of the two unit radius disks tangent to L_S and L_F, only one is a candidate for the middle arc of an LRL-path, and similar for RLR-paths.

For $d \geq 0$ and $0 \leq \alpha, \beta \leq 2\pi$, we define $\textsc{lsl}(d, \alpha, \beta)$ to be the length of the LSL-path from $(0, 0, \alpha)$ to $(d, 0, \beta)$. We define $\textsc{rsr}, \textsc{lsr}, \textsc{rsl}, \textsc{lrl}, \textsc{rlr}$ similarly, defining the length to be ∞ if no path of that type exists. The length of the shortest bounded-curvature path from S to F is then

$$\ell(d, \alpha, \beta) = \min \big\{ \textsc{lsl}(d, \alpha, \beta), \textsc{rsr}(d, \alpha, \beta), \textsc{lsr}(d, \alpha, \beta), \textsc{rsl}(d, \alpha, \beta), \\ \textsc{lrl}(d, \alpha, \beta), \textsc{rlr}(d, \alpha, \beta) \big\},$$

and our goal is to bound $\mathrm{dub}(d) = \sup_{0 \leq \alpha, \beta \leq 2\pi} \ell(d, \alpha, \beta) - d$. (Note that the supremum here is not always a maximum as the function ℓ is not continuous.)

We will often suppress the argument d for these functions when the distance d is fixed and understood.

Monotonicity of the Dubins Cost Function. Let $\square = [0, 2\pi]^2$ denote the range of (α, β). Consider two distances $d_1 < d_2$ and $(\alpha, \beta) \in \square$, and assume that we have a bounded-curvature path from $(0, 0, \alpha)$ to $(d_1, 0, \beta)$ of length $\ell \leq \mathrm{dub}(d_1) + d_1$. If this path has a horizontal tangent where the path is oriented to the right (in the direction of the positive x-axis), then we can insert a horizontal segment of length $d_2 - d_1$ at this point, and obtain a path from $(0, 0, \alpha)$ to $(d_2, 0, \beta)$ of length $\ell + (d_2 - d_1) \leq \mathrm{dub}(d_1) + d_2$. See, for instance, Fig. 2(a).

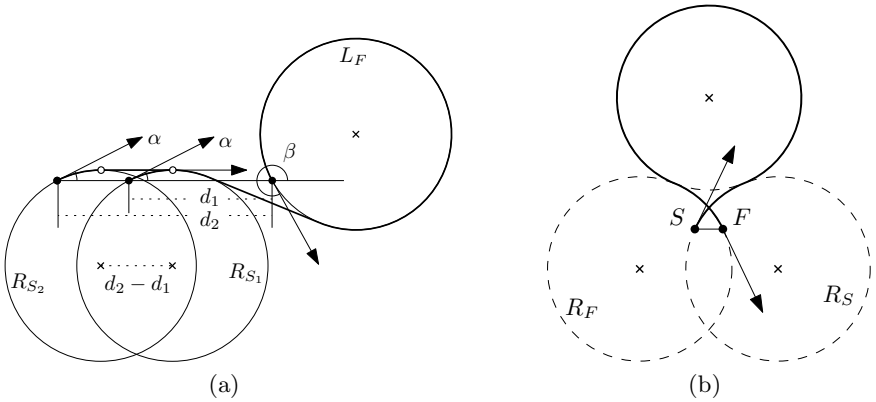


Fig. 2. (a) $\ell(d_2, \alpha, \beta) \leq \textsc{rsl}(d_1, \alpha, \beta) + (d_2 - d_1)$, (b) an RLR-path that does not have a right horizontal tangent

If this was possible for all $(\alpha, \beta) \in \square$, then it would imply that $\mathrm{dub}(d_2) \leq \mathrm{dub}(d_1)$, and it would follow that the Dubins cost function is monotone. Unfortunately, not all Dubins paths have horizontal tangents with the correct orientation (see Fig. 2(b) for an example), and so proving the monotonicity of the Dubins cost function will require much more work. However, we can prove that this holds for any CSC-path by showing that any of these path types must have a horizontal tangent:

Lemma 1. *Let $d_1 < d_2$, and $(\alpha, \beta) \in \square$. If there is a path of length ℓ of type RSR, LSL, LSR, or RSL from $(0, 0, \alpha)$ to $(d_1, 0, \beta)$, then there is a path of length $\ell + (d_2 - d_1)$ from $(0, 0, \alpha)$ to $(d_2, 0, \beta)$.*

Symmetries and a New Parameterization. For a fixed $d \geq 0$, determining $\mathrm{dub}(d)$ essentially amounts to finding $(\alpha, \beta) \in \square$ maximizing $\ell(d, \alpha, \beta)$ ("essentially" since the maximum may not actually be assumed).

We observe that the function $\ell(d, \alpha, \beta)$ has two symmetries; a point symmetry in (π, π) and a reflection around the line $\beta = 2\pi - \alpha$ (see Kim [4], or Goaoc et al. [2], or our full version [3]).

It follows that $\sup_{(\alpha,\beta)\in\square} \ell(d, \alpha, \beta) = \sup_{(\alpha,\beta)\in\Delta} \ell(d, \alpha, \beta)$, where Δ is the triangle with corners $(0, 0)$, (π, π), and $(0, 2\pi)$, or in other words the region $\Delta : 0 \leq \alpha \leq \pi, \alpha \leq \beta \leq 2\pi - \alpha$. In the following we will thus be able to restrict our considerations to the triangle Δ (see Fig. 3(a)).

We now introduce a new parameterization of the (α, β)-plane, which will sometimes be more convenient to work with: $\sigma = (\beta + \alpha)/2$ and $\delta = (\beta - \alpha)/2$. In other words, we have $\alpha = \sigma - \delta$ and $\beta = \sigma + \delta$.

In the (σ, δ)-representation, the triangle Δ is the region $0 \leq \sigma \leq \pi$ and $0 \leq \delta \leq \sigma$, or the bottom right half of the square $\Gamma = [0, \pi]^2$ (see Fig. 3(b)).

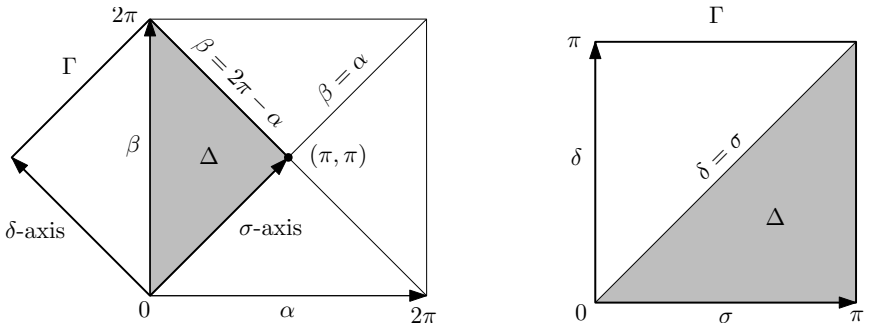


Fig. 3. Δ in two different representations

The following lemma can be shown by easy calculations:

Lemma 2. *The center distances in terms of σ and δ are:*

$$d_L^2 = |\ell_S \ell_F|^2 = d^2 - 4d \sin\delta \cos\sigma + 4\sin^2\delta \tag{2}$$

$$d_R^2 = |r_S r_F|^2 = d^2 + 4d \sin\delta \cos\sigma + 4\sin^2\delta \tag{3}$$

$$d_{LR}^2 = |\ell_S r_F|^2 = d^2 + 4d \cos\delta \sin\sigma + 4\cos^2\delta \tag{4}$$

$$d_{RL}^2 = |r_S \ell_F|^2 = d^2 - 4d \cos\delta \sin\sigma + 4\cos^2\delta \tag{5}$$

The Case $d = 0$. We first argue that $\mathrm{dub}(0) = 7\pi/3$. The case $d = 0$ is much easier since there is only one degree of freedom: Without loss of generality we

can assume $\alpha = 0$. It is easy to verify that for any β there is a CCC-path of length at most $7\pi/3$. For $\beta = \pi$, no Dubins path has length shorter than $7\pi/3$, and so $\mathrm{dub}(0) = 7\pi/3$. In the rest of this paper we can therefore mostly assume $d > 0$, and avoid some degeneracies.

The Three Cases. We subdivide the pairs of orientations $(\alpha, \beta) \in \Gamma$ (and hence in the triangle Δ) into three cases. We define:

$$A = \{(\alpha,\beta) \in \Gamma \mid d_{\mathrm{LR}}(\alpha,\beta) < 2 \text{ and } d_{\mathrm{RL}}(\alpha,\beta) < 2\},$$
$$B = \{(\alpha,\beta) \in \Gamma \mid d_{\mathrm{LR}}(\alpha,\beta) < 2 \text{ and } d_{\mathrm{RL}}(\alpha,\beta) \geq 2\},$$
$$C = \{(\alpha,\beta) \in \Gamma \mid d_{\mathrm{LR}}(\alpha,\beta) \geq 2\}.$$

For clarity, let us define the parts of A, B and C that lie inside the triangle Δ:

$$A^{\Delta} = A \cap \Delta, \quad B^{\Delta} = B \cap \Delta, \quad \text{and} \quad C^{\Delta} = C \cap \Delta.$$

Note that $(\alpha, \beta) \in A$ if and only if $L_S \cap R_F \neq \emptyset$ and $R_S \cap L_F \neq \emptyset$. Since $|\ell_S r_S| = 2$ and $|\ell_F r_F| = 2$, the triangle-inequality implies $d_{\mathrm{L}} < 4$ and $d_{\mathrm{R}} < 4$, so both LRL- and RLR-paths exist. We will concentrate entirely on these two path types in case A and prove that other path types cannot be shorter. Note that case A does not occur for $d \geq 2$, as we have $8 > d_{\mathrm{LR}}^2 + d_{\mathrm{RL}}^2 = 2d^2 + 8\cos^2\delta \geq 2d^2$.

Similarly, we have $(\alpha, \beta) \in B$ if and only if $L_S \cap R_F \neq \emptyset$ and $R_S \cap L_F = \emptyset$. As in case A, LRL-paths must exist, and $d_{\mathrm{RL}} \geq 2$ implies that RSL-paths exist as well. We will concentrate on RSL- and LRL-paths in case B and prove that other path types cannot be shorter.

Finally, $(\alpha, \beta) \in C$ if and only if $L_S \cap R_F = \emptyset$. This implies that LSR-paths exist. We will study LSR- and RSR-paths in case C.

We can now refine our Dubins-function $\mathrm{dub}(d)$ by defining the following three functions:

$$\mathrm{dub}_X(d) = \sup_{(\alpha,\beta)\in X^{\Delta}} \ell(d,\alpha,\beta) - d \quad \text{for } X \in \{A, B, C\}$$

and we have $\mathrm{dub}(d) = \max\{\, \mathrm{dub}_A(d),\ \mathrm{dub}_B(d),\ \mathrm{dub}_C(d)\,\}$.

Case C and a Lower Bound. Kim proved in his master thesis [4] that for $(\alpha, \beta) \in C^{\Delta}$, we have $\ell(d, \alpha, \beta) \leq 2\pi + d$ (a more polished proof can also be found in our full version [3]). We show that this bound is optimal. Since $\mathrm{dub}(d) \geq \mathrm{dub}_C(d)$, this establishes a lower bound for the Dubins cost function.

Lemma 3. *For any $d > 0$ we have $\mathrm{dub}_C(d) = 2\pi$.*

It remains to study cases A and B. For lack of space, we sketch the arguments for case A only in this extended abstract.

3 Regions of the Square Γ for $0 < d < 2$

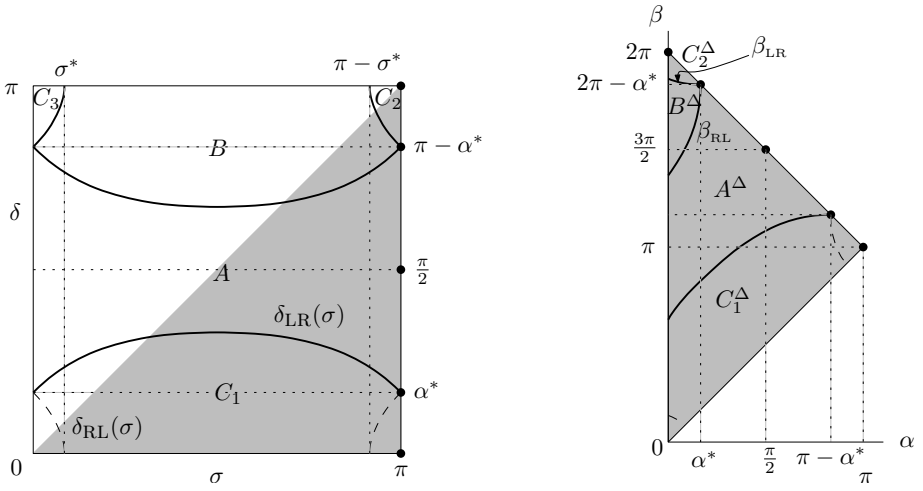
We will now describe the regions A, B, and C of the square Γ geometrically. For our purposes it will be sufficient to do this when $0 < d < 2$, so we assume this throughout this section.

We define the angle $\alpha^* = \arcsin \frac{d}{2}$, and observe that for $(\alpha, \beta) = (\alpha^*, 2\pi - \alpha^*)$ as well as for $(\alpha, \beta) = (\pi - \alpha^*, \pi + \alpha^*)$ we have $R_S = R_F$. Let us also define $\sigma^* = \arcsin \frac{d}{4}$.

Lemma 4. *For $0 < d < 2$,*

- *There is a curve $(\sigma, \delta_{LR}(\sigma))_{0 \le \sigma \le \pi}$ in Γ that connects the two points $(0, \alpha^*)$ and (π, α^*), lies strictly between $\delta = \alpha^*$ and $\delta = \pi/2$ except for its endpoints, and such that $d_{LR} = 2$ on the curve, $d_{LR} < 2$ between the curve and the line $\delta = \pi/2$, and $d_{LR} > 2$ below the curve;*
- *There is a curve $(\sigma, \delta_{RL}(\sigma))_{0 \le \sigma \le \sigma^*}$ in Γ that connects the two points $(0, \alpha^*)$ and $(\sigma^*, 0)$, lies strictly below $\delta = \alpha^*$ except for its left endpoint, and such that $d_{RL} = 2$ on the curve, $d_{RL} < 2$ between the curve and the line $\delta = \alpha^*$, and $d_{RL} > 2$ below the curve;*
- *For $\alpha^* \le \delta \le \pi/2$, we have $d_{RL} \le 2$ with equality only for the two points $(0, \alpha^*)$ and (π, α^*);*
- *For $\sigma^* < \sigma < \pi - \sigma^*$, $0 \le \delta \le \alpha^*$, we have $d_{RL} < 2$.*

We now exploit from Lemma 2 that $d_{\mathrm{RL}}(\sigma, \delta) = d_{\mathrm{LR}}(\sigma, \pi - \delta)$ and $d_{\mathrm{RL}}(\sigma, \delta) = d_{\mathrm{RL}}(\pi - \sigma, \delta)$ to obtain our desired subdivision. See Fig. 4.



(a) Subdivision of Γ (b) Subdivision of Δ

Fig. 4. The subdivision of Γ and Δ, shown for $d = 1$

- C_1 is the region $\delta \le \delta_{\mathrm{LR}}(\sigma)$. Inside this region we have $d_{\mathrm{LR}} \ge 2$.
- C_2 is the region $\pi - \sigma^* \le \sigma \le \pi$, $\pi - \delta_{\mathrm{RL}}(\sigma) \le \delta \le \pi$. Here we have $d_{\mathrm{LR}} \ge 2$ and $d_{\mathrm{RL}} \ge 2$.
- C_3 is the region $0 \le \sigma \le \sigma^*$, $\pi - \delta_{\mathrm{RL}}(\sigma) \le \delta \le \pi$. Here we have $d_{\mathrm{LR}} \ge 2$ and $d_{\mathrm{RL}} \ge 2$.
- A is the region $\delta_{\mathrm{LR}}(\sigma) < \delta < \pi - \delta_{\mathrm{LR}}(\sigma)$. In this region we have $d_{\mathrm{LR}} < 2$ and $d_{\mathrm{RL}} < 2$.

- Finally, B is the remaining region, where $\pi - \delta_{\text{LR}}(\sigma) \leq \delta$, but excluding $C_2 \cup C_3$. In this region we have $d_{\text{LR}} < 2$ and $d_{\text{RL}} \geq 2$.

It is clear from the description in Lemma 4 that the five regions A, B, C_1, C_2, and C_3 are σ-monotone, meaning that a line parallel to the δ-axis intersects each region in a single interval.

4 Explicit Expressions for the Length of *CCC*-Paths

In this section we develop explicit formulas for the length of *LRL*- and *RLR*-paths.

We start by a change of perspective, and consider all configurations where d_{L} is fixed. We choose a coordinate system where the line $\ell_S \ell_F$ is horizontal, and ℓ_S lies to the left of ℓ_F, see Fig 5. We have drawn the two unit-radius disks M^{L} and N^{L} tangent to L_S and L_F. The points of tangency are S_2^{L} and F_2^{L} for M^{L}, and S_1^{L} and F_1^{L} for N^{L}. Dubins [1] showed that the length of the middle circular arc of a *CCC*-path is larger than π, and so it lies on M^{L}.

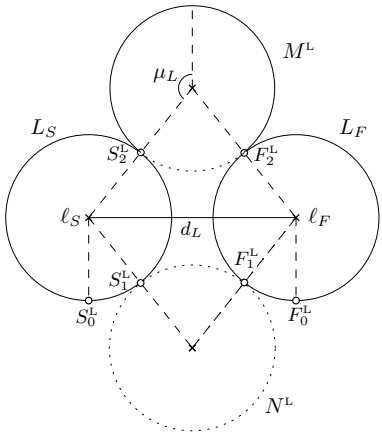


Fig. 5. Locations of S and F

So any *LRL*-path first follows a leftwards arc on L_S, then switches to M^{L} at S_2^{L}, follows the rightwards arc on M^{L} until it reaches F_2^{L}, and finally follows a leftwards arc on L_F. We note that the middle arc on M^{L} does not depend on the specific endpoints S and F, it is determined entirely by S_2^{L} and F_2^{L}, and therefore by d_{L}. Let μ_{L} denote half the length of the middle circular arc $S_2^{\text{L}} \overset{\cdots}{F_2^{\text{L}}}$. We have $\pi/2 < \mu_{\text{L}} \leq \pi$, and $4\sin(\pi - \mu_{\text{L}}) = d_{\text{L}}$, so that we have $\mu_{\text{L}} = \pi - \arcsin \frac{d_{\text{L}}}{4}$. The same considerations apply to *RLR*-paths. We define μ_{R} as half the length of the middle circular arc of the *RLR*-path, and obtain $\mu_{\text{R}} = \pi - \arcsin \frac{d_{\text{R}}}{4}$. Note that

$$\angle S_1^{\text{L}} \ell_S S_2^{\text{L}} = \angle F_2^{\text{L}} \ell_F F_1^{\text{L}} = 2\mu_{\text{L}} - \pi, \tag{6}$$

$$\angle S_1^{\text{R}} r_S S_2^{\text{R}} = \angle F_2^{\text{R}} r_F F_1^{\text{R}} = 2\mu_{\text{R}} - \pi. \tag{7}$$

Lemma 5. *We have*

- *S lies on the counter-clockwise arc $\overset{\cdots}{S_1^L S_2^L}$ of L_S if and only if $d_{RL} < 2$;*
- *F lies on the clockwise arc $\overset{\cdots}{F_2^R F_1^R}$ of R_F if and only if $d_{RL} < 2$;*
- *S lies on the clockwise arc $\overset{\cdots}{S_1^R S_2^R}$ of R_S if and only if $d_{LR} < 2$;*
- *F lies on the counter-clockwise arc $\overset{\cdots}{F_2^L F_1^L}$ of L_F if and only if $d_{LR} < 2$.*

Moreover, if $d_{RL} \geq 2$ and in addition $0 \leq \alpha \leq \pi/2$, then S lies on the counter-clockwise arc $\overset{\cdots}{S_0^L S_1^L}$ of L_S.

Lemma 5 allows us to develop explicit formulas for $\textsc{lrl}(\sigma,\delta)$ – the length of *LRL*-paths, and $\textsc{rlr}(\sigma,\delta)$ – the length of *RLR*-paths.

Lemma 6. *For any* σ,δ*, we have* $\textsc{lrl}(\sigma,\delta)\equiv 4\mu_{\scriptscriptstyle L}+2\delta \pmod{2\pi}$ *and* $\textsc{rlr}(\sigma,\delta)\equiv 4\mu_{\scriptscriptstyle R}-2\delta \pmod{2\pi}$*. Especially, for* $(\sigma,\delta)\in A$*, we have*

$$\textsc{lrl}(\sigma,\delta)=4\mu_{\scriptscriptstyle L}+2\delta-2\pi \qquad \textsc{rlr}(\sigma,\delta)=4\mu_{\scriptscriptstyle R}-2\delta.$$

Proof. An *LRL*-path consists of an initial left-turning arc of length γ_1 on L_S, a right-turning arc of length $2\mu_{\textsc{l}}$ on the middle disk, and a final left-turning arc of length γ_2 on L_F. This means that the total change in orientation is $\gamma_1-2\mu_{\textsc{l}}+\gamma_2$. On the other hand, since the initial orientation is α and the final orientation is β, this must be equal, up to multiples of 2π, to $\beta-\alpha=2\delta$. It follows that

$$\textsc{lrl}=\gamma_1+\gamma_2+2\mu_{\textsc{l}}=\gamma_1-2\mu_{\textsc{l}}+\gamma_2+4\mu_{\textsc{l}}\equiv 2\delta+4\mu_{\textsc{l}}.$$

For *RLR*-paths, we can similarly observe that $-\gamma_1+2\mu_{\textsc{r}}-\gamma_2\equiv 2\delta \pmod{2\pi}$ (here, γ_1 and γ_2 are the right-turning arcs) and obtain

$$\textsc{rlr}=\gamma_1+\gamma_2+2\mu_{\textsc{r}}=4\mu_{\textsc{r}}-(-\gamma_1+2\mu_{\textsc{r}}-\gamma_2)\equiv 4\mu_{\textsc{r}}-2\delta.$$

Let us now assume that $(\sigma,\delta)\in A$. We have $2\alpha^*<2\delta<2\pi-2\alpha^*$. On the other hand, $\gamma_1+\gamma_2-2\mu_{\textsc{l}}\geq -2\mu_{\textsc{l}}\geq -2\pi$. By Lemma 5, $S\in S_1^{\textsc{l}}\dot{S}_2^{\textsc{l}}$ and $F\in F_2^{\textsc{l}}\dot{F}_1^{\textsc{l}}$, and so we can extend the *LRL*-path to a complete clockwise loop. The loop uses additional left-turns ζ_1 on L_S and ζ_2 on L_F, and an additional right turn of length $2\mu_{\textsc{l}}$ on $N^{\textsc{l}}$. Let the length of the arc ζ_1 be δ_1 and that of ζ_2 be δ_2. The total turning angle of a clockwise loop is -2π, and thus $\gamma_1+\gamma_2+\delta_1+\delta_2-4\mu_{\textsc{l}}=-2\pi$. Since $2\mu_{\textsc{l}}\leq 2\pi$ this implies that $\gamma_1+\gamma_2-2\mu_{\textsc{l}}\leq 0$. From $-2\pi\leq\gamma_1+\gamma_2-2\mu_{\textsc{l}}\leq 0$ and $0\leq 2\alpha^*<2\delta<2\pi-2\alpha^*\leq 2\pi$, we conclude that $\gamma_1+\gamma_2-2\mu_{\textsc{l}}\equiv 2\delta \pmod{2\pi}$ implies $\gamma_1+\gamma_2-2\mu_{\textsc{l}}=2\delta-2\pi$. This shows that $\textsc{lrl}=4\mu_{\textsc{l}}+2\delta-2\pi$.

For *RLR*-paths, we could argue analogously, or we can simply observe that

$$\textsc{rlr}(\sigma,\delta)=\textsc{lrl}(\pi-\sigma,\pi-\delta)=4\mu_{\textsc{r}}(\sigma,\delta)-2\delta. \quad \square$$

5 *CCC*-Paths for $d<2$ and Case A

When $d<2$, both *LRL*- and *RLR*-paths exist for any $(\sigma,\delta)\in\Gamma$. In this section we analyze the length of these two *CCC*-paths for $0<d<2$.

We define three functions $\textsc{l}$, $\textsc{r}$, and $\textsc{c}$ on Γ:

$$\textsc{l}(\sigma,\delta)=4\mu_{\textsc{l}}(\sigma,\delta)+2\delta-2\pi \tag{8}$$

$$\textsc{r}(\sigma,\delta)=4\mu_{\textsc{r}}(\sigma,\delta)-2\delta \tag{9}$$

$$\textsc{c}(\sigma,\delta)=\min\{\textsc{l}(\sigma,\delta),\textsc{r}(\sigma,\delta)\}. \tag{10}$$

While these functions are defined and continuous everywhere on Γ, we have shown in Lemma 6 only that $\textsc{lrl}(\sigma,\delta)=\textsc{l}(\sigma,\delta)$ for $(\sigma,\delta)\in A\cup B$, and $\textsc{rlr}(\sigma,\delta)=\textsc{r}(\sigma,\delta)$ for $(\sigma,\delta)\in A$. It follows that $\textsc{c}(\sigma,\delta)$ is the length of the shortest *CCC*-path for $(\sigma,\delta)\in A$.

Recall that $\alpha^* = \arcsin \frac{d}{2}$. We define the following rectangle $\Xi \subset \Gamma$:

$$\Xi: \quad 0 \le \sigma \le \pi \quad \text{and} \quad \alpha^* \le \delta \le \pi - \alpha^*.$$

Note that $A \subset \Xi$ (see Fig. 4(a)). In the interior of Ξ, we have $\sin\delta > d/2$, which implies $\cos\delta^2 < 1 - d^2/4$. So for $\alpha^* < \delta \le \pi/2$, we have $d_{\mathrm{RL}} < 2$ by (5), and for $\pi/2 \le \delta < \pi - \alpha^*$, we have $d_{\mathrm{LR}} < 2$ by (4). By the triangle inequality it follows that $d_{\mathrm{L}} < 4$ and $d_{\mathrm{R}} < 4$.

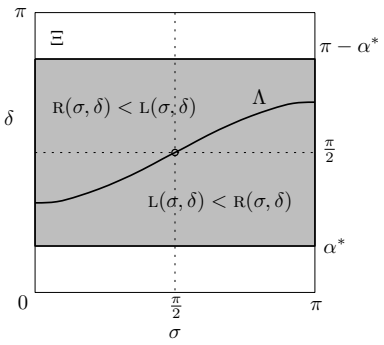


Fig. 6. The curve Λ where $\mathrm{L} = \mathrm{R}$

The following lemma is proven using the derivatives of the functions L and R.

Lemma 7. *For $(\sigma, \delta) \in \Xi$, the function*

- $\sigma \mapsto \mathrm{L}(\sigma, \delta)$ *is decreasing, while* $\sigma \mapsto \mathrm{R}(\sigma, \delta)$ *is increasing,*
- $\delta \mapsto \mathrm{L}(\sigma, \delta)$ *is increasing, while* $\delta \mapsto \mathrm{R}(\sigma, \delta)$ *is decreasing.*

Since $\mathrm{C}(\sigma, \delta)$ is continuous, it assumes its maximum on Ξ. The following is one of our key lemmas:

Lemma 8. *The function $\mathrm{C}(\sigma, \delta)$ has no local extremum in the interior of Ξ except at $(\pi/2, \pi/2)$.*

Proof. By Lemma 7, neither L nor R has a local extremum in the interior of Ξ, so any local extremum of $\mathrm{C}(\sigma, \delta)$ must be a point in the set Λ of points (σ, δ) with $\mathrm{L}(\sigma, \delta) = \mathrm{R}(\sigma, \delta)$. By Lemma 7, Λ is a δ-monotone curve. Since $\mathrm{L}(\sigma, \delta) = \mathrm{R}(\pi - \sigma, \pi - \delta)$, the curve Λ passes through the point $(\pi/2, \pi/2)$. By Lemma 7, this implies that $\mathrm{L}(\sigma, \delta) < \mathrm{R}(\sigma, \delta)$ for the quadrant $\pi/2 \le \sigma \le \pi$, $\alpha^* \le \delta \le \pi/2$, and that $\mathrm{R}(\sigma, \delta) < \mathrm{L}(\sigma, \delta)$ for the quadrant $0 \le \sigma \le \pi/2$, $\pi/2 \le \delta \le \pi - \alpha^*$ except at the point $(\pi/2, \pi/2)$. By point symmetry, we can restrict our attention to the range $\pi/2 < \sigma < \pi$, $\pi/2 < \delta < \pi - \alpha^*$.

Assume for a contradiction that $(\sigma, \delta) \in \Lambda$ is a local extremum of L, restricted to Λ. This implies that the gradient $\nabla\mathrm{L}(\sigma, \delta)$ and the normal of Λ in (σ, δ) are linearly dependent, by the method of Lagrange Multipliers. The normal of Λ is the gradient of $\mathrm{L}(\sigma, \delta) - \mathrm{R}(\sigma, \delta)$, so $\nabla\mathrm{L}(\sigma, \delta)$ and $\nabla\mathrm{R}(\sigma, \delta)$ must be linearly dependent.

Let $D_{\mathrm{L}} = d_{\mathrm{L}}\sqrt{1 - (d_{\mathrm{L}}/4)^2}$ and $D_{\mathrm{R}} = d_{\mathrm{R}}\sqrt{1 - (d_{\mathrm{R}}/4)^2}$. For the two vectors to be linearly dependent, we would have to have

$$D_{\mathrm{L}} \frac{\partial \mathrm{L}}{\partial \delta}(\sigma, \delta) + D_{\mathrm{R}} \frac{\partial \mathrm{R}}{\partial \delta}(\sigma, \delta) = 0,$$

which means $2D_{\mathrm{L}} - 2D_{\mathrm{R}} - 8\cos\delta\sin\delta = 0$. In the range under consideration, $-8\cos\delta\sin\delta > 0$. We will show that $D_{\mathrm{L}} > D_{\mathrm{R}}$, a contradiction. We have

$$\begin{aligned} 16(D_{\mathrm{L}}^2 - D_{\mathrm{R}}^2) &= d_{\mathrm{L}}^2(16 - d_{\mathrm{L}}^2) - d_{\mathrm{R}}^2(16 - d_{\mathrm{R}}^2) = (d_{\mathrm{L}}^2 - d_{\mathrm{R}}^2)(16 - (d_{\mathrm{L}}^2 + d_{\mathrm{R}}^2)) \\ &= -8d\cos\sigma\sin\delta(16 - (2d^2 + 8\sin^2\delta)). \end{aligned}$$

Since $\cos\sigma < 0$ and $d < 2$, the expression is positive. □

The proof of Lemma 8 implies that the maximum of the function C must happen either on the vertical side $\sigma = \pi, \pi/2 < \delta \leq \pi - \alpha^*$, or on the horizontal side $\delta = \pi - \alpha^*, \pi/2 < \sigma \leq \pi$. Let (σ_A, δ_A) be the point where the maximum of C is realized. The point (σ_A, δ_A) is unique if we require $\sigma_A + \delta_A > \pi$, and there is a symmetric point $(\pi - \sigma_A, \pi - \delta_A)$. Let us define the function $A(d)$ for $0 \leq d < 2$

$$A(d) = LRL(\sigma_A, \delta_A) = RLR(\sigma_A, \delta_A). \tag{11}$$

There is an important breakpoint at $d = \sqrt{2}$:

Lemma 9. *The maximum* $A(d)$ *occurs with* $\sigma_A = \pi$ *when* $0 \leq d \leq \sqrt{2}$*, and with* $\delta_A = \pi - \alpha^*$ *when* $\sqrt{2} \leq d < 2$*.*

Proof. Using (2) and (3), we have $d_L(\pi, \pi - \alpha^*) = 2d$ and $d_R(\pi, \pi - \alpha^*) = 0$, which implies by (8) and (9), $L(\pi, \pi - \alpha^*) = 4\pi - 6\alpha^*$ and $R(\pi, \pi - \alpha^*) = 2\pi + 2\alpha^*$.

Since $\alpha^* = \arcsin(d/2)$, we have $R(\pi, \pi - \alpha^*) < L(\pi, \pi - \alpha^*)$ for $d < \sqrt{2}$, equality for $d = \sqrt{2}$, and $R(\pi, \pi - \alpha^*) > L(\pi, \pi - \alpha^*)$ for $d > \sqrt{2}$. In the first case, Lemma 7 implies that the maximum must occur on the vertical side at $\sigma = \pi$, in the last case it must occur on the horizontal side at $\delta = \pi - \alpha^*$. For $d = \sqrt{2}$ the maximum occurs at the corner $(\sigma_A, \delta_A) = (\pi, \pi - \alpha^*)$. □

Again using the method of Lagrange Multipliers, we show that on the interval $0 \leq d \leq \sqrt{2}$, $\mathrm{dub}_A(d)$ is determined by the function $A(d)$.

Lemma 10. *On the interval* $0 \leq d \leq \sqrt{2}$*,*

- *the function* $d \mapsto A(d)$ *is monotonically increasing from* $7\pi/3$ *to* $5\pi/2$*,*
- *the function* $d \mapsto A(d) - d$ *is monotonically decreasing from* $7\pi/3$ *to* $5\pi/2 - \sqrt{2}$*,*
- $\max_{(\sigma,\delta)\in\Xi} C(d, \sigma, \delta) = A(d)$*.*

We now have all the tools to discuss case A. In case A, which occurs only for $d < 2$, we have $d_{LR} < 2$ and $d_{RL} < 2$. We will now justify that it suffices to study *CCC*-paths in this case, as no other path type can be shorter. Since *LSR*- and *RSL*-paths do not exist, it is enough to show the following lemma:

Lemma 11. *For* $(\sigma, \delta) \in A$*, we have* $LRL(\sigma, \delta) \leq LSL(\sigma, \delta)$ *and* $RLR(\sigma, \delta) \leq RSR(\sigma, \delta)$*.*

Proof. Let γ_1 and γ_2 be the length of the left-turning arcs of an *LRL*-path. By Lemma 5, the endpoints S and F lie on the counterclockwise arcs $\overset{\cdot\cdot}{S_1^L S_2^L}$ of L_S and $\overset{\cdot\cdot}{F_2^L F_1^L}$ of L_F. Consider now Fig. 5. The *LSL*-path turns left on L_S until S_0^L, goes along the tangent to F_0^L, then turns left on L_F until it reaches F. Since $\angle S_2^L \ell_S S_0^L = \mu_L$ and $|S_0^L F_0^L| = d_L$, we have

$$LSL - LRL = d_L + 2(2\pi - \mu_L) - 2\mu_L = d_L + 4(\pi - \mu_L) \geq 0$$

since $\mu_L \leq \pi$. The analogous argument shows that $RLR \leq RSR$. □

Lemma 12. *For* $0 < d < \sqrt{2}$*,* $\mathrm{dub}_A(d) = A(d) - d$*. In other words, the maximum is realized by the unique point* (σ_A, δ_A) *on the segment* $\sigma = \pi$*,* $\pi/2 \leq \delta \leq \pi - \alpha^*$ *where* $L(d, \sigma_A, \delta_A) = R(d, \sigma_A, \delta_A)$*.*

Proof. By Lemma 10, we have $\max_{(\sigma,\delta)\in\Xi} \mathrm{C}(d,\sigma,\delta) = \mathrm{A}(d)$, and the maximum is assumed at the point $(\sigma_\mathrm{A},\delta_\mathrm{A}) \in A^\Delta$. By Lemma 11, we have $\ell(d,\sigma,\delta) = \mathrm{C}(d,\sigma,\delta)$ for $(\sigma,\delta) \in A^\Delta$. Since $(\sigma_\mathrm{A},\delta_\mathrm{A}) \in A^\Delta \subset \Xi$, this means that $\mathrm{dub}_A(d) = \sup_{(\sigma,\delta)\in A^\Delta} \ell(d,\sigma,\delta) - d = \mathrm{C}(d,\sigma_\mathrm{A},\delta_\mathrm{A}) - d = \mathrm{A}(d) - d$. □

6 *RSL*-Paths for $d < 2$ and Case B

Case B is the situation where $d_{\mathrm{LR}}(\alpha,\beta) < 2$ and $d_{\mathrm{RL}}(\alpha,\beta) \geq 2$. The methods are similar to those of the previous section, and make use of the closed formulas for the derivatives of the length functions for *CSC*-paths derived by Goaoc et al. [2]. The full proofs can be found on the arXiv [3].

7 The Dubins Cost Function

We now put all the pieces together and summarize our results.

Theorem 1. *The function* $\mathrm{dub}(d)$ *has two breakpoints at* $\sqrt{2}$ *and* $d^* \approx 1.5874$. *For* $d < \sqrt{2}$, $\mathrm{dub}(d) = \mathrm{dub}_A(d) \leq \mathrm{dub}_A(0) = \frac{7\pi}{3}$. *For* $\sqrt{2} \leq d < d^*$, $\mathrm{dub}(d) = \mathrm{dub}_B(d) \leq \mathrm{dub}_B(\sqrt{2}) = \frac{5\pi}{2} - \sqrt{2}$. *For* $d \geq d^*$, *we have* $\mathrm{dub}(d) = 2\pi$.

References

1. Dubins, L.E.: On curves of minimal length with a constraint on average curvature, and prescribed initial and terminal positions and tangents. American Journal of Mathematics 79(3), 497–516 (1957)
2. Goaoc, X., Kim, H.-S., Lazard, S.: Bounded-curvature shortest paths through a sequence of points. Technical Report inria-00539957, INRIA (2010), http://hal.inria.fr/inria-00539957/en
3. Kim, H.-S., Cheong, O.: The cost of bounded curvature (2011), http://arxiv.org/abs/1106.6214
4. Kim, J.-H.: The upper bound of bounded curvature path. Master's thesis, KAIST (2008)
5. LaValle, S.M.: Planning Algorithms. Cambridge University Press (2006)
6. Lee, J.-H., Cheong, O., Kwon, W.-C., Shin, S.-Y., Chwa, K.-Y.: Approximation of Curvature-Constrained Shortest Paths through a Sequence of Points. In: Paterson, M. (ed.) ESA 2000. LNCS, vol. 1879, pp. 314–325. Springer, Heidelberg (2000)
7. Ma, X., Castañón, D.A.: Receding horizon planning for Dubins traveling salesman problems. In: 45th IEEE Conference on Decision and Control, pp. 5453–5458 (December 2006)
8. Le Ny, J., Feron, E., Frazzoli, E.: The curvature-constrained traveling salesman problem for high point densities. In: Proceedings of the 46th IEEE Conference on Decision and Control, pp. 5985–5990 (2007)
9. Savla, K., Frazzoli, E., Bullo, F.: Traveling salesperson problems for the Dubins vehicle. IEEE Transactions on Automatic Control 53, 1378–1391 (2008)

Optimally Solving a Transportation Problem Using Voronoi Diagrams

Darius Geiß*, Rolf Klein*, and Rainer Penninger*

University of Bonn, Institute of Computer Science I,
Friedrich-Ebert-Allee 144, D-53113 Bonn, Germany
wdarius@gmx.de, rolf.klein@uni-bonn.de, penninge@cs.uni-bonn.de

Abstract. In this paper we consider the following variant of the well-known Monge-Kantorovich transportation problem. Let S be a set of n point sites in $\mathbb{R}^d$. A bounded set $C \subset \mathbb{R}^d$ is to be distributed among the sites $p \in S$ such that (i), each p receives a subset C_p of prescribed volume and (ii), the average distance of all points z of C from their respective sites p is minimized. In our model, volume is quantified by a measure μ, and the distance between a site p and a point z is given by a function $d_p(z)$. Under quite liberal technical assumptions on C and on the functions $d_p(\cdot)$ we show that a solution of minimum total cost can be obtained by intersecting with C the Voronoi diagram of the sites in S, based on the functions $d_p(\cdot)$ equipped with suitable additive weights. Moreover, this optimum partition is unique, up to subsets of C of measure zero. Unlike the deep analytic methods of classical transportation theory, our proof is based on direct geometric arguments.

Keywords: Monge-Kantorovich transportation problem, earth mover's distance, Voronoi diagram with additive weights, Wasserstein metric.

1 Introduction

In 1781, Gaspard Monge [7] raised the following problem. Given two sets C and S of equal mass in $\mathbb{R}^d$, transport each mass unit of C to a mass unit of S at minimal cost. More precisely, given two measures μ and ν, find a map f satisfying $\mu(f^{-1}(\cdot)) = \nu(\cdot)$ that minimizes

$$\int d(z, f(z))\, \mathrm{d}\mu(z),$$

where $d(z, z')$ describes the cost of moving z to z'.

Because of its obvious relevance to economics, and perhaps due to its mathematical challenge, this problem has received a lot of attention. Even with the Euclidean distance as cost function d it is not at all clear in which cases an optimal map f exists. Progress by Appell [1] was honored with a prize by the

* This work was supported by the European Science Foundation (ESF) in the EUROCORES collaborative research project EuroGIGA/VORONOI.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 264–274, 2012.

Academy of Paris in 1887. Kantorovich [6] achieved a breakthrough by solving a relaxed version of Monge's original problem. In 1975, he received a Nobel prize in Economics; see Gangbo and McCann [4] for mathematical and historical details.

While usually known as the *Monge-Kantorovich transportation problem*, the minimum cost of a transportation is sometimes called *Wasserstein metric* or, in computer science, *earth mover's distance* between the two measures μ and ν. It can be used to measure similarity in image retrieval; see Rubner et al. [9]. If both measures μ and ν have finite support, Monge's problem becomes the minimum weight matching problem for complete bipartite graphs, where edge weights represent transportation cost; see Rote [8], Vaidya [11] and Sharathkumar et al. [10].

We are interested in the case where only measure ν has finite support. More precisely, we assume that a set S of n point sites p_i is given, and numbers $\lambda_i > 0$ whose sum equals 1. A body C of volume 1 must be split into subsets C_i of volume λ_i in such a way that the total cost of transporting, for each i, all points of C_i to their site p_i becomes a minimum. In this setting, volume is measured by some (continuous) measure μ, and transport cost $d(z, z')$ by some measure of distance.

Gangbo and McCann [4] report on the cases where either $d(z, z') = h(z - z')$ with a strictly convex function h, or $d(z, z') = l(|z - z'|)$ with a non-negative, strictly concave function l of the Euclidean norm. As a consequence of deep results on the general Monge-Kantorovich problem, they prove a surprising fact. The minimum cost partition of C is given by the additively weighted Voronoi diagram of the sites p_i, based on cost function d and additive weights w_i. In this structure, the Voronoi region of p_i contains all points z satisfying

$$d(p_i, z) - w_i \;<\; d(p_j, z) - w_j \text{ for all } j \neq i.$$

Villani [12] has more recently observed that this fact holds even for more general cost functions, that need not be invariant under translations, and in the case where both distributions are continuous.

Figure 1 depicts how to obtain this structure in dimension $d = 2$. For each point site $p \in S$ we construct in $\mathbb{R}^3$ the cone $\{(z, d(p, z) - w_p) \mid z \in \mathbb{R}^2\}$. The lower envelope of the cones, projected onto the XY–plane, results in the Voronoi diagram. Independently, Aurenhammer et al. [2] have studied the case where $d(z, z') = |z - z'|^2$. Here, a *power diagram* (compare [3]) gives an optimum splitting of C. In addition to structural results, they provide algorithms for computing the weights w_i. Regarding the case where the transportion cost from point z to site p_i is given by individual cost functions $d_{p_i}(z)$, it is still unknown how to compute the weights w_i.

In this paper we consider the situation where the cost $d(p_i, z)$ of transporting point z to site p_i is given by individual distance functions $d_{p_i}(z)$. We require that the weighted Voronoi diagram based on the functions $d_{p_i}(\cdot)$ is well-behaved, in the following sense. Voronoi regions may be disconnected, but together with the Voronoi diagram they form a finite cell decomposition of $\mathbb{R}^d$. Bisectors are

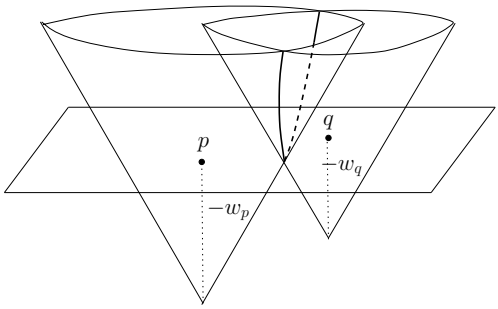


Fig. 1. An additively weighted Voronoi diagram as the lower envelope of cones

$(d-1)$–dimensional, and increasing weight w_i causes the bisectors of p_i to sweep d–space in a continuous way. These requirements are fulfilled for the Euclidean metric and, at least in dimension 2, if each site is assigned a strictly convex distance function.

We show that these assumptions are strong enough to obtain the result of [4,2]: The weighted Voronoi diagram based on the functions $d_{p_i}(\cdot)$ optimally solves the transportation problem, if weights are suitably chosen. Whereas [4] derives this fact from a more general measure-theoretic theorem, our proof uses direct geometric arguments. Furthermore, in comparison to [2], we allow more freedom regarding the choice of the distance functions $d_{p_i}(\cdot)$. The purpose of this paper is not to prove new results, but to give a new, intuitive proof for a general set of transport cost functions, that does not use deep methods of transportation theory but relies on basic geometric arguments.

After stating some definitions in Section 2 we generalize arguments from [2] to prove, in Section 3, that C can be split into parts of arbitrary given volumes by a weighted Voronoi diagram, for a suitable choice of weights. If C is connected, these weights are uniquely determined, up to addition of a constant. Then, in Section 4, a flow augmentation argument shows that such a partition is optimal, and unique.

2 Definitions

Let μ be a measure defined for all Lebesgue-measurable subsets of $\mathbb{R}^d$. We assume that μ vanishes exactly on the sets of Lebesgue measure zero.

Let S denote a set of n point sites in $\mathbb{R}^d$. For each $p \in S$ we are given a continuous function

$$d_p : \mathbb{R}^d \longrightarrow \mathbb{R}_{\geq 0}$$

that assigns, to each point z of $\mathbb{R}^d$, a nonnegative value $d_p(z)$ as the "distance" from site p to z.

For $p \neq q \in S$ and $\gamma \in \mathbb{R}$ we define

$$B_\gamma(p,q) := \{z \in \mathbb{R}^d \mid d_p(z) - d_q(z) = \gamma\}$$

and

$$R_\gamma(p,q) := \{z \in \mathbb{R}^d \mid d_p(z) - d_q(z) < \gamma\}.$$

The sets $R_\gamma(p,q)$ are open and increase with γ. Now let us assume that for each site $p \in S$ an additive weight w_p is given, and let

$$w = (w_1, w_2, \ldots, w_n)$$

denote the vector of all weights, according to some ordering $p_1, p_2, \ldots$ of S. Then, $B_{w_i - w_j}(p_i, p_j)$ is called the *additively weighted bisector* of p_i and p_j, and

$$\mathrm{VR}_w(p_i, S) := \bigcap_{j \neq i} R_{w_i - w_j}(p_i, p_j)$$

is the *additively weighted Voronoi region* of p_i with respect to S. It consists of all points z for which $d_{p_i}(z) - w_i$ is smaller than all values $d_{p_j}(z) - w_j$, by definition. As usual,

$$V_w(S) := \mathbb{R}^d \setminus \bigcup_i \mathrm{VR}_w(p_i, S)$$

is called the *additively weighted Voronoi diagram* of S; compare [3]. Clearly, $\mathrm{VR}_w(p_i, S)$ and $V_w(S)$ do not change if the same constant is added to all weights in w. Therefore, we may assume that $\min\{w_i | 1 \leq i \leq n\} = 0$ holds whenever this is convenient. Increasing a single value w_i will increase the size of p_i's Voronoi region.

Example. Let $d = 2$ and $d_p(z) = |p - z|$ the Euclidean distance. Given weights w_p, w_q, the bisector $B_{w_p - w_q}(p,q)$ is a line if $w_p = w_q$, and a hyperbola otherwise. Figure 2 (i) shows how $B_\gamma(p,q)$ sweeps across the plane as γ grows from $-\infty$ to ∞. It forms the boundary of the set $R_\gamma(p,q)$ which increases with γ. Each bounded set C in the plane will be reached at some point. From then on the volume of $C \cap R_\gamma(p,q)$ is continuously growing until all of C is contained in $R_\gamma(p,q)$. The increase in volume will be strict if each point z of C is an interior point, i. e., if C is open, and C is connected. Given n points p_j with additive weights w_j, raising the value of a single weight w_i will cause all sets $R_{w_i - w_j}(p_i, p_j)$ to grow until C is fully contained in the Voronoi region $V_w(p_i, S)$ of p_i.

In (ii) an additively weighted Voronoi diagram $V_w(S)$ based on the Euclidean distance is shown. It partitions the plane into 5 two-dimensional cells (Voronoi regions), and consists of 9 cells of dimension 1 (Voronoi edges without endpoints) and of 5 cells of dimension 0 (Voronoi vertices). Each cell is homeomorphic to an open sphere of appropriate dimension.

The next definition generalizes the above properties to the setting used in this paper.

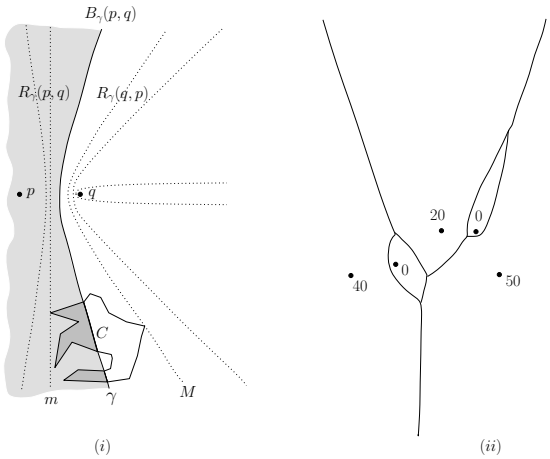


Fig. 2. (i) $R_\gamma(p,q)$ for increasing values of γ. (ii) An additively weighted Voronoi diagram.

Definition 1. *A system of continuous distance functions $d_p(\cdot)$, where $p \in S$, is called* admissible *if the following properties are fulfilled.*

(A) For all $p \neq q \in S$, and for each bounded open set $C \subset \mathbb{R}^d$, there exist values $m_{p,q}$ and $M_{p,q}$ such that $\gamma \longmapsto \mu\left(C \cap R_\gamma(p,q)\right)$ is continuously increasing from 0 to $\mu(C)$ as γ grows from $m_{p,q}$ to $M_{p,q}$, where $C \cap R_\gamma(p,q) = \emptyset$ if $\gamma \leq m_{p,q}$ and $C \subset R_\gamma(p,q)$ if $M_{p,q} \leq \gamma$.

(B) For every nonempty subset $T \subseteq S$ of S the Voronoi regions $VR_w(p_i,T)$, $p_i \in T$, and the Voronoi diagram $V_w(T)$ form a finite *cell decomposition of $\mathbb{R}^d$.*

From (A) we have $m_{q,p} = -M_{p,q}$ and $M_{q,p} = -m_{p,q}$ for all $p \neq q \in S$; compare Figure 2, (i). Property (B) ensures that Voronoi diagrams have reasonable structural properties. While Voronoi regions are not required to be connected, they may be split into only finitely many cells of dimension d each. Their boundaries consist of finitely many bisector pieces of dimension $d-1$, that are again bounded by lower dimensional cells consisting of points equidistant to three or more sites. The closures of all Voronoi regions together cover the whole space $\mathbb{R}^d$.

This definition rules out phenomena known for degenerate placement of sites in, e. g., the L_1 norm, where bisectors of points in d-space may be of dimension d. More information about cell decompositions can be found in Hatcher [5].

3 Partitions of Prescribed Size

Let $d_{p_i}(\cdot)$, $1 \leq i \leq n$, where $n \geq 2$, be an admissible system as in Definition 1, and let C denote a bounded and open subset of $\mathbb{R}^d$. Suppose we are given n real numbers $\lambda_i > 0$ such that $\lambda_1 + \lambda_2 + \ldots + \lambda_n = 1$ holds. The following theorem shows that we can use an addititively weighted Voronoi diagram based on the functions d_p to partition C into subsets of size $\lambda_i \cdot \mu(C)$.

Theorem 1. *There exists a weight vector* $w = (w_1, w_2, \ldots, w_n)$ *such that*

$$\mu(C \cap VR_w(p_i, S)) = \lambda_i \cdot \mu(C)$$

holds for $1 \leq i \leq n$. *Moreover, if* C *is pathwise connected then* w *is unique up to addition of a constant to all* w_i.

Proof. W. l. o. g. let $\mu(C) = 1$. The function

$$\Phi(w) := \sum_{i=1}^{n} |\mu(C \cap \mathrm{VR}_w(p_i, S)) - \lambda_i|$$

measures how close w comes to fulfilling the theorem. Since each function $\gamma \longmapsto \mu(C \cap R_\gamma(p_i, p_j))$ is continuous by Definition 1, (A), and because of

$$|\mu(C \cap \mathrm{VR}_w(p_i, S)) - \mu(C \cap \mathrm{VR}_{w'}(p_i, S))| \leq$$
$$\sum_{j \neq i} |\mu(C \cap R_{w_i - w_j}(p_i, p_j)) - \mu(C \cap R_{w'_i - w'_j}(p_i, p_j))|$$

we conclude that function Φ is continuous, too.

Existence. Let $D := \max\{M_{p,q} \mid p \neq q\}$ with $M_{p,q}$ as in Definition 1, (A). Note that $m_{p,q} = -M_{q,p}$ and $m_{p,q} < M_{p,q}$ together impliy that $D > 0$ is positive. On the compact set $[0, D]^n$ function Φ attains its minimum value at some argument w. If $\Phi(w) = 0$ we are done. Suppose that $\Phi(w) > 0$. Since the volumes of the Voronoi regions inside C add up to 1, there must be some sites p_j whose Voronoi regions have an intersection with C of volume $> \lambda_j$, while the volume of other region's intersections with C is too small (i.e. $< \lambda_j$).

Suppose there exists a set E of sites whose Voronoi regions, weighted with w, have empty intersections with C. Let $p_i \in E$. Thanks to property (A) of Definition 1, we can increase weight w_i until the Voronoi region of p_i has a non-empty intersection with C, of size less than λ_i and small enough so that no other region's intersection with C becomes empty. This procedure can be iterated for the other sites in E. Let w' denote the resulting weight vector. In the above process no point of S has changed from too small a Voronoi region into a region already too large. That is, $0 = \mu(C \cap \mathrm{VR}_w(p_i, S)) < \mu(C \cap \mathrm{VR}_{w'}(p_i, S)) < \lambda_i$ is fulfilled for all $p_i \in E$, and $\mu(C \cap \mathrm{VR}_{w'}(p_i, S)) \leq \mu(C \cap \mathrm{VR}_w(p_i, S))$ holds for all $p_i \in S \setminus E$. Hence, $\Phi(w') \leq \Phi(w)$ holds, by the (piecewise) linearity of Φ. By construction, all Voronoi regions weighted with w' now intersect C in a set of positive measure.

If $\Phi(w') > 0$ we do the following: We choose some value $\delta > 0$ and raise the weights of all sites whose Voronoi regions currently have too small an intersection with C by δ, increasing the size of at least one of those regions. Choosing δ small enough we ensure that the size of any region increased this way will stay too small, and the size of any region previously too large will also remain too large. Moreover, the size of the intersection of any Voronoi region with C will remain positive.

We repeat this process, each time choosing a new value δ, until $\Phi(w')$ decreases. If the increase of weights does not result in a decrease of $\Phi(w')$, the size $\mu(C \cap \mathrm{VR}_{w'}(p_j, S))$ of the region of at least one point p_j decreases from λ_j to some value $< \lambda_j$. Thus, if $\Phi(w')$ has not decreased after $n-2$ iterations, all regions have non-empty intersections with C that are either too small or too large. As we increase once more simultaneously the weights of all regions that are too small, their gain in size is caused only by losses of regions too large. For the resulting weight vector w'' this implies $\Phi(w'') < \Phi(w') \leq \Phi(w)$. (If $0 = \Phi(w')$ we have $\Phi(w') < \Phi(w)$ and continue with w' instead of w''.)

Let w''' result from w'' by subtracting the minimum weight w''_j from all w''_i. Since all Voronoi regions based on w''' have intersections of positive size with C, property (A) of Definition 1 implies that

$$m_{p_i,p_j} < w'''_i - w'''_j < M_{p_i,p_j} \leq D$$

holds for all $i \neq j$. Now because the minimum weight w'''_j is 0, the maximum weight w'''_i must be smaller than D. Together, this implies $w''' \in [0, D]^n$ and $\Phi(w''') = \Phi(w'') < \Phi(w)$—a contradiction!

Uniqueness. Suppose there are two weight vectors w, w' such that $w' - w \neq (c, c, \ldots, c)$ for any constant c, but $\Phi(w) = \Phi(w') = 0$. We may assume that there are entries $w'_j > w_j$; if not, we reverse the rôles of w' and w. Let $T \subset S$ be the set of sites p_i for which the difference $w'_i - w_i$ is maximal. By assumption, T is neither empty nor equal to S.

For the Voronoi diagram $V_w(S)$, remember that for each site $u \in S$ every connected component of the Voronoi region $\mathrm{VR}_w(u, S)$ is an open d - dimensional set, which we call a *cell* of the Voronoi region of u.

Let A_s and A_t be a cell of the Voronoi region of s and t, respectively, where $s \in S \setminus T$ and $t \in T$ are points of S. Furthermore, A_s and A_t are two cells whose intersection with C has positive measure. Let π be a path in C connecting A_s and A_t, and $\pi(r) : [0, 1] \longrightarrow C$ be a continuous parametrization of π. For any site $u \in S$ the function

$$F_u : r \longrightarrow \inf_{q \in \mathrm{VR}_w(u,S)} d(\pi(r), q)$$

is also continuous (where $d(x, y)$ denotes the Euclidean distance in $\mathbb{R}^d$ between points x and y), as well as the two lower envelopes

$$F^*_T(r) = \min\{F_t(r) \mid t \in T\}$$

and

$$F^*_{S \setminus T}(r) = \min\{F_s(r) \mid s \in S \setminus T\}.$$

Since $F_u(r)$ denotes the smallest Euclidean distance from $\pi(r)$ to any point in $\mathrm{VR}_w(u, S)$, clearly $\min\{F^*_T(r), F^*_{S \setminus T}(r)\} = 0$ holds for all $r \in [0, 1]$. If the converse were true for some $r \in [0, 1]$, then we could choose $\varepsilon > 0$ small enough such that every point contained in the open Euclidean ball $B_\varepsilon(\pi(r))$ of radius ε

centered at $\pi(r)$ would not belong to the Voronoi region of any point in S, but to the Voronoi diagram $V_w(S)$. The fact that C is an open set and $\pi(r) \in C$ implies that $B_\varepsilon(\pi(r)) \cap C$ has positive measure. This is a contradiction to Definition 1, (B), which implies that $V_w(S)$ has measure 0.

Let $\Delta(r) = F^*_{S\setminus T}(r) - F^*_T(r)$. $\pi(0) \in A_s$ and $\pi(1) \in A_t$ implies $\Delta(0) < 0$ and $\Delta(1) > 0$, respectively. By the intermediate value theorem there exists $\tau \in (0,1)$ where $\Delta(\tau) = 0$. Let $p = \pi(\tau)$. Note that $F^*_T(\tau) = F^*_{S\setminus T}(\tau)$ holds because at least one of the two values is 0, as mentioned above. We conclude that p cannot belong to the Voronoi region $\mathrm{VR}_w(u,S)$ of any point $u \in S$: since $\mathrm{VR}_w(u,S)$ is an open set, $p \in \mathrm{VR}_w(u,S)$ would imply that p is contained in $B_\epsilon(p) \subset \mathrm{VR}_w(u,S)$, if $\varepsilon > 0$ is suitably chosen. But this contradicts the fact $F^*_T(\tau) = F^*_{S\setminus T}(\tau) = 0$. From the same argument it follows that p must lie on the boundary of the Voronoi region of any site $u \in S$ where $F_u(\tau) = 0$, and that p lies on the weighted bisector $B_{w_u - w_v}(u,v)$ of any two sites $u, v \in S$ if $F_u(r) = F_v(r) = 0$. Let $s' \in S \setminus T$ and $t' \in T$ be two such sites. For convenience we assume w. l. o. g. that $s' = s$ and $t' = t$, and that p lies on the boundary of A_s and of A_t.

Let M be the set of sites $u \in S$ where $d_u(p) - w_u$ is minimal. Because M contains t, the set $T' := T \cap M$ is non-empty. By definition of point set T'

$$d_v(p) - w'_v < d_u(p) - w'_u$$

holds for any two sites $v \in T'$ and $u \in S \setminus T'$. Therefore, p is contained in the open set

$$\bigcap_{v \in T', u \in S\setminus T'} R_{w'_v - w'_u}(v,u)$$

and there exists some value $\varepsilon > 0$ such that $B_\varepsilon(p)$ is disjoint from $\mathrm{VR}_{w'}(u,S)$, $\forall u \in S \setminus T'$, and where $B_\varepsilon(p) \subseteq C$ holds. Since p lies on the boundary of A_s, $B_\varepsilon(p)$ contains an interior point $q \in A_s$. Now we choose $\delta > 0$ small enough so that $B_\delta(q)$ is a subset of $A_s \cap B_\varepsilon(p)$. The definition of point set T implies that $\mathrm{VR}_w(t,S) \subseteq \mathrm{VR}_{w'}(t,S)$ holds for any point $t \in T$. By construction, every point of $B_\delta(q) \subseteq A_s \cap C$ belongs in $\mathrm{V}_{w'}(S)$ to the Voronoi region of some point $t' \in T' \subseteq T$ (or to the Voronoi diagram $V_{w'}(S)$, whose measure is 0). We conclude that for some point $t' \in T'$, it holds that

$$\mu(C \cap \mathrm{VR}_w(t',S)) < \mu(C \cap \mathrm{VR}_{w'}(t',S))$$

—a contradiction to $\Phi(w) = 0 = \Phi(w')$.

It is easy to see why connectedness of C is a condition necessary for the uniqueness of w. Namely, assume that each connected component of C is completely contained in a separate Voronoi region. Now if the weights of the points in S are modified by some small amount then the resulting Voronoi diagram could still be the same inside C.

4 Optimality

Again, let $d_p(\cdot)$, $p \in S = \{p_1, p_2, \ldots, p_n\}$, be an admissible system as in Definition 1, and let C denote a bounded and open subset of $\mathbb{R}^d$. Moreover, let real numbers $\lambda_i > 0$ be given such that $\lambda_1 + \lambda_2 + \ldots + \lambda_n = 1$ holds.

By the existence part of Theorem 1, there exists a weight vector $w = (w_1, w_2, \ldots, w_n)$ satisfying

$$\mu(C \cap \mathrm{VR}_w(p_i, S)) = \lambda_i \cdot \mu(C) \text{ for } i = 1, \ldots, n.$$

Now we prove that this subdivision of C minimizes the transportation cost, i. e., the average distance from each point to its site. It is convenient to describe partitions of C by maps $f : C \longrightarrow S$ where, for each $p \in S$, $f^{-1}(p)$ denotes the region assigned to p. Let F_Λ denote the set of those maps f satisfying $\mu(f^{-1}(p_i)) = \lambda_i \cdot \mu(C)$ for $i = 1, \ldots, n$.

Theorem 2. *The partition of C into regions $C_i := C \cap VR_w(p_i, S)$ minimizes*

$$\mathit{cost}(f) := \sum_{p \in S} \int_{f^{-1}(p)} d_p(z)\, \mathrm{d}\mu(z)$$

over all maps $f \in F_\Lambda$. Any other partition of minimal cost differs at most by sets of measure zero from $(C_i)_i$.

Proof. Let f_w be defined by $f_w(C_i) = \{p_i\}$ for all i. Suppose there is a map $f \in F_\Lambda$ satisfying $\mathrm{cost}(f) < \mathrm{cost}(f_w)$. Let us consider the complete directed graph K whose vertices are the sites of S. The weight of edge (p_i, p_j) from p_i to p_j equals

$$\delta_{i,j} := \mu(C_i \cap f^{-1}(p_j)),$$

the volume of that part of C_i which is assigned to site p_j by f; we include the case $i = j$. Since $f \in F_\Lambda$, the weights of all incoming edges at p_i add up to $\lambda_i \cdot \mu(C)$. This equals $\mu(C_i)$, the sum of the weights of all outgoing edges.

Since f and f_w differ in cost, some edge (p_i, p_j) between different sites must carry a positive weight. Consequently, there must be an outgoing edge connecting p_j to some site $p_k \neq p_j$ whose weight is positive, and so on. Since graph K is finite we obtain, after renumbering vertices, a directed cycle $(p_1, p_2, \ldots, p_m, p_1)$ all of whose edge weights $\delta_{1,2}, \delta_{2,3}, \ldots, \delta_{m,1}$ are positive.

Let $\delta > 0$ denote the minimum of these weights. We modify map f such that it reassigns a subset $D_i \subseteq C_i$ of volume δ to p_i instead of p_{i+1}, for each p_i in the cycle. That is, $\delta_{i,i}$ becomes $\delta_{i,i} + \delta$, and $\delta_{i,i+1}$ turns into $\delta_{i,i+1} - \delta$. This reduces the weight of (at least) one edge of the cycle to zero.

Let f_1 denote the modified map. By construction, $f_1 \in F_\Lambda$. With $f_0 := f$ we obtain

$$\mathrm{cost}(f_1) - \mathrm{cost}(f_0) = \sum_{i=1}^{m} \int_{D_i} (d_{p_i}(z) - d_{p_{i+1}}(z))\, \mathrm{d}\mu(z) \qquad (1)$$

$$< \sum_{i=1}^{m} (w_i - w_{i+1}) \cdot \int_{D_i} 1 \, \mathrm{d}\mu(z) \tag{2}$$

$$= \sum_{i=1}^{m} (w_i - w_{i+1}) \cdot \delta \; = \; 0. \tag{3}$$

Line (2) follows from the fact that $z \in D_i \subseteq C_i \subseteq \mathrm{VR}_w(p_i, S)$ implies $d_{p_i}(z) - w_i < d_{p_{i+1}}(z) - w_{i+1}$. Line (3) holds because $w_{m+1} = w_1$ causes the sum to telescope to zero.

After $k \leq \binom{n}{2}$ repetitions of the above process, all edge weights $\delta_{i,j}$, where $i \neq j$, of graph K are equal to 0. That is, we obtain a sequence of maps $f_0, f_1, \ldots, f_k$ in F_Λ of strictly decreasing cost, where f_k maps, for each i, all of C_i to p_i, except possibly a set of measure 0. This yields the contradiction $\mathrm{cost}(f_w) = \mathrm{cost}(f_k) < \mathrm{cost}(f_1) < \mathrm{cost}(f_0) < \mathrm{cost}(f_w)$.

If we had an alternate optimal partition f that differs from f_w by sets of positive measure, we could apply the same argument to construct a map of even smaller cost, which has just shown to be impossible.

5 Conclusion and Future Work

We have given a direct geometric proof for the fact that additively weighted Voronoi diagrams can optimally solve some cases of the Monge-Kantorovich transportation problem, where one measure has finite support. Surprisingly, the existence of an optimal solution - the main mathematical challenge in the general case - is an easy consequence of our proof. In remains to be seen to which extent our assumptions on the distance functions d_{p_i} can be further generalized.

References

1. Appell, P.: Mémoire sur les déblais et les remblais des systèmes continues ou discontinus. Mémoires présentes par divers Savants à l'Academie des Sciences de l'Institut de France 29, 1–208 (1887)
2. Aurenhammer, F., Hoffmann, F., Aronov, B.: Minkowski-type theorems and least-squares clustering. Algorithmica 20, 61–76 (1998)
3. Aurenhammer, F., Klein, R.: Voronoi diagrams. In: Sack, J.R., Urrutia, G. (eds.) Handbook on Computational Geometry, pp. 201–290. Elsevier (1999)
4. Gangbo, W., McCann, R.J.: The geometry of optimal transportation. Acta Math. 177, 113–161 (1996)
5. Hatcher, A.: Algebraic Topology. Cambridge University Press (2001)
6. Kantorovich, L.: On a problem of Monge. Uspekhi Math. Nauk. 3, 225–226 (1948) (in Russian)
7. Monge, G.: Mémoire sur la théorie des déblais et de remblais. Histoire de l'Académie Royale des Sciences de Paris, avec les Mémoires de Mathématique et de Physique pour la même année 29, 666–704 (1781)
8. Rote, G.: Two applications of point matching. In: Abstracts of the 25th European Workshop on Computational Geometry (EuroCG 2009), pp. 187–189.

9. Rubner, Y., Tomasi, C., Guibas, L.J.: A metric for distributions with applications to image databases. In: Proceedings International Conference on Computer Vision (ICCV 1998), pp. 59–66 (1998)
10. Sharathkumar, R., Agarwal, P.K.: Algorithms for the transportation problem in geometric settings. In: Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012), pp. 306–317 (2012)
11. Vaidya, P.M.: Geometry helps in matching. SIAM J. Comput. 18, 1201–1225 (1989)
12. Villani, C.: Optimal Transport, Old and New. Grundlehren der mathematischen Wissenschaften, vol. 338. Springer (2009)

Unexplored Steiner Ratios in Geometric Networks*

Paz Carmi and Lilach Chaitman-Yerushalmi

Department of Computer Science,
Ben-Gurion University of the Negev, Israel

Abstract. In this paper we extend the context of Steiner ratio and examine the influence of Steiner points on the weight of a graph in two generalizations of the Euclidean minimum weight connected graph (MST). The studied generalizations are with respect to the weight function and the connectivity condition.

First, we consider the Steiner ratio of Euclidean minimum weight connected graph under the *budget allocation* model. The budget allocation model is a geometric version of a new model for weighted graphs introduced by Ben-Moshe et al. in [4].

It is known that adding auxiliary points, called Steiner points, to the initial point set may result in a lighter Euclidean minimum spanning tree. We show that this behavior changes under the budget allocation model. Apparently, Steiner points are not helpful in weight reduction of the geometric minimum spanning trees under the budget allocation model (BMST), as opposed to the traditional model.

An interesting relation between the BMST and the Euclidean square root metric reveals a somewhat surprising result: Steiner points are also redundant in weight reduction of the minimum spanning tree in the Euclidean square root metric.

Finally, we consider the Steiner ratio of geometric t-spanners. We show that the influence of Steiner points on reducing the weight of Euclidean spanner networks goes much further than what is known for trees.

1 Introduction

In our context, Steiner points are auxiliary points added to the original point set. Fermat was the first to consider the influence of Steiner points on the weight of Euclidean networks. Back in the 17th century he proposed the problem of finding a point that minimizes its distances from three given points in the plane [10]. Today, Steiner points are known mostly within the context of the *Steiner tree problem* [9,10]. This problem is a generalization of the problem proposed by Fermat. The three input points are replaced with any finite set of points in the

* Research is partially supported by Lynn and William Frankel Center for Computer Science and by grant 2240-2100.6/2009 from the German Israeli Foundation for scientific research and development (GIF) and by grant 680/11 from the Israel Science Foundation (ISF).

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 275–286, 2012.

plane and the objective is finding the[1] minimum weight network connecting all points in the given point set, where the weight is the sum of the weights of all edges. However, as opposed to the *minimum spanning tree (MST)* problem, in the Steiner tree problem Steiner points may be added to the input set. The optimum solution for the problem is referred to as the *Steiner minimum tree (SMT)*.

Steiner points are known to be effective in reducing the weight of the *Euclidean MST* for an input point set. Meaning, the *Euclidean SMT* may be lighter than the *Euclidean MST*. Throughout this paper, unless otherwise is mentioned, the distances between points in the input point set are defined by the Euclidean metric. Given a point set P, we denote by $MST(P)$ the minimum spanning tree of P and by $SMT(P)$ the Steiner minimum tree of P. The minimum ratio between the weight of the *Euclidean SMT* and the *Euclidean MST* among all possible input sets of points in the plane is called *Steiner ratio* and denoted by ρ. Formally, $\rho = \inf_P \{SMT(P)/MST(P)\}$. Gilbert and Pollak conjectured in [9] that $\rho = \sqrt{3}/2 \approx 0.866$, however they have only proved $\rho \leq \sqrt{3}/2$. A lower bound of $r_0 \approx 0.824$ was shown by Chung and Graham in [5].

In this paper we extend the context of Steiner ratio and examine the influence of Steiner points on the weight of geometric networks other than minimum spanning trees. We observe two geometric networks which are, in a sense, generalizations of Euclidean spanning trees. The first network is the *budget spanning tree*, which generalizes the traditional spanning trees with respect to the weight function. The second generalization is with respect to the connectivity condition. We strengthen the connectivity requirement to demand that every two points in the input set are connected with a path of length that approximates their Euclidean distance within a factor t. The resulting spanning graph is known in the literature as a t-spanner for P.

In order to present the *budget spanning tree* we should first introduce the *budget allocation* model. So far, researchers have considered the binary version of a network (graph), i.e., a link has a binary value that indicates whether the link exists in the network. In real life, when a budget is given to establish a network via links, the attributes of the connection varies with respect to its priority, and therefore, not all links are created equally. For example, consider a railway system that connects different cities. Naturally, we would like to have significantly faster trains on railway tracks between two large cities than on rails between two villages. Thus, considering all links in the same manner may miss the true nature of the network.

In this paper we address a geometric version of the model suggested by Ben-Moshe et al. in [4] for weighted graphs, to which we refer as the *budget allocation* model. The authors were motivated by communication networks modeled by communication graphs. They suggested a 'quality of service' model. The quality of service (QoS) of a link refers to different parameters such as delay time,

[1] Throughout the paper we may refer to **the** minimum geometric network (e.g., **the** MST) even though it is not unique; however, this does not affect the correctness of the proofs.

jitter, and packet error and depends on two main factors: the length of the link and its infrastructure. Hence, each edge in the graph is associated with a value between zero and one derived from its infrastructure to which we refer as a *budget*. The weight of an edge in this model, called *budgeted weight*, depends not only on its length, but also on the budget assigned to it. This *budgeted weight* represents the quality of service. More precisely, there is an inverse ratio between the quality of service and the *budgeted weight*, the higher the budget invested in a communication link, the lower its delay time. For the sake of simplicity, whenever the budget allocation model is addressed, we use the term *weight* when referring to the *budgeted weight*.

In this paper, we slightly modify and extend the model introduced in [4] and address different problems. The authors in [4] considered general weighted graphs, however, their motivation was communication networks which have a geometric nature. We address a geometric version of the model. We define a budget allocation for pairs in a given set of points in the plane or from another perspective, for the edges of the complete Euclidean graph. The geometric version of the budget allocation model may also be considered in the context of a railway system, mentioned earlier. The budget of a railway line between two stations indicates the resources grade. The budget weight of a railway line represents the quality of the ride which is derived from both the resources attributes and the length of the ride. The optimization problem that we address is allocating a fixed budget onto pairs of points from the input set where our objective is to create a minimum weight spanning tree and also examine the addition of Steiner points.

We refer to a minimum weight spanning tree induced by an optimal budget allocation as a *budget minimum spanning tree (BMST)*. We consider the influence of Steiner points on the weight of a $BMST$ measured by the parameter known as Steiner ratio. The Steiner ratio for budgeted trees ρ_b is defined as the smallest ratio between the weight of a BMST with and without permitting the use of Steiner points. Apparently, Steiner points lose their power in the budget allocation model, that is, their addition cannot reduce the weight of the BMST, and therefore the Steiner ratio ρ_b equals one.

An interesting relation between the BMST and the Euclidean square root metric (i.e., the metric that defines the distance between any two points p and q to be $\sqrt{|pq|}$) is revealed when considering the BMST problem. This relation implies a somewhat surprising conclusion: Steiner points are also redundant in the context of minimum spanning trees in the Euclidean square root metric.

The second geometric network that we consider in the context of its Steiner ratio is the Euclidean t-spanner. An Euclidean t-spanner for a set P of n points in the plane is a graph that spans P and whose edge set satisfies the following property: for every two points $p, q \in P$, there exists a path connecting them of a length that approximates the Euclidean distance between them with a factor t. In other words, the shortest path connecting them is of a length of at most $t|pq|$ (see [11] for a more extensive survey on the subject).

The contribution of Steiner points to 2-dimensional Euclidean spanners has been a subject of research within the context of planarity and the size of the spanner, i.e., the number of edges. Arikati et al. [3] showed that by allowing linear number of Steiner points one can obtain a plane $(\sqrt{2}+\epsilon)$-spanner (the stretch factor is defined only in terms of point-pairs of the input point set), where the construction of Arikati et al. requires $O(n/\epsilon^4)$ Steiner points. Abam et al. have proved in [1] that while any set P of n points in the plane admits a C-fault tolerant $(1+\epsilon)$-spanner of size $O(n \log n)$, if adding Steiner points is allowed, the size of the spanner reduces to $O(n)$. The influence of Steiner points on the weight of 1-dimensional Euclidean spanners has been considered by Elkin and Solomon in [8]. They have shown that Steiner points do not help in asymptotic weight reduction of 1-dimensional Euclidean t-spanners with hop-diameter $o(\log n)$.

We examine the influence of Steiner points on the weight of 2-dimensional Euclidean t-spanners. We define for every constant t the Steiner ratio ρ_t to be the smallest ratio between the weight of a minimum t-spanner with and without permitting the use of Steiner points. It turns out that the influence of Steiner points on reducing the weight of geometric spanner networks goes much further than what is known for trees. We show that for every constant t, ρ_t is smaller than the conjectured value of ρ. Moreover, we show that there is no lower bound on $\inf_t\{\rho_t\}$. Namely, for every $\epsilon > 0$ there exists $t > 1$ that satisfies $\rho_t < \epsilon$.

In the following sections we extend the context of Steiner ratio for two generalizations of the Euclidean MST.

2 Steiner Ratio for Budgeted Trees

In this section we extend the notion of Steiner ratio for a generalization of Euclidean trees, called budgeted trees. This generalization is with respect to the weight function.

In subsection 2.1 the *budget allocation* model is presented, and *budget minimum spanning trees* are introduced in subsection 2.2. Apparently, a minimum spanning tree of a given set of points in the plane is also a budget minimum spanning tree (only with different weight function over the edges) as proved in subsection 2.2. However, when allowing the addition of Steiner points to the spanning tree, the budget minimum spanning tree behaves differently than the traditional minimum spanning tree. In subsection 2.3, we show that the Steiner ratio for budgeted trees increases much above the Steiner ratio for trees. Actually, it reaches the maximum possible ratio 1.

2.1 Budget Allocation

Given a finite set of points in the plane P and a budget B, let $k(P) = (P, E_P)$ denote the complete Euclidean graph over P. A valid *budget allocation* for P is a function $B : E_P \to [0, 1]$, such that $\sum_{\{p,q\}\in E_P} B(p, q) = B$.

Let $|pq|$ denote the Euclidean distance between two points $p, q \in P$. Allocating a positive value to a pair of points $\{p, q\}$ implies that the resulting

budgeted weight of $\{p,q\}$ is $w_B(p,q)=\frac{|pq|}{B(p,q)}$, while allocating of a zero value implies $w_B(p,q)=0$. The *weighted distance* between a pair of points $p,q\in P$ is defined as $\delta_B(p,q)=\min\{\sum_{e\in R} w_B(e) : R \text{ is a simple path from } p \text{ to } q\}$. The graph *induced* by $B(\cdot)$ is defined as $G=(P,E)$, where $E=\{\{p,q\}\in E_P : w_B(p,q)>0\}$. Throughout the paper, when addressing the weight function of an induced graph we refer to the budgeted weight function $w_B(\cdot)$. We denote by $w_B(G)=\sum_{e\in E} w_B(e)$ the budgeted weight of the induced graph.

Given a graph $G=(P,E)$ and a budget B, a valid *budget allocation* for G is a positive real function $B : E \to (0,1]$, such that $\sum_{\{p,q\}\in E} B(p,q)=B$. This definition differs from the one in [4], which permits an assignment of a zero value to an edge. This modification simplifies the formalization in this paper, since each edge in the induced graph receives a positive budget allocation.

2.2 Budget Minimum Spanning Trees

In this subsection, we consider the *budget minimum spanning tree* problem (or the *BMST* problem for short). The objective is finding a budget allocation $B(\cdot)$ that induces the minimum weight spanning graph G of a given point set P, which is referred to as the *budget minimum spanning tree (BMST)* of P. Note that the *MST* problem can be considered as the *BMST* problem restricted to binary budget allocation and a budget of size $|P|-1$.

In this section we require $B=1$. Note that this demand is not restrictive, since any allocation of a budget of size 1 can be scaled to any budget B. Given a point set P, let $BM(P)(\cdot)$ denote the optimal solution for the *BMST* problem, i.e., for any budget allocation $B'(\cdot)$, $\sum_{\{p,q\}\in E_P} w_{BM(P)}(p,q)\le\sum_{\{p,q\}\in E_P} w_{B'}(p,q)$, and let $WB(P)$ denote the value of the optimal solution, meaning, $WB(P)=w_{BM(P)}(P)$. Given a graph $G=(V,E)$, let $BM(G)(\cdot)$ denote a budget allocation that minimizes its weight and let $WB(G)=w_{BM(G)}(G)$.

Observation 1. *Let P be a finite set of points in the plane. The BMST of P is a MST of P (only with different weight function over the edges).*

According to Observation 1, $BM(MST(P))(\cdot)$ can be extended to $BM(P)(\cdot)$ by assigning $BM(P)(p,q)=0$ for every $(p,q)\in E_p$, which is not an edge in $MST(P)$. In the following lemmas we determine $BM(G)(\cdot)$ and $WB(G)$ for general graphs. Lemma 1 is similar in principle to Lemma 2 in [4].

Lemma 1. *Given a graph $G=(P,E)$ and $e_1\in E$, let $b_1=BM(G)(e_1)$, $G'=(P,E\backslash\{e_1\})$ and $W'=WB(G')$, then $b_1=\frac{\sqrt{|e_1|}}{\sqrt{W'}+\sqrt{|e_1|}}$ and $WB(G)=\frac{|e_1|}{b_1}+\frac{W'}{1-b_1}$.*

Lemma 2. *Given a graph $G=(P,E)$, $WB(G)=(\sum_{e\in E}\sqrt{|e|})^2$.*

Proof. We prove the above by induction on $|E|$.
Base case: Let $G=(P,\{e\})$, then $BM(G)(e)=1$ and $WB(G)=|e|=(\sqrt{|e|})^2$.
Induction hypothesis: The claim holds for every $G=(P,E)$ with $|E|<n$.
Inductive step: Let $G=(P,E)$ be a graph with $|E|=n$. Let e_1 be an edge

in E and let $G' = (P, E \backslash \{e_1\})$. By the induction hypothesis, $W' = WB(G') = (\sum_{e \in E \backslash \{e_1\}} \sqrt{|e|})^2$. According to Lemma 1,

$$\begin{aligned} WB(G) &= |e_1| / \frac{\sqrt{|e_1|}}{\sqrt{W'} + \sqrt{|e_1|}} + W' / (1 - \frac{\sqrt{|e_1|}}{\sqrt{W'} + \sqrt{|e_1|}}) \\ &= (\sqrt{W'} + \sqrt{|e_1|})\sqrt{|e_1|} + (\sqrt{W'} + \sqrt{|e_1|})\sqrt{W'} = (\sqrt{W'} + \sqrt{|e_1|})^2 \\ &= (\sqrt{(\sum_{e \in E \backslash \{e_1\}} \sqrt{|e|})^2} + \sqrt{|e_1|})^2 = (\sum_{e \in E} \sqrt{|e|})^2. \end{aligned}$$

The above observation and lemmas imply the following theorem.

Theorem 1. *Let $P \subset \mathbb{R}^2$ be a finite set of points, then $WB(P) = (\sum_{e \in MST(P)} \sqrt{|e|})^2$.*

2.3 The Steiner Ratio of Budgeted Trees

We examine the benefit of Steiner points to weight reduction of budget minimum spanning trees. Explicitly, for a given set of points in the plane P, let $WS(P) = \min\{w_B(G) : G = (P \cup S, E)$ is an induced graph of a budget allocation $B(\cdot)$ for $P \cup S$ for a finite set of points $S \subseteq \mathbb{R}^2\}$. We are interested in the Steiner ratio in the terms of the new model, which is denoted and defined as $\rho_b = \inf_P\{WS(P)/WB(P)\}$. The following theorem states that in the budget allocation model not only the Steiner ratio increases above the upper bound on ρ, but it reaches 1, meaning, Steiner points have no contribution to weight reduction of budget minimum spanning trees.

Theorem 2. *The Steiner ratio of budgeted trees equals* 1, *i.e.,* $\rho_b = 1$.

Proof. Let P be a set of points in the plane, we show that for every $S \subset \mathbb{R}^2$, $WB(P) \leq WB(P \cup S)$ and conclude $\rho_b = 1$. Assume towards contradiction that a set of Steiner points S exists, such that $WB(P) > WB(P \cup S)$ (otherwise, we are done). Let $O_P = \{S \subset \mathbb{R}^2 : WB(P) > WB(P \cup S)\}$ and let $S \in O_P$ be the set of minimum cardinality in O_P that satisfies: $\forall S' \in O_P$, $WB(P \cup S) \leq WB(P \cup S')$. We denote by $\mathcal{B}$ be the set of all the optimal solutions to the *BMST* problem for the point set $P \cup S$ and by $d_B(s)$ the degree of the point s in the induced tree of a budget allocation $B(\cdot)$. Let $B^*(\cdot)$ be a budget allocation that satisfies $\forall B \in \mathcal{B}$, $\min_{s \in S} d_{B^*}(s) \leq \min_{s \in S} d_B(s)$. Let $T = (P \cup S, E)$ be the tree induced by $B^*(\cdot)$ and let $s \in S$ be a point with minimum degree in T among points in S. We build a tree $T' = (P \cup S \backslash \{s\}, E')$, such that

$$(\sum_{e' \in E'} \sqrt{|e'|})^2 \leq (\sum_{e \in E} \sqrt{|e|})^2. \tag{1}$$

By Theorem 1 we conclude $WB(P \cup S \backslash \{s\}) \leq WB(P \cup S)$ in contradiction to the minimality of $|S|$. In the rest of the proof we omit B^* from the notation $d_{B^*}(s)$.

Observation 2. *The degree of s in T satisfies $2 \leq d(s) \leq 5$.*

Let $q_1, q_2, ..., q_{d(s)}$ be the neighbors of s labeled in clockwise order and let $\alpha_i = \angle(q_{i-1}sq_i)$, for $1 \leq i \leq d(s)$, where $\alpha_1 = \angle(q_{d(s)}sq_1)$. We denote $e_i = \{s, q_i\}$, $E_s = \{e_i : 1 \leq i \leq d(s)\}$ and $e'_{i,j} = (q_i, q_j)$ (see Fig. 1). We assume w.l.o.g. that e_1 is the shortest edge in E_s and has a length 1 (due to scaling).

Throughout the proof we denote $h(x, y, \alpha) = \sqrt[4]{x^2 + y^2 - 2xy\cos(\alpha)} - \sqrt{x}$ and use the following observation regarding the behavior of this function.

Observation 3. *The function $h(x, y, \alpha)$ is maximized in the domain $(0 < x_1 \leq x \leq x_2) \wedge (0 < y_1 \leq y \leq y_2) \wedge (\frac{\pi}{3} < \alpha_1 \leq \alpha \leq \alpha_2 < \pi)$ when $x = x_1$, $y = y_2$ and $\alpha = \alpha_2$.*

Next we define a tree $T' = (P \cup S \setminus \{s\}, E')$ that satisfies inequality 1. We consider each possible degree of s:

- **Degree 2:** We define $E' = E \cup \{(e'_{1,2})\} \setminus E_s$. Thus, $\sqrt{|e'_{1,2}|} \leq \sqrt{|e_1| + |e_2|} \leq \sqrt{|e_1|} + \sqrt{|e_2|}$ and inequality (1) follows.
- **Degree 3:** We define $E' = E \cup \{e'_{1,2}, e'_{1,3}\} \setminus E_s$ (see Fig. 1). For inequality (1) to hold we need to show that $\sqrt{|e'_{1,2}|} + \sqrt{|e'_{1,3}|} \leq \sqrt{|e_1|} + \sqrt{|e_2|} + \sqrt{|e_3|}$. Rearranging the equation gives

$$\sqrt{|e'_{1,2}|} - \sqrt{|e_2|} + \sqrt{|e'_{1,3}|} - \sqrt{|e_3|} \leq \sqrt{|e_1|} = 1. \tag{2}$$

  By the law of cosines $|e'_{1,2}| = \sqrt{|e_2|^2 + 1 - 2|e_2|\cos(\alpha_2)}$ and $|e'_{1,3}| = \sqrt{|e_3|^2 + 1 - 2|e_3|\cos(\alpha_1)}$. Thus, inequality (2) is equivalent to the following: $h(|e_2|, 1, \alpha_2) + h(|e_3|, 1, \alpha_1) \leq 1$. Recall that $\pi/3 < \alpha_i \leq \pi$ and $|e_i| \geq 1$ (for $1 \leq i \leq 3$). According to Observation 3, $h(e_i, 1, \alpha_i)$ is maximized and receives the value $\sqrt{2} - 1$ in the above domain when $e_i = 1$ and $\alpha = \pi$. Hence, $h(|e_2|, 1, \alpha_2) + h(|e_3|, 1, \alpha_1) \leq 2(\sqrt{2} - 1) < 1$.

The cases of degree 4 and degree 5 were omitted due to space limitation.

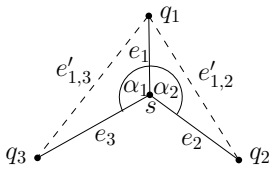


Fig. 1. The tree T' depicted in dashed lines as defined for degree 3 of s

Consider the Euclidean square root metric, i.e. the metric that defines the distance between every $p, q \in \mathbb{R}^2$ to be $\sqrt{|pq|}$. Given a set P of points in the plane, observe that the MST of P in the Euclidean square root metric is the MST of P in the Euclidean metric. The proof of Theorem 2 together with the aforementioned observation implies the following theorem.

Theorem 3. *The Steiner ratio of trees in the Euclidean square root metric equals* 1.

3 Steiner Ratio for t-Spanners

Given a finite set of points in the plane P, an *Euclidean t-spanner* for P is a graph G over P that satisfies the following property. Every two points $p, q \in P$ are connected by a path of length that approximates the Euclidean distance between them within a factor t. That is, the shortest path connecting them is of length at most $t|pq|$ (see [11] for more extensive survey on the subject). Minimum t-spanners are, in a sense, minimum weight connected graphs (MST) with strengthened connectivity requirement.

Although the contribution of Steiner points to weight reduction of trees have been well studied, no similar results have been shown for t-spanners. We examine the ratios between the weights of minimum t-spanners with and without Steiner points and reveal that they go much further than the Steiner ratio for trees.

Analogously to trees, we define a *Steiner t-spanner* as a t-spanner whose point set may contain additional points that are not members of the initial point set and a *minimum Steiner t-spanner* as a Steiner t-spanner with minimum weight. Steiner t-spanners may be considered in two main settings. In the first setting, the Steiner points are referred as members of the original point set and the stretch factor is determined by distances between any pair of points including Steiner points. In the second setting, the stretch factor is determined only with respect to the points in the input point set. Obviously, any graph that admits a Steiner t-spanner for a point set P according to the first setting also admits a Steiner t-spanner for P according to the second setting. We consider the first setting and therefore our results apply for both settings.

Throughout this section we use the following notations. Given a graph G, we denote by $\delta_G(p, q)$ the length of the shortest path between p and q in G. Given set of points $P \subset \mathbb{R}^2$, let $G_m(P, t)$ denote the minimum weight t-spanner for P, let $G_s(P, t)$ denote the minimum weight Steiner t-spanner for P, and let $W_m(p, t)$ and $W_s(p, t)$ denote their weights, respectively. The Steiner ratio for t-spanners is defined as $\rho_t = \inf_P\{W_s(P, t)/W_m(P, t)\}$. In the next subsections we show lower and upper bounds on the ratio ρ_t.

3.1 Lower Bound

Das et al. have introduced in [6] the *leapfrog property* and the *leapfrog theorem* for a set of points P in the 3-dimensional Euclidean space and used them to prove that the weight of the *greedy t-spanner* [2] for a point set P is $O(1)\cdot w(MST(P))$. In [7], those results were generalized to the k-dimensional space. More detailed modified proof is given in [11]. We use those results to bound ρ_t from below.

Theorem 4 (Theorem 15.1.11, [11]). *Let S be a set of n points in $\mathbb{R}^d$, let $t > 1$ be a real number, and let $G = (S, E)$ be the greedy t-spanner. The weight of G is $c_t \cdot w(MST(S))$, where c_t is a function of t.*

The following lemma introduces a lower bound on ρ_t.

Lemma 3. *For every constant t, $\rho_t \geq \frac{\rho}{c_t}$.*

Proof. Given a set of points $P \subset \mathbb{R}^2$ and a constant $t > 1$, let $G_s(P,t) = (V_s, E_s)$, then $W_s(P,t) \geq^{(a)} w(MST(V_s)) \geq^{(b)} \rho \cdot w(MST(P)) \geq^{(c)} W_m(P,t) \cdot \rho / c_t$, where inequality (a) is implied by the fact that the MST is the lightest connected graph over P, (b) is derived from the definition of ρ, and (c) follows from Theorem 4.

As previously mentioned, $\rho \geq r_0 \approx 0.824$ and thus we conclude the following.

Corollary 1. *For every constant t, $\rho_t \geq \frac{r_0}{c_t}$.*

3.2 Upper Bound

In this subsection we show upper bounds on the value of ρ_t. We prove that for small constant values of t, namely when $t \to 1$, ρ_t is remarkably smaller than the conjectured value of ρ, $\sqrt{3}/2$. Actually, for every $\epsilon > 0$, there exists a value t (which depends on ϵ) for which $\rho_t < \epsilon$. Moreover, we show that for every constant $t > 1$, $\rho_t < \sqrt{3}/2$.

In the following lemmas, we prove an upper bound b on ρ_t by suggesting a set of points P and a Steiner t-spanner $ST = (P \cup S, E_s)$, such that $\frac{w(ST)}{G_m(P,t)} \leq b$, and concluding $\rho_t \leq \frac{w(ST)}{G_m(P,t)} \leq b$.

Lemma 4. *For every $\epsilon > 0$ there exists a constant value t (which depends on ϵ), such that $\rho_t \leq \epsilon$.*

Proof. Consider a set of points $P = \{p_0, p_1, ..., p_n\}$, where p_0 is the center of a disk D with radius $r = n^3$ and $p_1, ..., p_n$ lie on a narrow fraction of D's boundary, such that $|p_i p_{i+1}| = 1$, for $1 \leq i < n$. Given $\epsilon > 0$, we show that $t = (r + 1 - \epsilon_1)/r$ for $0 < \epsilon_1 < 1$ (to be defined later), satisfies the inequality $\rho_t \leq \epsilon$.

Note that for sufficiently large r, $\frac{\sum_{k=i}^{j-1} |p_k p_{k+1}|}{|p_i p_j|} \leq t$ for every $0 < i, j \leq n$, however $\frac{|p_0 p_{i+1}| + |p_{i+1} p_i|}{|p_0 p_i|} = \frac{r+1}{r} > t$ and therefore $G_m(P,t) = (P, E)$, where $E = \{\{p_0, p_i\} : 1 \leq i \leq n\} \cup \{\{p_i, p_{i+1}\} : 1 \leq i < n\}$ and $W_m(P,t) = r \cdot n + n - 1$. We suggest the following Steiner t-spanner $ST = (P \cup \{s\}, E_s)$. Let r_0 be a point lying on the boundary of D with equal distances from $p_{n/2}$ and $p_{n/2+1}$. We locate s on the segment $\{p_0, r_0\}$, such that $|p_0 s| + |s p_1| = (r + 1) - \epsilon_1$ and define $E_s = \{\{s, p_i\} : 1 \leq i \leq n\}$. For every $1 < i \leq n$, $|s p_i| \leq |s p_1|$, which implies $\frac{|p_0 s| + |s p_i|}{|p_0 p_i|} \leq \frac{r + 1 - \epsilon_1}{r} = t$. One can verify that ST is indeed a t-spanner for $P \cup S$.

Let $x = |p_1 s|$ and $z = |r_0 s|$, then: (A) $x + (r - z) = r + 1 - \epsilon_1$. Note that $\angle(r_0 p_0 p_1) = \frac{n-1}{2r}$ and therefore $\angle(p_0 r_0 p_1) = \frac{\pi/2 - (n-1)}{4r} > \pi/2 - 1/n^2$. For sufficiently large n there exists $0 < \epsilon_1 < 1$ for which:
(B) $x = \sqrt{z^2 + (n-1)^2/4} - \epsilon_1$.

Solving the system of equations (A) and (B) yields $z = \frac{(n-1)^2}{8} - \frac{1}{2}$ and $x = \frac{(n-1)^2}{8} + \frac{1}{2} - \epsilon_1$. Since, $w(ST) \leq n \cdot x + (r - z) + n - 1$ we have

$$\frac{w(ST)}{G_m(P,t)} < \frac{x \cdot n + (r - z) + n - 1}{r \cdot n + n - 1}$$

$$= \frac{n((n-1)^2/8 + 1/2 - \epsilon_1 + 1) + n^3 - (n-1)^2/8 + 1/2 - 1}{n(n^3 + 1) - 1} \leq \frac{c}{n}$$

for some constant c. For sufficiently large n we have $\frac{c}{n} < \epsilon$. Hence, by setting n to be the maximum between the value that satisfies $\frac{c}{n} < \epsilon$ and the value that satisfies equation (A) we receive $\rho_t \leq \frac{w(ST)}{G_m(P,t)} < \epsilon$.

Next, we show that for every constant $t > 1$, $\rho_t < \sqrt{3}/2$. We begin by showing a tighter bound for $1 < t < 2$.

Claim 1. *For every constant* $1 < t < 2$ *and* $\epsilon > 0$, $\rho_t \leq (t+\epsilon)/(t+2)$.

Remark 1. By the above claim, for $t \to 1$ we have $\rho_t \leq 1/3 + \epsilon$ for every $\epsilon > 0$.

Proof. Consider the following set of points P of size $n = \frac{(2\epsilon^2+\epsilon(-2t^2+t-4)+2(t-1)t)}{(\epsilon(\epsilon-t^2-2))}$. Let $p_1, p_2, ..., p_n$ be the order of the points in P from left to right and let l_1 and l_2 be two lines parallel to the x-axis. For $0 < i \leq n$, the points $\{p_i : i \text{ is even}\}$ are located on l_2 and $\{p_i : i \text{ is odd}\}$ are located on l_1, such that for every $0 \leq i \leq n-1$, $|p_i, p_{i+1}| = 1$ and for every $0 \leq i \leq n-2$, $|p_i, p_{i+2}| = (2-\epsilon)/t$ (see Fig. 2). One can verify that $G_m(P,t) = (P,E)$, where $E = \{\{p_i, p_{i+1}\} : 0 \leq i \leq n-1\} \cup \{\{p_i, p_{i+2}\} : 0 \leq i \leq n-2\}$. Thus, $W_m(P,t) = (n-1) + (n-2)(2-\epsilon)/t$.

We define a Steiner t-spanner $ST = (P \cup S, E_s)$, where S is a set of $2(n-1)$ points, two on each edge of $\{\{p_i, p_{i+1}\} : 0 \leq i \leq n-1\}$ in distance $\epsilon_1 = t\epsilon/(2t-2+\epsilon)$ from each endpoint, and E_s is defined as follows. Let $q_1, q_2, ..., q_{2(n-1)}$ be the points $P \cup S$ ordered from left to right, then $E_s = \{\{q_i, q_{i+1}\} : 0 \leq i \leq 2(n-1)\} \cup \{\{q_{i-1}, q_{i+1}\} : 0 < i < 2(n-1) \wedge q_i \in P\}$ (see Fig. 2). Due to triangles similarity, the length of the edges from the second type is $\epsilon_1(2-\epsilon)/t$. One can verify that ST is indeed a t-spanner for $P \cup S$.

Thus, we have $w(ST) = (n-1) + (n-2)\epsilon(2-\epsilon)/(2t-2+\epsilon)$ and therefore,

$$\rho_t \leq \frac{w(ST)}{W_m(P,t)} = \frac{(n-1)+(n-2)\epsilon(2-\epsilon)/(2t-2+\epsilon)}{(n-1)+(n-2)(2-\epsilon)/t} = \frac{t+\epsilon}{t+2}.$$

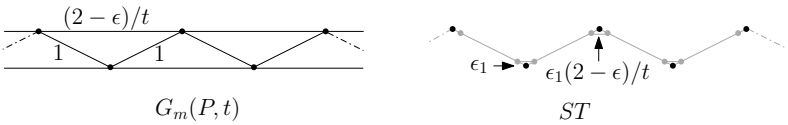


Fig. 2. Left: the graph $G_m(P,t)$. Right: the graph ST (including the point set S) as defined in the proof of Claim 1.

Lemma 5. *For every constant* t, $\rho_t < \sqrt{3}/2$.

Proof. For a constant $1 < t < 2$ the lemma follows from Claim 1. The proof for $2 \leq t \leq 6.3$ was omitted due to space limitation. The main ideas of the proof together with a proof for for $t > 6.3$ are given next.

For a constant $2 \leq t \leq 4$ we define a set P of 7 points with respect to the vertices of a regular hexagon. We locate the points in P inside the hexagon in distance $x(t)$ from its vertices as depicted in Fig. 3(a) together with the graph

$G_m(P,t)$ and the Steiner t-spanner that we suggest for P. The value $x(t)$ ensures $\frac{w(ST)}{W_m(P,t)} < \frac{\sqrt{3}}{2}$, hence $\rho_t < \frac{\sqrt{3}}{2}$.

For a constant $4 < t \leq 6.3$ we define a set P of 13 points as follows. Two points are located on a boundary of a circle within a distance $x(t)$ from each other and 5 points are equally spaced on the greater portion of the boundary between those two points. The remaining 6 points are located outside the circle with respect to every two adjacent points on the boundary, except for the two at distance $x(t)$ from each other, as a third vertex of an isosceles triangle (see Fig. 3(b)). The graph $G_m(P,t)$ and the Steiner t-spanner that we suggest are depicted in Fig. 3(b). The value $x(t)$ ensures $\frac{w(ST)}{W_m(P,t)} < \frac{\sqrt{3}}{2}$, which implies $\rho_t < \frac{\sqrt{3}}{2}$.

For $t > 6.3$ we define $P = \{p_1, ..., p_{2n+1}\}$ as follows. The points p_1 and p_{2n+1} are located on a boundary of a circle within a distance x (which depends on t) from each other and $n-1$ other points are equally spaced, with distances $\sqrt{3}$, on the greater portion of the boundary between p_1 and p_{2n+1}. Additional n points are located outside the circle with respect to every pair $\{p,q\}$ of adjacent points on the perimeter, except for the pair $\{p_1, p_{2n+1}\}$, as a third vertex of the isosceles triangle with a base $\{p,q\}$ and a side length $1 < y < \sqrt{3}$(see Fig. 3(c)). We define $y = \frac{\sqrt{2}\sqrt{t^2(2-\sqrt{3})-2t+2}+t(1+\sqrt{3})-2}{2t}$ and $x = \frac{(\sqrt{3}-y)(2-2y)}{(\sqrt{3}-2y)}n$.

Claim 2. *For sufficiently large n, $G_m(P,t) = (P,E)$, where E contains the cord $\{p_1, p_{2n+1}\}$ and all the equal sides of all the isosceles triangles (see Fig. 3(c)).*

We omit the proof of the above claim due to space limitations. The intuition for its correctness is the following. The lightest t-spanner that does not include the long edge $\{p_1, p_{2n+1}\}$ is the graph obtained by replacing one side of length y with the triangle base of length $\sqrt{3}$ (rather than adding some cord edges) in the required number of triangles. This spanner has the same weight as the spanner presented in the claim for sufficiently large n.

By Claim 2, $W_m(P,t) = 2yn + x$. We define a Steiner t-spanner $ST = (P \cup S, E_s)$, where S is the set of n Torricelli points of each three vertices of a triangle, i.e., inside each triangle $\triangle prq$ we locate a Steiner point s, such that $\angle psr = \angle rsq = \angle qsp = 2\pi/3$, and E_s is a set of $3n$ edges connecting each Torricelli point to the three corresponding triangle vertices. By the law of cosines the length of

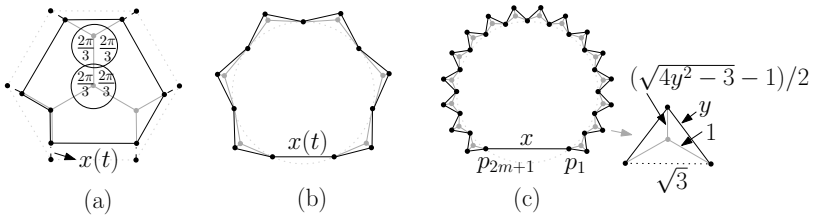


Fig. 3. The graphs $G_m(P,t)$ and ST are depicted in black and gray respectively, as defined in the proof of Lemma 5 for $2 \leq t \leq 4$ (a), $4 < t \leq 6.3$ (b) and $t > 6.3$ (c)

three edges inside each triangle are $1, 1$ and $(\sqrt{4y^2-3}-1)/2$. Since for every $q, r \in P \cup S$, $\delta_{ST}(q,r)/|qr| \leq \frac{2n}{x} = t$, ST is indeed a t-spanner for $p \cup S$.

Thus, we have $w(ST) = (2 + (\sqrt{4y^2-3}-1)/2)n$ and therefore,

$$\rho_t \leq \frac{(2+\sqrt{y^2+y+1})n}{2yn+x} = \frac{(2+\sqrt{y^2+y+1})}{2y+(\sqrt{3}-y)(2-2y)/(\sqrt{3}-2y)} <^{(*)} \sqrt{3}/2$$

The inequality $(*)$ holds for every $t > 6.3$.

References

1. Abam, M.A., de Berg, M., Farshi, M., Gudmundsson, J.: Region-fault tolerant geometric spanners. Discrete Comput. Geom. 41(4), 556–582 (2009)
2. Althöfer, I., Das, G., Dobkin, D.P., Joseph, D., Soares, J.: On sparse spanners of weighted graphs. Discrete & Computational Geometry 9, 81–100 (1993)
3. Arikati, S.R., Chen, D.Z., Chew, L.P., Das, G., Smid, M.H.M., Zaroliagis, C.D.: Planar Spanners and Approximate Shortest Path Queries Among Obstacles in the Plane. In: Díaz, J. (ed.) ESA 1996. LNCS, vol. 1136, pp. 514–528. Springer, Heidelberg (1996)
4. Benmoshe, B., Omri, E., Elkin, M.: Optimizing budget allocation in graphs. In: 23RD Canadian Conference on Computational Geometry, pp. 33–38 (2011)
5. Chung, F.R.K., Graham, R.L.: A new bound for euclidean steiner minimal trees. Ann. N.Y. Acad. Sci (1985)
6. Das, G., Heffernan, P.J., Narasimhan, G.: Optimally sparse spanners in 3-dimensional euclidean space. In: Symposium on Computational Geometry, pp. 53–62 (1993)
7. Das, G., Narasimhan, G., Salowe, J.S.: A new way to weigh malnourished euclidean graphs. In: SODA, pp. 215–222 (1995)
8. Elkin, M., Solomon, S.: Narrow-shallow-low-light trees with and without steiner points. SIAM J. Discret. Math. 25(1), 181–210 (2011)
9. Gilbert, E.N., Pollak, H.O.: Steiner minimal trees. SIAM J. App. Math. 16(1), 1–29 (1968)
10. Hwang, F.K., Richards, D.S., Winter, P.: The Steiner Tree Problem. Elsevier Science (1992)
11. Narasimhan, G., Smid, M.: Geometric Spanner Networks. Cambridge University Press (2007)

Geometric RAC Simultaneous Drawings of Graphs*

Evmorfia Argyriou[1], Michael Bekos[2], Michael Kaufmann[2], and Antonios Symvonis[1]

[1] School of Applied Mathematical & Physical Sciences,
National Technical University of Athens, Greece
{fargyriou,symvonis}@math.ntua.gr
[2] Institute for Informatics, University of Tübingen, Germany
{bekos,mk}@informatik.uni-tuebingen.de

Abstract. In this paper, we study the *geometric RAC simultaneous drawing problem*: Given two planar graphs that share a common vertex set but have disjoint edge sets, a geometric RAC simultaneous drawing is a straight-line drawing in which (i) each graph is drawn planar, (ii) there are no edge overlaps, and, (iii) crossings between edges of the two graphs occur at right-angles. We first prove that two planar graphs admitting a geometric simultaneous drawing may not admit a geometric RAC simultaneous drawing. We further show that a cycle and a matching always admit a geometric RAC simultaneous drawing, which can be constructed in linear time.

We also study a closely related problem according to which we are given a planar embedded graph G and the main goal is to determine a geometric drawing of G and its dual G^* (without the face-vertex corresponding to the external face) such that: (i) G and G^* are drawn planar, (ii) each vertex of the dual is drawn inside its corresponding face of G and, (iii) the primal-dual edge crossings form right-angles. We prove that it is always possible to construct such a drawing if the input graph is an outerplanar embedded graph.

1 Introduction

A *geometric right-angle crossing drawing* (or *geometric RAC drawing*, for short) of a graph is a straight-line drawing in which every pair of crossing edges intersects at right-angle. A graph which admits a geometric RAC drawing is called *right-angle crossing graph* (or *RAC graph*, for short). Motivated by cognitive experiments of Huang et al. [17], which indicate that the negative impact of an edge crossing on the human understanding of a graph drawing is eliminated in the case where the crossing angle is greater than seventy degrees, RAC graphs were recently introduced in [10] as a response to the problem of drawing graphs with optimal crossing resolution.

* The work of E.N. Argyriou 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: Heracleitus II. Investing in knowledge society through the European Social Fund. The work of M.A. Bekos is implemented within the framework of the Action "Supporting Postdoctoral Researchers" of the Operational Program "Education and Lifelong Learning" (Action's Beneficiary: General Secretariat for Research and Technology), and is co-financed by the European Social Fund (ESF) and the Greek State.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 287–298, 2012.

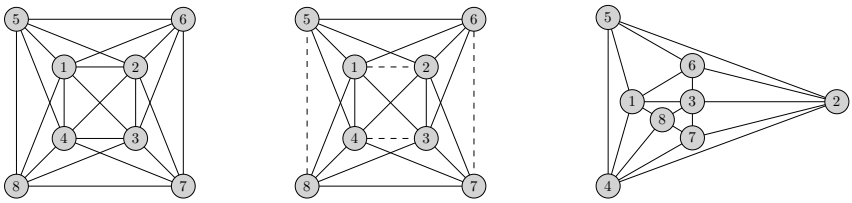


Fig. 1. (a) A graph with 8 vertices and 22 edges which does not admit a RAC drawing [11]. (b) A decomposition of the graph of Fig.1a into a planar graph (solid edges; a planar drawing is given in Fig.1c) and a matching (dashed edges), which implies that a planar graph and a matching do not always admit a GRacSim drawing; their union is not RAC.

Simultaneous graph drawing deals with the problem of drawing two (or more) planar graphs on the same set of vertices on the plane, such that each graph is drawn planar[1] (i.e., only edges of different graphs are allowed to cross). The *geometric* version restricts the problem to straight-line drawings. Besides its independent theoretical interest, this problem arises in several application areas, such as software engineering, databases and social networks, where a visual analysis of evolving graphs, defined on the same set of vertices, is useful.

Both problems mentioned above are active research topics in the graph drawing literature and positive and negative results are known for certain variations (refer to Section 2). In this paper, we study the *geometric RAC simultaneous drawing problem* (or *GRacSim* drawing problem, for short), i.e., a combination of both problems. Formally, the GRacSim drawing problem can be stated as follows: Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ be two planar graphs that share a common vertex set but have disjoint edge sets, i.e., $E_1 \subseteq V \times V$, $E_2 \subseteq V \times V$ and $E_1 \cap E_2 = \emptyset$. The main task is to place the vertices on the plane so that, when the edges are drawn as straight-lines, (i) each graph is drawn planar, (ii) there are no edge overlaps and (iii) crossings between edges in E_1 and E_2 occur at right-angles. Let $G = (V, E_1 \cup E_2)$ be the graph induced by the union of G_1 and G_2. Observe that G should be a RAC graph, which implies that $|E_1 \cup E_2| \leq 4|V| - 10$ [10]. We refer to this relationship as the *RAC-size constraint*.

Note that, in the ordinary geometric simultaneous drawing problem the input graphs are allowed to share edges, i.e., $E_1 \cap E_2$ is non-empty in general. For instance, it is known that there exists a planar graph and a matching that do not admit a geometric simultaneous drawing [7]. However, this does not immediately imply that a planar graph and a matching do not admit a GRacSim drawing either (since the graphs utilized in the proof of the corresponding theorem in [7] share a common edge). Fig.1 depicts an alternative and simpler technique to prove such negative results for GRacSim drawings, which is based on the fact that not all graphs that obey the RAC-size constraint are eventually RAC graphs. On the other hand, as we will shortly see, two planar graphs admitting a geometric simultaneous drawing may not admit a GRacSim drawing.

[1] In the graph drawing literature, the problem is known as "simultaneous graph drawing with mapping". For simplicity, we use the term "simultaneous graph drawing".

The GRacSim drawing problem is of interest since it combines two current research topics in graph drawing. Our motivation to study this problem rests on the work of Didimo et al. [10] who proved that the crossing graph of a geometric RAC drawing is bipartite[2]. Thus, the edges of a geometric RAC drawing of a graph $G = (V, E)$ can be partitioned into two sets E_1 and E_2, such that no two edges of the same set cross. So, the problem we study is, in a sense, equivalent to the problem of finding a geometric RAC drawing of an input graph (if one exists), given its crossing graph.

A closely related problem to the GRacSim drawing problem, refered to as *geometric Graph-Dual RAC simultaneous drawing problem* (or *GDual-GRacSim* for short), is the following: *Given a planar embedded graph G, determine a geometric drawing of G and its dual G^* (without the face-vertex corresponding to the external face) such that: (i) G and G^* are drawn planar, (ii) each vertex of the dual is drawn inside its corresponding face of G and, (iii) the primal-dual edge crossings form right-angles.*

This paper is structured as follows: In Section 2, we review relevant previous research. In Section 3, we demonstrate that two planar graphs admitting a geometric simultaneous drawing may not admit a GRacSim drawing. In Section 4, we prove that a cycle and a matching always admit a GRacSim drawing, which can be constructed in linear time. In Section 5, we study the GDual-GRacSim drawing problem. We show that given a planar embedded graph, a GDual-GRacSim drawing of the planar graph and its dual does not always exist. If the input graph is an outerplanar embedded graph, we present an algorithm that constructs a GDual-GRacSim drawing of the outerplanar graph and its dual. We conclude in Section 6 with open problems.

2 Related Work

Didimo et al. [10] were the first to study the geometric RAC drawing problem and proved that any graph with $n \geq 3$ vertices that admits a geometric RAC drawing has at most $4n - 10$ edges. Arikushi et al. [4] presented bounds on the number of edges of polyline RAC drawings with at most one or two bends per edge. Angelini et al. [1] presented acyclic planar digraphs that do not admit upward geometric RAC drawings and proved that the corresponding decision problem is $\mathcal{NP}$-hard. Argyriou et al. [3] proved that it is $\mathcal{NP}$-hard to decide whether a given graph admits a geometric RAC drawing (i.e., the upwardness requirement is relaxed). Di Giacomo et al. [8] presented tradeoffs on the maximum number of bends per edge, the required area and the crossing angle resolution. Didimo et al. [9] characterized classes of complete bipartite graphs that admit geometric RAC drawings. Van Kreveld [18] showed that the quality of a planar drawing of a planar graph (measured in terms of area required, edge-length and angular resolution) can be improved if one allows right-angle crossings. Eades and Liotta [11] proved that a *maximally dense RAC graph* (i.e., $|E| = 4|V| - 10$) is also 1-planar, i.e., it admits a drawing in which every edge is crossed at most once.

Regarding the geometric simultaneous graph drawing problem, Brass et al. [5] presented algorithms for drawing simultaneously (a) two paths, (b) two cycles and, (c) two caterpillars. Estrella-Balderrama et al. [14] proved that the problem of determining

[2] This can be interpreted as follows: "If two edges of a geometric RAC drawing cross a third one, then these two edges must be parallel."

whether two planar graphs admit a geometric simultaneous drawing is $\mathcal{NP}$-hard. Erten and Kobourov [13] showed that a planar graph and a path cannot always be drawn simultaneously. Geyer, Kaufmann and Vrt'o [16], showed that a geometric simultaneous drawing of two trees does not always exist. Angelini et al. [2] proved the same result for a path and a tree. Cabello et al. [7] showed that a geometric simultaneous drawing of a matching and (a) a wheel, (b) an outerpath or, (c) a tree always exists. For a quick overview of known results refer to Table 1 of [15].

Brightwell and Scheinermann [6] proved that the GDual-GRacSim drawing problem always admits a solution if the input graph is a triconnected planar graph. To the best of our knowledge, this is the only result which incorporates the requirement that the primal-dual edge crossings form right-angles. Erten and Kobourov [12], presented an $O(n)$ time algorithm that results into a simultaneous drawing but, unfortunately, not a RAC drawing of a triconnected planar graph and its dual on an $O(n^2)$ grid, where n is the number of vertices of G and G^*.

Before we proceed with the description of our results, we introduce some necessary notation. Let $G = (V, E)$ be a simple, undirected graph drawn on the plane. We denote by $\Gamma(G)$ the drawing of G. By $x(v)$ and $y(v)$, we denote the x- and y-coordinate of $v \in V$ in $\Gamma(G)$. We refer to the vertex (edge) set of G as $V(G)$ ($E(G)$). Given two graphs G and G', we denote by $G \cup G'$ the graph induced by the union of G and G'.

3 A Wheel and a Cycle: A Negative Result

In this section, we show that there exists a pair of planar graphs that admits a geometric simultaneous drawing, their union meets the RAC size constraint and they do not admit a GRacSim drawing (i.e, the class of graphs that admit GRacSim drawings is a subset of the class of graphs for which a simultaneous drawing is possible). We achieve this by showing that there exists a wheel and a cycle which do not admit a GRacSim drawing. Cabello et al. [7] have shown that a geometric simultaneous drawing of a wheel and a cycle always exists.

Our proof utilizes the *augmented triangle antiprism graph* [3,10], depicted in Fig. 2a. The augmented triangle antiprism graph contains two triangles $\mathcal{T}$ and $\mathcal{T}'$ (refer to the dashed and bold drawn triangles in Fig. 2a) and a "central" vertex v_0 incident to the vertices of $\mathcal{T}$ and $\mathcal{T}'$. If we delete the central vertex, the remaining graph corresponds to the skeleton of a triangle antiprism and it is commonly referred to as *triangle antiprism graph*. Didimo et al. [10] used the augmented triangle antiprism graph as an example of a maximally dense RAC graph (i.e., $|E| = 4|V| - 10$).

Lemma 1. *The geometric RAC drawings of the augmented triangle antiprism graph define exactly eight combinatorial embeddings.*

Sketch of proof. Due to space constraints we omit the detailed proof of this lemma. We simply note that (i) v_a and v'_a, (ii) v_b and v'_b, and, (iii) v_c and v'_c share the same neighbors. So, the eight combinatorial embeddings are implied by the drawing of Fig. 2a by mutually exchanging the positions of (i) v_a and v'_a, (ii) v_b and v'_b, and, (iii) v_c and v'_c. For an example refer to Fig. 2b. □

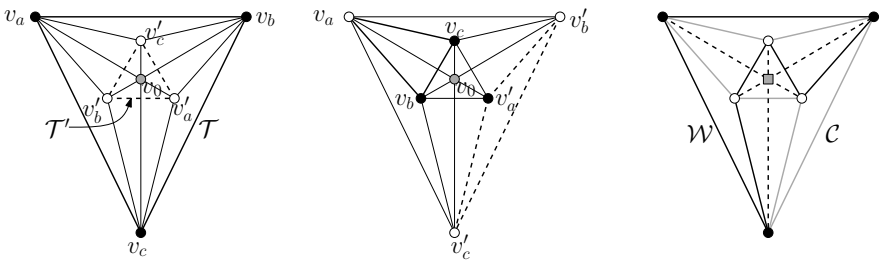


Fig. 2. (a)-(b) Two different RAC drawings of the augmented triangle antiprism graph with different combinatorial embeddings. (c) The union of wheel $\mathcal{W}$ (solid and dashed black edges) and cycle $\mathcal{C}$ (gray edges) is the augmented triangle antiprism graph.

Theorem 1. *There exists a wheel and a cycle which do not admit a* GRacSim *drawing.*

Proof. We denote the wheel by $\mathcal{W}$ and the cycle by $\mathcal{C}$. The counterexample is depicted in Fig.2c. The center of $\mathcal{W}$ is marked by a box, the spokes of $\mathcal{W}$ are drawn as dashed line-segments, while the rim of $\mathcal{W}$ is drawn in bold. Cycle $\mathcal{C}$ is drawn in gray. The graph induced by the union of $\mathcal{W}$ and $\mathcal{C}$ (which in a GRacSim drawing of $\mathcal{W}$ and $\mathcal{C}$ should be drawn with right-angle crossings) is the augmented triangle antiprism graph, which, by Lemma 1, has exactly eight RAC combinatorial embeddings. However, in none of them wheel $\mathcal{W}$ is drawn planar. This completes the proof. □

4 A Cycle and a Matching: A Positive Result

In this section, we first prove that a path and a matching always admit a GRacSim drawing and then we show that a cycle and a matching always admit a GRacSim drawing as well. Note that the union of a path and a matching is not necessarily a planar graph. Cabello et al. [7] provide an example of a path and a matching, which form a subdivision of $K_{3,3}$. We denote the path by $\mathcal{P}$ and the matching by $\mathcal{M}$. Let $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_n$ be the edges of $\mathcal{P}$ (see Fig.3). In order to keep the description of our algorithm simple, we will initially assume that n is even and $|E(\mathcal{M})| = n/2$. Later on this section, we will describe how to cope with the cases where n is odd or $|E(\mathcal{M})| < n/2$. Recall that by the definition of the GRacSim drawing problem, $\mathcal{P}$ and $\mathcal{M}$ do not share any edge, i.e., $E(\mathcal{P}) \cap E(\mathcal{M}) = \emptyset$.

The basic idea of our algorithm is to identify in the graph induced by the union of $\mathcal{P}$ and $\mathcal{M}$ a set of cycles $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$, $k \leq n/4$, such that: (i) $|E(\mathcal{C}_1)| + |E(\mathcal{C}_2)| + \ldots + |E(\mathcal{C}_k)| = n$, (ii) $\mathcal{M} \subseteq \mathcal{C}_1 \cup \mathcal{C}_2 \cup \ldots \cup \mathcal{C}_k$, and, (iii) the edges of cycle $\mathcal{C}_i$, $i = 1, 2, \ldots, k$ alternate between edges of $\mathcal{P}$ and $\mathcal{M}$. Note that properties (i) and (ii) imply that the cycle collection will contain half of $\mathcal{P}$'s edges and all of $\mathcal{M}$'s edges. In our drawing, these edges will not cross with each other. The remaining edges of $\mathcal{P}$ will introduce only right-angle crossings with the edges of $\mathcal{M}$.

Let $\mathcal{P}_{odd}$ be a subgraph of $\mathcal{P}$ which contains each second edge of $\mathcal{P}$, starting from its first edge, i.e., $E(\mathcal{P}_{odd}) = \{(v_i, v_{i+1});\ 1 \leq i < n,\ i \text{ is odd}\}$. In Fig.3, the edges

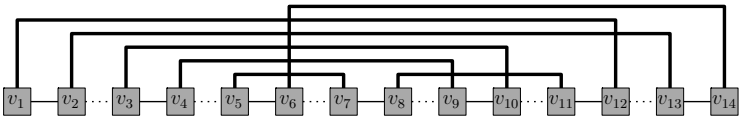


Fig. 3. An example of a path $\mathcal{P}$ and a matching $\mathcal{M}$. The path appears at the bottom of the figure. The edges of $\mathcal{M}$ are drawn bold, with two bends each. The edges of path $\mathcal{P}$ form two matchings, i.e., $\mathcal{P}_{odd}$ and $\mathcal{P} - \mathcal{P}_{odd}$. The edges of $\mathcal{P}_{odd}$ are drawn solid, while the edges of $\mathcal{P} - \mathcal{P}_{odd}$ dotted.

of $\mathcal{P}_{odd}$ are drawn solid. Clearly, $\mathcal{P}_{odd}$ is a matching. Since we have assumed that n is even, $\mathcal{P}_{odd}$ contains exactly $n/2$ edges. Hence, $|E(\mathcal{P}_{odd})| = |E(\mathcal{M})|$. In addition, $\mathcal{P}_{odd}$ covers all vertices of $\mathcal{P}$, and, $E(\mathcal{P}_{odd}) \cap E(\mathcal{M}) = \emptyset$. The later equation trivially follows from our initial hypothesis, which states that $E(\mathcal{P}) \cap E(\mathcal{M}) = \emptyset$. We conclude that $\mathcal{P}_{odd} \cup \mathcal{M}$ is a 2-regular graph. Thus, each connected component of $\mathcal{P}_{odd} \cup \mathcal{M}$ corresponds to a cycle of even length, which alternates between edges of $\mathcal{P}_{odd}$ and $\mathcal{M}$. This is the cycle collection mentioned above (see Fig. 4).

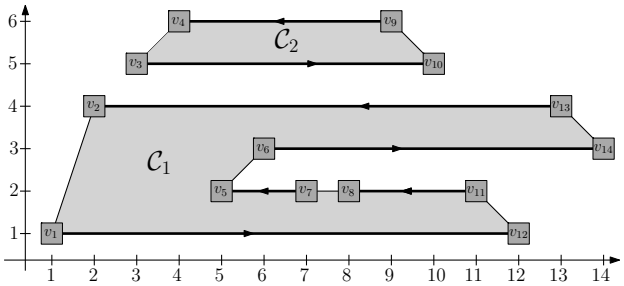
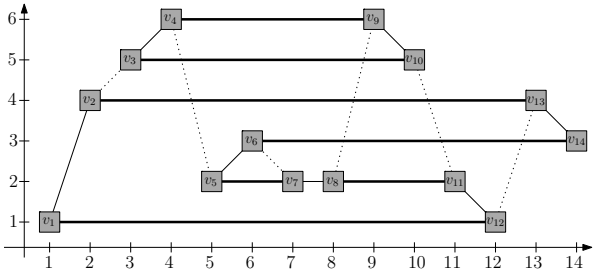


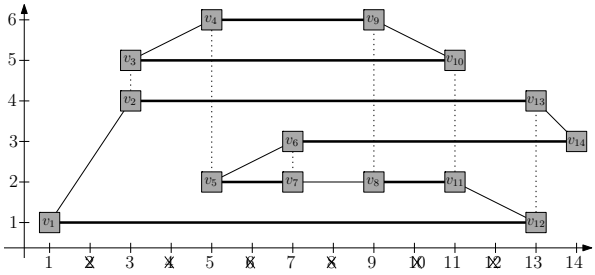
Fig. 4. $\mathcal{P}_{odd} \cup \mathcal{M}$ (of Fig. 3) consists of cycles $\mathcal{C}_1$ and $\mathcal{C}_2$. The edges of $\mathcal{P}_{odd}$ are drawn solid, while the edges of $\mathcal{M}$ are drawn bold.

Initially, we fix the x-coordinate of each vertex of $\mathcal{P}$ by setting $x(v_i) = i$, $1 \leq i \leq n$. This ensures that $\mathcal{P}$ is x-monotone and hence planar. Later on, we will slightly change the x-coordinate of some vertices of $\mathcal{P}$ (without affecting $\mathcal{P}$'s monotonicity). Note that the algorithm can be adjusted so that the x and y coordinates of each vertex are computed at the same time. We have chosen to compute them separately in order to simplify the presentation. The y-coordinate of each vertex of $\mathcal{P}$ is established by considering the cycles of $\mathcal{P}_{odd} \cup \mathcal{M}$.

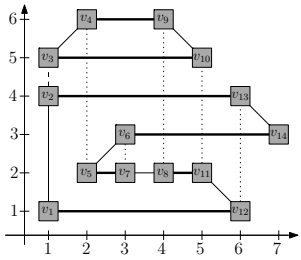
We draw each of these cycles in turn. More precisely, assume that zero or more cycles have been completely drawn and let $\mathcal{C}$ be the cycle in the cycle collection which contains the leftmost vertex, say v_i, of $\mathcal{P}$ that has not been drawn yet (initially, v_i is identified by v_1). Then, vertex v_i should be an odd-indexed vertex and thus (v_i, v_{i+1}) belongs to $\mathcal{C}$. Orient cycle $\mathcal{C}$ so that vertex v_i is the first vertex of cycle $\mathcal{C}$ and v_{i+1} is the last (see Fig. 4). Based on this orientation, we will draw the edges of $\mathcal{C}$ in a snake-like fashion, starting from vertex v_i and reaching vertex v_{i+1} last. The first edge to be drawn



(a) A drawing obtained by incorporating the edges of $\mathcal{P} - \mathcal{P}_{odd}$ into the drawing of Fig.4.



(b) A drawing obtained by moving the even-indexed vertices of $\mathcal{P}$ in the drawing of Fig.5a one unit to the right.



(c) A compact GRacSim drawing

Fig. 5. In all drawings, the edges of $\mathcal{P}_{odd}$ are drawn solid, while the edges of $\mathcal{P} - \mathcal{P}_{odd}$ dotted. The edges of $\mathcal{M}$ are drawn bold.

is incident to vertex v_i and belongs to $\mathcal{M}$. We draw it as a horizontal line-segment at the bottommost available layer in the produced drawing (initially, $L_1 : y = 1$). Since cycle $\mathcal{C}$ alternates between edges of $\mathcal{P}_{odd}$ and $\mathcal{M}$, the next edge to be drawn belongs to $\mathcal{P}_{odd}$ followed by an edge of $\mathcal{M}$. If we can draw both of them in the current layer without introducing edge overlaps, we do so. Otherwise, we employ an additional layer. We continue in the same manner, until edge (v_i, v_{i+1}) is reached in the traversal of cycle $\mathcal{C}$. This edge connects two consecutive vertices of $\mathcal{P}$ that are the leftmost in the drawing of $\mathcal{C}$. Therefore, edge (v_i, v_{i+1}) can be added in the drawing of $\mathcal{C}$ without introducing any crossings. Thus, cycle $\mathcal{C}$ is drawn planar.

So far, we have drawn all edges of $\mathcal{M}$ and half of the edges of $\mathcal{P}$ (i.e., $\mathcal{P}_{odd}$) and we have obtained a planar drawing in which all edges of $\mathcal{M}$ are drawn as horizontal, non-overlapping line segments. In the worst case, this drawing occupies $n/2$ layers.

We proceed to incorporate the remaining edges of $\mathcal{P}$, i.e, the ones that belong to $\mathcal{P} - \mathcal{P}_{odd}$, into the drawing (refer to the dotted drawn edges of Fig.5a). Since $x(v_i) = i$, $i = 1, 2, \ldots, n$, the edges of $\mathcal{P}$ do not cross with each other and therefore $\mathcal{P}$ is drawn planar. In contrast, an edge of $\mathcal{P} - \mathcal{P}_{odd}$ may cross multiple edges of $\mathcal{M}$, and, these crossings do not form right-angles (see Fig.5a). In order to fix these crossings, we suggest to move each even-indexed vertex of $\mathcal{P}$ one unit to the right (keeping its y-coordinate unchanged), except for the last vertex of $\mathcal{P}$. Then, the endpoints of the edges of $\mathcal{P} - \mathcal{P}_{odd}$ have exactly the same x-coordinate and cross at right-angles the edges of

$\mathcal{M}$ which are drawn as horizontal line-segments. The path remains x-monotone (but not strictly anymore) and hence planar. In addition, it is not possible to introduce vertex overlaps, since in the produced drawing each edge of $\mathcal{M}$ has at least two units length (recall that $E(\mathcal{P}) \cap E(\mathcal{M}) = \emptyset$). Since the vertices of the drawing do not occupy even x-coordinates, the width of the drawing can be reduced from n to $n/2+1$ (see Fig. 5b). We can further reduce the width of the produced drawing by merging consecutive columns that do not interfere in y-direction into a common column (see Fig. 5c). However, this post-processing does not result into a drawing of asymptotically smaller area.

In order to complete the description of our algorithm, it remains to consider the cases where n is odd or $|E(\mathcal{M})| < n/2$. Both cases can be treated similarly. If n is odd or $|E(\mathcal{M})| < n/2$, there exist vertices of $\mathcal{P}$ which are not covered by matching $\mathcal{M}$. As long as there exist such vertices, we can momentarily remove them from the path by contracting each subpath consisting of degree-2 vertices into a single edge. By this procedure, we obtain a new path $\mathcal{P}'$, so that $\mathcal{M}$ covers all vertices of $\mathcal{P}'$. If we draw $\mathcal{P}'$ and $\mathcal{M}$ simultaneously, then it is easy to incorporate the removed vertices in the produced drawing, since they do not participate in $\mathcal{M}$. The following theorem summarizes our result.

Theorem 2. *A path and a matching always admit a* GRacSim *drawing on an* $(n/2 + 1) \times n/2$ *integer grid. Moreover, the drawing can be computed in linear time.*

Proof. Finding the cycles of $\mathcal{P}_{odd} \cup \mathcal{M}$ can be done in $O(n)$ time, where n is the number of vertices of $\mathcal{P}$; we identify the leftmost vertex of each cycle and then we traverse it. Having computed the cycle collection of $\mathcal{P}_{odd} \cup \mathcal{M}$, the coordinates of the vertices are computed in $O(n)$ total time by a traversal of the cycle. □

We extend the algorithm that produces a GRacSim drawing of a path and a matching to also cover the case of a cycle $\mathcal{C}$ and a matching $\mathcal{M}$. Obviously, if we remove an edge from the input cycle, the remaining graph is a path $\mathcal{P}$. Then, we apply the developed algorithm and obtain a GRacSim drawing of $\mathcal{P}$ and $\mathcal{M}$, in which the first vertex of $\mathcal{P}$ is drawn at the bottommost layer (hence its incident edge in $\mathcal{M}$ is not crossed), and, the last vertex of $\mathcal{P}$ is drawn rightmost. With these two properties, we can add the removed edge, between the first and the last vertex of $\mathcal{P}$ without introducing new crossings. To achieve this, we move the first vertex of $\mathcal{P}$ at most $n/2 + 2$ units downwards (keeping its x-coordinate unchanged) and the last vertex of $\mathcal{P}$ at most $n/2 + 1$ units rightwards (keeping its y-coordinate unchanged). Then, the insertion in the drawing of the edge that closes the cycle does not introduce any crossings, as desired.

Theorem 3. *A cycle and a matching always admit a* GRacSim *drawing on an* $(n + 2) \times (n + 2)$ *integer grid. Moreover, the drawing can be computed in linear time.*

Theorem 4. *Let G be a simple connected graph that can be decomposed into a matching and either a path or a cycle. Then, G is a RAC graph.*

Proof. The argument trivially holds in the case where the input path (or cycle) is hamiltonian. If it is not hamiltonian, vertices that are not covered by the path (or cycle) are only incident to edges of the matching. Thus, they can be momentarily removed, compute a drawing of the remaining graph and easily insert them into the resulting drawing, since they are of degree 1. □

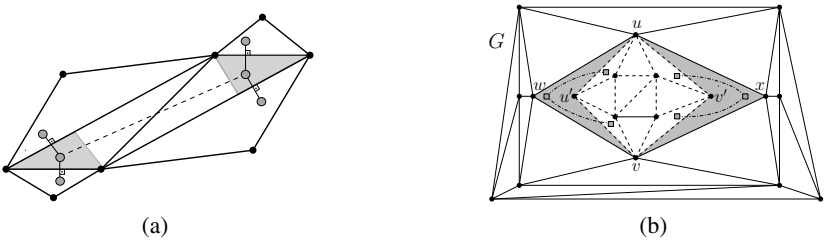
5 A Planar Graph and Its Dual: An Interesting Variation

In this section, we examine the GDual-GRacSim drawing problem. This problem can be considered as a variation of the GRacSim drawing problem, where the first graph (i.e., the planar graph) determines the second one (i.e., the dual) and places restrictions on its layout. As already stated in Section 2, Brightwell and Scheinermann [6] proved that the GDual-GRacSim problem always admits a solution if the input graph is a triconnected planar graph. For the general case of planar graphs, we demonstrate by an example that it is not always possible to compute such a drawing, and thus, we concentrate our study in the more interesting case of outerplanar graphs.

Initially, we consider the case where the planar drawing $\Gamma(G)$ of graph G is specified as part of the input and it is required that it remains unchanged in the output. We demonstrate by an example that it is not always feasible to incorporate G^* into $\Gamma(G)$ and obtain a GDual-GRacSim drawing of G and G^*. The example is illustrated in Fig. 6a. In the following, we prove that if the input graph is a planar embedded graph, then the GDual-GRacSim drawing problem does not always admit a solution, as well.

Theorem 5. *Given a planar embedded graph G, a* GDual-GRacSim *drawing of G and its dual G^* does not always exist.*

Proof. We prove a slightly stronger result by investigating all possible planar embeddings of a particular G for which we prove that a GDual-GRacSim drawing of G and its dual G^* does not exist. Graph G used to establish the theorem is depicted in Fig. 6b, where the vertices drawn as boxes belong to G^*. Observe that the subgraph drawn with dashed edges is triconnected planar. Thus, it has a unique planar embedding (up to a reflection). If we replace this subgraph by an edge, the remaining primal graph is also triconnected. Hence, the graph of our example is a subdivision of a triconnected graph, which implies that it has two planar combinatorial embeddings obtained by reflections of the triconnected planar subgraph, at u and v, i.e., either u' is to the "left" of v', or to its "right". Now, observe that the dual graph should have two vertices within the gray-colored faces of Fig. 6b (refer to the vertices drawn as boxes). Each of these two vertices is incident to two vertices of the dual that lie within the triangular faces of the dashed



(a) (b)

Fig. 6. (a) The input planar drawing of the primal graph G is sketched with black colored vertices and bold edges and should remain unchanged in the output. The vertices of the dual G^* are colored gray. Then, the dual's dashed drawn edge will inevitably introduce a non right-angle crossing. (b) An example of a planar embedded graph G for which the GDual-GRacSim does not admit a solution. The problematic faces are drawn in gray.

drawn subgraph of G, incident to the two gray-colored faces. Observe that in any RAC drawing of G and G^* both quadrilaterals $uu'vw$ and $uv'vx$ must be convex, which is impossible. □

Theorem 6. *Given an outerplane embedding of an outerplanar graph G, it is always possible to determine a* GDual-GRacSim *drawing of G and its dual G^*.*

Proof. The proof is given by a recursive geometric construction which computes a GDual-GRacSim drawing of G and its dual G^*. Consider an arbitrary edge (u, v) of the outerplanar graph that does not belong to its external face and let f and g be the faces to its left and the right side, respectively, as we move along (u, v) from vertex u to vertex v. Then, (f, g) is an edge of the dual graph G^*. Since the dual of an outerplanar graph is a tree, the removal of edge (f, g) results in two trees T_f and T_g that can be considered to be rooted at vertices f and g of G^*, respectively. For the recursive step of our algorithm, we assume that we have already produced a GDual-GRacSim drawing for T_f and its corresponding subgraph of G that satisfies the following invariant properties:

I-P1: *Edge (u, v) is drawn on the external face of the GDual-GSimRAC drawing constructed so far.* Let u and v be drawn at points p_u and p_v, respectively. Denote by $\ell_{u,v}$ the line defined by p_u and p_v.

I-P2: Let the face-vertex f be drawn at point p_f. *The perpendicular from point p_f to line $\ell_{u,v}$ intersects the line segment $p_u p_v$.* Let p be the point of intersection.

I-P3: *There exists two parallel semi-lines ℓ_u and ℓ_v passing from p_u and p_v, respectively, that define a semi-strip to the right of segment $p_u p_v$ that does not intersect the drawing constructed so far.* Denote this empty semi-strip by $R_{u,v}$.

We proceed to describe how to recursively produce a drawing for tree T_g and its corresponding subgraph of G so that the overall drawing is a GDual-GRacSim drawing for G and G^*. Refer to Fig.7a. Let p_g be a point in semi-strip $R_{u,v}$ that also belongs to the perpendicular line to line-segment $p_u p_v$ that passes from point p. Thus, the segment corresponding to the edge (f, g) of the dual crosses at right-angle the segment corresponding to the edge (u, v) of G, as required. If g is a leaf (i.e., all edges of g except (u, v) are edges of the external face), we can draw the remaining edges of face g as a polyline of appropriate number of points that goes around p_g and connects p_u and p_v.

Consider now the more interesting case where g is not a leaf in G^*. In this case, we draw two circles, say $\mathcal{C}_g$ and $\mathcal{C}'_g$, centered at p_g such that both lie entirely within semi-strip $R_{u,v}$ and do not touch neither ℓ_u nor ℓ_v. Assume that circle $\mathcal{C}'_g$ is the external of the two circles. From p_u draw the tangent to circle $\mathcal{C}_g$ and let a be the point it touches $\mathcal{C}_g$ and a' be the point to the right of a where the tangent intersects circle $\mathcal{C}'_g$ (see Fig.7a). Similarly, we define points b and b' based on the tangent from p_v to $\mathcal{C}_g$.

Let $k \geq 4$ be the number of vertices defining face g. The case where $k = 3$ will be examined later. Draw $k - 4$ points on the (a', b') arc, which is furthest from segment $p_u p_v$. These points, say $\{p_i \mid 1 \leq i \leq k - 4\}$, together with points p_u, p_v, a' and b' form face g. Observe that from point p_g, we can draw perpendicular lines towards each edge of the face. Indeed, line segments $p_g a$ and $p_g b$ are perpendicular to $p_u a'$ and $p_v b'$, respectively. The remaining edges of the face are chords of circle $\mathcal{C}'_g$ and thus, we can always draw perpendicular lines to their midpoints from the center p_g of the circle.

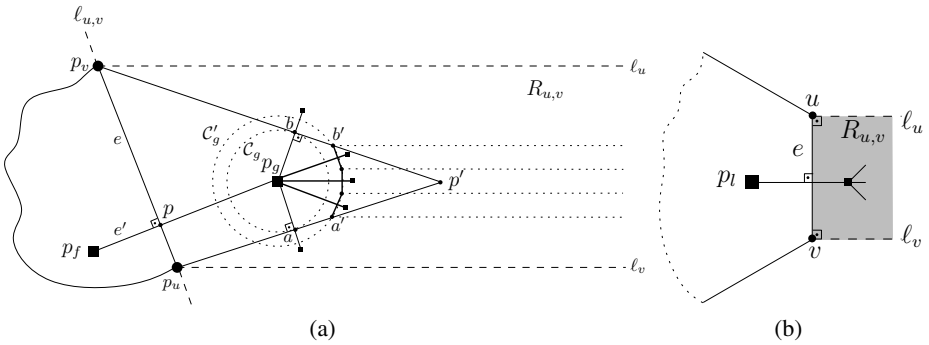


Fig. 7. (a) The recursion step of our algorithm, (b) The initial step of our algorithm

Now, from each of the newly inserted points of face g draw a semi-line that is parallel to semi-line ℓ_u and lies entirely in the semi-strip $R_{u,v}$. We observe all invariant properties stated above hold for each child of face g in the subtree T_g of the dual of G. Thus, our algorithm can be applied recursively. In the case where $k = 3$, we use the intersection of the two tangents, say p', as the third point of the triangular face. We have to be careful so that p' lies inside the semi-strip. However, we can always select a point p_g close to segment $p_u p_v$ and an appropriately small radius for circle $\mathcal{C}_g$, so that p' is inside $R_{u,v}$.

Now that we have described the recursive step, it remains to define how the recursion begins (see Fig. 7b). We start from any face of G that is a leaf at its dual tree, say face l. We draw the face as regular polygon, with face-vertex l mapped at its center, say p_l. Let $e = (u, v)$ be the only edge of the face that is internal to the outerplane embedding of G. W.l.o.g. assume that e is drawn vertically. Then, draw the horizontal semi-lines ℓ_u and ℓ_v from the endpoints of e in order to define semi-strip $R_{u,v}$. From this point on, the algorithm can recursively draw the remaining faces of G and G^*.

Note that the produced GDual-GRacSim drawing of G and its dual proves that producing such drawings is possible. The drawing is not particularly appealing since the height of the strips quickly becomes very small. However, it is a starting point towards algorithms that produce better layouts. Also note that the algorithm performs a linear number of "point computations" since for each face-vertex of the dual tree the performed computations are proportional to the degree of the face-vertex. However, the coordinates of some points may be non-rational numbers. □

6 Conclusion – Open Problems

In this paper, we introduced and examined geometric RAC simultaneous drawings. Our study raises several open problems. Among them are the following: (1) What other non-trivial classes of graphs, besides a matching and either a path or a cycle, admit a GRacSim drawing? (2) We showed that if two graphs admit a geometric simultaneous drawing, it is not necessary that they admit a GRacSim drawing. Finding a class of graphs (instead of a particular graph) with this property would strengthen this result. (3) A quite similar problem to the GRacSim drawing problem is the problem of drawing

two (or more) graphs on the same vertex set on the plane, such that each graph is drawn RAC (i.e., only edges of different graphs may introduce non-right angle crossings). Note that the class of graphs that admit such drawings contains the class of graphs for which a simultaneous drawing is possible. (4) Obtain more appealing GDual-GRacSim drawings for an outerplanar graph and its dual. Study the required drawing area.

References

1. Angelini, P., Cittadini, L., Di Battista, G., Didimo, W., Frati, F., Kaufmann, M., Symvonis, A.: On the perspectives opened by right angle crossing drawings. JGAA 15(1), 53–78 (2011)
2. Angelini, P., Geyer, M., Kaufmann, M., Neuwirth, D.: On a Tree and a Path with No Geometric Simultaneous Embedding. In: Brandes, U., Cornelsen, S. (eds.) GD 2010. LNCS, vol. 6502, pp. 38–49. Springer, Heidelberg (2011)
3. Argyriou, E.N., Bekos, M.A., Symvonis, A.: The Straight-Line RAC Drawing Problem Is NP-Hard. In: Černá, I., Gyimóthy, T., Hromkovič, J., Jefferey, K., Králović, R., Vukolić, M., Wolf, S. (eds.) SOFSEM 2011. LNCS, vol. 6543, pp. 74–85. Springer, Heidelberg (2011)
4. Arikushi, K., Fulek, R., Keszegh, B., Morić, F., Tóth, C.D.: Graphs that Admit Right Angle Crossing Drawings. In: Thilikos, D.M. (ed.) WG 2010. LNCS, vol. 6410, pp. 135–146. Springer, Heidelberg (2010)
5. Brass, P., Cenek, E., Duncan, C.A., Efrat, A., Erten, C., Ismailescu, D., Kobourov, S.G., Lubiw, A., Mitchell, J.S.B.: On simultaneous planar graph embeddings. Computational Geometry: Theory and Applications 36(2), 117–130 (2007)
6. Brightwell, G., Scheinerman, E.R.: Representations of planar graphs. SIAM Journal Discrete Mathematics 6(2), 214–229 (1993)
7. Cabello, S., van Kreveld, M.J., Liotta, G., Meijer, H., Speckmann, B., Verbeek, K.: Geometric simultaneous embeddings of a graph and a matching. JGAA 15(1), 79–96 (2011)
8. Di Giacomo, E., Didimo, W., Liotta, G., Meijer, H.: Area, Curve Complexity, and Crossing Resolution of Non-planar Graph Drawings. In: Eppstein, D., Gansner, E.R. (eds.) GD 2009. LNCS, vol. 5849, pp. 15–20. Springer, Heidelberg (2010)
9. Didimo, W., Eades, P., Liotta, G.: A characterization of complete bipartite graphs. Information Processing Letters 110(16), 687–691 (2010)
10. Didimo, W., Eades, P., Liotta, G.: Drawing graphs with right angle crossings. Theor. Comp. Sci. 412(39), 5156–5166 (2011)
11. Eades, P., Liotta, G.: Right angle crossing graphs and 1-planarity. In: 27th European Workshop on Computational Geometry (2011)
12. Erten, C., Kobourov, S.G.: Simultaneous embedding of a planar graph and its dual on the grid. Theory Computing 38(3), 313–327 (2005)
13. Erten, C., Kobourov, S.G.: Simultaneous embedding of planar graphs with few bends. JGAA 9(3), 347–364 (2005)
14. Estrella-Balderrama, A., Gassner, E., Jünger, M., Percan, M., Schaefer, M., Schulz, M.: Simultaneous Geometric Graph Embeddings. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 280–290. Springer, Heidelberg (2008)
15. Frati, F., Kaufmann, M., Kobourov, S.G.: Constrained simultaneous and near-simultaneous embeddings. JGAA 13(3), 447–465 (2009)
16. Geyer, M., Kaufmann, M., Vrto, I.: Two trees which are self-intersecting when drawn simultaneously. Discrete Mathematics 309(7), 1909–1916 (2009)
17. Huang, W., Hong, S.H., Eades, P.: Effects of crossing angles. In: IEEE Pacific Visualization Symp., pp. 41–46. IEEE (2008)
18. van Kreveld, M.: The Quality Ratio of RAC Drawings and Planar Drawings of Planar Graphs. In: Brandes, U., Cornelsen, S. (eds.) GD 2010. LNCS, vol. 6502, pp. 371–376. Springer, Heidelberg (2011)

Simultaneous Embeddings with Vertices Mapping to Pre-specified Points

Taylor Gordon

University of Waterloo

Abstract. We discuss the problem of embedding graphs in the plane with restrictions on the vertex mapping. In particular, we introduce a technique for drawing planar graphs with a fixed vertex mapping that bounds the number of times edges bend. An immediate consequence of this technique is that any planar graph can be drawn with a fixed vertex mapping so that edges map to piecewise linear curves with at most $3n + O(1)$ bends each. By considering uniformly random planar graphs, we show that $2n + O(1)$ bends per edge is sufficient on average.

To further utilize our technique, we consider simultaneous embeddings of k uniformly random planar graphs with vertices mapping to a fixed, common point set. We explain how to achieve such a drawing so that edges map to piecewise linear curves with $O(n^{1-\frac{1}{k}})$ bends each, which holds with overwhelming probability. This result improves upon the previously best known result of $O(n)$ bends per edge for the case where $k \geq 2$. Moreover, we give a lower bound on the number of bends that matches our upper bound, proving our results are optimal.

1 Introduction

Of fundamental importance to graph drawing is the problem of drawing graphs in the plane with restrictions on how vertices and edges are embedded. Indeed, discussions on *planar embeddings*, where vertices map to points and edges map to continuous non-crossing curves, were commensurate with the introduction of graph theory [5].

Bridges and Prussian cities aside, investigation into the properties of planar embeddings has been motivated by applications such as *information visualization* and *VLSI circuit design* (see [1], [9], [14]). These applications provide metrics for which certain embeddings become aesthetically or functionally preferable. For example, a situation might prefer that edges be drawn as straight lines.

A classic result of Fáry [11] showed that all *planar graphs* permit embeddings in the plane where each edge maps to a straight line segment (a result independently proven by Wagner [19] and Stein [17]). If we further restrict the vertices to map to points on an $(n-2) \times (n-2)$ grid, then a planar embedding can still be achieved with edges mapping to straight line segments [16].

On the other hand, if the vertex mapping is completely fixed, a straight-line embedding does not always exist. In fact, it was shown by Pach and Wenger [15] that if we require edges to be drawn as *polygonal curves* (piecewise linear

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 299–310, 2012.

curves) then there does not always exist an embedding with $o(n^2)$ total bends. Their results went further to show that this lower bound holds *asymptotically almost surely* for a uniformly random planar graph on n vertices; that is, the lower bound holds with probability 1 as n tends to infinity.

Kaufmann and Wiese [13] considered the case where the range of the vertex mapping is restricted to a fixed point set P of size n. They showed that any planar graph can be embedded so that each vertex maps to a unique point in P and each edge maps to a polygonal curve with at most 2 bends. This result is optimal in that there exists point sets (points on a line, for example) for which not all planar graphs can be drawn with edges bending at most once.

The problem of drawing graphs to minimize bends has also been discussed in regards to *simultaneous embeddings*. A simultaneous embedding is a drawing in the plane of k graphs $G_1, G_2, \ldots, G_k$, each over a common vertex set V, such that no two edges of one graph cross. The concept of a simultaneous embedding with this terminology was introduced in [6]. A related result of particular interest was discussed in [10] by Erten and Kobourov. They considered the special case of constructing a simultaneous embedding for when $k = 2$. Their results showed that 2 bends per edge suffice to construct a simultaneous embedding of two planar graphs.

One aim of our paper is to consolidate the above results on embedding graphs with restrictions on the vertex mapping into a single drawing technique. Lemma 3 establishes such a technique that optimally minimizes the number of bends (up to constant factors). Moreover, for the case where the vertex mapping is completely fixed, we give a result matching the constant factor of 3 on the number of bends per edge that was given in [2]. An advantage of our technique, however, is that it lends itself well to probabilistic analysis. Given a fixed vertex mapping, our technique gives at most $2n$ bends per edge on expectation for a *uniformly random planar graph*, by which we mean a graph sampled uniformly at random from the set of all planar graphs over the vertex set $V = \{1, 2, \ldots, n\}$.

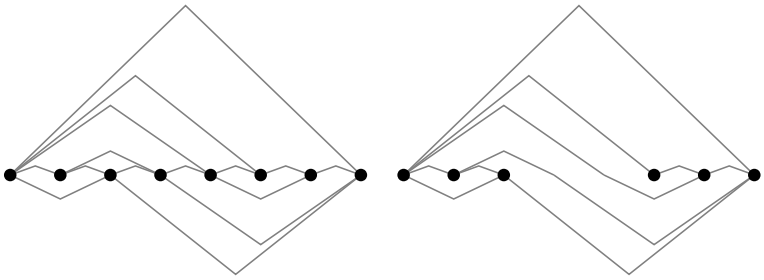
Another aim of our paper is to generalize our results to simultaneous embeddings. Our goal is to simultaneously embed planar graphs $G_1, G_2, \ldots, G_k$, each over a common vertex set V, so that each vertex uniquely maps to one of $n = |V|$ pre-specified points. Using Lemma 3, we give a construction for which each edge bends $O(n^{1-\frac{1}{k}})$ times with *overwhelming probability* if we assume that the k graphs are sampled uniformly at random; that is, each edge bends $O(n^{1-\frac{1}{k}})$ times with probability at least $1 - n^{-c}$ for any *fixed* constant c.

We go further to show that our result on simultaneous embeddings is optimal using information theory. That is, we use an encoding argument to prove a lower bound that matches our upper bound.

The drawing technique relies fundamentally on results related to book embeddings, which we introduce in Section 2. We describe the drawing technique in Section 3. Section 4 applies the drawing technique to the case of embedding a uniformly random planar graph with a fixed vertex mapping. The application of the drawing technique to simultaneous embeddings is described in Section 5. The proofs of the lower bounds are in Section 6.

2 Book Embeddings

A well-known result regarding *book embeddings* is that all Hamiltonian planar graphs have *book thickness* 2 (see [3]). A trivial consequence of this result is that a Hamiltonian planar graph can be embedded in the plane so that all vertices lie on a common line and all edges lie strictly above or below this line, except at their ends. Observe that in such an embedding, each edge can be drawn as a polygonal curve with at most 1 bend (see Fig. 1a for an example).



(a) Hamiltonian supergraph (b) Original graph

Fig. 1. The induced planar embedding of a graph from a book embedding

Any planar graph can be augmented to become 4-connected by subdividing each edge at most once and by adding additional edges. A classic theorem of Tutte [18] showed that all 4-connected planar graphs are Hamiltonian. It follows that we can always construct a Hamiltonian supergraph G' of a subdivision of a planar graph G by subdividing each original edge at most once[1]. From the Hamiltonian graph G', we can construct a book embedding (as in Fig. 1a), which induces an embedding of the original graph G (as in Fig. 1b). Observation 1 summarizes this embedding. Note that this embedding and its construction has been frequently described in graph drawing literature (as early as [1]).

Observation 1. *A planar graph G can be embedded in the plane so that*

1. *all vertices lie on a common line,*
2. *each edge bends at most once above the line, at most once below the line, and at most once on the line.*

3 Overview of the Drawing Technique

Let $G = (V, E)$ be a planar graph, and suppose that $\gamma : V \to R^2$ is a fixed vertex mapping. We define δ to be any vector in R^2 such that $\delta \cdot \gamma(u) = \delta \cdot \gamma(v)$, for $u, v \in V$, only if $u = v$ (here $\cdot$ is the standard *dot product* over R^2). That is, the

[1] G' can also be constructed in linear time by combining results from [4] and [8].

vertices in V map under γ to points at unique distances along the direction of the vector δ. Such a direction can trivially be seen to always exist.

Suppose that G is embedded as per Observation 1. For convenience, we will refer to this embedding as the *book embedding* of G and the line on which the vertices lie as the *spine*. We can assume without loss of generality that δ is aligned with the spine. Let $v_1, v_2, \ldots, v_n$ be the vertices in V as they occur along the direction of δ in the book embedding. We relate the mapping γ to this embedding of G using order-theoretic concepts.

Definition 2. *Let $\prec$ be a partial order over V such that $v_a \prec v_b$ if and only if $a \leq b$ and $\delta \cdot \gamma(v_a) \leq \delta \cdot \gamma(v_b)$. Similarly, let $\succ$ be a partial order over V such that $v_a \succ v_b$ if and only if $a \leq b$ and $\delta \cdot \gamma(v_a) \geq \delta \cdot \gamma(v_b)$.*

Thus, a *chain* with respect to $\prec$ is a set of vertices that occur along δ in the same order in both the book embedding of G and under the mapping γ. On the other hand, the vertices in a chain with respect to $\succ$ occur in the reversed order in the book embedding of G from their order under γ. Using this notation, we can state the effect of our drawing technique as follows.

Lemma 3. *Suppose that V is partitioned into $V_1, V_2, \ldots, V_r$ so that $v_a \in V_i$ and $v_b \in V_j$ satisfy $\delta \cdot \gamma(v_a) < \delta \cdot \gamma(v_b)$ if $i < j$. Then, if V_i forms a chain with respect to $\prec$ when i is odd and a chain with respect to $\succ$ when i is even, we can embed G in the plane with the vertex mapping γ using at most $3r + O(1)$ bends per edge.*

Proof. Without loss of generality, we can assume δ is directed horizontally. Thus, we can assume that

1. $v_1, v_2, \ldots, v_n$ are the vertices in G in the order they are mapped from left to right in the book embedding,
2. $V_1, V_2, \ldots, V_r$ map under γ to the point sets $P_1, P_2 \ldots, P_r$, respectively, such that all points in P_i occur left of all points in P_{i+1}, for $i = 1, 2, \ldots, r-1$,
3. for odd i, the vertices in V_i map to points in P_i with the same relative left-to-right order as they occur along the spine of the book embedding,
4. for even i, the vertices in V_i map to points in P_i with the reverse relative left-to-right order as they occur along the spine of the book embedding.

Thus, we can think of the vertex sets $V_1, V_2, \ldots, V_r$ as mapping to disjoint intervals $\Delta_1, \Delta_2, \ldots, \Delta_r$ along the x-axis, each (strictly) containing the points $P_1, P_2, \ldots, P_r$ respectively. See Fig. 2 for an example of such a configuration. We will return to this idea to show how to partially embed the edges in G inside each interval, but before doing so, we first introduce some terminology.

All points in the book embedding of G that intersect with the spine either correspond to a vertex in G or a point at which an edge crosses the spine. Let $\Gamma_1, \Gamma_2, \ldots, \Gamma_h$ correspond to these points in the order they occur along the spine from left to right. If Γ_i corresponds to a vertex v, then we define $\mathbf{vertex}(\Gamma_i) = v$. Furthermore, we define $\mathbf{top}(\Gamma_i)$ to be the set of edges incident to v that were embedded on the top page and $\mathbf{bottom}(\Gamma_i)$ to be those embedded on the bottom

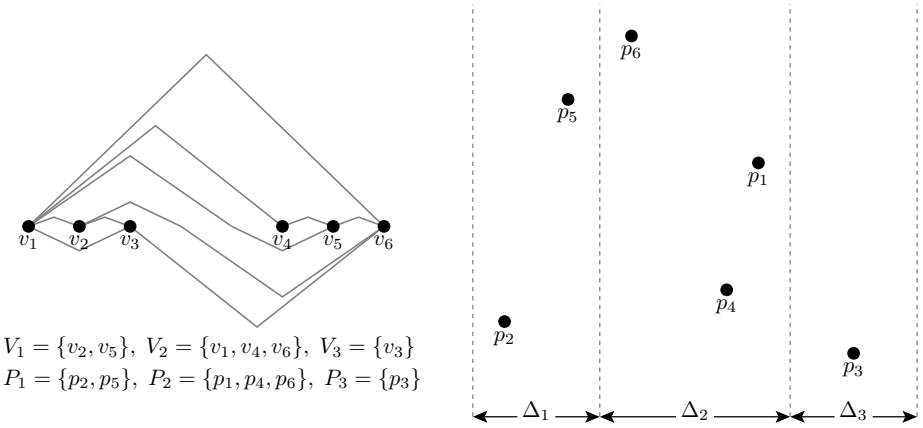


Fig. 2. An example configuration for a graph on 6 vertices

page. If Γ_i corresponded to a point at which edges crossed the spine, then Γ_i unambiguously refers to this edge.

We now describe how to draw the edges in G inside each of the intervals $\Delta_1, \Delta_2, \ldots \Delta_r$. We first consider an interval Δ_i, for which i is odd, with corresponding vertex set V_i and point set P_i. For $t = 1, 2, \ldots, h$, we will draw a set of vertical lines corresponding to Γ_t as follows (an example of the construction is shown in Fig. 4).

1. If $\mathbf{vertex}(\Gamma_t) = v \in V_i$, then we draw a vertical line above the point $\gamma(v)$ for each edge in $\mathbf{top}(\Gamma_t)$ and a vertical line below the point $\gamma(v)$ for each edge in $\mathbf{bottom}(\Gamma_t)$. We then join the ends of these vertical lines to $\gamma(v)$ as is shown in Fig. 3.
2. If $\mathbf{vertex}(\Gamma_t) = v \in V_j$, where $j > i$, then we draw a vertical line somewhere in the interval Δ_i for each edge in $\mathbf{top}(\Gamma_t)$.
3. If $\mathbf{vertex}(\Gamma_t) = v \in V_j$, where $j < i$, then we draw a vertical line somewhere in the interval Δ_i for each edge in $\mathbf{bottom}(\Gamma_t)$.
4. If Γ_t did not correspond to a vertex, we draw a vertical line for its unique edge.
5. All lines drawn for Γ_t occur left of all lines drawn for Γ_{t+1}.

Note that the final condition enforces that the vertical lines are separated into intervals, the first corresponding to edges from Γ_1, the second to edges from Γ_2, and so forth. It is clear that the first four conditions can easily be achieved. The last condition follows by the restriction on the left-to-right order of the points in P_i. Indeed, the vertices in V_i are mapped to points in P_i that occur from left to right in the same order as the points on the spine of the book embedding that defined $\Gamma_1, \Gamma_2, \ldots, \Gamma_h$.

For an interval Δ_i, for which i is even, the procedure is symmetric. The only difference is that the last condition is reversed so that the lines are drawn corresponding to those from Γ_h first, then those from Γ_{h-1}, and so forth. It

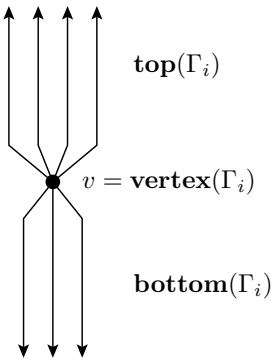


Fig. 3. A demonstration of how we join the edges in $\mathbf{top}(\Gamma_i)$ and $\mathbf{bottom}(\Gamma_i)$ to the vertex $\mathbf{vertex}(\Gamma_i)$

follows by the symmetric definition of V_i, for even i, why this construction can be achieved.

Thus, we can repeat the above process for each Δ-*interval* $\Delta_1, \Delta_2, \ldots, \Delta_r$. After this procedure, each Δ-interval will have a set of lines extending upwards and a set of lines extending downwards (some of which may correspond to the same edge). When i is odd, we say that the lines extending upward in Δ_i are *entering* Δ_i and those extending downward are *leaving*. When i is even, the definitions are reversed. We make a few observations about the configuration of these lines.

1. When i is odd, the lines in Δ_i define subintervals along the x-axis that correspond to $\Gamma_1, \Gamma_2, \ldots, \Gamma_h$ from left to right.
2. When i is even, the lines in Δ_i define subintervals along the x-axis that correspond to $\Gamma_h, \Gamma_{h-1}, \ldots, \Gamma_1$ from left to right.
3. For contiguous intervals Δ_i and Δ_{i+1}, the lines leaving Δ_i for a particular Γ_t correspond to the same set of edges as the lines entering Δ_{i+1} for Γ_t.

The last observation follows by construction. Clearly it holds for any Γ_t that corresponded to an edge crossing the spine. Suppose instead that $v = \mathbf{vertex}(\Gamma_t)$. If the lines leaving Δ_i corresponded to the edges in $\mathbf{top}(\Gamma_t)$, then $v \notin V_j$ for all $j \leq i$, implying that the lines entering Δ_{i+1} for Γ_t also correspond to $\mathbf{top}(\Gamma_t)$. On the other hand, if the lines leaving Δ_i corresponded to the edges in $\mathbf{bottom}(\Gamma_t)$, then $v \in V_j$, for some $j \leq i$, implying that the lines entering Δ_{i+1} for Γ_t also correspond to $\mathbf{bottom}(\Gamma_t)$.

We proceed to show how to join the vertical lines from contiguous Δ-intervals. Let B be an axis-aligned box containing all points in P (where P is the image of V under γ). For even i, suppose we were to rotate all vertical lines in the interval Δ_i clockwise by a small angle ϵ so that the lines remain parallel and only leave the interval Δ_i outside of the box B. Eventually, the lines drawn for each Γ_t in the interval Δ_i would intersect with the vertical lines drawn for Γ_t in the interval Δ_{i-1}. We can then terminate the lines drawn for Δ_{i-1} and Δ_i

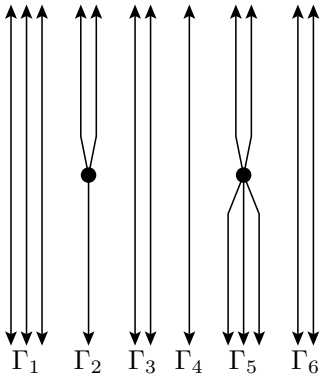


Fig. 4. An example of the vertical lines drawn for an interval Δ_i

at these intersection points, hence joining the lines drawn for Γ_t in Δ_{i-1} and Δ_i. Similarly, if we consider the intersection between the lines extending from Δ_i with the vertical lines extending upward from Δ_{i+1} (assuming Δ_{i+1} exists), we can again terminate these lines at the points the lines from a common Γ_t intersect. See Fig. 5 for an example of this procedure.

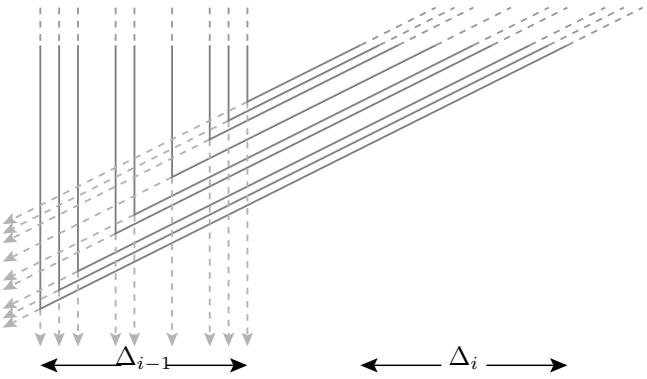


Fig. 5. An example of how we join the lines for an interval Δ_{i-1} with an interval Δ_i. Note that the lines need not be drawn identically spaced in both intervals.

By the previous observation, this procedure therefore joins the lines leaving Δ_{i-1} to those entering Δ_i and joins the lines leaving Δ_i to those entering Δ_{i+1}. Moreover, since the lines in each odd-indexed Δ-interval are left unrotated, we can repeat this procedure for each even-indexed Δ-interval. That is, for each Γ_t with $\mathbf{vertex}(\Gamma_t) = v$, the edges in $\mathbf{top}(\Gamma_t)$ are drawn from $\gamma(v)$ to a set of vertical lines entering Δ_1 in the same left-to-right order as these edges were drawn incident to v on the top page of the book embedding. Similarly, the edges

in **bottom**(Γ_t) are drawn from $\gamma(v)$ to a set of lines leaving Δ_r in the same left-to-right order as these edges were drawn incident to v on the bottom page. Furthermore, a line is drawn entering Δ_1 and leaving Δ_r for each edge that had crossed the spine in the book embedding.

To complete the desired embedding, we consider the vertical lines entering Δ_1 and the lines leaving Δ_r. The vertical lines entering Δ_1 correspond to the ends of the edges drawn on the top page of the book embedding (either where they are incident to a vertex or where they cross the spine). To join the two vertical lines corresponding to the same edge, we can use the embedding of the top page of the book embedding. The procedure is as follows. First, truncate the vertical lines at some common y-coordinate. Then, draw the top page of the book embedding above the vertical lines, excluding the region within some small distance ϵ from the spine (truncating the edges before they meet their incident vertices). We can then trivially connect the ends of the vertical lines to the ends of the truncated edges from the top page since they occur from left to right in the same order. See Fig 6 for a depiction of this procedure.

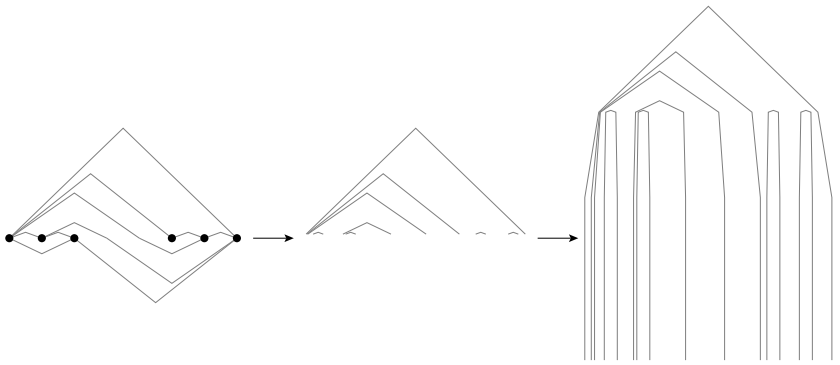


Fig. 6. An example of how we join the vertical lines entering Δ_1 by using the top page of the book embedding

To join the edges leaving Δ_r with each other, we consider two cases. If r is odd, then the lines leave downwards, and we can connect them using the bottom page in the same manner as we did with the lines entering Δ_1. If r is even, then the lines leave upwards and we can again join them by using the bottom page by simply rotating it to face the opposite direction.

We now bound the number of times an edge bends in the embedding. Recall that for each of the $\Gamma_1, \Gamma_2, \ldots, \Gamma_h$ we drew a set of piecewise linear curves through the intervals $\Delta_1, \Delta_2, \ldots, \Delta_r$. Each of these curves bent at most once for each of the intervals and $O(1)$ times where they connected to a vertex or joined another curve where they entered Δ_1 or left Δ_r. By using the embedding from Observation 1, it follows that each edge in G is associated with at most three of $\Gamma_1, \Gamma_2, \ldots, \Gamma_h$. Thus, it follows that each edge bends at most $3r+O(1)$ times. □

4 Drawing a Planar Graph with a Fixed Vertex Mapping

In Section 3, we established a technique for drawing a graph $G = (V, E)$ with a fixed vertex mapping γ, where the number of bends is proportional to the size of a partition $V_1, V_2, \ldots, V_r$ of V satisfying the conditions of Lemma 3. In this section, we discuss how to construct such a vertex partition for an arbitrary fixed vertex mapping.

Recall the definition of $\prec$ and $\succ$ from Definition 2. Clearly, any singleton set forms a chain with respect to both $\prec$ and $\succ$. Thus, by Lemma 3, we can use the partition $V_1 = \{v_1\}, V_2 = \{v_2\}, \ldots, V_n = \{v_n\}$ to embed G with any vertex mapping using $3n + O(1)$ bends per edge. This bound's constant factor matches the best known result of Badent et al. described in [2]. Using average-case analysis, we can improve the bound.

A *uniformly random planar graph* on n vertices is a graph sampled uniformly at random from the set of all planar graphs over the vertex set $V = \{1, 2, \ldots, n\}$. We consider two isomorphic graphs to be different if their vertex labelings differ. Suppose that we constructed a book embedding of such a graph as per Observation 1. If we do so in a manner that is independent from the labeling of the vertices, then we can assume that the vertices occur along the spine of the book embedding in a uniformly random order. To enforce independence, one could simply relabel the vertices uniformly at random before constructing the book embedding, reverting to the original labeling afterwards. Hence, we can make the following observation.

Observation 4. *A uniformly random planar graph G can be embedded in the plane so that*

1. *all vertices lie on a common line in a uniformly random order,*
2. *each edge bends at most once above the line, at most once below the line, and at most once on the line.*

By Observation 4, our analysis on random planar graphs reduces to an analysis of random permutations. The proof of the next theorem delineates this point.

Theorem 5. *A uniformly random planar graph $G = (V, E)$ can be embedded in the plane with a fixed vertex mapping using at most $2n + O(1)$ bends per edge on expectation.*

Proof. Let γ be a fixed vertex mapping, and let δ be a direction for which $\delta \cdot \gamma(u) = \delta \cdot \gamma(v)$, for $u, v \in V$, only if $u = v$. Let $p_1, p_2, \ldots, p_n$ be the points in the image of V under γ in the order they occur along δ. Define $v_1 = \gamma^{-1}(p_1), v_2 = \gamma^{-1}(p_2), \ldots, v_n = \gamma^{-1}(p_n)$.

Embed G as per Observation 4 so that the spine is aligned with the direction δ. For $i = 1, \ldots, n$, define $\alpha(v_i)$ to be the index along δ at which v_i occurs in this embedding. Thus, by Definition 2, $v_i \prec v_j$ if $i \leq j$ and $\alpha(v_i) \leq \alpha(v_j)$. Similarly, $v_i \succ v_j$ if $i \leq j$ and $\alpha(v_i) \geq \alpha(v_j)$.

By our choice of embedding, $\alpha(v_1), \alpha(v_2), \ldots, \alpha(v_n)$ is a uniformly random permutation of $1, 2, \ldots, n$. Construct a partition of V as follows. Let t_1 be the

largest index such that $\alpha(v_1), \alpha(v_2), \ldots, \alpha(v_{t_1})$ is increasing. Then, let t_2 be the largest index such that $\alpha(v_{t_1+1}), \alpha(v_{t_1+2}), \ldots, \alpha(v_{t_2})$ is decreasing. Repeat this process for $t = 3, \ldots, r$, maximizing increasing sequences when i is odd and decreasing sequences when i is even. The partition $V_1 = \{v_1, v_2, \ldots, v_{t_1}\}$, $V_2 = \{v_{t_1+1}, v_{t_1+2}, \ldots, v_{t_2}\}$, $\ldots$, $V_r = \{v_{t_{r-1}+1}, v_{t_{r-1}+2}, \ldots, v_{t_r}\}$ satisfies the conditions of Lemma 3 by construction. We can therefore construct the desired embedding of G so long as r is at most $\frac{2}{3}n + O(1)$.

Thus, to complete the proof we consider how large r is on average. Let X be the set of integers $1 < i < n$ for which $\alpha(v_{i-1}), \alpha(v_{i+1}) < \alpha(v_i)$ or $\alpha(v_{i-1}), \alpha(v_{i+1}) > \alpha(v_i)$. Clearly $r \leq |X| + 2$. Let $X_2, X_3, \ldots, X_{n-1}$ be indicator variables such that $X_i = 1$ if $i \in X$ and $X_i = 0$ otherwise. By linearity of expectation, it follows that

$$\mathbf{E}|X| = \sum_{i=1}^{n} \mathbf{E}X_i$$

and since $\mathbf{E}X_i = \mathbf{P}[X_i = 1] = 2/3$, it follows that $r \leq 2/3(n+1)$. □

5 Simultaneous Embeddings with a Fixed Vertex Range

In this section, we consider the problem of embedding k uniformly random planar graphs $G_1, G_1, \ldots, G_k$ over a common vertex set V, where the range of the vertex mapping γ is restricted to a fixed point set P of size $n = |V|$. As in the proof of Theorem 5, our analysis relies on properties of uniformly random permutations.

Lemma 6 ([7]). *Let $\pi_1, \pi_2, \ldots, \pi_k$ be uniformly random permutations over the set $S = \{1, 2, \ldots, n\}$. Then, there exists a partition $T_1, T_2, \ldots, T_r$ of S, where the elements in each part form an increasing subsequence in each of $\pi_1, \pi_2, \ldots, \pi_k$, such that r is $O(n^{1-\frac{1}{k+1}})$ with overwhelming probability.*

This bound was established by Brightwell in [7]. The following result follows by combining this bound with Lemma 3.

Theorem 7. *If $G_1, G_2, \ldots, G_k$ are uniformly random planar graphs, then we can embed each graph in the plane with a common vertex mapping $\gamma : V \to P$ so that all edges have $O(n^{1-\frac{1}{k}})$ bends each with overwhelming probability.*

Proof. Let δ be a direction for which $\delta \cdot p = \delta \cdot q$, for $p, q \in P$, only if $p = q$. Let $p_1, p_2, \ldots, p_n$ be the points in P in the order they occur along δ. Embed G_1 as per Observation 4 so that its spine is aligned with δ. Let $v_1, v_2, \ldots, v_n$ be the vertices in V in the order they occur along δ in this embedding. Embedding $G_2, \ldots, G_k$ in the same manner gives the corresponding vertex orders $\pi_2, \pi_3, \ldots, \pi_n$, where π_i is a uniformly random permutation of $v_1, v_2, \ldots, v_n$. By Lemma 6, it follows that V can be partitioned into $V_1, V_2, \ldots, V_r$ such that the vertices in V_i occur along δ in the same order in the embeddings of each of $G_1, G_2, \ldots, G_k$. Furthermore, r is $O(n^{1-\frac{1}{k}})$ with overwhelming probability.

Let $\{u_1, \ldots, u_{t_1}\} = V_1, \{u_{t_1+1}, \ldots, u_{t_2}\} = V_2, \ldots, \{u_{t_{r-1}+1}, \ldots, u_n\} = V_r$ such that u_i occurs before u_{i+1} along δ in the embedding of G_1, for all i, if $u_i, u_{i+1} \in V_j$

for some j. Consider the vertex mapping γ, defined such that $\gamma(u_1) = p_1, \gamma(u_2) = p_2, \gamma(u_n) = p_n$. By construction, each V_i forms a chain with respect to $\prec$ from Definition 2. Since Lemma 3 requires a partition that alternates between chains with respect to $\prec$ and $\succ$, we can introduce empty sets into our partition after each V_i, at most doubling its size. This extensions suffices since the empty set forms a chain with respect to both $\prec$ and $\succ$. Moreover, since r is $O(n^{1-\frac{1}{k}})$, so is this extension, and thus the claim follows by Lemma 3. □

6 Lower Bounds on the Number of Bends

In this section we prove that Theorem 7 is optimal by using an encoding argument. The technique relies on the following lemma.

Lemma 8. *If the planar graph G can be drawn in the plane with β total bends under a fixed vertex mapping that maps V to a convex point set, then G can be encoded using $n \lg \left(\frac{\beta+n}{n}\right) + O(n)$ bits.*

The proof of Lemma 8 is lengthy and is thus deferred to the full version of this paper [12]. The following theorem follows from Lemma 8 and information theoretic bounds on encodings of planar graphs.

Theorem 9. *Let $G_1, G_2, \ldots, G_k$ be uniformly random planar graphs over the vertex set V, and let P be a convex point set of size $|V| = n$. Then, in all simultaneous embeddings of $G_1, G_2, \ldots, G_k$ that map V to P, at least one of $G_1, G_2, \ldots, G_k$ has $\Omega(2^{1-\frac{1}{k}})$ total bends with overwhelming probability.*

Proof. Suppose that $G_1, G_2, \ldots, G_k$ can be drawn on P with β total bends for some vertex mapping γ. Since there are $n!$ possible vertex mappings, γ can be encoded using $\lg n!$ bits. Thus, $G_1, G_2, \ldots, G_k$ can be encoded using

$$kn \lg \left(\frac{\beta+n}{n}\right) + O(kn) + \lg n!$$

bits by Lemma 8. Since there are more than $n!$ planar graphs on n vertices, it follows that at least $\lg n! - \Delta$ bits are required to encode a uniformly random planar graph with probability at least $1 - 2^{-\Delta}$. It follows that

$$kn \lg \left(\frac{\beta+n}{n}\right) + O(kn) + \lg n! \geq k \lg n! - \Delta$$

with probability at least $1 - 2^{-\Delta}$. Thus, there exists a constant c for which

$$kn \lg \left(\frac{(\beta+n)2^{c+\frac{\Delta}{kn}} n^{\frac{1}{k}}}{n}\right) \geq kn \lg n$$

with probability at least $1 - 2^{-\Delta}$ by Stirling's approximation. Dividing a factor of kn and exponentiating both sides shows that the inequality

$$\frac{(\beta+n)2^{c+\frac{\Delta}{kn}}n^{\frac{1}{k}}}{n} \geq n$$

or equivalently,

$$\beta \geq \frac{n^{2-\frac{1}{n}}}{2^{c+\frac{\Delta}{kn}}} - n$$

holds with probability at least $1-2^{-\Delta}$. □

References

1. Aggarwal, A., Klawe, M., Shor, P.: Multilayer grid embeddings for VLSI. Algorithmica 6(1), 129–151 (1991)
2. Badent, M., Di Giacomo, E., Liotta, G.: Drawing colored graphs on colored points. In: Algorithms and Data Structures, pp. 102–113 (2007)
3. Bernhart, F., Kainen, P.: The book thickness of a graph. Journal of Combinatorial Theory, Series B 27(3), 320–331 (1979)
4. Biedl, T., Kant, G., Kaufmann, M.: On triangulating planar graphs under the four-connectivity constraint. Algorithmica 19(4), 427–446 (1997)
5. Biggs, N., Lloyd, E., Wilson, R.: Graph Theory, 1736–1936. Clarendon Press (1986)
6. Braß, P., Cenek, E., Duncan, C.A., Efrat, A., Erten, C., Ismailescu, D., Kobourov, S.G., Lubiw, A., Mitchell, J.S.B.: On Simultaneous Planar Graph Embeddings. In: Dehne, F., Sack, J.-R., Smid, M. (eds.) WADS 2003. LNCS, vol. 2748, pp. 243–255. Springer, Heidelberg (2003)
7. Brightwell, G.: Random k-dimensional orders: Width and number of linear extensions. Order 9(4), 333–342 (1992)
8. Chiba, N., Nishizeki, T.: The Hamiltonian cycle problem is linear-time solvable for 4-connected planar graphs. J. Algorithms 10(2), 187–211 (1989)
9. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall PTR, Upper Saddle River (1998)
10. Erten, C., Kobourov, S.G.: Simultaneous Embedding of Planar Graphs with Few Bends. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 195–205. Springer, Heidelberg (2005)
11. Fáry, I.: On straight-line representation of planar graphs. Acta Sci. Math. (Szeged) 11, 229–233 (1948)
12. Gordon, T.: Simultaneous embeddings with vertices mapping to pre-specified points. Arxiv preprint (2012)
13. Kaufmann, M., Wiese, R.: Embedding vertices at points: Few bends suffice for planar graphs. J. Graph Algorithms Appl. 6(1), 115–129 (2002)
14. Mead, C., Conway, L.: Introduction to VLSI Systems. Addison-Wesley Longman Publishing Co., Inc., Boston (1979)
15. Pach, J., Wenger, R.: Embedding Planar Graphs at Fixed Vertex Locations. In: Whitesides, S.H. (ed.) GD 1998. LNCS, vol. 1547, pp. 263–274. Springer, Heidelberg (1999)
16. Schnyder, W.: Embedding planar graphs on the grid. In: SODA 1990: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, pp. 138–148 (1990)
17. Stein, S.K.: Convex maps. In: Proceedings of the American Mathematical Society, pp. 464–466 (1951)
18. Tutte, W.T.: How to draw a graph. Proc. London Math. Soc. 13, 743–768 (1963)
19. Wagner, K.: Bemerkungen zum Vierfarbenproblem. Jahresbericht. German. Math.-Verein. 46, 26–32 (1936)

Multilevel Drawings of Clustered Graphs

Fabrizio Frati

School of Information Technologies – The University of Sydney
brillo@it.usyd.edu.au

Abstract. The *cluster adjacency graph* of a flat clustered graph $C(G,T)$ is the graph A whose vertices are the clusters in T and whose edges connect clusters containing vertices that are adjacent in G. A *multilevel drawing* of a clustered graph C consists of a straight-line c-planar drawing of C in which the clusters are drawn as convex regions and of a straight-line planar drawing of A such that each vertex $a \in A$ is drawn in the cluster corresponding to a and such that no edge $(a_1, a_2) \in A$ intersects any cluster different from a_1 and a_2. In this paper, we show that every c-planar flat clustered graph admits a multilevel drawing.

1 Introduction

A *clustered graph* is a pair $C(G,T)$, where G is a graph, called *underlying graph*, and T is a rooted tree, called *inclusion tree*, whose leaves are the vertices of G. Each internal node ν of T corresponds to the subset of vertices of G, called *cluster*, that are the leaves of the subtree of T rooted at ν. Throughout the paper, we assume that each path from the root of T to any leaf has the same number of edges, which is denoted by $h(T)$. We call *level* of a cluster μ the minimum number of edges in a path in T from μ to a leaf. Given a clustered graph $C(G,T)$, the *cluster adjacency graph at level i* is the graph A_i whose vertices are the clusters at level i and having an edge between two clusters μ and ν if any vertex in μ and any vertex in ν are connected by an edge of G. A clustered graph is *flat* if the height of T is at most two, i.e., no cluster different from the root contains other clusters. In a flat clustered graph $C(G,T)$, we say that the *cluster of* a vertex v of G is its parent in T and we denote it by $\mu(v)$, we call *clusters* only the children of the root, and we call *adjacency graph* A the adjacency graph at level one.

Clustered graphs find applications in several areas of computer science and hence they have been widely studied from a theoretical point of view. Several methods have been developed to compute a *good clustering* for a given graph G, that is, for constructing a (usually flat) clustered graph that has G as underlying graph and that has high edge density inside each cluster and few edges connecting vertices belonging to different clusters. See [13] for a survey on graph clustering. From a graph drawing perspective, the clustering is given as part of the problem's input, and the goal is to visualize the clustered graph in a *readable way*.

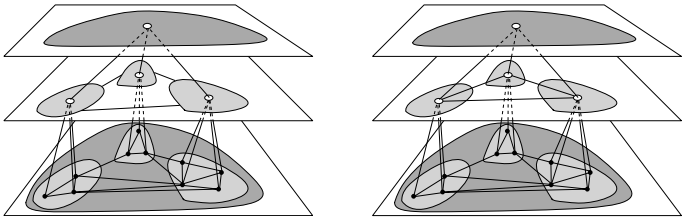
Clustered planarity (c-planarity for short) is a concept, introduced in [9], that generalizes planarity to clustered graphs and that has been recognized as *the* standard for readability of clustered graph drawings. A drawing of a clustered graph represents each cluster as a closed region of the plane containing all the vertices of the cluster; a drawing is *c-planar* if it contains no edge crossings (i.e., the drawing of the underlying graph

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 311–322, 2012.

is planar), no edge-region crossings (i.e., no edge intersects the border of a cluster more than once), and no region-region crossings (i.e., each two regions representing clusters are disjoint). A graph is c-planar if it admits a c-planar drawing. Designing an algorithm to test whether a clustered graph is c-planar (or prove that no efficient algorithm exists) is one of the most studied graph drawing problems. See [3,10] for two recent papers on this topic. Assuming that a clustered graph C is c-planar, algorithms and bounds are known for constructing c-planar drawings of C [2,6,7,12]. A particular attention has been devoted to *straight-line convex drawings*, that are c-planar drawings requiring edges to be straight-line segments and clusters to be convex regions. Every c-planar graph admits a straight-line convex drawing [5], even if the shape of each cluster is fixed in advance [1]. Straight-line convex drawings might require exponential area [8].

A *multilevel drawing* of a clustered graph $C(G,T)$ is a three-dimensional representation of C consisting of a straight-line convex drawing of C in the plane $z = 0$, of a planar drawing of the adjacency graph A_i at level i on the plane $z = i$, and of a straight-line planar drawing of T such that clusters at level i appear on the plane $z = i$. Multilevel drawings were introduced in [4], where it is shown how to obtain such representations by first computing a straight-line convex drawing of C in the plane $z = 0$, and by then vertically translating each cluster μ at level i to the plane $z = i$. The inclusion tree is visualized by choosing a point inside each cluster μ at level i and by connecting such a point to the points chosen for all the clusters at level $i-1$ (if $i > 1$) or to all the vertices that belong to μ (if $i = 1$). The adjacency among two clusters μ and ν at level i is represented by choosing any edge that connects a vertex in μ and a vertex in ν and by translating it to the plane $z = i$. The c-planarity of the drawing of C implies the planarity of the drawing of T and the planarity of the drawing of A_i. See Fig. 1(a) for an example of multilevel drawing.

In this paper we consider multilevel drawings in which the adjacency graph at level i is represented by a straight-line planar drawing. That is, for each $i \geq 1$, a cluster μ at level i is a point in the plane $z = i$ lying inside the region obtained by vertically translating cluster μ from the plane $z = 0$ to the plane $z = i$. We require the drawing of graph A_i to be straight-line and planar, and we require each edge (a,b) of A_i not to intersect the region obtained by vertically translating the cluster ρ from the plane $z = 0$ to the plane $z = i$, for each cluster ρ not containing a nor b. See Fig. 1(b) for an example of such a drawing. Observe that in our setting each cluster is represented in



(a) (b)

Fig. 1. (a) A multilevel drawing, as in [4]. (b) A multilevel drawing, as in this paper.

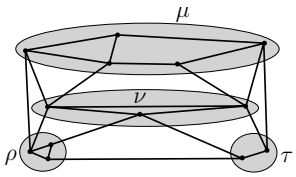


Fig. 2. A convex straight-line drawing Γ of a flat clustered graph such that, in every straight-line drawing of the cluster adjacency graph A, an edge of A intersects the border of a cluster twice

the drawing of A_i by a point in the plane $z = i$, while in the setting studied in [4], each cluster is represented in the drawing of A_i by a convex region in the plane $z = i$. We now formally define the multilevel drawings considered in this paper.

Definition 1. *A* multilevel drawing *of a clustered graph $C(G,T)$ consists of a convex straight-line drawing Γ of C and, for each $1 \leq i \leq h(T)$, of a straight-line planar drawing $\Gamma(A_i)$ of A_i, such that: (1) the point $a_i(\mu)$ representing a cluster μ in $\Gamma(A_i)$ is inside the region representing μ in Γ, for each $1 \leq i \leq h(T)$; and (2) each edge $(a_i(\mu), a_i(\nu))$ in $\Gamma(A_i)$ does not intersect the border of any cluster ρ different from μ and ν.*

Our main result is the following:

Theorem 1. *Every c-planar flat clustered graph admits a multilevel drawing.*

Not every convex straight-line drawing of a flat clustered graph allows for representing the adjacency graph A as required in Definition 1. Figure 2 shows a convex straight-line drawing such that, in any straight-line drawing of A, either the edge $(a(\mu), a(\rho))$ or the edge $(a(\mu), a(\tau))$ of A intersects the border of cluster ν, where $a(\alpha)$ denotes the vertex representing cluster α in A.

2 The Main Theorem

In this section we state our main theorem, which directly implies Theorem 1. First, it suffices to restrict the attention to *maximal* c-planar flat clustered graphs, that is, to c-planar flat clustered graphs $C(G,T)$ such that G is a maximal planar graph. Indeed, if C is not maximal, then it can be augmented to a maximal clustered graph C' by adding dummy edges without loosing c-planarity [11]. Then, a multilevel drawing of C' can be constructed and the inserted dummy edges can be deleted thus obtaining a multilevel drawing of C. In the following, all the considered clustered graphs are flat and maximal, even when not explicitly stated. We will assume that each clustered graph $C(G,T)$ is associated with a c-planar embedding that determines the faces of G. Moreover, we will denote a clustered graph also by $C(G,T,A)$, where A is the cluster adjacency graph of $C(G,T)$. The *outer face* of $C(G,T,A)$ is the clustered graph $C_o(G_o,T_o,A_o)$ such that G_o is the cycle delimiting the outer face of G, T_o is T restricted to the clusters that contain vertices of G_o, and A_o is the cluster adjacency graph of $C_o(G_o,T_o)$.

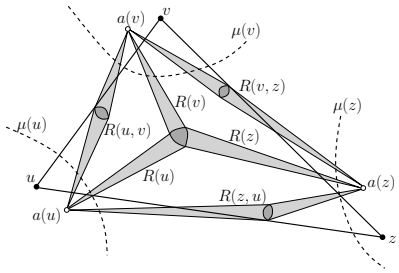


Fig. 3. Extension regions for a face $f = (u, v, z)$, if $\mu(u)$, $\mu(v)$, and $\mu(z)$ are distinct. The darker regions inside $R(u, v)$, $R(v, z)$, $R(z, u)$, and $R(u) \cup R(v) \cup R(z)$ are $S(u, v)$, $S(v, z)$, $S(z, u)$, and $S(u, v, z)$, respectively.

We introduce some geometric concepts. We denote by $CH(p, R)$ the convex hull of a convex region R and a point p, and by $R_1 \cup R_2$ the union of two convex regions R_1 and R_2. Given a triangle (u, v, z), we define a *side region* $S(u, v)$ as a convex region that intersects $\overline{uv}$ in exactly one point and whose every other point is internal to (u, v, z) and a *central region* $S(u, v, z)$ as a convex region entirely internal to (u, v, z). Side and central regions are used to define *extension regions*. In the inductive algorithm we will present in the next section, extension regions are associated to a multilevel drawing Γ' of a subgraph $C'(G', T', A')$ of the clustered graph $C(G, T, A)$ to be drawn; in the drawing Γ of $C(G, T, A)$ constructed by the algorithm, the edges of A not in A' are entirely contained inside the extension regions associated with Γ'. The geometric properties of the extension regions detailed in the next definition ensure that the edges of A not in A' do not cross edges of A' or clusters of T'.

Definition 2. *Consider a multilevel drawing Γ of a flat clustered graph $C(G, T, A)$. Let $f = (u, v, z)$ be a face of G, where vertices u, v, and z appear in this clockwise order around f. The* extension regions for f *are defined as follows (see Fig. 3).*

- *If $\mu(u)$, $\mu(v)$, and $\mu(z)$ are all distinct, then let $S(u, v, z)$ be a central region and $S(u, v)$, $S(v, z)$, and $S(z, u)$ be side regions inside (u, v, z) such that:*
  - *$CH(u, S(u, v, z))$, $CH(u, S(u, v))$, and $CH(u, S(z, u))$ do not intersect each other and do not intersect any region representing a cluster, except for $\mu(u)$;*
  - *$CH(v, S(u, v, z))$, $CH(v, S(u, v))$, and $CH(v, S(v, z))$ do not intersect each other and do not intersect any region representing a cluster, except for $\mu(v)$;*
  - *$CH(z, S(u, v, z))$, $CH(z, S(v, z))$, and $CH(z, S(z, u))$ do not intersect each other and do not intersect any region representing a cluster, except for $\mu(z)$;*
  - *for every point $p \in S(u, v, z)$, segments $\overline{pa(u)}$, $\overline{pa(v)}$, and $\overline{pa(z)}$ are in this clockwise order around p.*

  Then, the extension regions for f are:
  1. $R(u, v) = CH(a(u), S(u, v)) \cup CH(a(v), S(u, v))$;
  2. $R(v, z) = CH(a(v), S(v, z)) \cup CH(a(z), S(v, z))$;
  3. $R(z, u) = CH(a(z), S(z, u)) \cup CH(a(u), S(z, u))$;
  4. $R(u) = CH(a(u), S(u, v, z))$;

5. $R(v) = CH(a(v), S(u, v, z))$; *and*
6. $R(z) = CH(a(z), S(u, v, z))$.

– *If* $\mu(u) = \mu(v) \neq \mu(z)$, *then let* $S(v, z)$ *and* $S(z, u)$ *be side regions inside* (u, v, z) *such that:*
  - $CH(v, S(v, z))$ *and* $CH(u, S(z, u))$ *do not intersect each other and do not intersect any region representing a cluster, except for* $\mu(u)$*;*
  - $CH(z, S(v, z))$ *and* $CH(z, S(z, u))$ *do not intersect each other and do not intersect any region representing a cluster, except for* $\mu(z)$.

  Then, the extension regions for f *are:*
  1. $R(v, z) = CH(a(u), S(v, z)) \cup CH(a(z), S(v, z))$; *and*
  2. $R(z, u) = CH(a(u), S(z, u)) \cup CH(a(z), S(z, u))$.

– *If* $\mu(u) = \mu(v) = \mu(z)$, *then* f *has no extension regions.*

The next definition deals with the relationship between extensible regions, edges of A, and clusters in T in a multilevel drawing of a clustered graph $C(G, T, A)$. The kind of drawings defined in the following are used throughout the remainder of the paper.

Definition 3. *A multilevel drawing of a flat clustered graph* $C(G, T, A)$ *is called* extensible *if, for each face* f *of* G*, extension regions for* f *can be drawn so that (see Fig. 4): (1) for each face* f *of* G*, no extension region for* f *intersects an edge of* A*, except on its border; (2) for each two faces* f_1 *and* f_2 *of* G*, where* $f_1 \neq f_2$*, no extension region for* f_1 *intersects an extension region for* f_2*, except on its border; (3) for each face* f *of* G*, no two distinct extension regions for* f *intersect, except on their borders, unless they both comprise a central region* $S(u, v, z)$*; (4) each extension region* $R(u, v)$ *for a face* f *of* G *does not intersect any cluster other than* $\mu(u)$ *and* $\mu(v)$*; and (5) each extension region* $R(u)$ *for a face* f *of* G *does not intersect any cluster other than* $\mu(u)$.

The algorithm presented in the next section constructs an extensible drawing of a clustered graph for an arbitrary extensible drawing of the outer face. Thus, we define the concept of *completing a drawing of the outer face*.

Definition 4. *An extensible drawing* Γ *of a flat clustered graph* $C(G, T, A)$ completes *an extensible drawing* Γ_o *of the outer face* $C_o(G_o, T_o, A_o)$ *of* C *if* Γ *coincides with* Γ_o *when restricted to the vertices and edges of* G_o*, to the clusters in* T_o*, and to the vertices and edges of* A_o*, if all the vertices and edges of* A/A_o *lie in the extension regions of* Γ_o*, and if all the extension regions of* Γ *lie inside the extension regions of* Γ_o.

We are now ready to state our main theorem.

Theorem 2. *Let* C *be a flat clustered graph. Then, for every extensible drawing* Γ_o *of the outer face* C_o *of* C*, there exists an extensible drawing* Γ *of* C *that completes* Γ_o.

Observe that Theorem 2 implies Theorem 1, since an extensible drawing is a multilevel drawing. In the following two sections, we will prove a statement which is even stronger than the one in Theorem 2, namely that for every extensible drawing Γ_o of the outer face C_o of C, there exists an extensible drawing Γ of C that completes Γ_o, even if we assume that each cluster has to be represented by an arbitrary convex shape and if each vertex of

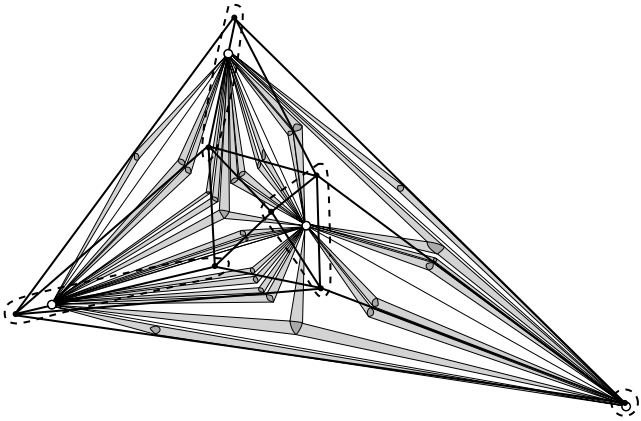


Fig. 4. An extensible drawing of a flat clustered graph $C(G, T, A)$. Vertices of G are black dots and edges of G are thick lines. The borders of the clusters are dashed lines. Vertices of A are white disks and edges of A are thin lines. The extension regions are gray.

A has to be mapped to the same point to which one of the vertices of the corresponding cluster is mapped to. However, both such conditions are not necessary for our inductive proof to work, hence we omitted them from the statement and we invite the reader to observe how the drawings we construct in the next two sections actually satisfy such conditions.

3 Base Cases

Our proof of Theorem 2 is by induction on the number of vertices. In this section we present the three base cases of such an induction. In Section 4 we will present the inductive cases. Denote by u, v, and z the vertices of G_o, where we assume w.l.o.g. that u, v, and z appear in this clockwise order along the outer face of G.

Base Case 1: *G has no internal vertices.* In this case, $C = C_o$ and the given extensible drawing Γ_o of C_o is an extensible drawing Γ of C that completes Γ_o.

Base Case 2: *G is K_4 and the only internal vertex x of G is such that $\mu(x)$ does not contain any vertex of G_o.* Refer to Fig. 5. Select any point p inside $S(u, v, z)$ (if $\mu(u)$, $\mu(v)$, and $\mu(z)$ are all different), or inside $S(v, z)$ (if $\mu(u) = \mu(v) \neq \mu(z)$). The cases in which $\mu(u) = \mu(z) \neq \mu(v)$ or $\mu(v) = \mu(z) \neq \mu(u)$ can be treated analogously to the case in which $\mu(u) = \mu(v) \neq \mu(z)$. Draw x at point p and draw $a(x)$ at point p. Connect x with u, v, and z, and connect $a(x)$ with $a(u)$, $a(v)$, and $a(z)$ ($a(u) = a(v)$ if $\mu(u) = \mu(v)$).

Next, we draw suitable extension regions. Denote by $f_1 = (u, v, x)$, $f_2 = (v, z, x)$, and $f_3 = (z, u, x)$ the three internal faces of G.

First, suppose that $\mu(u)$, $\mu(v)$, and $\mu(z)$ are all different. The extension regions $R(u, v)$, $R(v, z)$, and $R(z, u)$, for f_1, f_2, and f_3, respectively, have the same drawing as in Γ_o. Suppose that $a(v)$ is to the left of the half-line starting at x and passing

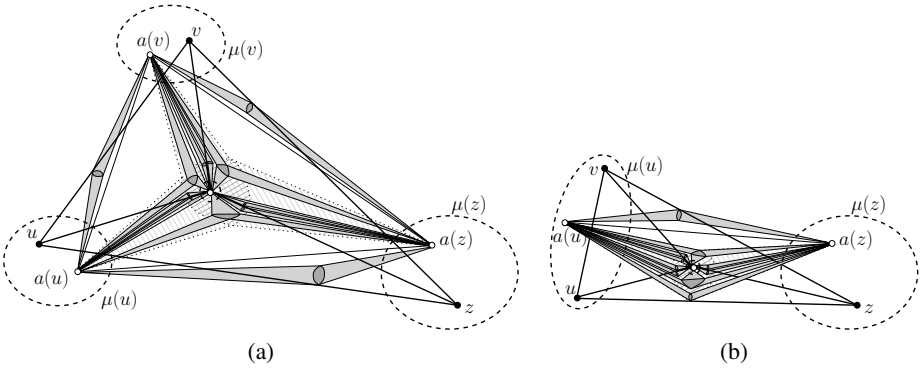


Fig. 5. Base Case 2, if (a) $\mu(u)$, $\mu(v)$, and $\mu(z)$ are all different, or if (b) $\mu(u) = \mu(v) \neq \mu(z)$

through v. The case in which $a(v)$ is to the right of the half-line starting at x and passing through v is analogous. Then, side region $S(v, x)$ for f_1 is drawn as a small region inside $S(u, v, z)$ touching (v, x) in a point p very close to x and side region $S(v, x)$ for f_2 is drawn as a small region inside $S(u, v, z)$ touching (v, x) in a point q very close to x such that q is farther from x than p. Observe that this implies that the extension regions $R(v, x)$ for f_1 and $R(v, x)$ for f_2 do not intersect. Extension regions $R(x, u)$ for f_1, $R(x, u)$ for f_3, $R(z, x)$ for f_2, and $R(z, x)$ for f_3 are drawn analogously. Central region $S(u, v, x)$ is also drawn as a small region inside $S(u, v, z)$; consider a point p such that p lies in the interior of $S(u, v, z)$, p lies inside f_1, and segments $\overline{pa(u)}$, $\overline{pa(v)}$, and $\overline{pa(x)}$ do not intersect any of the extension regions $R(v, x)$ and $R(x, u)$ for f_1. Observe that such a point always exists provided that side regions $S(v, x)$ and $S(x, u)$ for f_1 are small enough and sufficiently close to x. Then, $S(u, v, x)$ is any small convex region surrounding p. Such a construction implies that extension regions $R(u)$, $R(v)$, and $R(x)$ do not intersect extension regions $R(v, x)$ and $R(x, u)$ for f_1. Extension regions $R(v)$, $R(z)$, and $R(x)$ for f_2 and $R(z)$, $R(u)$, and $R(x)$ for f_3 are constructed analogously.

Second, suppose that $\mu(u) = \mu(v) \neq \mu(z)$. Extension region $R(v, z)$ for f_2 has the same drawing as in Γ_o. The construction of all other extension regions is the same as in the case in which $\mu(u)$, $\mu(v)$, and $\mu(z)$ are all different, with the only difference that the side and central regions that define the extension regions for f_1, f_2, and f_3 all lie inside the side region $S(z, u)$ (rather than inside the central region $S(u, v, z)$, which in this case does not exist) that defines the extension region $R(z, u)$ in the given extensible drawing Γ_o of C_o.

Observe that $\mu(u) = \mu(v) = \mu(z)$ can not happen, provided that $\mu(x)$ does not contain any vertex of G_o and that the embedding of C is c-planar.

It remains to draw cluster $\mu(x)$. Such a cluster is drawn as an arbitrary small region surrounding x. Denote by Γ the resulting drawing.

Lemma 1. *Γ is an extensible drawing of C completing Γ_o.*

Base Case 3: *G contains more than one internal vertex, does not contain any separating triangle, and does not contain any internal vertex u that is adjacent to a vertex v with $\mu(u) = \mu(v)$.*

Note that if u, v, and z and their incident edges are removed from G, the resulting graph G' is biconnected. Indeed, such a graph is connected because G is maximal and G_o is a cycle of three vertices. If G' is an edge (a, b), then, by the maximality of G, each of a and b is connected to the same two vertices of G_o, say u and v. Hence, either (u, v, a) or (u, v, b) is a separating triangle. If G' is not biconnected and has more than two vertices, then it contains a cut-vertex c. By the maximality of G, there exist two vertices of G_o, say u and v, that are both connected to c. Then, (u, v, c) is a separating triangle. It follows that G' is biconnected. Denote by $\mathcal{C}' = (uz, u_1, u_2, \ldots, u_U, uv, v_1, v_2, \ldots, v_V, vz, z_1, z_2, \ldots, z_Z)$ the cycle delimiting the outer face of G', where vertices $uz, u_1, u_2, \ldots, u_U, uv$, vertices $uv, v_1, v_2, \ldots, v_V$, vz, and vertices $vz, z_1, z_2, \ldots, z_Z, uz$ are neighbors of u, v, and z, respectively.

First, we draw $\mathcal{C}'$. Suppose that $\mu(u)$, $\mu(v)$, and $\mu(z)$ are distinct. Consider any point p in $S(u, v, z)$. Consider the wedge W_u with an angle smaller than $180°$ and delimited by the half-lines l_u and $l_{a(u)}$ centered at p and passing through u and $a(u)$, respectively. Suppose, w.l.o.g. up to a reflection of the drawing, that the clockwise rotation around p bringing l_u to coincide with $l_{a(u)}$ is smaller than $180°$. Let l_ϵ^+ (resp. l_ϵ^-) be the half-line starting at p obtained by clockwise rotating $l_{a(u)}$ by ϵ degrees (resp. by counterclockwise rotating l_u by ϵ degrees), for some arbitrarily small $\epsilon > 0$. Choose points p_{uv} on l_ϵ^+ and p_{uz} on l_ϵ^- arbitrarily close to p in such a way that segment $\overline{p_{uv}p_{uz}}$ crosses both segment $\overline{pu}$ and segment $\overline{pa(u)}$. Draw $\mathcal{C}'$ as a strictly-convex polygon such that uv is in p_{uv}, uz is in p_{uz}, vz is in p, the slopes of the edges of $\mathcal{C}'$ incident to u_i, for each $1 \leq i \leq U$ (resp. to v_i, for each $1 \leq i \leq V$, resp. to z_i, for each $1 \leq i \leq Z$), are arbitrarily close to the one of segment $\overline{p_{uv}p_{uz}}$ (resp. $\overline{p_{uv}p}$, resp. $\overline{p_{uz}p}$). Next, suppose that $\mu(u) = \mu(v) \neq \mu(z)$ (the cases in which $\mu(u) = \mu(z) \neq \mu(v)$ and $\mu(v) = \mu(z) \neq \mu(u)$ can be treated analogously). Consider any point p in $S(z, u)$. Consider any arbitrarily small segment $\overline{p_up_v}$ parallel to edge (u, v) and containing p. Draw $\mathcal{C}'$ as a strictly-convex polygon arbitrarily close to $\overline{p_up_v}$ such that the slope of every edge in $\mathcal{C}'$ is arbitrarily close to the one of $\overline{p_up_v}$, such that uz is mapped to p_u, and such that vz is mapped to p_v. Denote by $\Gamma_{\mathcal{C}'}$ the resulting drawing.

Second, we draw the vertices of G' not in $\mathcal{C}'$. Since $\Gamma_{\mathcal{C}'}$ is strictly-convex, a drawing $\Gamma_{G'}$ of G' having $\Gamma_{\mathcal{C}'}$ as outer face always exists (see, e.g., [14]). Let Γ_G be the straight-line drawing of G obtained by combining $\Gamma_{G'}$ and Γ_o.

Third, we draw the vertices of A different from $a(u)$, $a(v)$, and $a(z)$. Observe that, for any vertex x of G', there exists no vertex y in G such that $\mu(x) = \mu(y)$, by assumption and since every vertex of G' is an internal vertex of G. For each vertex x of G', draw $a(x)$ at the same point where x is drawn in Γ_G.

Fourth, we draw extension regions for the faces of G. Refer to Fig. 6. For each face $f = (x, y, t)$ of G such that $\{x, y, t\} \cap \{u, v, z\} = \emptyset$, side region $S(x, y)$ (resp. $S(y, t)$, resp. $S(t, x)$) for f is drawn as an arbitrarily small region inside f touching (x, y) (resp. (y, t), resp. (t, x)); central region $S(x, y, t)$ is drawn as follows: Consider a point p such that p lies inside f and segments $\overline{px}$, $\overline{py}$, and $\overline{pt}$ do not intersect any of the extension regions $R(x, y)$, $R(y, t)$, and $R(t, x)$ for f; such a point always exists provided that side

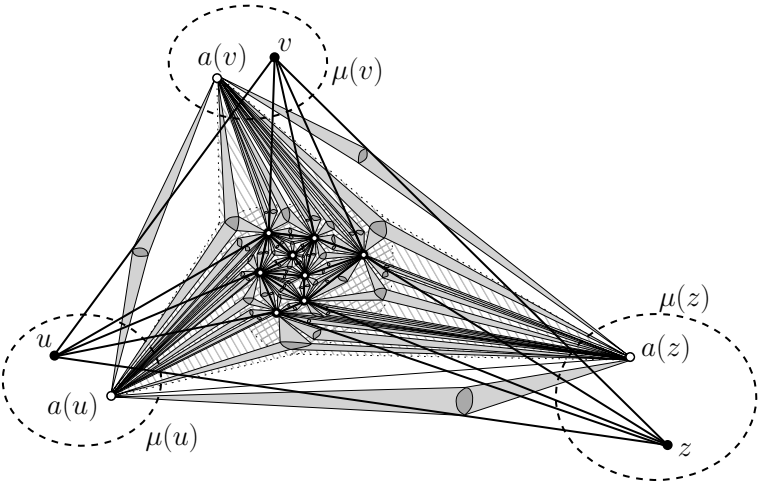


Fig. 6. Base Case 3, if $\mu(u)$, $\mu(v)$, and $\mu(z)$ are all different

regions $S(x, y)$, $S(y, t)$, and $S(t, x)$ are small enough; then, $S(x, y, t)$ is any arbitrary small convex region surrounding p. Next, we draw extension regions for the faces incident to u, v, and z. Consider any vertex x in $\mathcal{C}'$ adjacent to exactly one of u, v, and z, say to u. Let y and y' be the neighbors of x in $\mathcal{C}'$, where y, x, and y' appear in this clockwise order in $\mathcal{C}'$. Draw side regions $S(x, y)$ and $S(y', x)$ for faces (x, y, u) and (x, u, y') as arbitrarily small regions inside $S(u, v, z)$ (inside $S(z, u)$ if $\mu(u) = \mu(v)$), inside (x, y, u) and (x, u, y'), respectively, and touching (x, y) and (y', x), respectively. Suppose that $a(u)$ is to the left of the half-line starting at x and passing through u, the case in which it is to the right being analogous. Then, side region $S(u, x)$ for face (x, y, u) is drawn as a small region inside $S(u, v, z)$ (inside $S(z, u)$ if $\mu(u) = \mu(v)$), inside (x, y, u), and touching (u, x) in a point p very close to x. Side region $S(u, x)$ for face (x, u, y') is drawn as a small region inside $S(u, v, z)$ (inside $S(z, u)$ if $\mu(u) = \mu(v)$), inside (x, u, y'), and touching (u, x) in a point q very close to x such that q is farther from x than p. Observe that this implies that extension regions $R(u, x)$ for (x, y, u) and $R(u, x)$ for (x, u, y') do not intersect. Extension regions $R(y, u)$ for (x, y, u), $R(y, u)$ for the other face of G incident to edge (y, u), $R(u, y')$ for (x, u, y'), and $R(u, y')$ for the other face of G incident to edge (u, y') are drawn analogously. Central region $S(x, y, u)$ for (x, y, u) is drawn as follows: Consider a point p that lies inside $S(u, v, z)$ (inside $S(z, u)$ if $\mu(u) = \mu(v)$), inside (x, y, u), and such that segments $\overline{pa(u)}$, $\overline{pa(x)}$, and $\overline{pa(y)}$ do not intersect any of the extension regions $R(x, y)$, $R(y, u)$, and $R(u, x)$ for (x, y, u). Such a point always exists provided that side regions $S(x, y)$, $S(y, u)$, and $S(u, x)$ for (x, y, u) are small enough. Then, $S(u, v, x)$ is any arbitrarily small convex region surrounding p. Finally, we draw extension regions for the faces incident to two vertices out of u, v, and z as follows. Consider face (u, v, x). Extension regions $R(x, u)$ and $R(x, v)$ have been already drawn. Observe that, if $\mu(u) = \mu(v)$, no other region has to be drawn. Otherwise, extension region $R(u, v)$ has the same drawing as in Γ_o. Central region $S(u, v, x)$ is drawn as as follows: Consider a point p such that p lies

inside $S(u,v,z)$, inside (u,v,x), and such that segments $\overline{pa(u)}$, $\overline{pa(v)}$, and $\overline{px}$ do not intersect any of the extension regions $R(u,v)$, $R(v,x)$, and $R(x,u)$ for (u,v,x); such a point always exists provided that side regions $S(v,x)$ and $S(x,u)$ are small enough; then, $S(u,v,x)$ is any arbitrary small convex region surrounding p. The extension regions for the face incident to u and z and the extension regions for the face incident to v and z are drawn analogously.

Fifth, for each vertex x of G, we draw cluster $\mu(x)$ as an arbitrarily small convex region surrounding x. Denote by Γ the resulting drawing.

Lemma 2. *Γ is an extensible drawing of C completing Γ_o.*

4 Inductive Cases

In this section we present the inductive cases for the proof of Theorem 2.

Inductive Case 1: *G contains a separating triangle.* Suppose that G contains a separating triangle (u',v',z'). Denote by $C'_o(G'_o,T'_o,A'_o)$ the clustered graph such that G'_o is cycle (u',v',z'), T'_o is the subtree of T whose clusters contain at least one vertex of G'_o, and A'_o is the adjacency graph of $C'_o(G'_o,T'_o)$. Let $C^1(G^1,T^1,A^1)$ be the maximal flat clustered graph defined as follows. G^1 is the subgraph of G induced by all the vertices external to (u',v',z'), by u', by v', and by z'; T^1 is the subtree of T whose clusters contain at least one vertex of G^1; and A^1 is the adjacency graph of $C^1(G^1,T^1)$. Observe that C_o and C^1_o are the same graph. Further, let $C^2(G^2,T^2)$ be the clustered graph defined as follows. G^2 is the subgraph of G induced by all the vertices internal to (u',v',z'), by u', by v', and by z'. T^2 is the subtree of T whose clusters contain at least one vertex of G^2. A^2 is the adjacency graph of $C^2(G^2,T^2)$. Observe that C'_o and C^2_o are the same graph. Since (u',v',z') is a separating 3-cycle, the number of vertices of each of C^1 and C^2 is strictly less than the number of vertices of C. Hence, the inductive hypothesis applies and, for an arbitrary extensible drawing Γ_o of C^1_o, there exists an extensible drawing Γ^1 of C^1 completing Γ_o. Cycle (u',v',z') is a face f of G^1. By definition of extensible drawing, the drawing Γ^2_o of C^2_o in Γ^1 is an extensible drawing. Hence, the inductive hypothesis applies again and an extensible drawing Γ^2 of C^2 can be constructed completing Γ^2_o. Plugging Γ^2 into Γ^1 provides a drawing Γ of C.

Lemma 3. *Γ is an extensible drawing of C completing Γ_o.*

Inductive Case 2: *G contains an internal vertex u' that is adjacent to a vertex v' such that $\mu(u')=\mu(v')$.*

Refer to Fig. 7. Suppose that no separating triangle exists containing edge (u',v'), as otherwise Inductive Case 1 would apply. Since G is maximal, u' and v' have exactly two common neighbors z_1 and z_2, delimiting faces f_1 and f_2 with u' and v'. Contract edge (u',v') to a vertex w, that is, replace vertices u' and v' with a single vertex w that is connected to all the vertices u' and v' are connected to. Vertex w belongs to cluster $\mu(u')$. The resulting clustered graph $C'(G',T',A')$ is easily shown to be maximal, c-planar, and flat. In particular, the absence of separating triangles in G guarantees that G' is simple and maximal. Observe that C_o and C'_o are the same graph. Hence, the inductive hypothesis applies and, for an arbitrary extensible drawing Γ_o of C_o, there

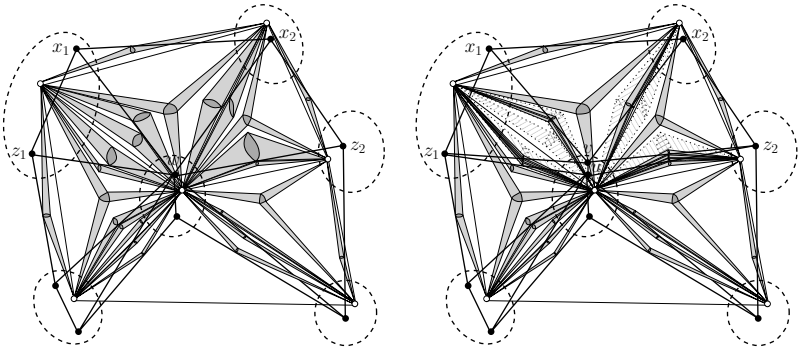


Fig. 7. (a) An extensible drawing Γ' of C'. (b) The extensible drawing Γ of C obtained from Γ'.

exists an extensible drawing Γ' of C' completing Γ_o. Then, consider a small disk D centered at w and consider any line l from w to an interior point of the segment between z_1 and z_2. Rename w to u' and insert v' on l, inside D, so that the order of the neighbors of u' in G is the required one. Connect u' and v' to their neighbors. It remains to show how to construct extension regions for the faces of G. The extension regions for each face not incident to v' are drawn as in Γ'. Denote by x_1 the common neighbor of v' and z_1 different from u' and by x_2 the common neighbor of v' and z_2 different from u'. Extension regions $R(u', z_1)$ and $R(z_1, v')$ for face (v', u', z_1) of G and extension region $R(z_1, v')$ for face (v', z_1, x_1) of G are drawn as subsets of the extension region $R(w, z_1)$ for the face (w, z_1, x_1) of G'. Extension regions $R(v', z_2)$ and $R(z_2, u')$ for face (v', u', z_2) of G and extension region $R(v', z_2)$ for face (v', z_2, x_2) of G are drawn as subsets of the extension region $R(w, z_2)$ for the face (w, z_2, x_2) of G'. For each neighbor x of v' different from z_1, z_2, and u', extension regions $R(v', x)$ for the two faces incident to edge (v', x) in G are drawn as a subset of the extension region $R(w, x)$ for one of the two faces incident to edge (w, x) in G'. All of the previously described extension regions for the faces of G can in fact be drawn inside the corresponding extension regions for the faces of G' provided that v' is close enough to u', so that edge (v', x) cuts one of the two side regions $S(w, x)$ touching edge (w, x). All the other extension regions for the faces incident to v' are drawn as the corresponding extension regions for the corresponding faces incident to w. All the extension regions for the faces not incident to u' and v' are drawn as in Γ'.

Lemma 4. *Γ is an extensible drawing of C completing Γ_o.*

It remains to prove that, if no inductive case applies, then we are in a base case.

Suppose that none of the inductive cases applies. If G does not contain internal vertices, then we are in Base Case 1. If G contains exactly one internal vertex, such a vertex can not be adjacent to any vertex of the outer face of G, as otherwise Inductive Case 2 would apply, hence we are in Base Case 2. If G contains more than one internal vertex, then observe that, since Inductive Case 1 and Inductive Case 2 do not apply, we are in Base Case 3. This concludes the proof of Theorem 2.

5 Conclusions

In this paper we have proved that every flat c-planar clustered graph admits a multilevel drawing. The algorithm we described constructs drawings requiring exponential area if a finite resolution rule is assumed to hold. However, this drawback is unavoidable, since there exist (flat) clustered graphs requiring exponential area in any straight-line drawing in which clusters are represented by convex regions [8]. It is an obvious open problem to extend our results to general c-planar clustered graphs. We suspect that our drawing techniques, together with some techniques to decompose non-flat clustered graphs into smaller non-flat clustered graphs presented in [1], might lead to a solution of the problem. However, we defer such an intuition to future research.

Acknowledgments. Thanks to Peter Eades for posing the problem studied in this paper and for useful discussions about it.

References

1. Angelini, P., Frati, F., Kaufmann, M.: Straight-line rectangular drawings of clustered graphs. Discrete & Computational Geometry 45(1), 88–140 (2011)
2. Di Battista, G., Drovandi, G., Frati, F.: How to draw a clustered tree. J. Discrete Algorithms 7(4), 479–499 (2009)
3. Di Battista, G., Frati, F.: Efficient c-planarity testing for embedded flat clustered graphs with small faces. J. Graph Alg. Appl. 13(3), 349–378 (2009)
4. Eades, P., Feng, Q.: Multilevel Visualization of Clustered Graphs. In: DiBattista, G. (ed.) GD 1997. LNCS, vol. 1353, pp. 101–112. Springer, Heidelberg (1997)
5. Eades, P., Feng, Q., Lin, X., Nagamochi, H.: Straight-line drawing algorithms for hierarchical graphs and clustered graphs. Algorithmica 44(1), 1–32 (2006)
6. Eades, P., Feng, Q., Nagamochi, H.: Drawing clustered graphs on an orthogonal grid. J. Graph Alg. Appl. 3(4), 3–29 (1999)
7. Feng, Q.: Algorithms for drawing clustered graphs. Ph. D. thesis. The University of Newcastle, Australia (1997)
8. Feng, Q., Cohen, R.F., Eades, P.: How to Draw a Planar Clustered Graph. In: Li, M., Du, D.-Z. (eds.) COCOON 1995. LNCS, vol. 959, pp. 21–30. Springer, Heidelberg (1995)
9. Feng, Q., Cohen, R.F., Eades, P.: Planarity for Clustered Graphs. In: Spirakis, P.G. (ed.) ESA 1995. LNCS, vol. 979, pp. 213–226. Springer, Heidelberg (1995)
10. Jelínková, E., Kára, J., Kratochvíl, J., Pergel, M., Suchý, O., Vyskocil, T.: Clustered planarity: Small clusters in cycles and Eulerian graphs. J. Graph Alg. Appl. 13(3), 379–422 (2009)
11. Jünger, M., Leipert, S., Percan, M.: Triangulating clustered graphs. Technical report. Zentrum für Angewandte Informatik Köln (2002)
12. Nagamochi, H., Kuroya, K.: Drawing c-planar biconnected clustered graphs. Discr. Appl. Math. 155(9), 1155–1174 (2007)
13. Schaeffer, S.E.: Graph clustering. Computer Science Review 1(1), 27–64 (2007)
14. Tutte, W.T.: How to draw a graph. Proc. London Math. Soc. 13(52), 743–768 (1963)

Outerplanar Graph Drawings with Few Slopes

Kolja Knauer[1,⋆], Piotr Micek[2,⋆⋆], and Bartosz Walczak[2,⋆⋆]

[1] Institut für Mathematik, Technische Universität Berlin
knauer@math.tu-berlin.de
[2] Theoretical Computer Science Department,
Faculty of Mathematics and Computer Science, Jagiellonian University
{micek,walczak}@tcs.uj.edu.pl

Abstract. We consider straight-line outerplanar drawings of outerplanar graphs in which the segments representing edges are parallel to a small number of directions. We prove that $\Delta - 1$ directions suffice for every outerplanar graph with maximum degree $\Delta \geqslant 4$. This improves the previous bound of $O(\Delta^5)$, which was shown for planar partial 3-trees, a superclass of outerplanar graphs. The bound is tight: for every $\Delta \geqslant 4$ there is an outerplanar graph of maximum degree Δ which requires at least $\Delta - 1$ distinct edge slopes for an outerplanar straight-line drawing.

1 Introduction

A *straight-line drawing* of a graph G is a mapping of the vertices of G into distinct points in the plane and of the edges of G into straight-line segments connecting the points representing their end-vertices and passing through no other points representing vertices. If it leads to no confusion, in notation and terminology, we make no distinction between a vertex and the corresponding point, and between an edge and the corresponding segment. The *slope* of an edge in a straight-line drawing is the family of all straight lines parallel to the segment representing this edge. The *slope number* of a graph G, introduced by Wade and Chu [1], is the smallest number s such that there is a straight-line drawing of G using s slopes.

Since at most two edges at each vertex can use the same slope, $\lceil \frac{\Delta}{2} \rceil$ is a lower bound for the slope number of a graph with maximum degree Δ. In general, graphs with maximum degree $\Delta \geqslant 5$ may have arbitrarily large slope number, see [2,3]. If the maximum degree of a graph is at most 3 then the slope number is at most 4 as shown by Mukkamala and Szegedy [4], improving a result of Keszegh et al. [5]. The question whether the slope number of graphs with maximum degree 4 is bounded by a constant remains open.

The situation is different for *planar* straight-line drawings, that is, straight-line drawings in which no two distinct edges intersect in a point other than a

⋆ Supported by DFG grant FE-340/8-1 as part of ESF EuroGIGA project GraDR.
⋆⋆ Supported by MNiSW grant 884/N-ESF-EuroGIGA/10/2011/0 as part of ESF EuroGIGA project GraDR.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 323–334, 2012.

common endpoint. It is well known that every planar graph admits a planar straight-line drawing [6,7,8]. The *planar slope number* of a planar graph G is the smallest number s such that there is a planar straight-line drawing of G using s slopes. Keszegh et al. [9] show that the planar slope number is bounded by a function of maximum degree. Their bound is exponential and their proof is non-constructive. Jelínek et al. [10] give an upper bound for the planar slope number of planar graphs of treewidth at most 3, which is $O(\Delta^5)$.

In the present paper we consider drawings of outerplanar graphs. As outerplanar graphs have treewidth at most 2, they admit planar drawings with $O(\Delta^5)$ slopes. A straight-line drawing of a graph G is *outerplanar* if it is planar and all vertices of G lie on the outer face. The *outerplanar slope number* of an outerplanar graph G is the smallest number s such that there is an outerplanar straight-line drawing of G using s slopes. Dujmović et al. [11] consider the outerplanar slope number as a function of the number of vertices. We provide a tight bound for the outerplanar slope number in terms of the maximum degree.

Main Theorem. *The outerplanar slope number of every outerplanar graph with maximum degree $\Delta \geqslant 4$ is at most $\Delta - 1$.*

It is easy to see, that the tightness of this bound is witnessed by a cycle with at least $2\Delta - 3$ vertices, where to each vertex we attach $\Delta - 2$ leaves.

Note that the tight bounds for the outerplanar slope number with respect to the maximum degree Δ are: 1 for $\Delta = 1$ and 3 for $\Delta \in \{2, 3\}$. The latter bound from above is implied by our theorem.

The proof of our theorem is constructive and yields a linear-time algorithm to produce drawings of the claimed kind.

2 Basic Definitions

Suppose we are given an outerplanar drawing of a connected graph G with maximum degree $\Delta \geqslant 4$. This drawing determines the cyclic ordering of edges around every vertex. We produce an outerplanar straight-line drawing of G with few edge slopes which preserves this ordering at every vertex. Our construction is inductive: it composes the entire drawing of G from drawings of subgraphs of G that we call bubbles.

We distinguish the *outer face* of G (the one that is unbounded in the given drawing of G and contains all vertices on the boundary) from the *inner faces*. The edges on the boundary of the former are *outer edges*, while all remaining ones are *inner edges*. A *snip* is a simple closed counterclockwise-oriented curve γ which

- passes through some pair of vertices u and v of G (possibly being the same vertex) and through no other vertex of G,
- on the way from v to u goes entirely through the outer face of G crossing no edges on the way,
- on the way from u to v (considered only if $u \neq v$) goes through inner faces of G possibly crossing some inner edges of G, each at most once.

Every snip γ defines a *bubble* H in G as the subgraph of G induced on the vertices lying on or inside γ. Note that H is a connected subgraph of G as γ crosses no outer edges. The oriented simple path P from u to v in H going counterclockwise along the boundary of the outer face of H is called the *root-path* of H. If $u = v$ then the root-path consists of that single vertex only. The *roots* of H are the vertices u and v together with all vertices of H incident to the edges crossed by γ. Note that vertices of H not being roots cannot have edges to $G - H$. Note also that the root-path and the roots of H do not depend on the particular snip γ used to define H. The order of roots along the root-path gives the *root-sequence* of H. A bubble with k roots is called a *k-bubble*. A special role in our proof is played by 1- and 2-bubbles.

Bubbles admit a natural decomposition, which is the base of our recursive drawing.

Lemma 1. *Let H be a bubble with root-path $v_1 \dots v_k$. Every component of $H - \{v_1, \dots, v_k\}$ is adjacent to either one vertex among $v_1, \dots, v_k$ or two consecutive vertices from $v_1, \dots, v_k$. Moreover, there is at most one component adjacent to v_i and v_{i+1} for $1 \leqslant i < k$.*

Proof. Let C be a connected component of $H - \{v_1, \dots, v_k\}$. As H itself is connected, C must be adjacent to a vertex from $v_1, \dots, v_k$. In order to get a contradiction suppose that C is connected to two non-consecutive vertices v_i and v_j. Let P be a simple v_i, v_j-path having all internal vertices in C. Let $P' = v_i \dots v_j$ be the subpath of the root-path of H connecting v_i and v_j. Since $v_1 \dots v_k$ is the root-path of H, all edges connecting the internal vertices of P' to $G - H$ are inner edges. Hence, also the edges of P' lie on inner faces which are not faces of H. The symmetric difference of all these inner faces considered as sets of edges is a simple cycle containing P' as a subpath. Let P'' denote the other v_i, v_j-subpath of that cycle. It is internally disjoint from P and P'. Moreover, P'' and P together enclose P' and thus the internal vertices of P' do not lie on the outer face—contradiction.

Now, to prove the second statement, suppose that for some i two components C and C' of $H - \{v_1, \dots, v_k\}$ are adjacent to both v_i and v_{i+1}. We find two internally disjoint v_i, v_{i+1}-paths P and P' through C and C', respectively. As in the above paragraph we use that $v_i v_{i+1}$ is contained in an inner face, which is not a face of H. The third path P'' is obtained from that face by deleting the edge v_i, v_{i+1}. It follows that P, P', P'' form a subdivision of $K_{2,3}$, which contradicts outerplanarity of G. □

Lemma 1 allows us to assign each component of $H - \{v_1, \dots, v_k\}$ to a vertex of P or an edge of P so that every edge is assigned at most one component. For a component C assigned to a vertex v_i, the graph induced on $C \cup \{v_i\}$ is called a *v-bubble*. If P consists of a single vertex with no component assigned to it, we consider that vertex alone to be a v-bubble. For a component C assigned to an edge $v_i v_{i+1}$, the graph induced on $C \cup \{v_i, v_{i+1}\}$ is called an *e-bubble*. If no component is assigned to an edge of P then we consider that edge alone an e-bubble. All v-bubbles of v_i in H are naturally ordered by their clockwise

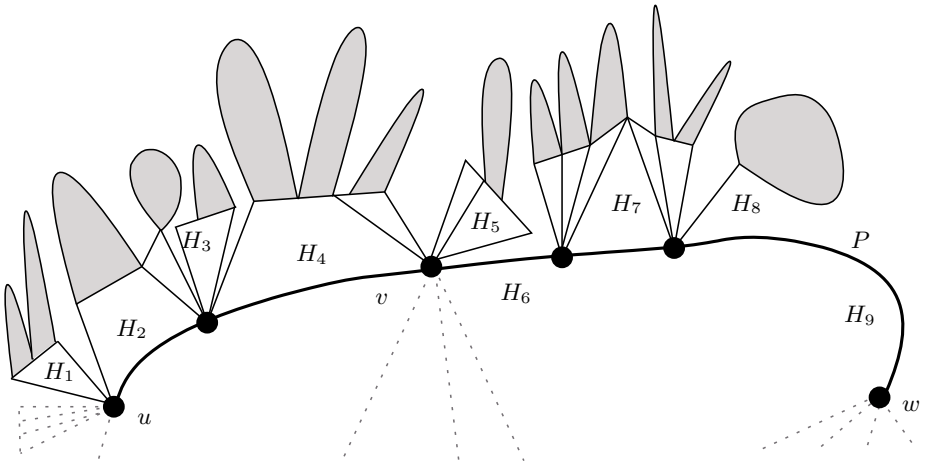


Fig. 1. A 3-bubble H with root-path P (drawn thick), root-sequence u, v, w (connected to the remaining graph by dotted edges), and splitting sequence into v- and e-bubbles $(H_1, \ldots, H_9)$. For example, (H_2, H_3, H_4) is a 2-bubble.

arrangement around v_i in the drawing. All this leads to a decomposition of the bubble H into a sequence $(H_1, \ldots, H_b)$ of v- and e-bubbles such that the naturally ordered v-bubbles of v_1 precede the e-bubble of v_1v_2, which precedes the naturally ordered v-bubbles of v_2, and so on. We call this sequence the *splitting sequence* of H and write $H = (H_1, \ldots, H_b)$. Since no single-vertex v-bubbles occur in the splitting sequence unless H is itself a single-vertex v-bubble the splitting sequence of H is unique. Note that v- and e-bubbles are special kinds of 1- and 2-bubbles, respectively. Every 1-bubble is a bouquet of v-bubbles. The splitting sequence of a 2-bubble may consist of several v- and e-bubbles. For an illustration see Fig. 1.

The general structure of the induction in our proof is covered by the following lemma (see Fig. 2):

Lemma 2.

2.1. Let H be a v-bubble rooted at v. Let $v^1, \ldots, v^k$ be the neighbors of v in H in clockwise order. Then $H - v$ is a k-bubble with root-sequence $v^1, \ldots, v^k$.

2.2. Let H be a v-bubble rooted at v_0. Let $v_0 \ldots v_n$ be an induced path going from v_0 counterclockwise along the outer face of H and such that $v_1, \ldots, v_{n-1}$ are not cut-vertices in H. Then $H - \{v_0, \ldots, v_n\}$ has a unique component H' adjacent to both v_0 and v_n. Moreover, let X be the subgraph of H induced on $v_0, \ldots, v_n$ and the vertices of H'. Let $v_0^1, \ldots, v_0^{k_0}, v_1$ be the neighbors of v_0 in X in clockwise order. For $1 \leqslant i \leqslant n-1$ let $v_{i-1}, v_i^1, \ldots, v_i^{k_i}, v_{i+1}$ be the neighbors of v_i in X in clockwise order. Let $v_{n-1}, v_n^1, \ldots, v_n^{k_n}$ be the neighbors of v_n in X in clockwise order. Then H' is a bubble with root-

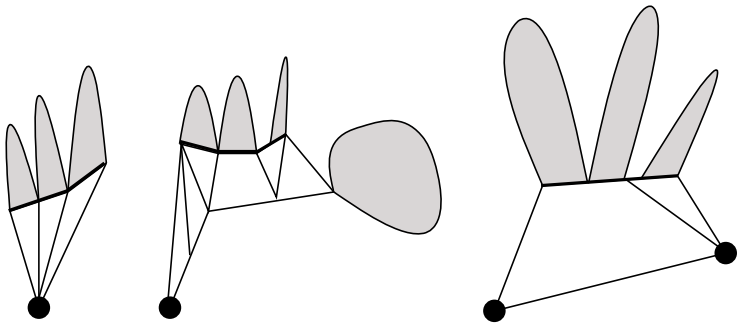


Fig. 2. Three ways of obtaining smaller bubbles from v- and e-bubbles. The new root-path is drawn thick.

sequence $v_0^1, \ldots, v_0^{k_0}, v_1^1, \ldots, v_1^{k_1}, \ldots, v_n^1, \ldots, v_n^{k_n}$, *where* $v_i^{k_i}$ *and* v_{i+1}^1 *may coincide for* $0 \leqslant i \leqslant n-1$.

2.3. Let H be an e-bubble with roots u, v. Let $u^1, \ldots, u^k, v$ be the neighbors of u in H in clockwise order and $u, v^1, \ldots, v^\ell$ be the neighbors of v in H in clockwise order. Then $H - \{u, v\}$ is a bubble with root-sequence $u^1, \ldots, u^k$, $v^1, \ldots, v^\ell$, where u^k and v^1 may coincide.

Proof. Omitted. □

3 Bounding Regions

Depending on the maximum degree Δ of G define the set S of $\Delta - 1$ slopes to consist of the horizontal slope and the slopes of vectors $\mathbf{f}_1, \ldots, \mathbf{f}_{\Delta-2}$ where

$$\mathbf{f}_i = (-\tfrac{1}{2} + \tfrac{i-1}{\Delta-3}, 1) \quad \text{for } i = 1, \ldots, \Delta - 2.$$

An important property of S is that it cuts the horizontal segment L from $(-\frac{1}{2}, 1)$ to $(\frac{1}{2}, 1)$ into $\Delta - 3$ segments of equal length $\frac{1}{\Delta-3}$. We construct an outerplanar straight-line drawing of G using only slopes from S and preserving the given cyclic ordering of edges at each vertex of G.

The essential tool in proving that our construction does not make bubbles overlap are bounding regions. Their role is to bound the space of the plane occupied by bubbles. The bounding region for a bubble is parametrized by ℓ and r which depend on the degrees of the roots in the bubble. Let v be a point in the plane. For a vector x let $R(v; x) = \{v + \alpha x : \alpha \geqslant 0\}$. We define $LB(v; \ell)$ to be the set consisting of v and all points p with $p_y \geqslant v_y$. If $\ell > 0$ then we furthermore require that

- p lies on $R(v; \mathbf{f}_1)$ or to the right of it — if $\ell = 1$,
- $p_x > v_x$ — if $\Delta = 4$ and $\ell = 2$,

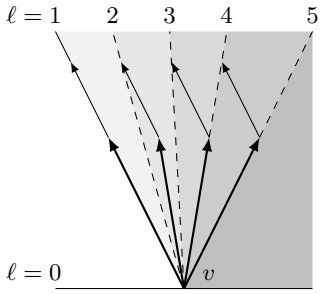


Fig. 3. Boundaries of $LB(v;\ell)$ for $\Delta = 6$. Vectors $\mathbf{f}_i$ at v are indicated by thick arrows. Vectors $\frac{1}{2}\mathbf{f}_1$ at $v + \mathbf{f}_i$ are indicated by thin arrows. Note that $\mathbf{f}_3$ lies on the boundary of $LB(v;4)$.

- p lies to the right of $R(v; \mathbf{f}_\ell + \frac{1}{\Delta-4}\mathbf{f}_1)$ if $\Delta \geqslant 5$ and $2 \leqslant \ell \leqslant \Delta - 2$,
- p lies to the right of $R(v; \mathbf{f}_{\Delta-2})$ if $\ell = \Delta - 1$.

See Fig. 3 for an illustration. Similarly, $RB(v;r)$ consists of v and all points p with $p_y \geqslant v_y$. If $r < \Delta - 1$ we furthermore require that

- p lies to the left of $R(v; \mathbf{f}_1)$ if $r = 0$,
- $p_x < v_x$ if $\Delta = 4$ and $r = 1$,
- p lies to the left of $R(v; \mathbf{f}_r + \frac{1}{\Delta-4}\mathbf{f}_{\Delta-2})$ if $\Delta \geqslant 5$ and $1 \leqslant r \leqslant \Delta - 3$,
- p lies on $R(v; \mathbf{f}_{\Delta-2})$ or to the left of it if $r = \Delta - 2$.

Now, for points u, v in the plane such that $u_y = v_y$ and $u_x \leqslant v_x$ we define bounding regions as follows:

$$B(uv;\ell,r) = LB(u;\ell) \cap RB(v;r) \quad \text{for } 0 \leqslant \ell, r \leqslant \Delta - 1,$$
$$\bar{B}(uv;\ell,r;h) = B(uv;\ell,r) \cap \{p : p_y < u_y + h\} \quad \text{for } 0 \leqslant \ell, r \leqslant \Delta - 1 \text{ and } h > 0.$$

We denote $B(vv;\ell,r)$ simply by $B(v;\ell,r)$ and $\bar{B}(vv;\ell,r;h)$ simply by $\bar{B}(v;\ell,r;h)$. Note that the bottom border of a bounding region is always included, the left border (if exists) is included if $\ell = 1$, the right border (if exists) is included if $r = \Delta - 2$, and the top border (if exists) is never included. If $\ell > r$ then $B(v;\ell,r) = \{v\}$.

We use $B(v;\ell,r)$ and $\bar{B}(v;\ell,r;h)$ to bound drawings of 1-bubbles H with root v such that $r - \ell + 1 = d_H(v)$. Note that every 1-bubble drawn inside $B(v;\ell,r)$ can be scaled to fit inside $\bar{B}(v;\ell,r;h)$ for any $h > 0$ without changing slopes. We use $B(uv;\ell,r)$ and $\bar{B}(uv;\ell,r;h)$ with $u \neq v$ to bound drawings of 2-bubbles H whose root-path starts at u and ends at v, such that $\ell = \Delta - d_H(u)$ and $r = d_H(v) - 1$. Here H cannot be scaled if the positions of both u and v are fixed, so the precise value of h matters. However, every 2-bubble drawn inside $B(uv;\ell,r)$ can be scaled to fit inside $\bar{B}(uw;\ell,r;h)$ for any $h > 0$ without changing slopes, where w is some point of the horizontal line containing uv.

Lemma 3. *Bounding regions have the following geometric properties ($*$ denotes any relevant value).*

3.1. If $u'v' \subseteq uv$, $\ell' \geqslant \ell$, and $r' \leqslant r$ then $B(u'v';\ell',r') \subseteq B(uv;\ell,r)$.

3.2. If $i < \ell$ then a vector at u in direction $\mathbf{f}_i$ points outside $B(uv;\ell,*)$ to the left of it. If $i > r$ then a vector at v in direction $\mathbf{f}_i$ points outside $B(uv;*,r)$ to the right of it.

3.3. If u,v,w are consecutive points on a horizontal line and $\ell - 1 \geqslant r + 1$ then $B(uv;*,r) \cap B(vw;\ell,*) = \{v\}$.

Moreover, the following holds for $\Delta \geqslant 5$.

3.4. For $l < r$, $h > 0$, $u' = u + h\mathbf{f}_\ell$, and $v' = v + h\mathbf{f}_r$ we have $\bar{B}(u'v';1,\Delta - 2;\frac{h}{\Delta-4}) \subseteq \bar{B}(uv;\ell,r;\frac{\Delta-3}{\Delta-4}h)$.

3.5. If u,v,w are consecutive points on a horizontal line, $\ell' \geqslant \ell$, $r' \leqslant \Delta - 3$, and $r \geqslant 1$ then $\bar{B}(uv;\ell',r';(\frac{\Delta-3}{\Delta-4})^2|vw|) \subseteq B(uw;\ell,r)$.

3.6. If u,v,w,x are consecutive points on a horizontal line, $|uv| = |wx| \leqslant |vw|$, $\ell \geqslant 2$, and $r \leqslant \Delta - 3$ then $\bar{B}(uv;*,r;\frac{\Delta-3}{\Delta-4}|uv|) \cap \bar{B}(wx;\ell,*;\frac{\Delta-3}{\Delta-4}|wx|) = \emptyset$.

Proof. Statement 3.1 clearly follows from the definition. Statement 3.2 is implied by the definition and for $\Delta \geqslant 5$ by the fact that

$$\begin{aligned} \mathbf{f}_\ell + \tfrac{1}{\Delta-4}\mathbf{f}_1 &= \mathbf{f}_{\ell-1} + \tfrac{1}{\Delta-4}\mathbf{f}_{\Delta-3} && \text{for } \ell = 2,\dots,\Delta-2, \\ \mathbf{f}_r + \tfrac{1}{\Delta-4}\mathbf{f}_{\Delta-2} &= \mathbf{f}_{r+1} + \tfrac{1}{\Delta-4}\mathbf{f}_2 && \text{for } r = 1,\dots,\Delta-3. \end{aligned}$$

Statement 3.2 directly yields 3.3: the vector at v in direction $\mathbf{f}_{r+1}$ points outside both $B(uv;*,r)$ and $B(vw;\ell,*)$. To see 3.4 note that the point $u' + \frac{\alpha}{\Delta-4}\mathbf{f}_1$, which is the top-left corner of $\bar{B}(u'v';1,\Delta-2;\frac{h}{\Delta-4})$, equals $u + h(\mathbf{f}_\ell + \frac{1}{\Delta-4}\mathbf{f}_1)$. Hence it lies at the top-left corner of $B(uv;\ell,r)$. Similarly, the top-right corner of $\bar{B}(u'v';1,\Delta-2;\frac{h}{\Delta-4})$ lies at the top-right corner of $B(uv;\ell,r)$. To prove 3.5 it suffices to consider the case $r = 1$ and $r' = \Delta - 3$. The top-right corner of $\bar{B}(uv;\ell',r';(\frac{\Delta-3}{\Delta-4})^2|vw|)$ is

$$\begin{aligned} v + \tfrac{\Delta-3}{\Delta-4}|vw|(\mathbf{f}_{\Delta-3} + \tfrac{1}{\Delta-4}\mathbf{f}_{\Delta-2}) &= v + |vw|(\mathbf{f}_{\Delta-2} + \tfrac{1}{\Delta-4}\mathbf{f}_1 + \tfrac{\Delta-3}{\Delta-4}\cdot\tfrac{1}{\Delta-4}\mathbf{f}_{\Delta-2}) \\ &= w + \tfrac{\Delta-3}{\Delta-4}|vw|(\mathbf{f}_1 + \tfrac{1}{\Delta-4}\mathbf{f}_{\Delta-2}). \end{aligned}$$

Therefore, it lies on the right side of $B(uw;\ell,1)$ and the conclusion of 3.5 follows. Finally, for the proof of 3.6 it suffices to consider the case $\ell = 2$, $r = \Delta - 3$, and $|uv| = |vw| = |wx| = \lambda$. The top-right corner of $\bar{B}(uv;*,\Delta-3;\frac{\Delta-3}{\Delta-4}\lambda)$ and the top-left corner of $\bar{B}(wx;2,*;\frac{\Delta-3}{\Delta-4}\lambda)$ are respectively

$$\begin{aligned} v + \lambda(\mathbf{f}_{\Delta-3} + \tfrac{1}{\Delta-4}\mathbf{f}_{\Delta-2}) &= v + \lambda(\mathbf{f}_{\Delta-2} + \tfrac{1}{\Delta-4}\mathbf{f}_2), \\ w + \lambda(\mathbf{f}_2 + \tfrac{1}{\Delta-4}\mathbf{f}_1) &= w + \lambda(\mathbf{f}_1 + \tfrac{1}{\Delta-4}\mathbf{f}_{\Delta-3}). \end{aligned}$$

They coincide if $\Delta = 5$, otherwise the former lies to the left of the latter. □

4 The Drawing

We present the construction of a drawing first for $\Delta \geqslant 5$ and then for $\Delta = 4$. Both constructions follow the same idea but differ in technical details.

Lemma 4. *Suppose* $\Delta \geqslant 5$.

4.1. Let H *be a* 1*-bubble with root* v*. Suppose that the position of* v *is fixed. Let* ℓ *and* r *be such that* $0 \leqslant \ell, r \leqslant \Delta - 1$ *and* $r - \ell + 1 = d_H(v) \leqslant \Delta - 1$*. Then there is a straight-line drawing of* H *inside* $B(v; \ell, r)$*.*

4.2. Let H *be a* 2*-bubble with first root* u *and last root* v*. Suppose that the positions of* u *and* v *are fixed on a horizontal line so that* u *lies to the left of* v*. Let* $\ell = \Delta - d_H(u)$ *and* $r = d_H(v) - 1$*. Then there is a straight-line drawing of* H *inside* $\bar{B}(uv; \ell, r; \frac{\Delta-3}{\Delta-4}|uv|)$ *such that the root-path of* H *is drawn as the segment* uv*.*

4.3. Let H *be a* k*-bubble with roots* $v_1, \ldots, v_k$ *in this order along the root-path. If* $k = 1$ *then suppose* $d_H(v_1) \leqslant \Delta - 2$*, otherwise suppose* $d_H(v_1), d_H(v_k) \leqslant \Delta - 1$*. Suppose that for some* $\lambda > 0$ *the positions of* $v_1, \ldots, v_k$ *are fixed in this order on a horizontal line so that* $|v_1v_2| = \ldots = |v_{k-1}v_k| = \lambda$*. Then there is a straight-line drawing of* H *inside* $\bar{B}(v_1v_k; 1, \Delta - 2; \frac{\Delta-3}{\Delta-4}\lambda)$ *such that the root-path of* H *is drawn as the segment* v_1v_k*.*

The drawings claimed above use only slopes from S *and preserve the order of edges around each vertex* w *of* H *under the assumption that if there are edges connecting* w *to* $G - H$ *then they are drawn in the correct order outside the considered bounding region.*

Proof. The proof constructs the required drawing by induction on the size of H. The construction we are going to describe clearly preserves the order of edges at every vertex of H and uses only slopes from S, and we do not explicitly state this observation anywhere further in the proof.

Let H be a k-bubble with splitting sequence $(H_1, \ldots, H_b)$. Assuming that the lemma holds for any bubble with fewer vertices than H has, we prove that H satisfies 4.1 if $k = 1$, 4.2 if $k = 2$, and 4.3 for any k.

First we prove 4.1 and 4.2 for the case that $b \geqslant 2$ and one of H_1, H_b is a v-bubble. By symmetry we consider only the case of H_1 being a v-bubble. Let u and v denote respectively the first and the last root of H. Define $Y = (H_2, \ldots, H_b)$, $r_1 = \ell + d_{H_1}(u) - 1$, and $\ell_Y = \Delta - d_Y(u)$. By the induction hypothesis we can draw H_1 inside $B(v_1; \ell, r_1)$ and Y inside $\bar{B}(uv; \ell_Y, r; \frac{\Delta-3}{\Delta-4}|uv|)$ (in case $k = 1$ we drop the restriction on the height of the latter bounding region). We scale the drawing of H_1 to make it so small that it lies entirely below all vertices of Y not contained in the root-path and (for $k = 2$) below the horizontal line bounding from above the requested bounding region of H. Since $r_1 + 1 = \ell_Y$ and by 3.2, our scaled drawing of H_1 lies to the left of the leftmost edge at the root u of Y. Thus the drawings of H_1 and Y do not overlap. By 3.1 they both fit within $\bar{B}(uv; \ell, r; \frac{\Delta-3}{\Delta-4}|uv|)$.

Now we consider the case of H being a v-bubble rooted at v and such that $d_H(v) \leqslant \Delta - 1$. We are to prove that 4.1 holds for H. If v is the only vertex of H, the conclusion is trivial. Thus we assume that H has at least two vertices, that is, $\ell \leqslant r$.

Suppose $1 \leqslant \ell, r \leqslant \Delta - 2$. Define $H' = H - v$. By 2.1 the graph H' is an $(r - \ell + 1)$-bubble with root-sequence formed by the neighbors $v^\ell, \ldots, v^r$ of v in H listed in the clockwise order around v. Put the vertices $v^\ell, \ldots, v^r$ at points $v + \mathbf{f}_\ell, \ldots, v + \mathbf{f}_r$, respectively. This way $v^\ell, \ldots, v^r$ lie on a common horizontal line L and partition L into segments of length $\frac{1}{\Delta-3}$. Now, if $\ell = r$ then by the induction hypothesis the 1-bubble H' can be drawn inside $B(v^r; 0, d_{H'}(v^r) - 1)$ as well as inside $B(v^r; \Delta - d_{H'}(v^r), \Delta - 1)$. Choose the former drawing if $\ell = r = \Delta - 2$, the latter if $\ell = r = 1$, or any of the two otherwise. After appropriate scaling the chosen drawing fits within $B(v; r, r)$. If $\ell < r$ then apply the induction hypothesis to draw H' inside $\bar{B}(v^\ell v^r; 1, \Delta - 2; \frac{1}{\Delta-4})$. It follows from 3.4 that this bounding region is included in $B(v; \ell, r)$.

It remains to consider the following cases for v-bubble H: $\ell = 0$ or $r = \Delta - 1$. They cannot hold simultaneously as $d_H(v) \leqslant \Delta - 1$. By symmetry it is enough to consider only one of them. Thus we assume $1 \leqslant \ell \leqslant r = \Delta - 1$.

If v has only one neighbor in H, say w, then $\ell = r = \Delta - 1$ and clearly $H' = H - v$ is a 1-bubble rooted at w. Put w horizontally to the right of v. Draw H' inside $B(w; \Delta - d_{H'}(w), \Delta - 1)$ by the induction hypothesis, scaling the drawing appropriately to fit it within $B(v; \Delta - 1, \Delta - 1)$.

Now suppose that v has at least two neighbors in H and therefore $\ell < r = \Delta - 1$. Let $P = w_0 \ldots w_n$ be the simple path of length $n \geqslant 1$ starting at $w_0 = v$ and going counterclockwise along the outer face of H so that

- the edge $w_0 w_1$ preceeds the root-angle in the clockwise order around $w_0 = v$,
- the vertices $w_1, \ldots, w_{n-1}$ have degree Δ and are not cut-vertices in H,
- the vertex w_n has degree at most $\Delta - 1$ or is a cut-vertex in H.

Note that if $n = 1$ then the second condition is satisfied vacuously. Since the degrees of $w_1, \ldots, w_{n-1}$ are at least 3 and by outerplanarity, P is an induced path. Thus by 2.2 the graph $H - P$ has exactly one component H' adjacent to both w_0 and w_n. All other components of $H - P$ are adjacent to w_n. Together with w_n they form a (possibly trivial) 1-bubble Y rooted at w_n. Let X denote the subgraph of H induced on $w_0, \ldots, w_n$ and the vertices of H'. Define $r_X = d_X(w_n) - 1$ and $\ell_Y = \Delta - d_Y(w_n)$. Let $w_0^\ell, \ldots, w_0^{\Delta-2}$ be the neighbors of w_0 in X ordered clockwise. Let $w_i^1, \ldots, w_i^{\Delta-2}$ be the neighbors of w_i in X ordered clockwise, for $1 \leqslant i \leqslant n - 1$. Let $w_n^1, \ldots, w_n^{r_X}$ be the neighbors of w_n in X ordered clockwise. Note that $w_i^{\Delta-2}$ and w_{i+1}^1 may coincide, for $0 \leqslant i \leqslant n-1$. It follows from 2.2 that H' is a bubble with root-sequence $w_0^\ell, \ldots, w_0^{\Delta-2}, w_1^1, \ldots, w_1^{\Delta-2}, \ldots, w_n^1, \ldots, w_n^{r_X}$. For $i = 0, \ldots, n - 1$ define

$$\lambda_i = \begin{cases} 1 & \text{if } w_i^{\Delta-2} = w_{i+1}^1, \\ \frac{\Delta-2}{\Delta-3} & \text{if } w_i^{\Delta-2} \neq w_{i+1}^1. \end{cases}$$

Put the vertices $w_1, \ldots, w_n$ in this order from left to right on the horizontal line going through w_0 in such a way that $|w_i w_{i+1}| = \lambda_i$. Put each vertex w_i^j at point

$w_i + \mathbf{f}_j$. Note that if $w_i^{\Delta-2}$ and w_{i+1}^1 are the same vertex then they end up at the same point. All w_i^j lie on a common horizontal line L at distance 1 to the segment w_0w_n and partition L into segments of length $\frac{1}{\Delta-3}$. Define

$$B_X = \bar{B}(w_0w_n; \ell, r_X; \tfrac{\Delta-3}{\Delta-4}),$$
$$B_Y = \bar{B}(w_n; \ell_Y, \Delta-1; 1).$$

Draw H' inside $\bar{B}(w_0^\ell w_n^{r_X}; 1, \Delta-2; \frac{1}{\Delta-4})$ using the induction hypothesis. Note that if H' is a 1-bubble then $n = 1$ and the root of H' has at least two edges outside H' (namely, the ones going to w_0 and w_1), and therefore the first case of the induction hypothesis yields its drawing inside the claimed bounding region. By 3.4 this bounding region is contained in B_X. Draw Y inside B_Y using the induction hypothesis and scaling. This way it lies entirely below the line L. By 3.2 the drawing of Y lies to the right of the edge $w_nw_n^{r_X}$ and thus does not overlap with the drawing of X. Clearly, B_X and B_Y are contained in $B(w_0; \ell, \Delta-1)$. This completes the proof for the case of H being a 1-bubble.

Now, suppose $k = 2$. We are to show that 4.2 holds for H. Suppose first that H is a single e-bubble with roots u and v. If H consists of the edge uv only then the conclusion follows trivially. Thus assume that H is not the single edge uv. This implies that $d_H(u), d_H(v) \geqslant 2$ and therefore $\ell \leqslant \Delta-2$ and $r \geqslant 1$. Let $u^\ell, \ldots, u^{\Delta-2}, v$ denote the neighbors of u in H ordered clockwise, and let $u, v^1, \ldots, v^r$ be the neighbors of v in H ordered clockwise. Let $H' = H - \{u, v\}$. By 2.3 the graph H' is a bubble with all w_0^i and w_1^i being the roots. Define

$$h = \begin{cases} |uv| & \text{if } u^{\Delta-2} = v^1, \\ \frac{\Delta-3}{\Delta-2}|uv| & \text{if } u^{\Delta-2} \neq v^1. \end{cases}$$

Put each vertex u^i at point $u + h\mathbf{f}_i$ and each vertex v^i at point $v + h\mathbf{f}_i$. Note that if $u^{\Delta-2}$ and v^1 are the same vertex then they end up at the same point. All u^i and v^i lie on a common horizontal line and partition it into segments of length $\frac{h}{\Delta-3}$. Draw H' inside $\bar{B}(u^\ell v^r; 1, \Delta-2; \frac{h}{\Delta-4})$ using the induction hypothesis. This bounding region is contained in $\bar{B}(uv; \ell, r; \frac{\Delta-3}{\Delta-4}h)$ by 3.4. Since $h \leqslant |uv|$, we have $\bar{B}(uv; \ell, r; \frac{\Delta-3}{\Delta-4}h) \subseteq \bar{B}(uv; \ell, r; \frac{\Delta-3}{\Delta-4}|uv|)$.

Now, consider the case that H is a 2-bubble but not an e-bubble. Let $w_0 \ldots w_n$ be the root-path of H. Thus $w_0 = u$, $w_n = v$, and $n \geqslant 2$. Suppose that none of the edges $w_0w_1, \ldots, w_{n-1}w_n$ is a bridge in H. This implies that $d_H(u), d_H(v) \geqslant 2$ and therefore $\ell \leqslant \Delta-2$ and $r \geqslant 1$. We split H into the e-bubble H_1 and the rest $Y = (H_2, \ldots, H_b)$ being a 2-bubble with roots w_1 and w_n. Define $r_1 = d_{H_1}(w_1) - 1$ and $\ell_Y = \Delta - d_Y(w_1)$. The assumption that $w_0w_1, \ldots, w_{n-1}w_n$ are not bridges yields $r_1 \geqslant 1$ and $\ell_Y \leqslant \Delta-2$. Let $w_0^\ell, \ldots, w_0^{\Delta-2}, w_1$ denote the neighbors of w_0 in H_1 ordered clockwise. Similarly, let $w_0, w_1^1, \ldots, w_1^{r_1}$ be the neighbors of w_1 in H_1 ordered clockwise. Define

$$\alpha = \begin{cases} 1 & \text{if } w_0^{\Delta-2} = w_1^1, \\ \frac{\Delta-3}{\Delta-2} & \text{if } w_0^{\Delta-2} \neq w_1^1. \end{cases}$$

Fix the position of w_1 on the segment w_0w_n so that $\alpha|w_0w_1| = \frac{\Delta-3}{\Delta-4}|w_1w_n| = h$. Put each vertex w_0^i at point $w_0 + h\mathbf{f}_i$ and each vertex w_1^i at point $w_1 + h\mathbf{f}_i$. Note that if $w_0^{\Delta-2}$ and w_1^1 are the same vertex then they end up at the same point. All w_0^i and w_1^i lie on a common horizontal line L at distance h to the segment w_0w_n and partition L into segments of length $\frac{h}{\Delta-3}$. Define

$$B_1 = \bar{B}(w_0w_1; \ell, r_1; \tfrac{\Delta-3}{\Delta-4}h),$$
$$B_Y = \bar{B}(w_1w_n; \ell_Y, r; h).$$

Let $H_1' = H_1 - \{w_0, w_1\}$. By 2.3 the graph H_1' is a bubble with all w_0^i and w_1^i being the roots. Draw H_1' inside $\bar{B}(w_0^\ell w_1^{r_1}; 1, \Delta-2; \frac{h}{\Delta-4})$ using the induction hypothesis. By 3.4 this bounding region is contained in B_1. Moreover, $r_1 \leqslant \Delta-3$ as w_1w_2 is not a bridge. This together with 3.5, the choice of w_1, and the fact that $h \leqslant |w_0w_n|$ imply that $B_1 \subseteq \bar{B}(w_0w_n; \ell, r; \frac{\Delta-3}{\Delta-4}|w_0w_n|)$. To complete the drawing of H, apply the induction hypothesis to draw Y inside B_Y. This way it lies entirely below L and therefore does not overlap with the drawing of H_1'. By 3.2 it also lies to the right of the edge $w_1w_1^{r_1}$. Clearly, $B_Y \subseteq \bar{B}(w_0w_n; \ell, r; \frac{\Delta-3}{\Delta-4}|w_0w_n|)$.

If H is a 2-bubble with root-path $w_0 \dots w_n$ and some edge w_iw_{i+1} is a bridge in H, then H splits into three bubbles: X with root-path $w_0 \dots w_i$, $Y = \{w_iw_{i+1}\}$ being a trivial e-bubble, and Z with root-path $w_{i+1} \dots w_n$. Define

$$r_X = \begin{cases} \ell + d_X(w_0) - 1 & \text{if } i = 0, \\ d_X(w_i) - 1 & \text{if } i \geqslant 1, \end{cases}$$
$$\ell_Z = \begin{cases} r - d_Z(w_n) + 1 & \text{if } i+1 = n, \\ \Delta - d_Z(w_{i+1}) & \text{if } i+1 \leqslant n-1. \end{cases}$$

We are free to choose the positions of w_i (unless $i = 0$) and w_{i+1} (unless $i+1 = n$) on the segment w_0w_n. So we take care that X and Z are small enough, that is, that after applying the induction hypothesis to draw X inside $\bar{B}(w_0w_i; \ell, r_X; |w_iw_{i+1}|)$ and Z inside $\bar{B}(w_{i+1}w_n; \ell_Z, r; |w_iw_{i+1}|)$, these bounding regions do not intersect and are both contained in $\bar{B}(w_0w_n; \ell, r; \frac{\Delta-3}{\Delta-4}|w_0w_n|)$.

It remains to prove 4.3 for H. If $k = 1$ then the claim follows directly from 4.1 and 3.1 by scaling. Thus assume $k \geqslant 2$. There is a splitting of H into 2-bubbles $X_1, \dots, X_{k-1}$ so that the splitting sequences of $X_1, \dots, X_{k-1}$ together form the splitting sequence $(H_1, \dots, H_b)$ of H. Therefore,

- the roots of X_i are v_i, v_{i+1} for $i = 1, \dots, k-1$,
- $X_{i-1} \cap X_i = \{v_i\}$ for $i = 2, \dots, k-1$.

Apply 4.2 to draw each X_i inside $\bar{B}(v_iv_{i+1}; \Delta - d_{X_i}(v_i), d_{X_i}(v_{i+1}) - 1; \frac{\Delta-3}{\Delta-4}\lambda)$. Consecutive bounding regions do not overlap by 3.3, while non-consecutive ones are disjoint by 3.6. By 3.1 they are all contained in $\bar{B}(v_1v_k; 1, \Delta-2; \frac{\Delta-3}{\Delta-4}\lambda)$. □

Lemma 5. *Suppose* $\Delta = 4$.

5.1. Let H be a 1-bubble with root v. Suppose that the position of v is fixed. Let ℓ and r be such that $0 \leqslant \ell, r \leqslant 3$ and $r - \ell + 1 = d_H(v) \leqslant 3$. Then there is a straight-line drawing of H inside $B(v; \ell, r)$.

5.2. Let H be a 2-bubble with first root u and last root v. Suppose that the positions of u and v are fixed on a horizontal line so that u lies to the left of v. Let $\ell = 4 - d_H(u)$ and $r = d_H(v) - 1$. Then there is a straight-line drawing of H inside $B(uv; \ell, r)$ such that the root-path of H is drawn as the segment uv.

The drawings claimed above use only slopes from S and preserve the order of edges around each vertex w of H under the assumption that if there are edges connecting w to $G - H$ then they are drawn in the correct order outside the considered bounding region.

Proof. Omitted. □

Now, to prove the Main Theorem, pick any vertex v of G of degree less than Δ (such a vertex always exists in an outerplanar graph), fix its position in the plane, and apply 4.1 or 5.1 to the graph G considered as a 1-bubble with root v.

Acknowledgments. We thank Vít Jelínek and Dömötör Pálvölgyi for introducing us to the problem at the meeting in Prague in summer 2011.

References

1. Wade, G.A., Chu, J.H.: Drawability of complete graphs using a minimal slope set. The Computer Journal 37(2), 139–142 (1994)
2. Barát, J., Matoušek, J., Wood, D.R.: Bounded-degree graphs have arbitrarily large geometric thickness. Electron. J. Combin. 13(1), #R3, 14 (2006)
3. Pach, J., Pálvölgyi, D.: Bounded-degree graphs can have arbitrarily large slope numbers. Electron. J. Combin. 13(1), #N1, 4 (2006)
4. Mukkamala, P., Szegedy, M.: Geometric representation of cubic graphs with four directions. Comput. Geom. 42(9), 842–851 (2009)
5. Keszegh, B., Pach, J., Pálvölgyi, D., Tóth, G.: Drawing Cubic Graphs with at Most Five Slopes. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 114–125. Springer, Heidelberg (2007)
6. Fáry, I.: On straight line representation of planar graphs. Acta Univ. Szeged. Sect. Sci. Math. 11, 229–233 (1948)
7. Koebe, P.: Kontaktprobleme der konformen Abbildung. Berichte Verhande. Sächs. Akad. Wiss. Leipzig, Math.-Phys. Klasse 88, 141–164 (1936)
8. Wagner, K.: Bemerkungen zum Vierfarbenproblem. Jahresber. Deutsch. Math. Verein. 46, 26–32 (1936)
9. Keszegh, B., Pach, J., Pálvölgyi, D.: Drawing Planar Graphs of Bounded Degree with Few Slopes. In: Brandes, U., Cornelsen, S. (eds.) GD 2010. LNCS, vol. 6502, pp. 293–304. Springer, Heidelberg (2011)
10. Jelínek, V., Jelínková, E., Kratochvíl, J., Lidický, B., Tesař, M., Vyskočil, T.: The Planar Slope Number of Planar Partial 3-Trees of Bounded Degree. In: Eppstein, D., Gansner, E.R. (eds.) GD 2009. LNCS, vol. 5849, pp. 304–315. Springer, Heidelberg (2010)
11. Dujmović, V., Eppstein, D., Suderman, M., Wood, D.R.: Drawings of planar graphs with few slopes and segments. Comput. Geom. 38(3), 194–212 (2007)

Fáry's Theorem for 1-Planar Graphs

Seok-Hee Hong[1], Peter Eades[1], Giuseppe Liotta[2], and Sheung-Hung Poon[3]

[1] University of Sydney, Australia
{seokhee.hong,peter.eades}@sydney.edu.au
[2] University of Perugia, Italy
liotta@diei.unipg.it
[3] National Tsing Hua University, Taiwan
spoon@cs.nthu.edu.tw

Abstract. A *plane graph* is a graph embedded in a plane without edge crossings. Fáry's theorem states that every plane graph can be drawn as a straight-line drawing, preserving the embedding of the plane graph. In this paper, we extend Fáry's theorem to a class of *non-planar* graphs. More specifically, we study the problem of drawing *1-plane graphs* with straight-line edges. A 1-plane graph is a graph embedded in a plane with at most one crossing per edge. We give a characterisation of those 1-plane graphs that admit a straight-line drawing. The proof of the characterisation consists of a linear time testing algorithm and a drawing algorithm. Further, we show that there are 1-plane graphs for which every straight-line drawing has exponential area. To the best of our knowledge, this is the first result to extend Fáry's theorem to non-planar graphs.

1 Introduction

Since the 1930s, a number of researchers have investigated planar graphs. A beautiful and classical result, known as *Fáry's Theorem*, asserts that every *plane graph*, that is, every planar topological embedding of a planar graph has a planar straight-line drawing [8].

Since then, many straight-line drawing algorithms for plane graphs have followed [6,13]. In the 1960s, the first algorithm for constructing a planar straight-line drawing was given by Tutte [17]. In 1980s, efficient algorithms for constructing planar straight-line drawing were given by Read [15] and Chiba et al. [2]. In 1990s, de Fraysseix et al. [5] showed that a *quadratic area* planar straight-line grid drawing could be efficiently obtained. Indeed, planar straight-line drawing is one of the most popular drawing conventions in Graph Drawing [6,13].

More recently, researchers have investigated topological graphs that are "almost" planar, in some sense. An interesting example is *1-plane* graphs, that is, topological graphs with at most one crossing per edge. Some mathematical results for 1-plane graphs are known [1,4,14,16]; in particular, Pach and Toth [14] proved that a 1-plane graph with n vertices has at most $4n - 8$ edges, which is a tight upper bound. Korzhik and Mohar [11] proved that testing whether a graph has a 1-plane embedding is NP-complete.

In this paper, we study *straight-line representations* of *1-plane* graphs, and give a characterisation of those 1-plane graphs that admit a straight-line drawing; our results

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 335–346, 2012.

extend Fáry's Theorem. Fundamentally, there are two 1-plane graphs that cannot be drawn with straight-line edges. One is a 1-plane graph consisting of a path of length 3, called the *bulgari* graph (see Fig. 1(a)). The other is a 1-plane graph consisting of two paths of length two, called the *gucci* graph (see Fig. 1(b)).

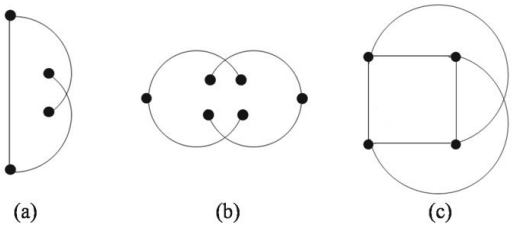


Fig. 1. (a) The bulgari graph; (b) the gucci graph; (c) the bad K4 graph

The following theorem summarises the main results of this paper.

Theorem 1. *A 1-plane graph G admits a straight-line 1-planar drawing if and only if G contains neither the bulgari graph nor the gucci graph*[1]*. Furthermore, there is a linear time testing algorithm to test such conditions, and a linear time drawing algorithm to construct such a drawing if it exists.*

We would like to emphasize that our result is concerned with *topological embeddings* of 1-plane graphs. To illustrate this, note that the underlying graph of bulgari graph is a path of length three and has a straight-line planar drawing. However, the topological embedding of the path of length three, shown in Figure 1(a), has no straight-line 1-planar drawing.

To our best knowledge, Theorem 1 is the first result to extend Fáry's theorem to non-planar graphs. Furthermore, in contrast to many known mathematical results [4,14,16] and hardness results [11] on 1-planar graphs, our results are constructive and algorithmic. The proof of Theorem 1 occupies the remainder of this paper. We also give an *exponential* lower bound for the area of a straight-line 1-planar grid drawing.

Section 3 shows that the absence of the bulgari and gucci graphs is necessary for a straight-line drawing. Sufficiency is established by presenting an algorithm with an augmentation step, in Section 4 and a drawing step, in Section 5. The exponential lower bound on area is given in Section 6.

2 Preliminaries

A *topological graph* $G = (V, E)$ is a representation of a simple graph in the plane where each vertex is a point and each edge is a Jordan curve between the points representing

[1] Note added in proof: The first part of Theorem 1 was independently proved by Thomassen (Rectilinear Drawings of Graphs, *Journal of Graph Theory*, 12 (3), 335-341, 1988).

its endpoints. A *geometric graph* is a topological graph whose edges are represented by straight-line segments.

Two edges *cross* if they have a point in common, other than their endpoints. The point in common is a *crossing*. To avoid some pathological cases, some constraints apply: (i) An edge does not contain a vertex other than its endpoints; (ii) No edge crosses itself; (iii) Edges must not meet tangentially; (iv) No three edges share a crossing.

A *1-planar* graph is a graph that can be drawn in a plane with at most one crossing per edge. A *1-plane* graph is a topological graph with at most one crossing per edge.

Suppose that G is a topological graph. The graph that is obtained by replacing every crossing point of G with a vertex is the *planarisation* of G and is denoted by G^*. The vertices of G^* arising from crossings in G are called *crossing vertices*. From the computational point of view, a 1-plane graph can be represented as a plane embedding of G^*. The algorithms in this paper assume that the input is given in this way, and that constant time adjacency testing can be done (for example, using the data structures in [3]).

The *neighborhood* $N(u)$ of a vertex u in a planar graph is the circular list of vertices adjacent to u, in clockwise order. If G is a topological graph and γ is a crossing vertex of G^*, then the neighborhood $N(\gamma)$ of γ consists of a 4-tuple (a, b, c, d) where the edges (a, c) and (b, d) in G cross at γ.

3 Necessity

We first prove the *necessity* of Theorem 1.

Theorem 2. *Neither the bulgari graph nor the gucci graph admit a straight-line 1-planar drawing. Further, there is a linear time algorithm to test whether a 1-plane graph G contains a bulgari or gucci subgraph.*

Sketch of Proof: Consider the planarisation of the bulgari graph: there is one cycle of length three, and one of the vertices of this 3-cycle is a crossing γ. In any straight-line drawing of the bulgari graph, this 3-cycle forms a triangle, and the interior angle of this triangle at γ must be less than π. However, this interior angle is formed by three of the four angles formed by the edges that cross at γ. This is clearly impossible. A similar argument applies to the gucci graph, using the fact that the four angles in a straight-line quadrilateral add to 2π.

Testing whether a 1-plane graph G contains a bulgari subgraph can be done in linear time by checking the neighborhood of each crossing. Testing for a gucci subgraph is a little more complex. Suppose that edge (a, c) crosses edge (b, d). We consider the subgraph H_{ab} of G^* consisting of all pairs of crossing edges that have endpoints a and b; see Fig. 4(c). One can check for gucci subgraphs in H_{ab} by traversing the clockwise order of edges around a and b; this can be done in time proportional to the sum of the degrees of a and b. Executing this check over all such subgraphs H_{ab} tests for gucci subgraphs in linear time. □

We next prove the *sufficiency* by presenting a drawing algorithm. The overall algorithm consists of two steps: an augmentation step in Section 4 and a drawing step in Section 5.

4 Sufficiency: The Augmentation Algorithm

This Section shows that we can augment a 1-plane graph G by adding edges without crossings, while preserving the straight-line drawability of G.

4.1 Red-Maximal 1-Plane Graphs and Red Augmentation

The edges of a 1-plane graph G that have no crossing are called *red* edges. A *red augmentation* $G' = (V, E')$ of $G = (V, E)$ is a 1-plane graph with $E \subseteq E'$ such that no edge in $E' - E$ has a crossing. A 1-plane graph is *red-maximal* if there is no non-incident pair of vertices a, b that share a face. That is, the addition of any edge makes a crossing. The red-maximal 1-plane graphs have nice properties (see Lemmas 2 and 3), which are helpful for the drawing algorithm in Section 5.

We show how to construct a red-maximal red augmentation of a 1-plane graph, preserving the absence of bulgari and gucci subgraphs.

Theorem 3. *Suppose that G is a 1-plane graph with no bulgari or gucci subgraph. Then there is a red-maximal red augmentation G^+ of G with no bulgari or gucci subgraph. Furthermore, G^+ can be computed in linear time.*

The proof of Theorem 3 consists of an algorithm that adds edges one at a time. The first step, described in Section 4.2, considers pairs of nonadjacent vertices that are endpoints of two edges that cross. The second step, described in Section 4.3, triangulates any remaining faces. The algorithm can be implemented in linear time using appropriate data structures.

4.2 The First Step: Adding Edges around Crossings

The aim of this Section is to present an algorithm that takes a 1-plane graph G with no bulgari or gucci subgraphs, and adds edges until each crossing is surrounded by a 4-cycle. More precisely, we prove the following Lemma.

Lemma 1. *Suppose that $G = (V, E)$ is a 1-plane graph, with no bulgari or gucci subgraph. Then there is a red augmentation G' of G such that G' has no bulgari or gucci subgraph, and for each crossing γ, the neighborhood $N(\gamma)$ of γ in the planarisation G'^* of G' induces a complete subgraph of size 4 in G'.*

The proof of Lemma 1 has two parts. For the first part, we need to define some notation. Suppose that G is a 1-plane graph, and γ is a crossing between edges (a, c) and (b, d) in G. The edge (a, γ) of G^* may occur immediately before (b, γ) in the clockwise order of edges around γ, as in Fig. 2(a); in this case we say that γ is *clockwise with respect to the (ordered) pair* (a, b). In Fig. 3(a), the crossing γ is *anticlockwise with respect to the pair* (a, b).

Further, there may be two ways to add the edge (a, b), as in Fig. 3. One way introduces a bulgari subgraph, the other does not. It is easy to distinguish between these: if γ is anticlockwise with respect to (a, b), then a bulgari subgraph is created if and only if the 3-cycle (a, b, γ) is clockwise, and vice versa. In other words, the augmentation

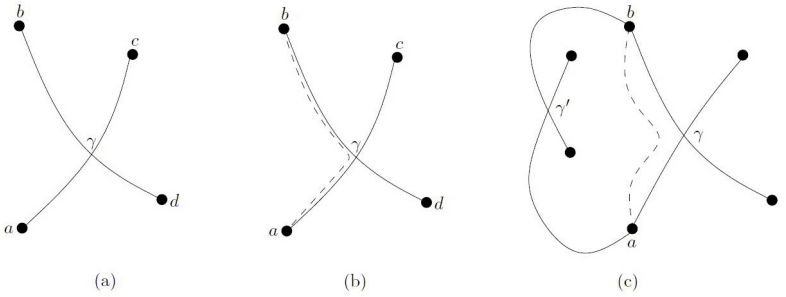


Fig. 2. (a) A crossing; (b) Adding the edge (a, b), following the path (a, γ, b); (c) A 1-plane graph G containing crossing vertices γ and γ'

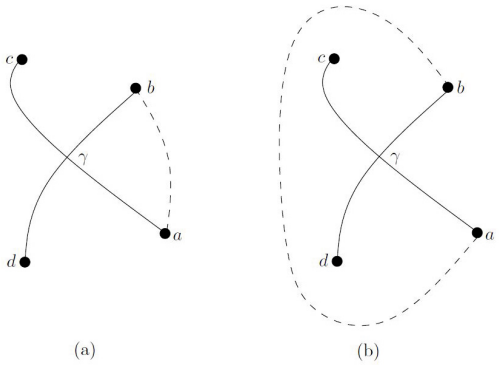


Fig. 3. (a) The crossing γ is anticlockwise with respect to (a, b), and the 3-cycle (a, b, γ) is anticlockwise. (b) The crossing γ is anticlockwise with respect to (a, b), and the 3-cycle (a, b, γ) is clockwise.

avoids a bulgari subgraph if and only if the 3-cycle (a, b, γ) has the same orientation as the crossing γ with respect to (a, b).

We next show that it is always possible to route (a, b) such that the 3-cycle (a, b, γ) has the same orientation as the crossing γ with respect to (a, b), that is, such that the neighborhood of γ does not contain a bulgari subgraph.

Assume without loss of generality that γ is clockwise with respect to (a, b), as in Fig. 2(a). We define a curve $r(a, \gamma, b)$ that is arbitrarily close to the edges (a, γ) and (γ, b), as in Fig. 2(b). Note that $r(a, \gamma, b)$ begins at a such that it is immediately before the edge (a, γ) in the clockwise order of edges around a. At the other end, it is immediately after b in the clockwise order of edges around b. Since G is 1-planar, the curve $r(a, \gamma, b)$ does not cross any edge of G (the edges (a, c) and (b, d) cross at γ and so have no other crossing). We route the edge (a, b) on the curve $r(a, \gamma, b)$; we denote the resulting graph by $G +_\gamma (a, b)$. The following property of $G +_\gamma (a, b)$ is immediate.

Proposition 1. *Suppose that G is a 1-plane graph with no gucci and no bulgari subgraph, and γ is a crossing in G between the edges (a, c) and (b, d). Then $G+_\gamma(a, b)$ is a*

1-plane graph with no gucci subgraph, and the induced subgraph of the neighborhood $N(\gamma)$ of γ does not contain a bulgari subgraph.

Note that Proposition 1 is not enough to prove Lemma 1. There may be another crossing vertex γ' that has a and b as neighbors. The problem is that $G +_\gamma (a, b)$ may contain a bulgari subgraph in the neighborhood of γ', as in Fig. 2(c). Let Γ_{ab} denote the set of crossings that have both a and b as a neighbor. Next we show how to choose $\gamma \in \Gamma_{ab}$ such that $G +_\gamma (a, b)$ does not contain any bulgari subgraph.

Proposition 2. *For every pair a, b of nonadjacent vertices in G such that Γ_{ab} is nonempty, there is a crossing vertex $\gamma \in \Gamma_{ab}$ such that $G +_\gamma (a, b)$ has no bulgari subgraph.*

Sketch of Proof: Let H_{ab} be the subgraph of G^* consisting of all pairs of crossing edges that have endpoints a and b, as in Fig. 4(c). Note that each crossing in Γ_{ab} is either clockwise or anticlockwise with respect to (a, b). Using the fact that G has no gucci subgraph, one can traverse the circular order around a to find a unique pair γ_j, γ_{j+1} of crossings such that γ_j is anticlockwise and γ_{j+1} is clockwise. We can deduce that routing (a, b) on the curve $r(a, \gamma_{j+1}, b)$ (or, equivalently, on the curve $r(a, \gamma_j, b)$) does not introduce a bulgari subgraph. The traversal of edges around a can be executed in time proportional to the degree of a. Performing this operation over all such subgraphs H_{ab} takes linear time. □

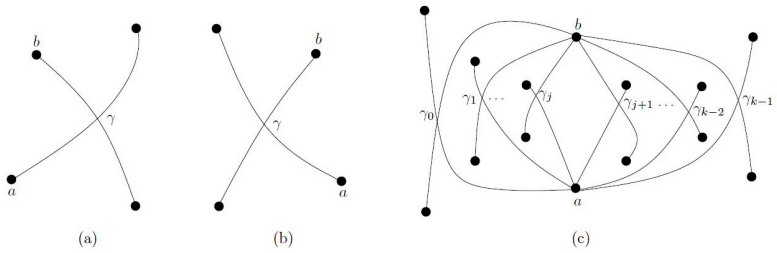


Fig. 4. (a) γ is *clockwise* with respect to the (a, b); (b) γ is *anticlockwise* with respect to the (a, b); (c) H_{ab}

One can repeatedly add edges using the methods defined in the proof of Propositions 1 and 2 to give an augmentation such that each crossing is surrounded by a 4-cycle. No new crossings are introduced, and thus no gucci subgraphs are introduced. Further, by Proposition 2, no bulgari subgraphs are introduced. This completes the proof of Lemma 1.

4.3 The Second Step: Triangulating Remaining Faces

Suppose that a and b are two nonadjacent vertices in the 1-plane graph G' after the first step is applied, as in Lemma 1. Further suppose that a and b share a face f. We can add

the edge (a, b) inside f, without crossing any edge. The graph remains 1-plane. Since a and b were non-adjacent in G', we can deduce from Lemma 1 that Γ_{ab} is empty. Hence no bulgari or gucci subgraph is introduced by adding the edge (a, b). Repeatedly adding edges in this fashion, we can ensure that every face with no crossings is a triangle.

This completes the proof of Theorem 3.

5 Sufficiency: The Drawing Algorithm

In this Section, we present a linear time algorithm for drawing 1-plane graphs with straight-line edges. The input of the drawing algorithm is a red-maximal augmentation G^+ with no bulgari or gucci subgraph.

5.1 Properties of Red-Maximal 1-Plane Graphs

The first result lists some simple properties of red-maximal 1-plane graphs. Informally, the Lemma states that the structure of a red-maximal 1-plane graph is relatively simple; this helps with the drawing algorithm.

Lemma 2. *Let G^+ be a red-maximal 1-plane graph that does not contain a bulgari or gucci subgraph, and let G^* be the planarisation of G^+.*

(a) Every face of G^* is a simple cycle.
(b) If γ is a crossing in G^+ then $N(\gamma)$ induces a 4-clique.
(c) If f is an internal face of G^* with no crossing vertex, then f is a 3-cycle.
(d) If f is an internal face of G^* with a crossing vertex, then f is either a 3-cycle, a 4-cycle or a 5-cycle.
(e) If f is the outer face of G^*, then f has no crossing vertices.
(f) If f is the outer face of G^*, then f is either a 3-cycle or a 4-cycle. If f is a 4-cycle, then it induces a 4-clique with a crossing.
(g) If γ is a crossing between edges (a, c) and (b, d), then there is a path P of red edges from a to b such that the cycle C in G^* formed by the edges (a, γ) and (γ, b), and P contains no vertices strictly inside C.
(h) If γ is a crossing in G^+, then there is a simple cycle C of red edges such that
  (i) The vertices of $N(\gamma)$ appear on C in the same order as their clockwise order around γ, and
  (ii) γ is inside C.

Connectivity plays an important role in our drawing algorithm. It is straightforward to observe that a red-maximal 1-plane graph is one-connected. Using Lemma 2, we can prove a stronger result.

Lemma 3. *Let G^+ be a red-maximal 1-plane graph that does not contain a bulgari or gucci subgraph, and let G_r be the subgraph of red edges of G^+. Then both G^+ and G_r are biconnected.*

5.2 Decomposition of Biconnected Graphs into Triconnected Components

From Lemma 3, the red-maximal 1-plane graph G^+ output by the augmentation process is biconnected; it is a simple exercise to show that there are red-maximal 1-plane graphs that are not triconnected. Thus we use the SPQR tree [7], which represents a decomposition of biconnected graphs into triconnected components, to draw G^+. In this paper, we use a slight modification of the SPQR tree without Q-nodes, called the *SPR tree*. We first define basic terminology.

Each node ν in the SPR tree is associated with a graph called the *skeleton* of ν, denoted by $\sigma(\nu)$. There are three types of nodes ν in the SPR tree: (i) S-node: $\sigma(\nu)$ is a simple cycle with at least 3 vertices; (ii) P-node: $\sigma(\nu)$ consists of two vertices connected by at least 3 edges; (iii) R-node: $\sigma(\nu)$ is a simple triconnected graph.

We treat the SPR tree as a rooted tree by choosing a node ν^* as its root. Let ρ be the parent of ν. The graph $\sigma(\rho)$ has exactly one *virtual edge* e in common with $\sigma(\nu)$. We denote the graph formed from $\sigma(\nu)$ by deleting its parent virtual edge as $\sigma^-(\nu)$. When G is a plane graph, $\sigma(\nu)$ and $\sigma^-(\nu)$ are plane graphs with embedding induced from G.

5.3 Algorithm for Constructing a Straight-Line 1-Planar Drawing

We now present the main theorem of this Section.

Theorem 4. *Let G^+ be a red-maximal 1-plane graph with no bulgari or gucci subgraph. Then there is a linear time algorithm to construct a straight-line 1-planar drawing of G^+.*

Proof. We present a divide-and-conquer algorithm using the SPR tree. The overall approach is similar to that used for star-shaped drawings [9]. However, our algorithm is much simpler due to the nice properties described in Lemma 2. We first describe the basic process, then give details.

We use the SPR tree of the red subgraph G_r of G^+. We choose the root to be the node ν^* whose skeleton contains the vertices on the outer face. We recursively compute a straight-line drawing of G^+ in a top-down manner, using the SPR tree rooted at ν^*.

Roughly speaking, we recursively process each node ν in the SPR tree as follows. First we construct a drawing D_ν of $\sigma(\nu)$ in a given convex polygon P_ν. Then we re-insert crossing edges in the corresponding face of D_ν with straight-line edges. Finally, for each child μ of ν, we define a convex polygon P_μ and replace the virtual edges in D_ν with a drawing of $\sigma(\mu)$. This process continues until we reach the leaf nodes.

We use the *convex drawing algorithm* of Chiba et al. [2] as a subroutine. This algorithm takes a convex polygon P_ν and the plane graph $\sigma(\nu)$ as input, and produces a straight-line *convex* drawing D_ν of $\sigma(\nu)$, that is, each face of D_ν is a convex polygon. Since each face of D_ν is a convex polygon, we can re-insert the crossing edges as straight-lines, without introducing any new crossings.

In fact, we process each node ν differently, based on its type. For R-nodes and root S-node, we construct a convex drawing D_ν of $\sigma(\nu)$, then insert the crossing edges, then define a drawing area $P\mu$ for each child node μ of ν. For P-nodes and non-root S-nodes, the main task is to define a drawing area for each child node μ and a convex polygon P_μ for $\sigma(\mu)$.

Lemma 2 shows that the skeleton $\sigma(\nu)$ has a relatively simple structure. Thus, we can define the convex polygon P_ν for ν as follows: (i) R-node ν: either a triangle or a rhombus (see Figure 5); (ii) S-node ν: either a triangle or a trapezoid.

Next we describe details of the drawing algorithm for each type of node ν.

(i) R-node ν: We first construct a convex drawing D_ν of $\sigma(\nu)$ for the root node (or $\sigma(\nu)^-$ for non-root node) with outer boundary P_ν, noting that $\sigma(\nu)$ (resp., $\sigma(\nu)^-$) is a triconnected (resp., internally-triconnected) plane graph. If $\nu = \nu^*$ is the root, then we choose P_{ν^*} to be a triangle, since the outer face of $\sigma(\nu^*)$ is a 3-cycle. Otherwise, we define a convex polygon P_ν with one of the following three shapes:

- left triangle (see Figure 5(a)) or right triangle (see Figure 5(b)): We use this shape when the edge $e = (s, t)$, which corresponds to the separation pair (s, t), exists as a *real* edge. In this case, we define P_ν using the outer face of $\sigma(\nu)$ including e.
- rhombus shape (see Figure 5(c)): We use this shape, when $e = (s, t)$ is a *virtual* edge introduced in the decomposition. In this case, we define P_ν^- using the outer face of $\sigma(\nu)^-$ without e.

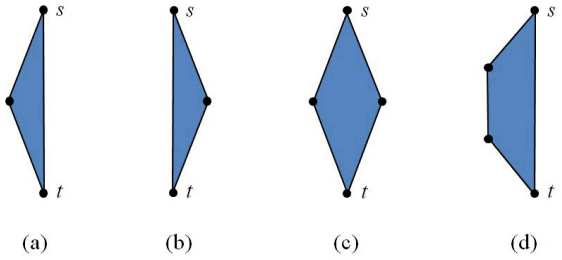


Fig. 5. Shape of P_ν: (a) left triangle; (b) right triangle; (c) rhombus; (d) left trapezoid

Next we re-insert the crossing edges in the corresponding face in D_ν as straight-line segments. Since the faces of D_ν are convex polygons, the drawing remains 1-planar. After inserting crossing edges, we proceed to define a drawing region and a convex polygon P_μ for drawing $\sigma(\mu)$ of each child node μ recursively.

(ii) S-node ν: If $\nu = \nu^*$ is the root, from Lemma 2, the outer face of $\sigma(\nu^*)$ is a 3-cycle or a 4-cycle. Thus we draw $\sigma(\nu^*)$ as a triangle or a rectangle. Next we re-insert the crossing edges, if $\sigma(\nu^*)$ is a 4-cycle. Then we define a drawing region for each child node μ recursively.

If ν is a non-root node, then we first draw $\sigma(\nu)^-$ as a path. Then, the main task is to define a drawing area and a convex polygon P_μ for drawing $\sigma(\mu)$ of each child node μ_i recursively.

(iii) P-node ν: Here the main task is to define a drawing area and a convex polygon P_μ for drawing $\sigma(\mu)$ of each child node μ_i recursively. For R-node child μ, we define P_μ as either a triangle or a rhombus. For S-node child μ, we define P_μ as either a triangle or a trapezoid.

Suppose that vertices s and t form the separation pair for ν, and that the virtual edges between s and t are $u_1, u_2, \ldots, u_m$, in left-right order as in Fig. 6(a). Denote the corresponding children of ν as $\mu_1, \mu_2, \ldots, \mu_m$.

First, consider the case where there is a real edge between s and t; suppose that e occurs between u_k and u_{k+1} in left-right order. The polygons P_{μ_i} must be drawn in a specific order. We first draw $\sigma(\mu_1)$ in the polygon P_{μ_1}, and re-insert crossing edges to form the drawing D_{μ_1}. Then, we can define a drawing area for $\sigma(\mu_2)$ with a convex polygon P_{μ_2}; we choose P_{μ_2} so that it does not overlap with any edge already drawn in D_{μ_1}. Then we re-insert crossing edges in the drawing D_{μ_2}. We repeat this process until we process $\sigma(\mu_k)$. Similarly, we can process $\mu_{k+1}, \mu_{k+2}, ..., \mu_m$ symmetrically beginning with μ_m and working toward μ_{k+1}.

More specifically, based the ordering, we define a *left* triangle (or a left trapezoid) for $\mu_1, \mu_2, ..., \mu_k$, and a *right* triangle (or a right trapezoid) for $\mu_{k+1}, \mu_{k+2}, ..., \mu_m$ to avoid edge crossings. Figure 6 shows an example.

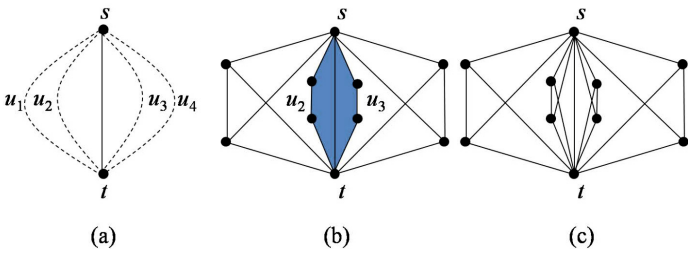


Fig. 6. Defining drawing areas for the children of a P-node

Now consider the case where there is no real edge between s and t. If there is a crossing edges between the two consecutive child nodes μ_k and μ_{k+1} of a P-node, then we add the real edge e in the ordering between between u_k and u_{k+1}, and treat the case as described above.

We now briefly discuss the correctness and time complexity of the algorithm. It is clear that the resulting drawing D^+ is a straight-line drawing of G^+ that retains the same topological embedding as G^+. Since the convex drawing algorithm draws $\sigma(\nu)$ of R-nodes ν with straight-line edges such that each face is drawn as convex polygon, we can re-insert the crossing edges in the corresponding face with straight-lines, without introducing any further crossings.

Since we define a drawing area and a convex polygon P_μ for the boundary of $\sigma(\mu)$, for each child node μ of ν, after the crossing edge re-insertion step, we can draw each $\sigma(\mu)$ without introducing any new crossings. Note that there are no crossing edges between $\sigma(\mu_i)$ and $\sigma(\mu_j)$, where μ_i and μ_j are children of ν, since otherwise they are connected by the corresponding 4-cycle of the crossing edges, due to Lemma 2.

When we replace each virtual edge, which corresponds to a child μ of ν, in the convex drawing D_ν of $\sigma(\nu)$, we can define a convex polygon P_μ for the boundary of $\sigma(\mu)$ thin enough not to create any new crossings.

It is clear that the overall algorithm runs in linear time, since the SPR tree can be constructed in linear time [7], and the convex drawing algorithm by Chiba et al. [2] runs in linear time. This completes the proof of Theorem 4. □

6 Lower Bound on Area

In this Section we present a 1-plane graph, illustrated in Fig. 7, for which any straight-line 1-planar drawing has exponential area.

Theorem 5. *For all $k > 1$, there is a 1-plane graph G_k with $2k$ vertices and $2k - 2$ edges such that any straight-line 1-planar drawing in which of G_k every vertex has integer coordinates has area at least 2^{k-1}.*

Sketch of Proof: One can show that in any straight-line drawing of the graph G_k illustrated in Fig. 7, the area of the triangle $(\gamma_j, a_{j+1}, b_{j+1})$ is at least twice the area of the triangle $(\gamma_{j+1}, a_{j+2}, b_{j+2})$, for $1 \leq j \leq k - 2$. □

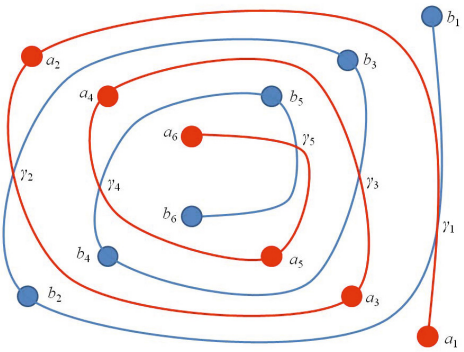


Fig. 7. A 1-plane graph G_k for which every straight-line 1-plane grid drawing has exponential area. The case $k = 6$ is illustrated.

7 Final Remarks

The algorithm presented in this paper takes a 1-plane graph G as input and produces a straight-line 1-planar drawing of G if such a drawing exists. The drawings produced have exponential area; we show that this is unavoidable. The algorithm extends Fáry's Theorem; indeed, if a plane graph is input, then our algorithm produces a straight-line planar drawing. As such, it opens the way for the investigation of a straight-line drawings for a number of other classes of graphs that are "almost" planar. For example, a *quasi-plane graph* is a topological embedding of a graph in which no three edges mutually cross. Is there an algorithm for computing a straight-line drawing of a quasi-plane graph?

References

1. Borodin, O.V., Kostochka, A.V., Raspaud, A., Sopena, E.: Acyclic colouring of 1-planar graphs. Discrete Applied Mathematics 114(1-3), 29–41 (2001)
2. Chiba, N., Yamanouchi, T., Nishizeki, T.: Linear time algorithms for convex drawings of planar graphs. In: Progress in Graph Theory, pp. 153–173. Academic Press (1984)
3. Chrobak, M., Eppstein, D.: Planar Orientations with Low Out-degree and Compaction of Adjacency Matrices. Theor. Comput. Sci. 86(2), 243–266 (1991)
4. Fabrici, I., Madaras, T.: The structure of 1-planar graphs. Discrete Mathematics 307(7-8), 854–865 (2007)
5. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica 10(1), 41–51 (1990)
6. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice-Hall (1999)
7. Di Battista, G., Tamassia, R.: On-line planarity testing. SIAM J. on Comput. 25(5), 956–997 (1996)
8. Fáry, I.: On straight line representations of planar graphs. Acta Sci. Math. Szeged 11, 229–233 (1948)
9. Hong, S., Nagamochi, H.: An algorithm for constructing star-shaped drawings of plane graphs. Comput. Geom. 43(2), 191–206 (2010)
10. Hopcroft, J.E., Tarjan, R.E.: Dividing a graph into triconnected components. SIAM J. on Comput. 2, 135–158 (1973)
11. Korzhik, V.P., Mohar, B.: Minimal Obstructions for 1-Immersions and Hardness of 1-Planarity Testing. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 302–312. Springer, Heidelberg (2009)
12. Kuratowski, K.: Sur le problme des courbes gauches en topologie. Fund. Math. 15, 271–283 (1930)
13. Nishizeki, T., Rahman, M.S.: Planar Graph Drawing. World Scientific (2004)
14. Pach, J., Toth, G.: Graphs drawn with few crossings per edge. Combinatorica 17(3), 427–439 (1997)
15. Read, R.C.: A new method for drawing a planar graph given the cyclic order of the edges at each vertex. Congr. Numer. 56, 31–44 (1987)
16. Suzuki, Y.: Optimal 1-planar graphs which triangulate other surfaces. Discrete Mathematics 310(1), 6–11 (2010)
17. Tutte, W.T.: How to draw a graph. Proc. of the London Mathematical Society 13, 743–767 (1963)

Constant Time Enumeration of Bounded-Size Subtrees in Trees and Its Application

Kunihiro Wasa[1], Yusaku Kaneta[1,⋆], Takeaki Uno[2], and Hiroki Arimura[1]

[1] IST, Hokkaido University, Sapporo, Japan
{wasa,y-kaneta,arim}@ist.hokudai.ac.jp
[2] National Institute of Informatics, Tokyo, Japan
uno@nii.jp

Abstract. By the motivation to discover patterns in massive structured data in the form of graphs and trees, we study a special case of the k-subtree enumeration problem, originally introduced by (Ferreira, Grossi, and Rizzi, ESA'11, 275-286, 2011), where an input graph is a tree of n nodes. Based on reverse search technique, we present the first constant delay enumeration algorithm that lists all k-subtrees of an input tree in $O(1)$ worst-case time per subtree. This result improves on the straightforward application of Ferreira et al's algorithm with $O(k)$ amortized time per subtree when an input is restricted to tree. Finally, we discuss an application of our algorithm to a modification of the graph motif problem for trees.

1 Introduction

By emergence of massive structured data in the form of trees and graphs, there have been increasing demands on efficient methods that discovers many of interesting patterns or regularity hidden in collections of structured data [1,2,13,14]. For instance, the proximity pattern mining problem [8,9] is a class of such pattern discovery problems, where an algorithm is requested to find all collections of items satisfying proximity constraints in a given discrete structure. For example, the proximity string search problem [9] and the graph motif problem [5,8] are popular examples of such proximity pattern discovery problems.

In this paper, we consider the *k-subtree enumeration problem*, which is originally introduced by Ferreira, Grossi, and Rizzi [6], where an instance consists of an undirected graph G of n nodes and a positive integer $k \geq 1$, and the task is to find all k-subtrees, a connected and acyclic node subsets consisting of exactly k nodes in G. Ferreira *et al.* [6] presented the first output-sensitive algorithm that lists all k-subtrees in a graph G of size n in $O(sk)$ total time and $O(n)$ space, in other words, in $O(k)$ *amortized time* per subtree, where n is the number of edges of an input graph and s is the number of solutions. However, it has been an open question whether there exists a faster enumeration algorithm that solves this problem.

⋆ Presently at Rakuten Research, Tokyo, Japan.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 347–359, 2012.

As a main result of this paper, we present the first *constant delay enumeration algorithm* for the k-subtree enumeration problem in *trees*. More precisely, our algorithm lists all k-subtrees of an input tree T of size n in constant worst-case time per subtree using $O(n)$ preprocessing and space. Our algorithm is based on reverse search technique, proposed by Avis and Fukuda [3], as in the algorithm by Ferreira *et al.* [6] for general input graphs. However, unlike their algorithm [6], our algorithm achieves the best possible enumeration complexity. Finally, we discuss an application of our algorithm to a modification of the graph motif problem for trees.

1.1 Related Work

The k-subtree enumeration problem considered in this paper is closely related to a well-known graph problem of enumerating all spanning trees in an undirected graph G [11]. For this problem, Tarjan and Read [11] first presented an $O(ns)$ time and $O(n)$ space algorithm in 1960's, where n is the number of edges in G. Recently, Shioura, Tamura, and Uno [10] presented $O(n+s)$ time and $O(n)$ space algorithm. Unfortunately, it is not easy to extend the algorithms for spanning tree enumeration to subtree enumeration.

One of our motivation comes from application to the graph motif problem (GMP, for short). Given a bag of k labels, called a pattern, and an input graph G, called a text, GMP asks to find a k node subgraph of G whose multi-set of labels is identical to P. Lacroix *et al.* [8] introduced the problem with application to biology and presented an FPT algorithm with $k = O(1)$, and NP-hardness in general. Then, Fellows *et al.* [5] showed that the problem is NP-hard even for trees of degree 3, and presented an improved FPT algorithm. Sadakane *et al.* [9] studied the string version of GMP, and presented linear-time algorithms.

Although there are increasing number of studies on GMP [5,8], there are few attempts to apply efficient enumeration algorithms to this problem. Ferreira, Grossi, and Rizzi [6] mentioned above is one of such studies. Recent studies [2,13,14] in data mining applied efficient enumeration algorithms to discovery of interesting substructures from massive structured data in the real world.

1.2 Organization of This Paper

In Sec. 2, we define basic definitions on the k-subtree problem. In Sec. 3, we first introduces a family tree, and in Sec. 4, then, we present a constant delay algorithm that solves the k-subtree enumeration problem. Sec. 5 gives an application to the graph motif problem. Finally, in Sec. 6, we conclude.

2 Preliminaries

In this section, we give basic definitions and notation for trees and their subtrees. For the definitions not found here, please consult textbooks, e.g., [4]. For a set S, we denote by $|S|$ the number of elements in S. In this paper, all graphs are simple (without self-loops or parallel edges).

Trees. A *rooted tree* is a directed connected acyclic graph $T = (V(T), E(T), \mathsf{root}(T))$, where $V = V(T)$ is a set of *nodes*, $E = E(T) \in V^2$ is a set of *directed edges*, and $\mathsf{root}(T) \in V$ is a distinguished node, called the *root* of T. For each directed edge $(u, v) \in E$, we call u the *parent* of v and v a *child* of u. We assume that every node v except the root has the unique parent. The *size* of T is denoted by $|T| = |V(T)| = n$. We say that nodes u and v are *siblings* each other if they have the same parent. For each node v, we denote the unique parent of v by $\mathsf{pa}(v)$, and the set of all children of a node u by $Ch(v) = \{\, w \in V \mid (v, w) \in E \,\}$.

We define the ancestor-descendant relation $\preceq$ as follows: For any pair of nodes u and $v \in V$, if there is a sequence of nodes $\pi = (v_0 = u, v_1, \cdots, v_k = v)$ $(k \geq 0)$, where $(v_{i-1}, v_i) \in E$ for every $i = 1 \ldots k$, then we define $u \preceq v$, and say that u is an *ancestor* of v, or v is a *descendant* of u. If $u \preceq v$ but $u \neq v$, then and u is a *proper ancestor* of v, denoted by $u \prec v$. , or v is a *proper descendant* of u. For any node v, we denote by $T(v)$ the *set of all descendants* of v in T.

DFS-Numbering. In this paper, we regard an input tree T of size $n \geq 0$ as an ordered tree as follows. We first assume an arbitrary fixed ordering among siblings. Then, we number all nodes of T from 1 to n by the *DFS-numbering*, which is the preorder numbering in the depth-first search [4] on nodes in T. In what follows, we identify the node and the associated node number, and thus, write $V = \{1, \cdots, n\}$. Thus, we can write $u \leq v$ (resp. or $u < v$) if the numbering of u is smaller or equal to (resp. smaller than) that of v. As a basic property of a DFS-numbering, we have the next lemma.

Lemma 1. *For any $u, v \in V$, the DFS-numbering on T satisfies the following properties (i), (ii), and (iii):*

(i) If v is a proper descendant of u, i.e., $u \prec v$, then $u < v$ holds.
(ii) If v is a properly younger sibling of u, then $u < v$ holds.
(iii) Suppose that $x \not\preceq y$ and $y \not\preceq x$. For any nodes x', y' such that $x' \succeq x$ and $y' \succeq y$, $x < y$ implies that $x' < y'$.

The Family of k-Subtrees. Let $1 \leq k \leq n = |T|$. A *k-subtree* in a tree T is a connected and acyclic subgraph of T, as an undirected graph, consisting of exactly k nodes. Since T is a tree, any connected subgraph is obviously acyclic, and thus, it is completely specified by its node set. Therefore, if it is clearly understood, we often identify a connected node set S such that $|S| = k$ with the k-subtree, where S is *connected* if its induced subgraph $T(S)$ is connected. In what follows, we denote by $\mathcal{S}_k = \mathcal{S}_k(T)$ the *family of all k-subtrees* of T.

For a k-subtree S in T, we denote by $\mathsf{root}(S)$ and $L(S)$ the root and the set of leaves of S, respectively. The *border set* is the set $B(S) = Ch(S) \setminus S = \{\, y \in Ch(x) \mid x \in S, y \notin S \,\}$ that consists of all nodes lying immediately outside of S. $L(S)$ and $B(S)$ are anti-chain of nodes in T w.r.t. the ancestor-descendant relation $\preceq$. We define the *interior* and *exterior* of S, respectively, by $Int(S) = S \setminus L(S)$ and $Ext(S) = T(\mathsf{root}(S)) \setminus (S \cup B(S))$. We can easily see that interior, leaves, border, and exterior are mutually disjoint subsets of $T(\mathsf{root}(S))$. For a subset $A \subseteq S$, we define the *head* and the *tail* of S by the elements $\min(A)$ and $\max(A)$, respectively. We define the *weight* of a k-subtree S by the sum

$w(S) = \sum_{v \in V(S)} v$, of the DFS numbers of the nodes in S, where w is bounded from below by $w_{\min} = \frac{1}{2}k(k+1) \geq 0$.

Next, we introduce a class of subtrees in special form, called serial trees as follows. Let S be any k-subtree. Then, S is *serial* if it is *serial* in their shape, that is, whose nodes are consecutively numbered from its root r to the rightmost leaf $r+k-1$, and thus, denoted by $\mathsf{sertree}(v,k) = \{\, r+i \mid i = 0, \ldots, k-1 \,\}$. S is *non-serial* if it is not serial.

Lemma 2 (DFS-numbering lemma). *For any k-subtree S in T, then*

(a) If S is non-serial, then there is some $\min(S) < v < \max(S)$ such that $v \in B(S)$.
(b) If S is non-serial, then $B(S) \neq \emptyset$ and $\min(B(S)) < \max(L(S))$ hold.
(c) If S is serial and $B(S) \neq \emptyset$, then $\max(L(S)) < \min(B(S))$ holds.

Proof. (a) If S is non-serial, then there is some $v \in V(T) \setminus S$ such that $\min(S) < v < \max(S)$. We can find some $v' \in B(S)$ such that $\min(S) < v' < \max(S)$ and v' is an ancestor of v. Furthermore, if we take the smallest such v, then $v' = \min(B(S))$. (b) Since $\max(L(S)) = \max(S)$ holds, the result follows from Claim (a). (c) If S is serial, there is no border node between $\min(S)$ and $\max(S)$. Since any border node is below $\mathsf{root}(S)$, it is properly larger than $\max(S)$. ■

Enumeration Algorithms. We introduce terminology for enumeration algorithms according to Goldberg [7] and Uno [12]. An *enumeration algorithm* for an enumeration problem Π is an algorithm $\mathcal{A}$ that receives an *instance* I and outputs all *solutions* S in the answer set $S(I)$ into a write-only output stream O without duplicates. Let $n = ||I||$, $m = |S(I)|$ be the input and the output size on I. We say that $\mathcal{A}$ is of *amortized constant time* if the total running time of $\mathcal{A}$ for computing all solutions on I is linear in m. For a polynomial $p(\cdot)$, $\mathcal{A}$ is of *constant delay* using preprocessing $p(n)$ if the *delay*, which is the maximum computation time between two consecutive outputs, is bounded by a constant $c(n)$ after preprocessing in $p(n)$ time. As a computation model, we adopt the usual RAM [4]. Now, we state our problem below.

Problem 1 (k-subtree enumeration in a tree). Given an input tree T and an integer k, enumerate all the k-subtrees of T.

This problem is a special case of the k-subtree problem, studied by Ferreira, Grossi, and Rizzi [6], where an input graph is a tree. They showed an efficient enumeration algorithm that lists all k-subtrees in $O(k)$ amortized time per subtree for a general class of undirected graphs. Therefore, our goal is to devise an efficient algorithm that lists all k-subtrees in $O(1)$ worst-case time per subtree.

3 The Parent-Child Relationship among k-Subtrees

3.1 Basic Idea: A Family Tree

Our algorithm is designed based on *reverse search* technique by Avis and Fukuda [3]. In the reverse search technique, we define a tree-shaped search route on

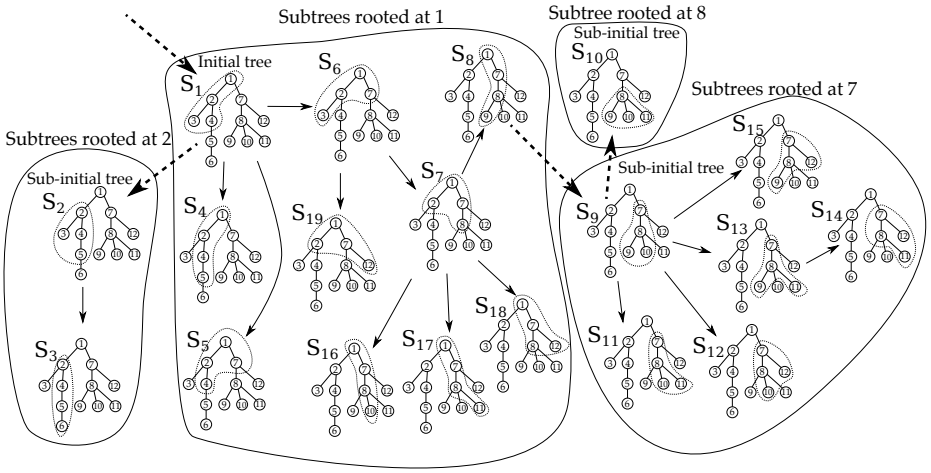


Fig. 1. A family tree for all of nineteen k-subtrees of an input tree T_1 of size $n = 11$, where $k = 4$. In this figure, each set of nodes surrounded by a dotted circle indicates a k-subtree, and each arrow (resp. dashed arrow) indicates the parent-child relation defined by the parent function $\mathcal{P}^1$ of type I (resp. $\mathcal{P}^2$ of type II). We observe that the arrows among each set of subtrees in a large circle indicates a sub-family tree of type I, and the dotted arrows among the set of the initial and sub-initial k-subtrees form the unique sub-family tree of type II.

solutions, called a *family tree*. Let $T = (V(T), E(T), \mathsf{root}(T) = 1)$ be an input tree on $V(T) = \{1, \ldots, n\}$ with DFS-numbering, and let $k \geq 1$ be any positive integer.

A family tree for the class $\mathcal{S}_k(T)$ of all k-subtrees in T is a spanning tree $\mathcal{F}_k(T) = (\mathcal{S}_k(T), \mathcal{P}_k, \mathcal{I}_k(T))$ over $\mathcal{S}_k(T)$ as node set. Given a family tree $\mathcal{F}_k$, we can enumerate all solutions using backtracking starting from the root $\mathcal{I}_k$. In what follows, we omit the subscript k and T, and thus write $\mathcal{F}$ or $\mathcal{F}(T)$ for $\mathcal{F}_k(T)$ if it is clear from context.

In the family tree, its root node is given by the unique serial tree $\mathcal{I}(T) = \mathsf{sertree}(1, k)$, called the *initial tree*, whose node set ranges from 1 to k. The collection of *reverse edges* is given by a function $\mathcal{P} : \mathcal{S}(T) \setminus \{\mathcal{I}\} \to \mathcal{S}(T)$, called the *parent function*, that assigns the unique parent $\mathcal{P}(S)$ to each child k-subtree S except the initial tree. Precise definitions of $\mathcal{I}$ and $\mathcal{P}$ will be given later.

Example 1. In Fig. 1, we show an example of a family tree for all of nineteen k-subtrees of an input tree T_1 of size $n = 11$, where $k = 4$.

A basic idea of our algorithm is to partition the search space $\mathcal{S}(T)$ into almost mutually disjoint subspaces $\mathcal{S}(T) = (\cup_{r \in V(T)} \mathcal{S}^1(T, r)) \cup \mathcal{S}^2(T)$, where elements in collections $\mathcal{S}^1(T, r)$ and $\mathcal{S}^2(T)$ are called k-subtrees *of type I and type II*, respectively. Then, we can separately build family trees for each collections of k-subtrees.

More precisely, as we will see later, for any node r of T, called a *subroot*, the collection $\mathcal{S}^1(T, r)$ consists of all k-subtrees S whose root is r. We refer to

such k-subtrees as *r-rooted* k-subtrees. For the collection $\mathcal{S}^1(T,r)$, we can define a *sub-family tree*, denoted by $\mathcal{F}^1(T,r) = (\mathcal{S}^1(T,r), \mathcal{P}^1, \mathcal{I}^1(T,r))$, which will be given by introducing the parent function $\mathcal{P}^1$ for non-serial subtrees in Sec. 3.3. Interestingly, the initial tree $\mathcal{I}^1(T,r)$ of this collection, called the *sub-initial tree*, is naturally determined to be the serial k-subtree rooted at r by the property of $\mathcal{P}^1$ to be shown in Lemma 5 of Sec. 3.3. On the other hand, the collection $\mathcal{S}^2(T)$ of type II consists of the sub-initial trees of all collections $\cup_r \mathcal{S}^1(T,r)$ of type I, which actually are all serial k-subtrees in T. Then, the unique *sub-family tree*, denote by $\mathcal{F}^2(T) = (\mathcal{S}^2(T), \mathcal{P}^2, \mathcal{I}^2(T))$, for collection $\mathcal{S}^2(T)$ will be given by the parent function $\mathcal{P}^2$ for serial subtrees in Sec. 3.4.

3.2 Traversing k-Subtrees

We efficiently traverse between two k-subtrees, R and $S \in \mathcal{S}_k(T)$. Suppose we are to visit S from R. Then, we first delete a leaf $\ell \in L(R)$ from R, and next, add a border node $\beta \in B(R)$ to R. Unfortunately, this construction is not always sound, meaning that, sometimes, a certain combination of ℓ and β violates the connectivity condition on S. The next technical lemma precisely describes when this degenerate case happens and how to avoid it.

Lemma 3 (connectivity). *Let R be any k-subtree R of size $k \geq 2$. Suppose that $\ell \in R$ and $\beta \notin R$ are any nodes of T that are properly below* $\mathsf{root}(R)$. *Then, the set $S = (R \setminus \{\ell\}) \cup \{\beta\}$ is k-subtree iff $\ell \in L(R)$, $\beta \in B(R)$, and $\beta \notin Ch(\ell)$.*

Then, the next technical lemma is useful in showing identity.

Lemma 4 (identity). *Let R and $S \subseteq V(T)$ be two k-subtrees. If we take $S = (R \setminus \{\ell\}) \cup \{\beta\}$ and $\hat{R} = (S \setminus \{\hat{\beta}\}) \cup \{\hat{\ell}\}$ for some nodes $\ell \in R, \beta \notin R, \hat{\ell} \notin S$, and $\hat{\beta} \in S$, then we have that $R = \hat{R}$ holds iff $\ell = \hat{\ell}$ and $\beta = \hat{\beta}$ hold.*

3.3 A Sub-family Tree for k-Subtrees of Type I

Firstly, for each subroot $r \in V(T)$, we describe how to build a sub-family tree $\mathcal{F}^1(T,r)$ for the subspace $\mathcal{S}^1(T,r)$ of all r-rooted k-subtrees of type I. Suppose that $|T(r)| \geq k$. Then, the corresponding sub-family tree $\mathcal{F}^1(T,r) = (\mathcal{S}^1(T,r), \mathcal{P}^1, \mathcal{I}^1)$ is given as follows. The node set is the collection $\mathcal{S}^1(T,r)$. The sub-initial tree $\mathcal{I}^1 = \mathcal{I}^1(T,r)$ is given as a serial tree containing r as its root. Actually, such a serial tree is uniquely determined by the k-subtree $\mathcal{I}^1$ consisting of k nodes $\{\, r+i \,|\, i = 0, \ldots, k-1 \,\}$.

Next, we give the parent function $\mathcal{P}^1$ from $\mathcal{S}^1(T,r) \setminus \{\mathcal{I}^1\}$ to $\mathcal{S}^1(T,r)$ as follows.

Definition 1 (the parent of k-subtree of type I). Let $S \in \mathcal{S}^1(T,r) \setminus \{\mathcal{I}^1\}$ be any non-serial k-subtree rooted at $r \in V$. Then, the *parent* of S is the k-subtree $\mathcal{P}^1(S) = (S \setminus \{\ell\}) \cup \{\beta\}$ obtained from S by deleting a node $\ell \in L(S)$ and adding a node $\beta \in B(S)$ satisfying the conditions that $\ell = \max(L(S))$ and $\beta = \min(B(S))$. Then, we say that S is a type-I child of $\mathcal{P}^1(S)$.

Table 1. Change of the membership of nodes w.r.t. $\mathcal{P}^1$ and Child_1 operations, where I, L, B and E are regions of k-subtree S, called interior, leaves, border, and exterior, respectively. In this figure, the leaf ℓ_* (resp. the border node β) in R corresponds to the border node β (resp. the leaf ℓ) in S.

(a) From child S to parent R via $\mathcal{P}^1$, where ℓ_* is removed and β_* is added.

node	R	S	node	R	S
$\mathsf{pa}(\beta_*)$	L	I	$\mathsf{pa}(\ell_*)$	I	L
β_*	B	L	ℓ_*	L	B
$Ch(\beta*)$	E	B	$Ch(\ell*)$	B	E

(b) From parent R to child S via Child_1, where ℓ is removed and β is added.

node	R	S	node	R	S
$\mathsf{pa}(\ell)$	I	L	$\mathsf{pa}(\beta)$	L	I
ℓ	L	B	β	B	L
$Ch(\ell)$	B	E	$Ch(\beta)$	E	B

Lemma 5. *If $S \in \mathcal{S}^1(T,r) \setminus \{\mathcal{I}^1\}$, then $\mathcal{P}^1(S)$ is uniquely determined, and a well-defined k-subtree of T. Furthermore, $w(\mathcal{P}^1(S)) < w(S)$ holds.*

Proof. Since S is non-serial, $\beta < \ell$ from Lemma 2. We have $\beta \notin Ch(\ell)$ because if we assume that $\beta \in Ch(\ell)$ then $\ell < \beta$ from Lemma 1, and thus the contradiction is derived. It immediately follows from (iii) of Lemma 3 that $\mathcal{P}^1(S)$ is connected. Since $\beta < \ell$ again, we have $w(\mathcal{P}^1(S)) = w(S) - \ell + \beta < w(S)$. ■

From Lemma 5, it is natural to have $\mathcal{I}^1(T,r)$ as the sub-initial tree of $\mathcal{S}^1(T,r)$. The next lemma describes what happens when we apply $\mathcal{P}^1$ to S.

Lemma 6 (update of lists). *Let S be any non-serial k-subtree and $R = \mathcal{P}^1(S)$. Then, (1) S and R satisfy the conditions in Table 1 (a) before and after application of $\mathcal{P}^1$. (2) The second maximum element in $L(S)$ becomes* $\max(L(R))$. *(3) The second minimum element in $B(S)$ becomes* $\min(B(R))$.

For example, in Fig. 1, we observe that subtree S_6 is the parent of S_7 of type I since the maximum leaf is $\ell = 8$ and the minimum border node is $\beta = 3$, where $L(S_7) = \{2, 8\}$ and $B(S_7) = \{3, 4, 9, 10, 11, 12\}$.

3.4 A Sub-family Tree for k-Subtrees of Type II

To enumerate the whole $\mathcal{S}(T)$, it is sufficient to compute the r-rooted k-serial tree $\mathcal{I}^1(T,r)$ for each possible subroot r in T, and then to enumerate $\mathcal{S}^1(T,r)$ starting from $\mathcal{I}^1(T,r)$. We see, however, that this approach is difficult to implement in constant delay because it is impossible to compute $\mathcal{I}^1(T,r)$ from scratch in the constant time. To overcome this difficulty, we want to directly build a sub-family tree $\mathcal{F}^2(T)$ of a collection $\mathcal{S}^2(T)$ of all sub-initial trees for collections $\cup_r \mathcal{S}^1(T,r)$.

The sub-family tree is given by $\mathcal{F}^2(T) = (\mathcal{S}^2(T), \mathcal{P}^2, \mathcal{I}^2)$, where $\mathcal{S}^2(T)$ is the collection, $\mathcal{I}^2(T) = \mathsf{sertree}(1,k)$ is the initial tree that is the unique serial tree with root 1, and $\mathcal{P}^2$ is the parent function from $\mathcal{S}^2(T,r) \setminus \{\mathcal{I}^2\}$ to $\mathcal{S}^2(T,r)$. Then, we define function $\mathcal{P}^2$ as follows.

Definition 2 (the parent of k-subtree of type II). Let S be any serial k-subtree other than $\mathcal{I}^2$. Then, the *parent* of S is the k-subtree $\mathcal{P}^2(S) = (S \setminus \{\ell\}) \cup \{\beta\}$ obtained from S by deleting the node $\ell = \max(L(S))$ and adding the node $\beta = \mathsf{pa}(\mathsf{root}(S))$. Then, we say that S is a type-II child of $\mathcal{P}^2(S)$.

Lemma 7. *If $S \in \mathcal{S}^2(T) \setminus \{\mathcal{I}^2\}$, then $\mathcal{P}^2(S)$ is uniquely determined, and a well-defined k-subtree of T. Furthermore, $w(\mathcal{P}^2(S)) < w(S)$ holds.*

Proof. If S is not the initial tree, β is always defined. Since ℓ is a leaf of S, (i) of Lemma 3 shows that $S' = (S \setminus \{\beta\})$ is connected. Since β is adjacent to $\mathsf{root}(S)$, clearly, $\mathcal{P}^2(S) = (S' \cup \{\ell\})$ is also connected. Since $\beta < v$ for any node v in S, we have $w(\mathcal{P}^2(S)) = w(S) - \ell + \beta < w(S)$. ■

For example, in Fig. 1, we observe that subtree S_8 is the parent of S_9 of type II since the max. leaf is $\ell = 10$ and the parent of the root is $\beta = 1$, where $L(S_9) = \{9, 10\}$ and $B(S_9) = \{11, 12\}$.

3.5 Putting Them together

Now, we define the *master family tree* $\mathcal{F}(T) = (\mathcal{S}_k(T), \mathcal{P}_k, \mathcal{I}_k)$, for the class $\mathcal{S}_k(T)$ of all k-subtrees in an input tree T. Let the master initial tree $\mathcal{I}_k$ be the initial tree $\mathcal{I}^2$, and let the master parent function $\mathcal{P}$ be the union of two parent functions $\mathcal{P}^1$ and $\mathcal{P}^2$ defined in the previous subsections.

Theorem 1. *The master family tree $\mathcal{F}(T)$ forms a spanning tree over $\mathcal{S}(T)$.*

Proof. Suppose that starting from any $S \in \mathcal{S}_k(T)$, we repeatedly apply the parent function $\mathcal{P}_k$ to S. Then, we have a sequence of k-subtrees $S_0(= S), S_1, \ldots, S_i, \ldots$, where $i \geq 0$. From Lemma 5 and Lemma 7, the corresponding properly decreasing sequence of $w(S_0) > w(S_1) > \cdots > w(S_i) > \cdots$ has at most finite length since $w(S_i) \geq 0$. Since any subtree other than $\mathcal{I}_k$ has the unique parent, the above sequence of k-subtrees eventually reaches the master subtree $\mathcal{I}_k$ in finite time. ■

For example, in Fig. 1, we see that the family tree $\mathcal{F}(T_1)$ is a spanning tree on $\mathcal{S}(T_1)$ rooted at $\mathcal{I}_k = S_1$. From Theorem 1 above, we can enumerate all k-subtrees in $\mathcal{S}_k$ starting from S_1 by depth-first search on $\mathcal{F}_k$ using backtracking.

4 The Constant Delay Enumeration Algorithm

In this section, we present an efficient backtracking algorithm that enumerates all k-subtrees of an input tree T in $O(1)$ delay using $O(n)$ preprocessing. The remaining task is to invert the reverse edges in $\mathcal{P}$ to compute the children from a given parent. We describe this process according to the types of a child S.

4.1 Generation of Non-serial k-Subtrees

We first consider the case that a child S is non-serial (*Type I*). In our algorithm, we keep these nodes as pointers to nodes in the implementation.

Definition 3. We define the candidate sets $\mathsf{DelList}(R)$ and $\mathsf{AddList}(R)$ for deleting nodes ℓ and adding nodes β, respectively, as follows: $\mathsf{DelList}(R) = \{\ell \in L(R) \mid \ell < \max(B(R))\}$, $\mathsf{AddList}(R) = \{\beta \in B(R) \mid \beta > \min(L(R))\}$.

Definition 4 (child generation of type I). Given a k-subtree R in T, we define the k-subtree $\mathsf{Child}_1(R, \ell, \beta) = (R \setminus \{\ell\}) \cup \{\beta\}$ for (i) any $\ell \in \mathsf{DelList}(R)$ and (ii) any $\beta \in \mathsf{AddList}(R)$ such that (iii) β is not a child of ℓ.

The next lemma describes what happens when we apply Child_1 to R.

Lemma 8 (update of lists). *Let R be any k-subtree and $S = \mathsf{Child}_1(R, \ell, \beta)$ be defined, where a leaf $\ell \in \mathsf{DelList}(R)$ is removed from and a border node $\beta \in \mathsf{AddList}(R)$ is added to R. Then, (1) R and S satisfy the conditions in Table 1 (b) before and after application of Child_1. (2) The leaf ℓ becomes the minimum border node in S. (3) The border node β becomes the maximum leaf in S.*

Now, we show the correctness of $\mathsf{Child}_1(\cdot)$ as follows.

Theorem 2 (correctness of Child_1). *Let R and S be any k-subtree of T and S be non-serial. Then, (1) $R = \mathcal{P}^1(S)$, if and only if (2) $S = \mathsf{Child}_1(R, \ell, \beta)$ for (i) some $\ell \in \mathsf{DelList}(R)$ and (ii) some $\beta \in \mathsf{AddList}(R)$ such that (iii) $\beta \notin Ch(\ell)$.*

Proof. Firstly, we can easily obtain a statement that $\mathsf{Child}_1(R, \ell, \beta)$ is non-serial from Lemma 2 and Lemma 3. (1) $\Rightarrow$ (2): Suppose that $R = \mathcal{P}^1(S)$. Then, R is obtained from S by removing $\ell_* = \max(L(S))$ and $\beta_* = \min(B(S))$. From Lemma 6, we see that $\max(L(R)) < \ell_*$ and $\beta_* < \min(B(R))$. If we put $\beta = \ell_*$ and $\ell = \beta_*$, then we can show that β and ℓ are a border node and a leaf in R, respectively, that satisfy the pre-condition of Child_1 in Def. 4. Therefore, we can apply $\mathsf{Child}_1(R, \ell, \beta)$, and then, we obtain the new child from R by removing $\ell = \beta_*$ and adding $\beta = \ell_*$ from R. From Lemma 4, the child is identical to the original subtree S. (2) $\Rightarrow$ (1): In this direction, we suppose that $S = \mathsf{Child}_1(R, \ell, \beta)$ for some $\ell \in L(R)$ and $\beta \in B(R)$ satisfying the conditions (i)–(iii). Then, S is obtained from R by removing ℓ from and adding β to R. From Lemma 8, ℓ becomes $\min(B(R))$ and β becomes $\max(L(R))$. Thus, if we put $\beta_* = \ell$ and $\ell_* = \beta$, then β_* and ℓ_* satisfies the pre-condition of $\mathcal{P}^1$ in Def. 1. By applying $\mathsf{Child}_1(R, \ell, \beta)$, we obtain S from R by removing $\ell_* = \beta$ from and adding $\beta_* = \ell$ to R. From Lemma 4, the child is identical to the original subtree S. Hence, the result is proved. ∎

4.2 Generation of Serial k-Subtrees

Next, we consider the special case to generate a serial subtree as a child k-subtree S of a given parent k-subtree (*Type II*). A k-subtree R is a *pre-serial subtree* if (i) $\mathsf{root}(R)$ has only one child v such that $|T(v)| \geq k$, and (ii) v satisfies that $R(v)$ is a serial $(k-1)$-subtree of T with root v.

Lemma 9. *R is a pre-serial k-subtree of T if and only if $\mathsf{root}(R)$ has a single child v and the equality $\max(L(R)) = \mathsf{root}(R) + k - 1 = v + k - 2$ holds. Furthermore, we can check this condition in constant time.*

Proof. The result follows from that a pre-serial k-subtree is obtained from a serial $(k-1)$-subtree by attaching the new root as the parent of its root. ∎

Algorithm 1. Constant delay enumeration for all k-subtrees in a tree

```
1: procedure EnumSubTreeMain(T, k)
2:     Input: T: a rooted tree of size n, k: size of subtrees (1 ≤ k ≤ n);
3:     Number the nodes of T by the DFS-numbering;
4:     Compute the initial k-subtree I_k;
5:     Update the related lists and pointers;
6:     EnumSubTreeRec(I_k, T, k);

7: procedure EnumSubTreeRec(S, T, k)
8:     Print S;
9:     for each ℓ ∈ DelList(S) do // See Sec. 4.1
10:        for each β ∈ AddList(S) such that β ∉ Ch(max(L(S))) do
11:            S ← Child_1(S, ℓ, β) by updating the related lists and pointers;
12:            EnumSubTreeRec(S, T, k);
13:            S ← P^1(S) by restoring the related lists and pointers;
14:    if S is a k-pre-serial tree then // See Sec. 4.2
15:        S ← Child_2(S) by updating the related lists and pointers;
16:        EnumSubTreeRec(S, T, k);
17:        S ← P^2(S) by restoring the related lists and pointers;
```

Definition 5 (child generation of type II). For any pre-serial k-subtree R, we define $S = \mathsf{Child}_2(R) = (R \backslash \{\mathsf{root}(R)\}) \cup \{\beta_*(R)\}$, where $\beta_*(R) = \min(B(R))$.

Theorem 3 (correctness of Child_2). *Let R and S be any k-subtrees of T. Then, the following (1) and (2) hold:*

(1) If R is pre-serial, then (i) $S = \mathsf{Child}_2(R)$ implies (ii) $R = \mathcal{P}^2(S)$.
(2) If S is serial, then (ii) $R = \mathcal{P}^2(S)$ implies (i) $S = \mathsf{Child}_2(R)$.

Proof. From Lemma 2 and Lemma 9, if R is a pre-serial k-subtree, then we have $\beta_*(R) = \max(R) + 1$ and $\mathsf{Child}_2(R)$ is serial. (1) Suppose that $S = \mathsf{Child}_2(R)$ with deleting $r = \mathsf{root}(R)$ and adding $\beta_*(R)$. Since r is the root of $\mathsf{root}(S)$ and $\beta_*(R)$ is the largest node in S, application of $\mathcal{P}^2$ to S yields R. (2) Suppose that R is obtained from S by $\mathcal{P}^2$ with adding the parent r' of $\mathsf{root}(S)$ and deleting $\beta = \max(S)$. Since S is serial, we have $\max(R) = \max(S) - 1 = \beta - 1$ and then $\beta_*(R) = \max(R) + 1 = (\beta - 1) + 1 = \beta$. Thus, we obtain S if we apply $\mathsf{Child}_2(\cdot)$ to R by deleting $\mathsf{root}(R)$ and adding $\beta_*(R)$. This completes the proof. ■

4.3 The Proposed Algorithm

In Algorithm 1, we present the main procedure EnumSubTreeMain and a subprocedure EnumSubTreeRec that enumerates all k-subtrees in an input tree T of size n in constant delay. Starting from $\mathcal{I}(T)$, EnumSubTreeRec recursively computes all child k-subtrees from its parent by the children generation method in this section.

To efficiently find deleting nodes ℓ in $\mathsf{DelList}(R)$ and adding nodes β in $\mathsf{AddList}(R)$ (resp. the parent of $\mathsf{root}(R)$) by Child_1 (resp. Child_2) that satisfy

the conditions (i)–(iii) of Def. 4 (resp. Def. 5), EnumSubTreeRec maintains the lists of nodes $L(R)$ and $B(R)$ in the increasing order of DFS-numbering during the computation. This is done by attaching two pointers $\texttt{prev}_*$ and $\texttt{next}_*$ to each node v in T for implementing doubly-linked lists in addition to the standard pointers in the leftmost child, the rightmost child and right sibling representation of trees [4]. The algorithm also has two pointers $\hat{\ell}$ and $\hat{\beta}$. The pointer $\hat{\ell}$ always points to the maximum leaf of $\mathsf{DelList}(R)$ and the pointer $\hat{\beta}$ points to the minimum border node of $\mathsf{AddList}(R)$, according to Table 1 (b).

Lemma 10 (time complexity of update). *Assuming the above representation, a data structure for the above lists and pointers can be implemented to run in $O(1)$ worst case time per update using $O(n)$ time preprocessing on RAM.*

Proof. Initialization of the data structure is done in $O(n)$ time by once traversing an input tree T. At each request for update, we dynamically redirect pointers $\texttt{prev}_*$ and $\texttt{next}_*$ when a single node or a node list is deleted or added to R to maintain the values of $L(S), B(S), \hat{\ell}$, and $\hat{\beta}$ according to Table 1 (b) of Lemma 8 in the case of Child_1. We can use a similar procedure to maintain lists and pointers in the case of Child_2. Under this assumption, these operations can be implemented in the claimed complexity. ■

We have the main theorem of this paper.

Theorem 4 (constant delay algorithm for k-subtree enumeration). *Given an input rooted tree T of size n, and a positive integer $k \geq 1$, Algorithm 1 solves the k-subtree enumeration problem in constant worst-case time per subtree using $O(n)$ preprocessing and space.*

Proof. By the construction of EnumSubTreeRec in Algorithm 1, we observe that each iteration of recursive call generates at least one solution. To achieve constant delay enumeration, we need a bit care to represent subtrees and to perform recursive call. From Lemma 10, each call performs constant number of update operations when it expand the current subtree to descendants. Therefore the remaining thing is to estimate the book-keeping on backtrack. This is done as follows: When a recursive procedure call is made, we apply a constant number of operations on candidate lists and record them on a stack as in Lemma 10, and when the procedure comes back to the parent subtree, we apply the inverse of the recorded operations on the lists in constant time as in Lemma 10 to reclaim the running state in constant time. To improve the $O(d)$ time output overhead with backtrack from node v of depth $d = O(n)$ to a shallow ancestor, we use alternating output technique (see, e.g., Uno [12]) to reduce it to exactly $O(1)$ time per solution. Combining the above arguments, we proved the theorem. ■

This result improves on the straightforward application of Ferreira et al's algorithm [6] with $O(k)$ amortized time per subtree when an input is restricted to tree.

5 Application to the Graph Motif Problem for Trees

We consider the restricted version of graph motif problem where an input graph is a tree. Let $\mathcal{C}$ be a set of colors. The *graph motif problem* is the problem of, given a vertex colored graph $G = (V, E)$ and a multi-set P of colors with total frequency of colors k, to find a k-subtree $S \subseteq V$ whose multi-set of colors $C(S)$ is identical to P. We denote by s the *number of all k-subtrees in a tree T*. From Theorem 4, we have the following result.

Theorem 5. *Given an input tree T of size n, a multi-set P of colors with size m, and a positive integer $k \geq 1$, the graph motif problem for tree is solvable in $O(s + n + m)$ total time using $O(n)$ space.*

Proof. We use a histgram $C : \mathcal{C} \to \mathbf{N}$ for the frequencies of colors. Initially, the algorithm sets the counter value to be $C[c] \leftarrow (-1) \cdot P[c]$ for each color in $O(|\mathcal{C}|)$ time, and also setup Algorithm 1 in $O(n)$ time. Then, the algorithm enumerates all the k-subtrees of T by in $O(1)$ delay. For each enumerated k-subtree S with removed node ℓ (or added node β, resp.), we increment (or decrement, resp.) the counter value $C[c]$ by one according to the color c of the node. We can detect the matching of P at some S by testing if all counter values equal zero in $O(1)$ time with appropriate data structure. Hence, the result is proved. ■

We can easily show that $s = n^{\Theta(k)}$. In the worst case, our algorithm is not faster than a straightforward exhaustive search algorithm with $O(mn^k)$ total time in asymptotic sense. However, in practice, our algorithm can be faster when the number s is much smaller than $n^{\Theta(k)}$ and k is relatively large.

6 Conclusion

In this paper, we studied the k-subtree enumeration problem in rooted trees. As a main result, we presented an efficient algorithm. That solve this problem in constant worst-case time per subtree. We also discussed application to graph motif problem.

References

1. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann (1999)
2. Asai, T., Abe, K., Kawasoe, S., Arimura, H., Sakamoto, H., Arikawa, S.: Efficient substructure discovery from large semi-structured data. In: SDM 2002 (2002)
3. Avis, D., Fukuda, K.: Reverse search for enumeration. Discrete Applied Mathematics 65, 21–46 (1993)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. The MIT Press (2001)
5. Fellows, M., Fertin, G., Hermelin, D., Vialette, S.: Sharp Tractability Borderlines for Finding Connected Motifs in Vertex-Colored Graphs. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 340–351. Springer, Heidelberg (2007)

6. Ferreira, R., Grossi, R., Rizzi, R.: Output-Sensitive Listing of Bounded-Size Trees in Undirected Graphs. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 275–286. Springer, Heidelberg (2011)
7. Goldberg, L.A.: Polynomial space polynomial delay algorithms for listing families of graphs. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, STOC 1993, pp. 218–225. ACM, New York (1993)
8. Lacroix, V., Fernandes, C.G., Sagot, M.-F.: Motif search in graphs: Application to metabolic networks. IEEE/ACM TCBB 3, 360–368 (2006)
9. Sadakane, K., Imai, H.: Fast algorithms for k-word proximity search. IEICE Trans. Fundam. Electron., Comm., and Comp. E84-A(9), 2311–2318 (2001)
10. Shioura, A., Tamura, A., Uno, T.: An optimal algorithm for scanning all spanning trees of undirected graphs. SIAM J. Comput. 26(3), 678–692 (1997)
11. Tarjan, R.E., Read, R.C.: Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. Networks 5(3), 237–252 (1975)
12. Uno, T.: Two general methods to reduce delay and change of enumeration algorithms. Technical Report NII-2003-004E. National Institute of Informatics (2003)
13. Uno, T., Asai, T., Uchida, Y., Arimura, H.: An Efficient Algorithm for Enumerating Closed Patterns in Transaction Databases. In: Suzuki, E., Arikawa, S. (eds.) DS 2004. LNCS (LNAI), vol. 3245, pp. 16–31. Springer, Heidelberg (2004)
14. Zaki, M.J.: Efficiently mining frequent trees in a forest. In: KDD, pp. 71–80 (2002)

External Memory Soft Heap, and Hard Heap, a Meldable Priority Queue

Alka Bhushan* and Sajith Gopalan

Department of Computer Science and Engineering,
Indian Institute of Technology Guwahati,
Guwahati-781039, India
{alka,sajith}@iitg.ac.in
http://www.iitg.ac.in

Abstract. An external memory version of soft heap that we call "External Memory Soft Heap" (EMSH) is presented. It supports `Insert`, `Findmin`, `Deletemin` and `Meld` operations and as in soft heap, it guarantees that the number of corrupt elements in it is never more than ϵN, where N is the total number of items inserted in it, and ϵ is a parameter of it called the error-rate. For $N = O(Bm^{M/2(B+\sqrt{m})})$, the amortised I/O complexity of an `Insert` is $O(\frac{1}{B}\log_m \frac{1}{\epsilon})$, where M is the size of the main memory, B is the size of a disk block and $m = M/B$. `Findmin`, `Deletemin` and `Meld` all have non-positive amortised I/O complexities.

When we choose an error rate $\epsilon < 1/N$, EMSH stays devoid of corrupt nodes, and thus becomes a meldable priority queue that we call "hard heap". The amortised I/O complexity of an `Insert`, in this case, is $O(\frac{1}{B}\log_m \frac{N}{B})$, over a sequence of operations involving N `Insert`s. `Findmin`, `Deletemin` and `Meld` all have non-positive amortised I/O complexities. If the inserted keys are all unique, a `Delete` (by key) operation can also be performed at an amortised I/O complexity of $O(\frac{1}{B}\log_m \frac{N}{B})$. A balancing operation performed at appropriate intervals on a hard heap ensures that the number of I/Os performed by a sequence of S operations on it is $O(\frac{S}{B} + \frac{1}{B}\sum_{i=1}^{S}\log_m \frac{N_i}{B})$, where N_i is the number of elements in the heap before the ith operation.

Keywords: Data Structure, I/O, External Memory Data Structure, Priority Queue.

1 Introduction

A priority queue is a data structure used for maintaining a set S of elements, each with a key drawn from a linearly ordered set K, and typically supports the following operations:

1. `Insert`(S, x): Insert element x into S.
2. `Findmin`(S): Return the element with the smallest key in S.

* Present address: GISE Lab, Department of Computer Science and Engineering, Indian Institute of Technology Bombay, India-400076. abhushan@cse.iitb.ac.in

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 360–371, 2012.

3. `Deletemin`(S): Return the element with the smallest key in S and remove it from S.
4. `Delete`(S, x): Delete element x from S.
5. `Delete`(S, k): Delete the element with key $k \in K$ from S.

Algorithmic applications of priority queues abound [2,7].

Soft heap is an approximate meldable priority queue devised by Chazelle [5], and supports `Insert`, `Findmin`, `Deletemin`, `Delete`, and `Meld` operations. A soft heap, may at its discretion, corrupt the keys of some elements in it, by revising them upwards. A `Findmin` returns the element with the smallest current key, which may or may not be corrupt. A soft heap guarantees that the number of corrupt elements in it is never more than ϵN, where N is the total number of items inserted in it, and ϵ is a parameter of it called the error-rate. A `Meld` operation merges two soft heaps into one new soft heap. This data structure is used in computing minimum spanning trees [6] in the fastest known in-core algorithm for that problem. Soft heap has also been used for finding exact and approximate medians and percentiles optimally, and for approximate sorting [5]. An alternative simpler implementation of soft heap is given by Kaplan and Zwick [9].

Various external memory models have been used to design algorithms and data structures intended to work on large data sets that do not fit in the main memory. We work with the external memory model of Aggarwal and Vitter [1]. This model defines the following parameters: N is the number of input data elements, M is the size of the main memory and B is the size of a disk block. It is assumed that $2B < M < N$. In an I/O operation one block of data is transferred between disk and internal memory. The measure of performance of an algorithm in this model is the number of I/Os it performs. The number of I/Os needed to read (write) N contiguous elements from (to) the disk is $\text{Scan}(N) = \Theta(N/B)$. The number of I/Os required to sort N items is $\text{Sort}(N) = \Theta(\frac{N}{B} \log_{M/B} \frac{N}{B})$ [1]. For all realistic values of N, B, and M, $\text{Scan}(N) < \text{Sort}(N) \ll N$. Let m denote M/B.

I/O efficient priority queues have been reported before [3,8,10]; none of them is a meldable priority queue. A meldable priority queue (MPQ) is presented in literature [4] which is an extension of external memory heap (EMH) given in [8]. The MPQ maintains a sequence of priority queues (EMHs) $PQ_1, PQ_2, \ldots, PQ_l$. But, MPQ does not follow all characteristics of a priority queues and differs from the other priority queues in the following ways: (i) the `Deletemin` operation performed in MPQ may not return the minimum element because it always deletes the minimum element from the last EMH (ii) The `Meld` operation in MPQ can be performed only on the last two EMHs.

In this paper, we present an external memory version of soft heap that permits batched operations and is called as "External Memory Soft Heap" (EMSH). As far as we know, this is the first implementation of soft heap on an external memory model. It supports `Insert`, `Findmin`, `Deletemin` and `Meld` operations. Similar to in-core version of soft heap, it guarantees that the number of corrupt elements in it is never more than ϵN. For $N = O(Bm^{M/2(B+\sqrt{m})})$, the

amortised I/O complexity of an `Insert` is $O(\frac{1}{B}\log_m \frac{1}{\epsilon})$. `Findmin`, `Deletemin` and `Meld` all have non-positive amortised I/O complexities. It is not yet clear to us how EMSH can be used to improve the bound for computing minimum spanning trees. However, other problems mentioned in [5] (e.g., finding exact and approximate medians and percentiles optimally, approximate sorting) can be computed in $O(N/B)$ I/Os using EMSH.

When we choose an error rate $\epsilon < 1/N$, EMSH stays devoid of corrupt nodes, and thus becomes a meldable priority queue that we call "hard heap". The amortised I/O complexity of an `Insert`, in this case, is $O(\frac{1}{B}\log_m \frac{N}{B})$. `Findmin`, `Deletemin` and `Meld` all have non-positive amortised I/O complexities. If the inserted keys are all unique, a `Delete` (by key) operation can also be performed at an amortised I/O complexity of $O(\frac{1}{B}\log_m \frac{N}{B})$. We also show that a balancing operation performed at appropriate intervals on a hard heap ensures that the number of I/Os performed by a sequence of S operations is $O(\frac{S}{B} + \frac{1}{B}\sum_{i=1}^{S}\log_m \frac{N_i}{B})$, where N_i is the number of elements in the heap before the ith operation. Our heap requires some main memory space to perform the operations in claimed amortised costs. Even with such restriction, our hard heap can be used for computing water flow across a terrain using the algorithm given in [4] with same I/O complexity, instead of using the meldable priority queue presented in it.

This paper is organised as follows: Section 2 describes the data structure. The correctness of the algorithm is proved in Section 3. The amortised I/O analysis of the algorithm is presented in Section 4. EMSH with $\epsilon < 1/N$ is discussed in Section 5.

2 The Data Structure

An EMSH consists of a set of trees on disk. The nodes of the trees are classified as follows: (i) a node without a child is a leaf (ii) a node without a parent is a root (iii) a node is internal, if it is neither a leaf, nor a root. Every non-leaf in the tree has at most $\sqrt{m}$ children. Nodes hold pointers to their children.

Every node has a rank associated with it at the time of its creation. The rank of a node never changes. All children of a node of rank k are of rank $k-1$. The rank of a tree T is the rank of T's root. The rank of a heap H is $\max\{\text{rank}(T) \mid T \in H\}$. An EMSH can have at most $\sqrt{m}-1$ trees of any particular rank.

Each element held in the data structure has a key drawn from a linearly ordered set. We will treat an element and its key as indistinguishable. Each instance of EMSH has an associated error-rate $\epsilon > 0$. Define $r = \log_{\sqrt{m}} 1/\epsilon$. Nodes of the EMSH with a rank of at most r are called *pnodes* (for "pure nodes"), and nodes with rank greater than r are called *cnodes* (for "corrupt nodes"). Each *pnode* holds an array that contains elements in sorted order. A tree is a ptree if its rank is at most r, and a ctree otherwise.

We say that a pnode p satisfies the *pnode invariant* (PNI), if

p is a non-leaf and the array in p contains at most $B\sqrt{m}$ and at least $B\sqrt{m}/2$ elements, or

p is a leaf and the array in p contains at most $B\sqrt{m}$ elements.

Note that a pnode that satisfies PNI may contain less than $B\sqrt{m}/2$ elements, if it is a leaf.

Every cnode has an associated doubly linked list of *listnodes*. A cnode holds pointers to the first and last listnodes of its list. The size of a list is the number of listnodes in it. Each listnode holds pointers to the next and previous listnodes of its list; the next (resp., previous) pointer of a listnode l is null if l is the last (resp., first) of its list. Each listnode contains at most $B\sqrt{m}$, and unless it is the last of a list, at least $B\sqrt{m}/2$ elements. The last listnode of a list may contain less than $B\sqrt{m}/2$ elements.

Let s_k be defined as follows:

$$s_k = \begin{cases} 0 & \text{if } k \leq r \\ 2 & \text{if } k = r+1 \\ \lceil \frac{3}{2} s_{k-1} \rceil & \text{if } k > r+1 \end{cases}$$

We say that a cnode c that is a non-leaf and is of rank k satisfies the *cnode invariant* (CNI), if the list of c has a size of at least $\lfloor s_k/2 \rfloor + 1$. A leaf cnode always satisfies CNI.

Every cnode has a *ckey*. For an element e belonging to the list of a cnode v, the ckey of e is the same as the ckey of v; e is called corrupt if its ckey is greater than its key.

An EMSH is said to satisfy the heap property if the following conditions are met: For every cnode v of rank greater than $r+1$, the ckey of v is smaller than the ckey of each of v's children. For every cnode v of rank $r+1$, the ckey of v is smaller than each key in each of v's children. For every pnode v, each key in v is smaller than each key in each of v's children.

For each rank i, we maintain a bucket B_i for the roots of rank i. We store the following information in B_i:

1. the number of roots of rank i in the EMSH; there are at most $\sqrt{m} - 1$ such roots.
2. pointers to the roots of rank i in the EMSH.
3. if $i > r$ and $k = \min\{\text{ckey}(y) \mid y \text{ is a root of rank } i\}$, then all elements of a listnode of the list associated with the root of rank i, whose ckey value is k; this listnode will not be the last of the list, unless the list has only one listnode.
4. if $i \leq r$, then the n smallest of all elements in the roots of rank i, for some $n \leq B\sqrt{m}/2$
5. a pointer suffixmin$[i]$, defined in the next paragraph.

We define the minkey of a tree as follows: for a ptree T, the minkey of T is defined as the smallest key in the root of T; for a ctree T, the minkey of T is the ckey of the root of T. The minkey of a bucket B_i is the smallest of the minkeys of the trees of rank i in the EMSH; B_i holds pointers to the roots of these trees. The suffixmin pointer of B_i points to the bucket with the smallest minkey among $\{B_x \mid x \geq i\}$.

For each bucket, we keep the items in 1, 2 and 5 above, and at most a block of the elements (3 or 4 above) in main memory. When all elements of the block are deleted by `Deletemin`s, the next block is brought in. The amount of main memory needed for a bucket is, thus, $O(B + \sqrt{m})$. As we shall show later, the maximum rank in the data structure, and so the number of buckets is $O(\log_{\sqrt{m}}(N/B))$. Therefore, if $N = O(Bm^{M/2(B+\sqrt{m})})$ the main memory suffices for all the buckets. (See Subsection 2.1).

We do not keep duplicates of elements. All elements and listnodes that are taken into the buckets are physically removed from the respective roots. But these elements and listnodes are still thought of as belonging to their original positions.

A bucket B_i becomes *empty*, when all the elements in it have been deleted.

2.1 The Operations

In this section we discuss the `Insert`, `Deletemin`, `Findmin`, `Meld`, `Sift` and `Fill-Up` operations on EMSH. The first four are the basic operations. The last two are auxiliary. The `Sift` operation is invoked only on non-leaf nodes that fail to satisfy the pnode-invariant (PNI) or cnode-invariant (CNI), whichever is relevant. When the invocation returns, the node will satisfy the invariant. Note that PNI applies to pnodes and CNI applies to cnodes. `Fill-Up` is invoked by the other operations on a bucket when they find it empty.

`Insert`. For each heap, a buffer of size $B\sqrt{m}$ is maintained in the main memory. If an element e is to be inserted into heap H, store it in the buffer of H. The buffer stores its elements in sorted order of key values. If the buffer is full (that is, e is the $B\sqrt{m}$-th element of the buffer), create a new node x of rank 0, and copy all elements in the buffer into it. The buffer is now empty. Create a tree T of rank 0 with x as its only node. Clearly, x is a root as well as a leaf. Construct a new heap H' with T as its sole tree. Create a bucket B_0 for H', set the number of trees in it to 1, and include a pointer to T in it. Invoke `Meld` operation on H and H'.

`Deletemin`. A `Deletemin` operation is to delete and return an element with the smallest key in the EMSH. The pointer suffixmin[0] points to the bucket with the smallest minkey. A `Deletemin` proceeds as in Algorithm 1. Note that if after a `Deletemin`, a root fails to satisfy the relevant invariant, then a `Sift` is not called immediately. We wait till the next `Fill-Up` or `Meld`. (While deletions happen in buckets, they are counted against the root from which the deleted elements were taken. Therefore, a deletion can cause the corresponding non-leaf root node of rank less than r (more than r) to fail the relevant invariant by having less than $B\sqrt{m}/2$ elements (list of size less than $\lfloor s_k/2 \rfloor + 1$)).

Recall that we keep at most a block of $B[i]$'s elements in main memory. When all elements of the block are deleted by `Deletemin` operations, the next block is brought in.

Algorithm 1. Deletemin

Let B_i be the bucket pointed by suffixmin[0];
let e be the smallest element in the insert buffer;
if the key of e is smaller than the minkey of B_i **then**
  delete e from the insert buffer, and
  return e;
end if
if $i \leq r$ **then**
  let e be the element with the smallest key in bucket B_i (every tree in the EMSH has a minkey not smaller than the key of e);
  delete e from B_i;
  if B_i is not empty, **then**
    update its minkey value;
  end if
else
  let x be the root of the tree T that lends its minkey to B_i (the ckey of x is less than or equal to all keys in the pnodes and all ckeys);
  B_i holds elements from a listnode l of x;
  let e be an element from l; delete e from l;
end if
if B_i is empty **then**
  fill it up with an invocation to `Fill-Up`(), and update B_i's minkey value;
end if
update the suffixmin pointers of buckets $B_i, \ldots, B_0$;
return e;

`Findmin`. A `Findmin` returns the same element that a `Deletemin` would. But the element is not deleted from the EMSH. Therefore, a `Findmin` does not need to perform any of the updates that a `Deletemin` has to perform on the data structure.

As it is an in-core operation, a `Findmin` does not incur any I/O.

`Meld`. In the `Meld` operation, two heaps H_1 and H_2 are to be merged into a new heap H. It is assumed that the buckets of the two heaps remain in the main memory.

Combine the input buffers of H_1 and H_2. If the total number of elements exceeds $B\sqrt{m}$, then create a new node x of rank 0, move $B\sqrt{m}$ elements from the buffer into it, leaving the rest behind, create a tree T of rank 0 with x as its only node, create a bucket B'_0, set the number of trees in it to 1, and include a pointer to T in it.

Let $B_{1,i}$ (resp., $B_{2,i}$) be the i-th bucket of H_1 (resp., H_2). Let max denote the largest rank in the two heaps H_1 and H_2. The `Meld` is analogous to the summation of two $\sqrt{m}$-radix numbers of max digits. At position i, buckets $B_{1,i}$ and $B_{2,i}$ are the "digits"; there could also be a "carry-in" bucket B'_i. The "summing" at position i produces a new $B_{1,i}$ and a "carry-out" B'_{i+1}. B'_0 will function as the "carry in" for position 0.

The `Meld` proceeds as in Algorithm 2.

`Sift`. The `Sift` operation is invoked only on non-leaf nodes that fail to satisfy PNI or CNI, whichever is relevant. When the invocation returns, the node will satisfy the invariant. We shall use in the below a procedure called extract that is to be invoked only on cnodes of rank $r+1$, and pnodes, and is defined in Algorithm 3.

Suppose `Sift` is invoked on a node x. This invocation could be recursive, or from `Meld` or `Fill-Up`. `Meld` and `Fill-Up` invoke `Sift` only on roots. Recursive invocations of `Sift` proceed top-down; thus, any recursive invocation of `Sift` on x must be from the parent of x. Also, as can be seen from the below, as soon as a non-root fails its relevant invariant (PNI or CNI), `Sift` is invoked on it. Therefore, at the beginning of a `Sift` on x, each child of x must satisfy PNI or CNI, as is relevant.

If x is a pnode (and thus, PNI is the invariant violated), then x contains less than $B\sqrt{m}/2$ elements. Each child of x satisfies PNI, and therefore has, unless it is a leaf, at least $B\sqrt{m}/2$ elements. Invoke extract(x). This can be done in $O(\sqrt{m})$ I/Os by performing a $\sqrt{m}$-way merge of x's children's arrays. For each non-leaf child y of x that now violates PNI, recursively invoke `Sift`(y). Now the size of x is in the range $[B\sqrt{m}/2, B\sqrt{m}]$, unless all of x's children are empty leaves.

If x is a cnode of rank $r+1$, then CNI is the invariant violated. The children of x are of rank r, and are thus pnodes. There are two possibilities: (A) This `Sift` was invoked from a `Fill-Up` or `Meld`, and thus x has one listnode l left in it. (B) This `Sift` was invoked recursively, and thus x has no listnode left in it. In either case, to begin with, invoke extract(x), and invoke `Sift`(y) for each non-leaf child y of x that now violates PNI. The number of elements gathered in x is $B\sqrt{m}/2$, unless all of x's children are now empty leaves.

Suppose case (A) holds. Create a new listnode l', and store in l' the elements just extracted into x. If l' has a size of $B\sqrt{m}/2$, insert l' at the front of x's list; else if l and l' together have at most $B\sqrt{m}/2$ elements, then merge l' into l; else, append l' at the end of the list, and transfer enough elements from l' to l so that l has a size of $B\sqrt{m}/2$.

If case (B) holds, then if x has nonempty children, once again, extract(x), and invoke `Sift`(y) for each non-leaf child y of x that now violates PNI. The total number of elements gathered in x now is $B\sqrt{m}$, unless all of x's children are empty leaves. If the number of elements gathered is at most $B\sqrt{m}/2$, then create a listnode, store the elements in it, and make it the sole member of x's list; otherwise, create two listnodes, insert them into the list of x, store $B\sqrt{m}/2$ elements in the first, and the rest in the second.

In both the cases, update the ckey of x so that it will be the largest of all keys now present in x's list.

If x is a cnode of rank greater than $r+1$, then while the size of x is less than s_k, and not all children of x hold empty lists, do the following repeatedly: (i) pick the child y of x with the smallest ckey, (ii) remove the last listnode of x and merge it with the last listnode y, if they together have at most $B\sqrt{m}$

Algorithm 2. Meld

for $i = 0$ to max+1 **do**
  if only one of $B_{1,i}$, $B_{2,i}$ and B'_i exists **then**
    that bucket becomes $B_{1,i}$; `Fill-Up` that bucket, if $B_{1,i}$ is empty ;
    there is no carry-out;
  else
    if $i \leq r$ **then**
      if $B_{1,i}$ (resp., $B_{2,i}$) contains elements **then**
        send the elements of $B_{1,i}$ (resp., $B_{2,i}$) back to the roots from which they were taken;
        for each root x pointed by $B_{1,i}$ or $B_{2,i}$ **do**
          if x does not satisfy PNI, invoke `Sift`(x);
        end for
      end if
    else
      if $B_{1,i}$ (resp., $B_{2,i}$) and the last listnode l_1 (resp., l_2) of the root x_1 (resp., x_2) with the smallest ckey in $B_{1,i}$ (resp., $B_{2,i}$) have sizes $< B\sqrt{m}/2$ each, **then**
        merge the elements in $B_{1,i}$ (resp., $B_{2,i}$) into l_1 (resp., l_2);
      else
        store the elements in $B_{1,i}$ (resp., $B_{2,i}$) in a new listnode l and insert l into the list of x_1 (resp., x_2) so that all but the last listnode will have $\geq B\sqrt{m}/2$ elements;
        if x_1 (resp., x_2) does not satisfy CNI, then `Sift` it;
      end if
    end if
  end if
  if the total number of root-pointers in $B_{1,i}$, $B_{2,i}$ and B'_i is $< \sqrt{m}$ **then**
    move all root-pointers to $B_{1,i}$; `Fill-Up` $B_{1,i}$;
    delete $B_{2,i}$ and B'_i; There is no carry-out;
  else
    create a tree-node x of rank $i + 1$;
    pool the root-pointers in $B_{1,i}$, $B_{2,i}$ and B'_i;
    take $\sqrt{m}$ of those roots and make them children of x; `Sift`(x);
    create a carry-out bucket B'_{i+1};
    place in it a pointer to x; this is to be the only root-pointer of B'_{i+1};
    move the remaining root-pointers into $B_{1,i}$; `Fill-Up` $B_{1,i}$;
    delete $B_{2,i}$ and B'_i;
  end if
end for
update the suffixmin pointers;

Algorithm 3. Extract

extract(x)
begin
let N_x be the total number of elements in all the children of x
put together; extract the smallest $\min\{B\sqrt{m}/2, N_x\}$ of those elements and store them in x;
end

Algorithm 4. Fill-Up

```
if i ≤ r then
  for each root x in B_i that does not satisfy PNI do
    Sift(x);
  end for
  Let N_i be the total number of elements in all the roots of B_i put together;
  extract the smallest min{B√m/2, N_i} of those and store them in B_i;
else
  for each root x in B_i that does not satisfy CNI do
    Sift(x);
  end for
  pick the root y with the smallest ckey in B_i;
  copy the contents of one listnode l of y's list (not the last one) into B_i;
  remove l from the list of y.
end if
```

elements, (iii) merge the resultant list of y to the resultant list of x such that all but the last listnode will have at least $B\sqrt{m}/2$ elements, (iv) set the ckey of x to the ckey of y, and (v) invoke `Sift`(y) recursively. If merging of two list nodes (step ii) is not required, then the concatenation merely updates $O(1)$ pointers. Merging, when it is needed, incurs $O(\sqrt{m})$ I/Os.

The `Sift` operation removes all leaves it renders empty. An internal node becomes a leaf, when all its children are removed.

`Fill-Up`. The `Fill-Up` operation is invoked by `Deletemin` and `Meld` on a bucket B_i when those operations find B_i empty. B_i is filled up using the Algorithm 4.

A bucket remembers, for each element e in it, the root from which e was extracted. This is useful when the `Meld` operation sends the elements in the bucket back to their respective nodes.

Even if a `Fill-Up` moves all elements of a root x without children into the bucket, x is retained until all its elements are deleted from the bucket. (A minor point: For $i \leq r$, if the roots in the bucket all have sent up all their elements into the bucket, are without children, and have at most $B\sqrt{m}/2$ elements together, then all of them except one can be deleted at the time of the `Fill-Up`.)

The Memory Requirement. The following lemma establishes the largest rank that can be present in a heap. The proof is by induction on k and is omitted due to lack of space.

Lemma 1. *There are at most* $\frac{N}{B\sqrt{m}^{k+1}}$ *tree-nodes of rank* k *when* N *elements have been inserted into it.*

Therefore, if there is at least one node of rank k in the heap, then $\frac{N}{B\sqrt{m}^{k+1}} \geq 1$, and so $k \leq \log_{\sqrt{m}} \frac{N}{B}$. Thus, the rank of the EMSH is at most $\log_{\sqrt{m}} \frac{N}{B}$. Note that there can be at most $\sqrt{m} - 1$ trees of the same rank.

The main memory space required for a bucket is $O(B + \sqrt{m})$. So, the total space required for all the buckets is $O((B + \sqrt{m}) \log_{\sqrt{m}} \frac{N}{B})$. We can store all buckets in main memory, if we assume that $(B + \sqrt{m}) \log_{\sqrt{m}} \frac{N}{B} = O(M)$. This assumption is valid for all values of $N = O(Bm^{M/2(B+\sqrt{m})})$. Assume the modest values for M and B given in [12]: Then, if $N < 10^{900}$, which is practically always, the buckets will all fit in the main memory.

3 A Proof of Correctness

If the heap order property is satisfied at every node in the EMSH before an invocation of `Sift`(x), then it will be satisfied after the invocation returns too. This can be easily shown and details are omitted due to lack of space.

In every other operation of the EMSH, all data movements between nodes are achieved through `Sifts`. Thus, they too cannot violate the heap order property.

When we note that a `Fill-Up` on a bucket B_i moves into it a set of elements with smallest keys or the smallest ckey from its roots, and that the suffixmin pointer of B_0 points to the bucket with the smallest minkey among $\{B_x \mid x \geq 0\}$, we have the following Lemma.

Lemma 2. *If there is no cnode in the EMSH, then the element returned by* `Deletemin` *will be the one with the smallest key in the EMSH. If there are cnodes, and if the returned element is corrupt (respectively, not corrupt), then its ckey (respectively, key) will be the smallest of all keys in the pnodes and ckeys of the EMSH.*

For all $k > r$, and for every nonleaf x of rank k that satisfies CNI, the size of the list in x is at least $\lfloor s_k/2 \rfloor + 1$. For a root x of rank $k > r$, when the size of its list falls below $\lfloor s_k/2 \rfloor + 1$, `Sift`$(x)$ is not invoked until at least the next invocation of `Fill-Up` or `Meld`.

The following lemmas give an upperbound on the size of the list. Due to lack of space, their proofs are omitted.

Lemma 3. *For all $k > r$, and for every node x of rank k, the size of the list in x is at most $3s_k$.*

Lemma 4. *For all values of $k > r$,*

$$\left(\frac{3}{2}\right)^{k-r-1} \leq s_k \leq 2\left(\frac{3}{2}\right)^{k-r} - 1$$

Using Lemmas 1, 3 and 4, we can prove the following lemma which gives an upperbound on number of corrupted elements at any point of time in an EMSH.

Lemma 5. *If $m > 110$, at any time there are at most ϵN corrupt elements in the EMSH, where N is the total number of insertions performed.*

4 An Amortised I/O Analysis

The amortised complexity of each operation is given in the the following lemma. Due to lack of space, the proof is omitted here.

Lemma 6. *In EMSH, the amortised complexity of an* `Insert` *is* $O(\frac{1}{B}\log_m \frac{1}{\epsilon})$. `Findmin`, `Deletemin` *and* `Meld` *all have non-positive amortised complexities.*

5 Hard Heap: A Meldable Priority Queue

When an EMSH has error-rate $\epsilon = 1/(N+1)$, no element in it can be corrupt. In this case, EMSH becomes an exact priority queue, which we call hard heap. In it every node is a pnode, and every tree is a ptree. `Deletemin`s always report the exact minimum in the hard heap. The height of each tree is $O(\log_m \frac{N}{B})$, as before. But, since all nodes are pnodes, the amortised cost of an insertion is $O(\frac{1}{B}\log_m \frac{N}{B})$ I/Os. The amortised costs of all other operations remain unchanged. The absence of corrupt nodes will also permit us to implement a `Delete` operation in a similar manner as in [3,8,10]. The amortised cost of a `Delete` is $O(\frac{1}{B}\log_m \frac{N}{B})$ I/Os, the same as that of an `Insert`.

The actual cost of a `meld` of two hard heap's with N elements each is $O(\sqrt{m}\log_m \frac{N}{B})$ I/Os; the amortised cost of the `meld` is subzero. But this is the case only if the buckets of both the heaps are in the main memory. Going by our earlier analysis in Section 2.1, if $N = O(Bm^{M/2k(B+\sqrt{m})})$ then k heaps of size N each can keep their buckets in the main memory.

The buckets of the heaps to be melded could be kept in the secondary memory, and brought into the main memory, and written back either side of the meld. The cost of this can be accounted by an appropriate scaling of the amortised complexities. However, operations other than `meld` can be performed only if the buckets are in the main memory.

In hard heap, unlike in the other external memory priority queues, elements move only in the upward direction. That makes hard heap easier to implement. The external memory heap of [8], used in meldable priority queue [4] is a balanced heap, and therefore, incurs a balancing cost. But, in it the number of I/Os performed by a sequence of S operations is $O(\frac{S}{B} + \frac{1}{B}\sum_{i=1}^{S}\log_m \frac{N_i}{B})$, where N_i is the number of elements remaining in the heap before the i-th operation; this is helpful when the `insert`s and `deletemin`s are intermixed so that the number of elements remaining in the data structure at any time is small.

In comparison, hard heap is not a balanced tree data structure. It does not use a costly balancing procedure like the heaps of [8,10]. However, for a sequence of N operations, the amortised cost of each operation is $O(\frac{1}{B}\log_m \frac{N}{B})$ I/Os.

We can make the amortised cost depend on N_i, the number of elements remaining in the hard heap before the i-th operation, at the cost of adding a balancing procedure. In a sequence of operations, whenever N_I, the number of `insert`s performed, and N_D, the number of `deletemin`s performed, satisfy $N_I - N_D < N_I/\sqrt{m}$, and the height of the hard heap is $\log_{\sqrt{m}} \frac{N_I}{B}$, delete the remaining $N_I - N_D$ elements from the hard heap, insert them back in, and set

the counts N_I and N_D to $N_I - N_D$ and zero respectively; we can think of this as the end of an *epoch* and the beginning of the next in the life of the hard heap. The sequence of operations, thus, is a concatenation of several epochs. Perform an amortised analysis of each epoch independently. The cost of reinsertions can be charged to the elements actually deleted in the previous epoch, thereby multiplying the amortised cost by a factor of $O(1 + \frac{1}{\sqrt{m}})$. It is easy to see that now the number of I/Os performed by a sequence of S operations is $O(\frac{S}{B} + \frac{1}{B}\sum_{i=1}^{S} \log_m \frac{N_i}{B})$.

Number of Comparisons in Heap Sort: To sort, insert the N input elements into an initially empty hard heap, and then perform N `deletemin`s. When a node of rank 0 is created, $O(\log_2(B\sqrt{m}))$ comparisons per element are performed. When an elements moves from a node to its parent, it participates in a $\sqrt{m}$-way merge; a $\sqrt{m}$-way merge that outputs k elements requires to perform only $O(\log_2 \sqrt{m})$ comparisons per element. Since the number of levels in the hard heap is at most $\log_{\sqrt{m}} N/B\sqrt{m}$, the total number of comparisons performed by one element is $\log_2 N$. Each `deletemin` operation can cause at most $\log_{\sqrt{m}}(N/\sqrt{mB})$ comparisons among the suffixmin pointers. Thus, the total number of comparisons is $O(N \log_2 N)$.

References

1. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Communications of the ACM 31(9), 1116–1127 (1998)
2. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The design and analysis of computer algorithms, Massachusetts, Addison-Wesley (1974)
3. Arge, L.: The buffer tree: A technique for designing batched external data structures. Algorithmica 37(1), 1–24 (2003)
4. Arge, L., Revsbaek, M., Zeh, N.: I/O-efficient computation of water flow across a terrain. In: Proc. 26th ACM Symposium on Computational Geometry, pp. 403–412 (2010)
5. Chazelle, B.: The soft heap: an approximate priority queue with optimal error rate. Journal of the ACM 47(6), 1012–1027 (2000)
6. Chazelle, B.: A minimum spanning tree algorithm with inverse-Ackermann type complexity. Journal of the ACM 47(6), 1028–1047 (2000)
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press, Cambridge (1990)
8. Fadel, R., Jakobsen, K.V., Katajainen, J., Teuhola, J.: Heaps and heapsort on secondary storage. Theoretical Computer Science 220, 345–362 (1999)
9. Kaplan, H., Zwick, U.: A simpler implementation and analysis of Chazelle's soft heaps. In: Proc. 20th ACM -SIAM Symposium on Discrete Algorithms, pp. 477–485 (2009)
10. Kumar, V., Schwabe, E.: Improved algorithms and data structures for solving graph problems in external memory. In: Proc. 8th IEEE Symp. on Parallel and Distributed Processing, pp. 169–177 (1996)
11. Meyer, U., Zeh, N.: I/O-Efficient Undirected Shortest Paths. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 434–445. Springer, Heidelberg (2003)
12. Mungala, K., Ranade, A.: I/O-complexity of graph algorithms. In: Proc. 10th ACM-SIAM Symposium on Discrete Algorithms, pp. 687–694 (1999)

Partially Specified Nearest Neighbor Search

Tomas Hruz and Marcel Schöngens

Institute of Theoretical Computer Science, ETH Zurich, Switzerland
{tomas.hruz,schoengens}@inf.ethz.ch

Abstract. We study the Partial Nearest Neighbor Problem that consists in preprocessing n points $\mathcal{D}$ from d-dimensional metric space such that the following query can be answered efficiently: Given a query vector $Q \in \mathbb{R}^d$ and an axes-aligned query subspace represented by $S \in \{0,1\}^d$, report a point $P \in \mathcal{D}$ with $\mathrm{d}_S(Q,P) \leq \mathrm{d}_S(Q,P')$ for all $P' \in \mathcal{D}$, where $\mathrm{d}_S(Q,P)$ is the distance between Q and P in the subspace S. This problem is related to similarity search between feature vectors w.r.t. *a subset* of features. Thus, the problem is of great practical importance in bioinformatics, image recognition, etc., however, due to exponentially many subspaces, each changing distances significantly, the problem has a considerable complexity. We present the first exact algorithms for ℓ_2- and ℓ_∞-metrics with linear space and sub-linear worst-case query time. We also give a simple approximation algorithm, and show experimentally that our approach performs well on real world data.

1 Introduction

One of the fundamental problems in the study of index structures is the *Nearest Neighbor (NN) Problem*: For a database $\mathcal{D}$ that contains n points from a d-dimensional metric space, build a data structure such that given a query point $Q \in \mathbb{R}^d$, one can efficiently find a point $P \in \mathcal{D}$ closest to Q. Such a point P is called *closest point* or *nearest neighbor* of Q and closest is meant with respect to the underlying metric. To stay in a well-studied space, in this paper we consider the ℓ_2 and ℓ_∞-metric, though the presented approach also works for other ℓ_p-norms. By $\mathrm{d}(P,Q)$ we denote the distance between two points $P,Q \in \mathbb{R}^d$, and, if not stated otherwise, it refers to the ℓ_2-distance. We understand the points in $\mathcal{D}$ as feature representations, or feature vectors, of real world objects. Hence, a nearest neighbor represents an object similar to the object the query represents.

The Nearest Neighbor Problem has been studied extensively in the last decades. However, in some important practical applications the problem formulation above is too restrictive in the sense that one is often interested in finding a point in $\mathcal{D}$ that is similar to a query with respect *to subset of features*. This leads to the definition of the *Partial Nearest Neighbor (PNN) Problem*: Build a data structure such that for a query point $Q \in \mathbb{R}^d$ together with a query subspace represented by a vector $S \in \{0,1\}^d$, one can efficiently find a point $P = (p_1, p_2, \ldots, p_d) \in \mathcal{D}$ that is closest to $Q = (q_1, q_2, \ldots, q_d)$ with respect to the subspace $S = (s_1, s_2, \ldots, s_d)$. More formally, in Euclidean space

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 372–383, 2012.

we search for a point P with $\mathrm{d}_S(P,Q) \leq \mathrm{d}_S(P',Q)$ for all $P' \in \mathcal{D}$ where $\mathrm{d}_S(P,Q) = \left(\sum_{i=1}^{d} s_i(p_i - q_i)^2\right)^{1/2}$. The subspace represented by S is always orthogonal to the coordinate axes. The dimension of the subspace is denoted by w and we may say a query has w *relevant dimensions*. There are exponentially many subspaces and in general for each subspace the point distances change significantly, which is the main reason for the problem's complexity. Our approach considers small to medium feature space, which is supported by our experimental evaluation that shows good results for up to 64 dimensions (Section 4).

From the application perspective it is often sufficient to find a point in $\mathcal{D}$ that is relatively close to the query point, but not necessarily the closest. This relaxation is captured by the *Approximate Partial Nearest Neighbor (APNN) Problem* in which the objective is to find a point $P \in \mathcal{D}$ with $\mathrm{d}_S(P,Q) \leq (1+\gamma)\mathrm{d}_S(P',Q)$ for all $P' \in \mathcal{D}$ and a predefined $\gamma > 0$. In the case in which we restrict ourselves to the full space, that means $S = (1,1,\dots)$, the problem specializes to the well-known *Approximate Nearest Neighbor Problem* [14].

The PNN problem occurs in many application areas such as bioinformatics, image recognition, business data mining, and many more. A specific example, which was the motivation to study the problem, is knowledge retrieval in gene expression databases, in which biologists are interested in finding genes with a similar response profile to a given gene. In the database, a feature vector represents a gene and a feature represents the gene's activity in a certain anatomical part of the organism like a brain cell, muscle cell, blood cell, etc. The anatomical parts are organized in classes according to an ontology, in which the number of classes is the dimension d of the full feature space (see for example [25,13]). The set of classes according to which the search is made changes dynamically, hence, this problem is essentially the (A)PNN Problem.

For the classical NN Problem worst-case efficient algorithms have been developed in the last decades, but they are still missing for the PNN Problem. A trivial approach is to preprocess a NN data structure for each of the 2^d subspaces, such that the query time is the time needed to find the correct NN data structure plus its query time. This approach has exponential space requirements which especially unpractical for main memory data structures. We focus on data structures that have only linear space requirements. Known data structures for nearest neighbor and range searching were studied assuming constant d, thus the influence of d on the query time is usually not known but believed to be exponential. Obtaining good bounds is sophisticated even in the case $d = 2$ [20]. Hence, in the query time analysis we express the dependence of d as some unknown function $\mathrm{g}(d)$.

Our Results. Space is the most critical resource when dealing with the PNN problem (even for moderate $d \geq 10$). Already for small data sets there is no chance to deal with the brute force' exponential overhead in space, which is not a worst-case but a tight estimate. Consequently, we consider linear space data structures, which come with a worse but still sub-linear query time.

The presented method combines epsilon-nets and range searching data structures to provide an efficient and simple approach to the PNN problem, leading to the first linear space algorithm that for the ℓ_2- and ℓ_∞-metric has a guaranteed worst-case query time sub-linear in n. This result can easily be extended to ℓ_p-metrics for fixed p.

For the Euclidean metric and a d-dimensional database, we obtain a query time of $\mathcal{O}\left(\mathrm{g}(d)n^{1-1/(2d-4)+\delta}\right)$ w.h.p., for $\delta > 0$, and $d > 3$ using $\mathcal{O}(dn)$ space (Lemma 4). On the other hand, the known lower bounds on space decomposition [23,15] give less hope to obtain fundamentally better exact algorithms.

For the ℓ_∞-metric we obtain an exact algorithm that has $\mathcal{O}\left(dn^{1-1/d}\right)$ query time w.h.p. while using $\mathcal{O}(dn)$ space (Lemma 5). The approach is very flexible and can be adapted to any metric that can be described by a constant number of bounded degree polynomials (using Lemma 3), which also includes the ℓ_1-metric.

We present a $\sqrt{w}$-approximation algorithm that has the same complexity as the ℓ_∞-metric case (Lemma 7). This approach is of high practical relevance: An experimental evaluation shows that it is more than 60 times faster than a linear scan while the approximation ratio is only 1.2 on average, and the maximal rank over all queries was always less than 0.025% of the database (Section 4).

2 Related Work

In theory and practice little is known about the PNN Problem, which might be due to the problem complexity that manifests in an exponential number of possible subspaces. There are some heuristics for the problem that perform well on real-world data, but usually they have no sublinear worst-case bound on the query time, or do not guarantee a good solution quality.

The problem was first considered in a paper by Eastman and Zemankova in 1982 [9] in which KD Trees are used as the data structure of choice. The authors prove that for uniformly distributed database points under the maximum norm the expected query time is $\mathcal{O}\left(dn^{1-1/d}\right)$. With the KD tree, our approach achieves the same result under the maximum norm, but for any point distribution.

A theoretical paper by Andoni et. al. [2] treats a special case of the PNN problem, namely, the case if all but one feature are specified. The authors study the problem in high-dimensional space in which the dimension d is not considered constant. A $(1+\gamma)$-approximation can be obtained in time $\mathcal{O}\left(d^3 n^{0.5+\delta}\right)$ using $\mathcal{O}\left(d^2 n^{c(1/\gamma^2+1/\delta^2)}\right)$ space for any $\delta > 0, \gamma > 0$, and a constant c. The result makes use of the LSH scheme [14], which is a very successful approach for the classical NN Problem in high-dimensional space. However, the approach has not been extended to smaller relevant dimensions than $w = d - 1$. In contrast our approach works for all w in an axes-parallel setting.

In [5] the authors develop and experimentally examine a heuristic method to solve the PNN Problem in the ℓ_p-metric using R-trees (resp. R*-trees). The algorithm uses an ordered active page list (APL) which contains the R-tree nodes (enclosing rectangles) sorted in ascending order according to the distance between query and the nearest edge of the enclosing rectangle. This is essentially

a min-heap, which is initialized with the tree's root node. The algorithm iteratively pops the first node in the APL and pushes the node's children into the APL if their distance to the query is smaller than the current candidate nearest neighbor. The algorithm provides accurate results and it is fast in many cases as the experiments in [5] show, however, it cannot provide a worst-case time complexity better than a linear scan.

A heuristic for partial range searching, which is related to the PNN Problem, was proposed in [16] in which the authors study partial vector approximation files (VA-files) [22]. One dimensional VA-files with variable interval distribution are constructed with the consequence that the variable intervals allow to order the VA-files according to the so-called sensitivity. The sensitivity is defined as number of VA interval end-points lying in the search range. It allows to correct results in many situations. This heuristic has no guarantee on the accuracy of the results, nor a sublinear worst-case guarantee on the query time.

3 Our Approach

We show that a combination of epsilon-nets and range searching can be used to obtain the first exact algorithm for the PNN Problem. Furthermore, the algorithm can be modified such that one obtains better query time for the APNN Problem. The algorithm is based on the concepts of epsilon-nets and range spaces: A *range space* $\mathfrak{S}$ is a 2-tuple $(\mathcal{X}, \mathfrak{R})$, where $\mathcal{X}$ is a usually infinite set and $\mathfrak{R}$ is a collection of subsets of $\mathcal{X}$. The elements of $\mathcal{X}$ are called points and the elements of $\mathfrak{R}$ are called *ranges*. For example, a range could be an axes-parallel, in some dimensions unbounded, hyper-rectangle $\mathcal{H}_{L,R} = \{(x_1, x_2, \ldots, x_d) \mid l_i \leq x_i \leq r_i\}$, represented by two vectors $L = (l_1, l_2, \ldots, l_d)$ and $R = (r_1, r_2, \ldots, r_d)$, where $l_i, r_i \in \mathbb{R} \cup \{-\infty, \infty\}$. Let us denote the set of all hyper-rectangles by $\mathfrak{H}$ so that we can specify the range space $(\mathbb{R}^d, \mathfrak{H})$. To simplify notation, we always denote a vector from $\mathbb{R}^d$ by a capital letter and its components by the lowercase.

An important measure for the complexity of a range space is its VC-dimension: A range space $(\mathcal{X}, \mathfrak{R})$ has *VC-dimension* h if there exists a subset $X \subset \mathcal{X}$ of maximal cardinality h such that $\{\mathcal{R} \cap X \mid \mathcal{R} \in \mathfrak{R}\}$ equals the power-set of X [11]. For example, consider the range space of all half-spaces in $\mathbb{R}^2$: There exists a 3-point set $\mathcal{P}$ in $\mathbb{R}^2$ such that any subset of $\mathcal{P}$ can be generated by cutting $\mathcal{P}$ with a half-space. This is not possible for any 4-point set in $\mathbb{R}^2$. Hence, the VC-dimension of the half-space range space in $\mathbb{R}^2$ is 3.

Range spaces with small VC-dimension have the property that for any set $\mathcal{D} \subset \mathcal{X}$ there exist a small subset that is sensitive to large ranges. Such sets are called epsilon-nets: For a finite n-point set $\mathcal{D} \subset \mathcal{X}$ of a range space $(\mathcal{X}, \mathfrak{R})$, a subset $\mathcal{N}$ of $\mathcal{D}$ is called *epsilon-net* for a parameter $\varepsilon \leq 1/2$ if for any range $\mathcal{R} \in \mathfrak{R}$ with $|\mathcal{R} \cap \mathcal{D}| \geq \varepsilon n$ there is at least one point in $\mathcal{R} \cap \mathcal{N}$. Epsilon-nets were introduced to computational geometry in an influential paper by Haussler and Welzl [11], who show that epsilon-nets of small size exist for range space of small VC-dimension:

Lemma 1 (Epsilon-net Lemma [21]). *Let $(\mathcal{X}, \mathfrak{R})$ be a range space with finite VC-dimension $h \geq 2$. For an absolute constant c, a parameter $\varepsilon \leq 1/2$ and an n-point subset $\mathcal{D}$ of $\mathcal{X}$, there exists an epsilon-net $\mathcal{N}$ of $\mathcal{D}$ for $(\mathcal{X}, \mathfrak{R})$ of size at most $(ch/\varepsilon)\log(1/\varepsilon)$.*

For many natural ranges, such as half-spaces, balls, and ranges defined by small-degree polynomials, the VC-dimension is usually small. Our approach uses properties of epsilon-nets that get emphasized by the following reformulation of their definition: A subset $\mathcal{N} \subset \mathcal{D}$ is an epsilon-net for a range space $(\mathcal{X}, \mathfrak{R})$, if for any range $\mathcal{R} \in \mathfrak{R}$ with $\mathcal{R} \cap \mathcal{N} = \emptyset$, the cardinality of $\mathcal{R} \cap \mathcal{D}$ is bounded by εn.

The *Range Searching Problem* for a range space $(\mathcal{X}, \mathfrak{R})$ and an n-point database $\mathcal{D} \subset \mathcal{X}$ can be formulated as follows: Preprocess $\mathcal{D}$ such that for a query range $\mathcal{R} \in \mathfrak{R}$ one can either efficiently report or count all points in $\mathcal{D} \cap \mathcal{R}$. We refer to these variants as the *Range Reporting Problem* and *Range Counting Problem* respectively. The Range Searching Problem for the range space $(\mathbb{R}^d, \mathfrak{H})$, in which the ranges are axes-parallel hyper-rectangles, is important in theory and practice [8,10], thus one usually refers to it as the *Orthogonal Range Searching Problem*. Range queries that are unbounded in some dimensions are usually called *partial range queries* [10].

NN Search. We now show how to combine epsilon-nets and range reporting to answer PNN queries. We first explain the algorithm for the special case of the NN Problem in Euclidean space, and then adapt the algorithm for the PNN Problem with other metrics.

On an intuitive basis, for a NN query $Q \in \mathbb{R}^d$ we find a nearest neighbor if we can report and check all points in the d-dimensional Euclidean ball $\mathcal{B}^d_{Q,r}$ that is centered at Q and has an appropriate radius r. The problem in this procedure is not the range reporting, since efficient algorithms exist, but the determination of a suitable radius r. On the one hand, if r is too large the performance of the procedure may degenerate to a linear scan. On the other hand, if r is too small, there is no point in the intersection and hence no nearest neighbor is found. The idea to solve this problem is to generate an epsilon-net $\mathcal{N}$ on $\mathcal{D}$ and use the distance of a nearest neighbor in $\mathcal{N}$ to bound r.

We start with the definition of an appropriate range space. A d-dimensional Euclidean ball is defined as $\mathcal{B}^d_{Q,r} = \left\{X \in \mathbb{R}^d \mid \mathrm{d}(X, Q) < r\right\}$ and the set of all Euclidean balls, denoted by $\mathfrak{B}$, equals $\left\{\mathcal{B}^d_{Q,r} \mid Q \in \mathbb{R}^d, r \geq 0\right\}$. For the Euclidean distance the range space $(\mathbb{R}^d, \mathfrak{B})$ is *suitable* with respect to NN search. Note that it is only a technical detail to define the Euclidean balls to be open, which is clarified later. The following scheme is the basis for all subsequent algorithms.

Preprocessing: First, we build a range searching data structure for $(\mathbb{R}^d, \mathfrak{B})$ on $\mathcal{D}$. Secondly, for the range space $(\mathbb{R}^d, \mathfrak{B})$ and a parameter $\varepsilon > 0$ we generate an epsilon-net $\mathcal{N}$ on $\mathcal{D}$. This can be done by random sampling [11,21] or deterministically as described in [6]. The latter algorithm has a running time exponential in the VC-dimension, hence we stick to random sampling which comes at the cost of obtaining Las-Vegas algorithms. The overall preprocessing time is dominated by the preprocessing of the range searching data structure.

Query algorithm: When processing a query $Q \in \mathbb{R}^d$, we compare each point of $\mathcal{N}$ with Q and obtain a nearest neighbor N of the epsilon-net. We set the radius $r = \mathrm{d}(Q, N)$ and ask the range searching data structure to report all points in $\mathcal{B}^d_{Q,r} \cap \mathcal{D}$. These points are finally compared with Q to obtain a nearest neighbor in $\mathcal{D}$. If no points are in this range, then N is not only a nearest neighbor of the epsilon-net, but also of $\mathcal{D}$.

We require an efficient range searching data structure for the range space $(\mathbb{R}^d, \mathfrak{B})$. Using a standard lifting transform [8], in which all points are lifted to the $(d+1)$-dimensional paraboloid, the ball range searching problem transforms to the half-space range searching problem [19]. The most efficient range searching data structure for this problem is presented by Chan [7] and for linear space it achieves a query time of $\mathcal{O}\left(\mathrm{g}(d) n^{1-1/\lfloor d/2 \rfloor}\right)$ w.h.p.. As an intermediate result we obtain a Lemma for the NN Problem.

Lemma 2. *Given an n-point set $\mathcal{D} \subset \mathbb{R}^d$ a nearest neighbor of a query $Q \in \mathbb{R}^d$ can be found in time $\mathcal{O}\left(\mathrm{g}(d) n^{1-1/\lceil d/2 \rceil}\right)$ w.h.p., using $\mathcal{O}(dn)$ space.*

For a proof see [12]. For linear space and $d > 3$, our result already attains the best known query time for NN search [7,19]. The dependence of d should roughly be of the same order since the same techniques are used.

PNN Search. The principle feature of our approach is that it is extensible to the PNN problem. The distance between two points is now measured with respect to the subspace S as defined by d_S. Instead of querying ranges that have the form of balls, we query ranges that have the form $\mathcal{C}_{Q,S,r} = \left\{X \in \mathbb{R}^d \mid \mathrm{d}_S(Q, X) < r\right\}$. Their projection onto S equals $\mathcal{B}^w_{Q,S}$, so we may refer to $\mathcal{C}_{Q,S,r}$ as a *partially defined ball*. For example when $d = 3$ and the dimension of S is $w = 2$, the range corresponds to a cylinder of radius r. Let us define the range space $(\mathbb{R}^d, \mathfrak{C})$ for which $\mathfrak{C} = \left\{\mathcal{C}_{Q,S,r} \mid Q \in \mathbb{R}^d, S \in \{0,1\}^d, r \geq 0\right\}$.

In order to build a range searching data structure for $(\mathbb{R}^d, \mathfrak{C})$ we require an efficient description of the ranges in $\mathfrak{C}$. Though, there are exponentially many subspaces, the ranges in $\mathfrak{C}$ can be described by a single polynomial of the form $\mathrm{f}(x_1, \ldots, x_d, q_1, \ldots, q_d, s_1, \ldots, s_d, r) = \sum_{i=1}^d s_i(q_i - x_i)^2 - r^2 \leq 0$. The corresponding range space has VC-dimension of $\mathcal{O}\left(d^2\right)$ (see [21], Proposition 10.3.2). A range searching data structure that queries the desired ranges is described by Agarwal and Matoušek [1]. We restate a simplified version of this result to show the flexibility of our approach. Combining with the latest results on Tarski-Cell decompositions [15] leads to the following lemma:

Lemma 3 ([1,15]). *Let $\mathrm{f}(x_1, \ldots, x_d, a_1, \ldots, a_p)$ be a $(d+p)$-variate polynomial where d, p, and the degree of f is bounded. Let $(\mathbb{R}^d, \mathfrak{R})$ be a range space with $\mathfrak{R} = \{\mathcal{R}_{A_1,\ldots,A_u} \mid A_1, \ldots, A_u \in \mathbb{R}^u\}$, for a fixed constant u, and $\mathcal{R}_{A_1,\ldots,A_u} = \left\{X \in \mathbb{R}^d \mid \mathrm{f}(X, A_1) \leq 0, \ldots, \mathrm{f}(X, A_u) \leq 0\right\}$. Then, the range searching problem for $(\mathbb{R}^d, \mathfrak{R})$ can be solved with $\mathcal{O}(n)$ space, and $\mathcal{O}\left(n^{1-1/b+\delta}\right)$ query time for $\delta > 0$. The parameter $b = d$ for $d \leq 3$ and $b = 2d - 4$ for $d > 3$.*

This Lemma also allows the construction of a PNN data structure for ℓ_p-metrics with fixed p. We only need to adapt the polynomial f. Note that for

increasing p the degree of the polynomial and thus the VC-dimension increases, and consequently, the query time of the range searching algorithm increases. On the other hand, in the case of the ℓ_∞-metric, the induced partially defined balls are hyper-rectangles, thus, a KD Tree [8] is a suitable structure. The advantage of our algorithm is that it can handle different range searching data structures. Hence, we formulate our main result on the PNN problem in a generic way, more precisely, for a generic metric that has suitable range space $\mathfrak{S}$ of VC-dimension h. For a range searching data structure build on $\mathcal{D}$ which uses $\mathrm{s}_{\mathfrak{S}}(n)$ space, let $\mathrm{t}^*_{\mathfrak{S}}(n)$ be the preprocessing time and let $\mathrm{t}_{\mathfrak{S}}(n,m)$ be the time it takes to report $m = |\mathcal{D} \cap \mathcal{R}|$ points in the intersection of a query range $\mathcal{R}$.

Theorem 1. *Let d_S be a distance function and $\mathfrak{S}$ a suitable range space of VC-dimension h. Then, an n-point set $\mathcal{D} \subset \mathbb{R}^d$ can be preprocessed in time $\mathrm{t}^*_{\mathfrak{S}}(n)$ into a data structure of $\mathrm{s}_{\mathfrak{S}}$ space, such that a PNN query Q, S can be answered in time $\mathcal{O}\left(h\sqrt{n}\log n + \mathrm{t}_{\mathfrak{S}}(n,\sqrt{n})\right)$ w.h.p..*

For a proof see [12]. Theorem 1 and the data structure from Lemma 3 directly imply an algorithm for the Euclidean PNN problem. The VC-dimension of the range space $(\mathbb{R}^d, \mathfrak{C})$ is at most $\mathcal{O}\left(d^2\right)$ (see [21], Proposition 10.3.2).

Lemma 4. *An n-point set $\mathcal{D} \subset \mathbb{R}^d$ from Euclidean space can be preprocessed in time $\mathcal{O}\left(\mathrm{g}(d)n\log n\right)$ into a $\mathcal{O}\left(dn\right)$ space data structure, such that an PNN query Q, S can be answered in time $\mathcal{O}\left(\mathrm{g}(d)n^{1-1/b+\delta}\right)$ w.h.p., for $\delta > 0$ and for $b = d$ if $d \leq 4$ and $b = (2d-4)$ otherwise.*

Opposed to the brute force approach, with $\mathcal{O}\left(2^d n\right)$ space, $\mathcal{O}\left(\mathrm{g}(d)n\log n\right)$ preprocessing time and $\mathcal{O}\left(\mathrm{g}(d)n^{1-1/\lceil d/2\rceil}\right)$ query time, we obtain a worse query time, however, we save space exponentially in d.

As we stated above, the ℓ_∞-metric uses the range space $(\mathbb{R}^d, \mathfrak{H})$, that has VC-dimension $2d$. There are reasonable fast and well-studied data structures for $(\mathbb{R}^d, \mathfrak{H})$ such as KD Trees and Range Trees [8]. If we use a KD Tree for range searching, which reports m points in a partially defined range in time $\mathrm{t}_{(\mathbb{R}^d,\mathfrak{H})}(n,m) = \mathcal{O}\left(dn^{1-1/d} + dm\right)$ [17], and apply Theorem 1 we obtain the following Lemma.

Lemma 5. *Given an n-point set $\mathcal{D} \subset \mathbb{R}^d$ together with the ℓ_∞-metric. There is a data structure of $\mathcal{O}\left(dn\right)$ space, which can be constructed in $\mathcal{O}\left(dn\log n\right)$ time, that answers a PNN query Q, S in time $\mathcal{O}\left(dn^{1-1/d}\right)$ w.h.p..*

Approximate PNN Search. The performance of the range searching algorithm for Euclidean distance (Lemma 3) might not be satisfying for practical applications. Even with more efficient Tarski-cell decompositions, which are the current limiting factor in the range searching data structure, the worst-case time cannot be better than $\mathcal{O}\left(\mathrm{g}(d)n^{1-1/d}\right)$ [23]. Consequently, it might be worth to consider approximation algorithms that work well on real world data. We present a simple and practically efficient $\sqrt{w}$-approximation algorithm that queries partially specified axes-parallel hyper-rectangles instead of $\mathcal{C}_{Q,S,r}$. Better approximation ratios require the study of data structures for approximate partial ball range reporting.

For simplicity, we describe a specific unbounded hyper-rectangle $\mathcal{H}_{Q,S,r}$ with center Q and radius r that is equal to $\mathcal{H}_{L,R}$ with $l_i = q_i - r$ ($r_i = q_i + r$ resp.) if $s_i = 1$ and $l_i = -\infty$ ($r_i = \infty$ resp.) otherwise. As above, an epsilon-net for the range space $(\mathbb{R}^d, \mathfrak{C})$ is denoted by $\mathcal{N}$ and the point of the epsilon-net closest to Q with respect to S is denoted by $N \in \mathcal{N}$. Let $r = \mathrm{d}_S(Q, N)$ be the radius of the desired range $\mathcal{C}_{Q,S,r}$.

The smallest enclosing rectangle $\mathcal{H}_{Q,S,r}$, called *outer cube range*, is not suitable as a query range since its volume grows exponentially faster than the volume of $\mathcal{C}_{Q,S,r}$. Thus, we use the hyper-rectangle $\mathcal{H}_{Q,S,r/\sqrt{w}}$ of maximal size that completely fits inside the desired range $\mathcal{C}_{Q,S,r}$. Let us call this range *inner cube*. The resulting point size is bounded by the epsilon-net, however, we can only get an $\sqrt{w}$-approximation as the subsequent lemma states.

Lemma 6. *Given a point N of an n-point set $\mathcal{D}$, a query point Q with w-dimensional query subspace S, and a radius $r = \mathrm{d}_S(Q, N)$. If the hyper-rectangle $\mathcal{H}_{Q,S,r/\sqrt{w}}$ is queried, then the returned point is a $\sqrt{w}$-approximate PNN of $\mathcal{D}$.*

For a proof see [12]. As a consequence we can obtain a $\sqrt{w}$-approximation data structures for the Euclidean PNN Problem.

Lemma 7. *Given an n-point set $\mathcal{D} \subset \mathbb{R}^d$ from Euclidean space and suppose we use a KD Tree as range searching data structure. For a PNN query Q, S we obtain a $\sqrt{w}$-approximation in time $\mathcal{O}\left(dn^{1-1/d}\right)$ w.h.p., consuming $\mathcal{O}(dn)$ space and using $\mathcal{O}(dn \log n)$ preprocessing time.*

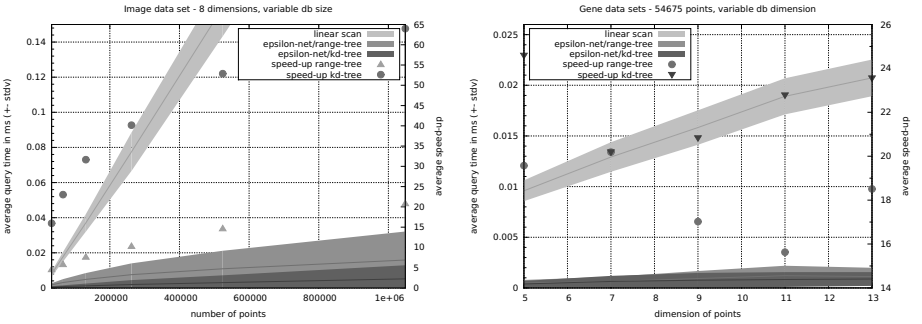


Fig. 1. The average over 10000 queries is shown as straight lines within filled area that represents the standard deviation. The speed-up is shown as bullets and triangles with axes on the right. Left: The query time depending on the database size. Right: The query time depending on the database dimension.

It is also possible to use different, application adapted range searching data structures, as for example the Priority R-tree [4] (or PR-tree) if the goal is an IO-optimal and practically efficient algorithm for memory hierarchies.

4 Experiments

We implemented the proposed APNN algorithm for the Euclidean distance using KD Tree and Range Tree data structure to show that the theoretical foundations are valid and the concept performs well on a wide range of real world data. The performance results also indicate good behavior for the exact PNN data structure using the ℓ_∞-metric, since the same range searching data structures can be used. The implementation was done using C++ in a standard way [8] without any optimizations to exploit architecture related specialties. The PNN query points were generated by randomly perturbing points from the database and, unless stated otherwise, the query subspace $S \subset \{0,1\}^d$ was generated by setting each component to 1 with probability 1/2. For each experiment we performed 10000 queries to get reliable results.

For the experimental evaluation we used data sets from two completely different domains to emphasize the data independence of our approach. The first data set comes from a gene database and contains feature vectors that represent the genes by their activity in multiple experimental settings and were all extracted from the Genevestigator database [13]. A feature represents an anatomical class (category) such as a certain type of neoplasmic cells or cells from different parts of the human brain. An element p_i of the feature vector representing gene $P \in \mathcal{D}$ corresponds to the gene's activity in a cell of class i. The activity is a floating point number in $[0, 100]$. We consider three data sets show in Table 2. We want to emphasize the practical relevance of the query type we use: it is the same type as widely used in the data mining application of Genevestigator.

Table 1. This table summarizes the dependence of $|\mathcal{C}_{Q,S,r} \cap \mathcal{D}|$ for various experimental settings (columns from left: data set number, number of points, number of dimensions, epsilon-net size)

#	n	d	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{C} \cap \mathcal{D}\rvert$ (avg)	(stdv)	(max)
4	2^{14}	8	2^{10}	14.5 (0.09%)	14.7	143 (0.87%)
4	2^{15}	8	2^{10}	30.4 (0.09%)	31.6	284 (0.87%)
4	2^{16}	8	2^{10}	57.8 (0.09%)	59.3	576 (0.88%)
4	2^{17}	8	2^{10}	118.0 (0.09%)	116.2	1181 (0.90%)
4	2^{18}	8	2^{10}	246.8 (0.09%)	250.7	2388 (0.91%)
4	2^{19}	8	2^{10}	515.4 (0.10%)	509.3	5487 (1.05%)
4	2^{20}	8	2^{10}	1074.2 (0.10%)	1055.3	8735 (0.83%)
1	54675	5	2^{10}	51.4 (0.09%)	52.6	517 (0.95%)
2	54675	7	2^{10}	47.9 (0.09%)	48.5	535 (0.98%)
3	54675	9	2^{10}	51.7 (0.09%)	53.5	521 (0.95%)
3	54675	11	2^{10}	56.7 (0.10%)	54.7	555 (1.02%)
3	54675	13	2^{10}	46.9 (0.09%)	48.2	570 (1.04%)

The second data set comes from an image collection crawled by Standford Universities WebBase project [3,24]. We extracted an 8-dimensional (data set #4) and a 64-dimensional (data set #5) color histogram of $n = 2^{20}$ images such that each image is represented by its color intensities. This type of data was recently used in a study dedicated to the NN problem [18] and it is also well suitable for PNN queries: a PNN query asks for an image that is similar to the query image with respect to a subset of colors. The coordinates of a point are floating point numbers from the interval $[0, 100]$.

We use randomly sampled ε-nets to study their failure probability. Our experiments indicate that small epsilon-nets bound the number of points in a query range extremely well. The average and standard deviation (see Table 1) of the number of points in the query range is very small. Not only the average, but

also the maximal number of points behaves well and is roughly what Lemma 1 predicts. We observe that the dependence on the number of points n and dimension d is nearly linear. One could roughly state that if we used 1000 points for the epsilon-net over 10000 queries, then a range cuts roughly a 1/1000-fraction of points on the average (Table 1).

There are several heuristics that naturally stem from our approach, which have guarantees on the approximation ratio but not on the query time. Nevertheless, our experimental evaluation indicates that the worst-case query time does not occur in practice. A heuristic to approximate $\mathcal{C}_{Q,S,r}$, denoted by *volume-fit cube*, corresponds to a hyper-rectangle that is centered at Q and its projection onto S has the same volume as a w-dimensional Euclidean ball $\mathcal{B}_w(r)$. Thus, the edge length of the hyper-rectangle equals $\mathrm{vol}(\mathcal{B}_w(r))^{1/w}$. We obtain the hyper-rectangle $\mathcal{H}_{L,R}$ with $l_i = q_i - r\pi^{1/2}/(2\Gamma(w/2+1)^{1/w})$ and $r_i = q_i + r\pi^{1/2}/(2\Gamma(w/2+1)^{1/w})$ for the relevant dimensions. This hyper-rectangle exceeds the boundaries of $\mathcal{B}_w(r)$, and thus the number of reported points is no longer bounded by the epsilon-net, so the query time might degenerate to a linear scan. The approximation ratio is $2\Gamma(w/2+1)/\pi^{1/2}$ and is much smaller than $\sqrt{w}$ as the dimension grows. This heuristic performs markedly as Table 2 and Figure 2 indicate.

Table 2. Approximation ratio for the KD Tree implementation using the image data sets (columns from left: data set number, number of points, number of dimensions, approximation ratio average, standard deviation, maximum, and average rank)

#	n	d	range	ar (avg)	(stdv)	(max)	rank
4	2^{20}	8	inner cube	1.08	0.15	2.30	153.67
5	2^{20}	64	inner cube	1.01	0.29	5.66	87.50
4	2^{20}	8	volume-fit	1.02	0.05	1.45	81.72
5	2^{20}	64	volume-fit	1.02	0.09	2.74	54.80

Evaluation. We show results for the algorithms' performance in terms of query time, space requirements, approximation quality and rank of the answer. The query time is compared with a linear scan and we define the speed-up to be the ratio of the average query time of a linear scan and the average query time of our implementation. A discussion on the performance of our approach compared with the standard KD tree nearest neighbor query performance can be found in [12].

The dependence of the query time on the size of the data-base is presented in Figure 1. We observe a sub-linear dependence of the query time, and the algorithm performs better if a KD tree is used for range searching. A factor of over 60 times faster than a linear scan gives strong evidence for the algorithms practical usability. The results were obtained for an approximation with the inner cube range that has a theoretical bound on the approximation ratio of 2.8, however, the measured average approximation ratio was way less, roughly around 1.08 (see Table 2).

The dependence of the database dimension on the query time is shown in Figure 1. The ranges were approximated with the inner cube range and we observe that the dimensionality of the data set has only a negligible impact on the query time, at least for dimensions up to 13.

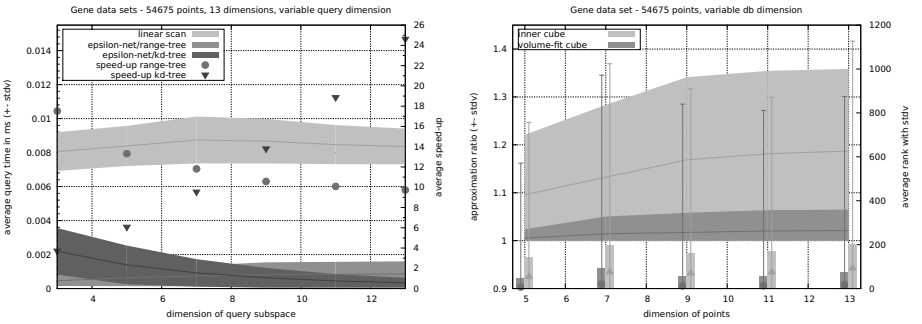


Fig. 2. The description of this figure is the same as of Figure 1. Left: Query time depending on the dimension of the query subspace. Right: The quality of the approximation (average over 10k queries: straight lines; standard deviation: filled area) and rank of the answer point (average over 10k queries: box; standard deviation: whisker).

In Figure 2 and Table 2 we present the measured approximation quality in terms of approximation ratio and rank of the answer. The *rank* is the number of points that are closer to the query than the point found. Figure 2 shows that the obtained approximation ratios are reasonable small. The maximal rank is less than than 0.025 percent of the database.

We also studied the behavior of our algorithm for data with medium dimensionality between 10^1 and 10^2. The 64-dimensional data set #5 using the KD Tree as range searching data structure has a query time that is still a factor of 5 to 10 faster than the linear scan. The theoretical observation would suggest that the approximation ratio is fairly large for this dimension, however, as Table 2 shows, the approximation ratio and rank are only slightly worse than for the low dimensional case.

References

1. Agarwal, P.K., Matoušek, J.: On Range Searching with Semialgebraic Sets. Discrete and Computational Geometry 11(1), 393–418 (1994)
2. Andoni, A., Indyk, P., Krauthgamer, R., Nguyen, H.L.: Approximate Line Nearest Neighbor in High Dimensions. In: SODA 2009: Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 293–301. ACM (2009)
3. Arasu, A., Cho, J., Garcia-Molina, H., Paepcke, A., Raghavan, S.: Searching the Web. Technical Report 2000-37. Stanford InfoLab (2000)
4. Arge, L., Berg, M.D., Haverkort, H., Yi, K.: The priority R-tree: A practically efficient and worst-case optimal R-tree. ACM Trans. Algorithms 4, 9:1–9:30 (2008)
5. Bernecker, T., Emrich, T., Graf, F., Kriegel, H.-P., Kröger, P., Renz, M., Schubert, E., Zimek, A.: Subspace Similarity Search: Efficient k-NN Queries in Arbitrary Subspaces. In: Gertz, M., Ludäscher, B. (eds.) SSDBM 2010. LNCS, vol. 6187, pp. 555–564. Springer, Heidelberg (2010)
6. Brönnimann, H., Chazelle, B., Matoušek, J.: Product Range Spaces, Sensitive Sampling, and Derandomization. SIAM Journal on Computing 28(5), 1575 (1999)

7. Chan, T.M.: Optimal Partition Trees. In: SCG 2010: Proceedings of the 2010 Annual Symposium on Computational Geometry, pp. 1–10. ACM (2010)
8. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry - Algorithms and Applications, 2nd edn. Springer (2000)
9. Eastman, C.M., Zemankova, M.: Partially Specified Nearest Neighbor Searches Using k-d Trees. Information Processing Letter 15(2), 53–56 (1982)
10. Goodman, J.E., O'Rourke, J. (eds.): Handbook of Discrete and Computational Geometry, 2nd edn. CRC Press (2004)
11. Haussler, D., Welzl, E.: Epsilon-nets and Simplex Range Queries. In: SCG 1986: Proceedings of the 2nd Annual Symposium on Computational Geometry, p. 71. ACM (1986)
12. Hruz, T., Schöngens, M.: Partially Specified Nearest Neighbor Search. Technical Report 762. Department of Computer Science, ETH Zurich (2012)
13. Hruz, T., Wyss, M., et al.: RefGenes: identification of reliable and condition specific reference genes for RT-qPCR data normalization. BMC Genomics 12(1), 156 (2011)
14. Indyk, P., Motwani, R.: Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. In: STOC 1998: Proceedings of the 30th Annual ACM Symposioum on Theory of Computing, pp. 604–613. ACM (1998)
15. Koltun, V.: Almost Tight Upper Bounds for Vertical Decompositions in Four Dimensions. In: FOCS 2001: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, pp. 56–65. IEEE (2001)
16. Kriegel, H., Kroger, P., Schubert, M., Zhu, Z.: Efficient Query Processing in Arbitrary Subspaces Using Vector Approximations. In: SSDBM 2006: Proceedings of the 18th International Conference on Scientific and Statistical Database Management, pp. 184–190 (2006)
17. Lee, D.T., Wong, C.: Worst-case Analysis for Region and Partial Region Searches in Multidimensional Binary Search Trees and Balanced Quad Trees. Acta Informatica 9(1), 23–29 (1977)
18. Lv, Q., Josephson, W., Wang, Z., Charikar, M., Li, K.: Multi-probe LSH: Efficient Indexing for High-Dimensional Similarity Search. In: VLDB 2007: Proceedings of the 33rd International Conference on Very Rarge Data Bases, pp. 950–961 (2007)
19. Matoušek, J.: Reporting Points in Halfspaces. Computational Geometry 2, 169–186 (1992)
20. Matoušek, J.: On Constants for Cuttings in the Plane. Discrete & Computational Geometry 20(4), 427–448 (1998)
21. Matoušek, J.: Lecture Notes on Discrete Geometry. Sp (2002)
22. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann Publishers Inc. (2005)
23. Sharir, M., Shaul, H.: Ray Shooting Amid Balls, Farthest Point from a Line, and Range Emptiness Searching. In: SODA 2005: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 525–534 (2005)
24. Stanford WebBase Project, http://diglib.stanford.edu:8091/~testbed/doc2/WebBase
25. Zimmermann, P., Laule, O., Schmitz, J., Hruz, T., Bleuler, S., Gruissem, W.: Genevestigator transcriptome meta-analysis and biomarker search using rice and barley gene expression databases. Molecular Plant 1(5), 851 (2008)

Multi-pattern Matching with Bidirectional Indexes

Simon Gog[1,⋆], Kalle Karhu[2,⋆⋆], Juha Kärkkäinen[3,⋆⋆⋆], Veli Mäkinen[3,†], and Niko Välimäki[3,‡]

[1] Department of Computing and Information Systems
University of Melbourne
simon.gog@unimelb.edu.au
[2] Department of Computer Science and Engineering
Aalto University
kalle.karhu@aalto.fi
[3] Department of Computer Science
University of Helsinki
{tpkarkka,vmakinen,nvalimak}@cs.helsinki.fi

Abstract. We study multi-pattern matching in a scenario where the pattern set is to be matched to several texts and hence indexing the pattern set is affordable. This kind of scenarios arise, for example, in metagenomics, where pattern set represents DNA of several species and the goal is to find out which species are represented in the sample and in which quantity. We develop a generic search method that exploits bidirectional indexes both for the pattern set and texts, and analyze the best and worst case running time of the method on worst case text. We show that finding the instance of the search method with minimum best case running time on worst case text is NP-hard. The positive result is that an instance with logarithm-factor approximation to minimum best case running time can be found in polynomial time using a bidirectional index called affix tree. We further show that affix trees can be simulated, in reduced space, using bidirectional variant of compressed suffix trees.

1 Introduction

Metagenomics studies genomic material taken from environmental samples [10]. Typically millions of short DNA reads are produced from the sample with length

⋆ Also affiliated with Ulm University. Supported by the Australian Research Council grant DP110101743.
⋆⋆ Supported by Academy of Finland grant 134287.
⋆⋆⋆ Supported by Academy of Finland grant 118653 (ALGODAN).
† Also affiliated with Helsinki Institute for Information Technology HIIT. Supported by Academy of Finland grants 140727 and 250345.
‡ Also affiliated with Helsinki Institute for Information Technology HIIT. Supported by Academy of Finland grant 118653 (ALGODAN) and Helsinki Doctoral Programme in Computer Science (Hecse).

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 384–395, 2012.

of each read varying between 30 to 400 nucleotides depending on sequencing technology, and subsequent sequence analysis tries to identify the species present in the sample. Sequence analysis can be either *fragment assembly* -based or *read alignment* -based. In the former approach the reads are first assembled into *contigs* (longer fragments glued together based on read overlaps) and then compared against reference genomes to locate statistically significant local alignments. In the latter approach the reads are directly aligned to reference genomes. We will focus on this latter approach.

Aligning metagenomics reads to the reference genomes can be done using e.g. software packages building on the *Burrows-Wheeler transform (BWT)* [1] and on the *FM-index* [4] concept. The FM-index provides a way to index a reference genome within its compressed size exploiting BWT. This index provides so-called *backward search* principle that enables very fast exact string matching on the indexed sequence. Extensions of this idea provide very efficient methods for doing read alignment, see `bowtie` [16], `bwa` [17], `SOAP2` [3], `readaligner` [19].

In this paper, we propose an approach for multi-pattern matching that takes the special characteristics of metagenomics read alignment into account. The methods above align each read separately without exploiting the fact that reads sets typically cover the same genomic position many times. Also repetitive areas cause similar reads to be produced. In metagenomics scenario, the same read set is to be aligned to several reference genomes, and hence it is affordable to spend time in preprocessing the read set.

We show how to take the similar content in read set into account with *bidirectional FM-index* [15,23] built both on the read set and on each reference genome. The idea is that we first select a set of subpatterns from the read set such that they cover the whole set (each read contains one of the subpatterns). Each such subpattern is searched for separately in the bidirectional FM-index of a reference genome *and* from the bidirectional FM-index of the read set. We then show how to continue the search synchronizing the two bidirectional indexes so that common parts of the reads preceding and succeeding the subpattern occurrence are scanned simultaneously.

We also study the optimization problem of selecting the optimal subpatterns to minimize the search time. We assume a worst case scenario where all reads are found in the reference genome. We then analyze the best and worst case running time of our synchronized bidirectional search and show that optimal selection of subpatterns to minimize best case search time is NP-hard. However, a greedy approach for selection of subpatterns yields logarithm factor approximation to the best case search time. We show that the greedy approach can be implemented efficiently using *affix trees* [24,18]. We further show that one can replace affix trees in the computation with a bidirectional variant of compressed suffix trees.

Our approach is currently limited to exact searching; see Sect. 4 for discussion on extensions to approximate search.

Related Work. Our work is motivated by the experimental study by Karhu [13] showing that subpattern selection from read set can reduce the total search time in the case of several reference genomes. The difference to our work is

that bidirectional indexes were not used in [13], but a more direct approach of aligning the covered reads to the occurrences of subpatterns was used. Also the optimization of subpattern selection was not studied there. Recently, Gagie et al. [8] gave the first theoretical improvement for indexed multi-pattern matching over the approach of searching each pattern separately. It is shown in [8] that given FM-index for text of length n and the LZ77 parse of the concatenation of p patterns of total length M and maximum individual length m, one can count the occurrences of each pattern in a total of $O((z+p)\log M \log m \log^{1+\epsilon} n)$ time, where z is the number of phrases in the parse. The result suites very well the metagenomics application considered here, as the LZ77 parsing needs to be constructed only once for the read set. However, in this application $p = M/c$, where c is a constant (typically $30 \leq c \leq 400$), and the logarithm factors may cancel out the improved search functionality in practice; $O(M)$ search time can be achieved searching each pattern separately from FM-index. Our synchronized bidirectional search does not provide good worst case bounds, but we claim that it is more suitable for this case of short patterns: In case of short patterns, subpattern cover captures the largest and most frequent repetitions also found by LZ77 parse, but at the same time the search algorithm has constant coefficients.

The paper is organized as follows. We define the basic concepts in Sections 2. In Sect. 3 we first introduce the synchronized bidirectional search algorithm and analyse its best and worst case running time, and then study the hardness of subpattern selection leading to a greedy approximation algorithm and its implementation with affix trees. Then we show that affix trees can be replaced with a more space-efficient bidirectional variant of compressed suffix trees. We discuss the future work in Sect. 4.

2 Preliminaries

A *string* $S = S[1, n] = S[1]S[2] \cdots S[n]$ is a *sequence* of *symbols* (a.k.a. characters or letters). Each symbol is an element of an ordered *alphabet* $\Sigma = \{1, 2, \ldots, \sigma\}$. A *substring* of S is written $S[i\,..\,j] = S[i]S[i+1] \cdots S[j]$. A *prefix* of S is a substring of the form $S[1\,..\,j]$, and a *suffix* is a substring of the form $S[i\,..\,n]$. If $i > j$ then $S[i\,..\,j] = \varepsilon$, the empty string of length $|\varepsilon| = 0$. The *lexicographical order* "$<$" among strings is defined in the obvious way. A *text* string $T[1\,..\,n]$ is a string terminated by the special symbol $T[n] = \$ \notin \Sigma$, smaller than any other symbol in Σ. *Suffix array* SA is an array of length n with $\texttt{SA}[i]$ corresponding to the starting position of the i-th smallest suffix in S.

2.1 Bidirectional FM-Index

A *bidirectional index* of string T consists of a *forward* and *reverse* index. The forward index supports backward search in T, and the reverse index in T^R, where T^R denotes the reversed string of T. Lam et al. [15] and Schnattinger et al. [23] showed how to synchronize the forward and reverse index to support bidirectional search. Let P denote a pattern, and let $[sf\,..\,ef]$ denote the SA

interval of the suffixes of T whose prefixes match P, and $[sr\,..\,er]$ denote the suffixes of T^R matching P^R. Now a bidirectional search step allows us to find out the new interval corresponding to either cP or Pc for any symbol $c \in \Sigma$. The new interval is empty if the pattern is not found.

We require the following operations. The *direction* of the operation is given by the parameter $d \in \{\mathsf{left}, \mathsf{right}\}$:

- `pushChar`$(d, c, [sf\,..\,ef], [sr\,..\,er])$: Assume that $[sf\,..\,ef]$ and $[sr\,..\,er]$ correspond to the pattern P. The operation returns new intervals corresponding to the concatenated pattern cP if $d = \mathsf{left}$, or Pc if $d = \mathsf{right}$. The operation returns an empty interval if the concatenated pattern does not exists. Both [15] and [23] show how to support this operation. The latter uses wavelet tree for the task and supports the operation in $O(\log \sigma)$ time.
- `getBranches`$(d, [sf\,..\,ef], [sr\,..\,er])$: Returns a subset of symbols, that is, all symbols $c \in \Sigma$ having a non-empty `pushChar`$(d, c, [sf\,..\,ef], [sr\,..\,er])$ interval. If $d = \mathsf{left}$, we return the set of distinct symbols occurring in $T_{BWT}[sf\,..\,ef]$, and if $d = \mathsf{right}$, we return the distinct symbols occurring in $T^R_{BWT}[sr\,..\,er]$. This can be done in $O(\log \sigma)$ time per distinct symbol with a wavelet tree [9].

The space usage for the bidirectional FM-index is twice that of an FM-index based on wavelet tree, i.e. $2n \log \sigma + o(n \log n)$ bits [20] for a text of length n.

3 Multi-pattern Matching

3.1 Bidirectional Search

We construct a bidirectional index for both the text T and the set of patterns $P_1, P_2, \ldots, P_p$. More precisely, the pattern index is constructed for the concatenated string $\#P_1\#P_2\#\cdots\#P_p\#\$$, where $\#$ is a special separator symbol that does not occur in any of patterns. Let N and M denote the total length of the text and the concatenated string of patterns, respectively. The pattern index stores SA samples only at separator symbol positions. This requires $p \log M$ bits of space, which might be too much for patterns shorter than $\log M$, but allows $O(1)$ time locate for SA ranges $[i\,..\,j]$ that are prefixed by $\#$.

We assume that the subpattern P is given as input, and the task is to locate occurrences of patterns $P_1, P_2, \ldots, P_p$, that contain subpattern P, in the text T. In other words, for every P_i that has an occurrence of P, we must output all occurrences of P_i in T. We proceed with the search as follows.

First, we search the subpattern P in the text index by forward and backward search to retrieve the SA intervals $[sf, ef]$ and $[sr, er]$. These intervals are empty if P does not occur in T. We repeat the same search in the pattern index — we can assume that the subpattern P appears in the pattern index at least once. Fig. 1 gives an algorithm (rows 15–19) that starts with intervals corresponding to empty pattern (rows 15–16) and then searches P using the `pushChar`() operation. The rest of the search extends P recursively to both

Algorithm `extend`$([sf\,..\,ef],[sr\,..\,er],[sf'\,..\,ef'],[sr'\,..\,er'],P,d)$
(1) **if** $(P[1]=\#)$ **and** $(P[|P|]=\#)$ **then**
(2) Report that patterns $[sf'\,..\,ef']$ occur at positions $[sf\,..\,ef]$.
(3) **return**
(4) **if** $(P[1]=\#)$ **then** $d \leftarrow \mathsf{right}$
(5) **if** $(P[|P|]=\#)$ **then** $d \leftarrow \mathsf{left}$
(6) $C \leftarrow$ `getBranches`$(d,[sf\,..\,ef],[sr\,..\,er])$
(7) $C' \leftarrow$ `getBranches`$(d,[sf'\,..\,ef'],[sr'\,..\,er'])$
(8) **if** $(d=\mathsf{left})$ **then** $d' \leftarrow \mathsf{right}$ **else** $d' \leftarrow \mathsf{left}$
(9) **for each** $c \in C \cap C'$ **do**
(10) **if** $(d=\mathsf{left})$ **then** $P' \leftarrow cP$ **else** $P' \leftarrow Pc$
(11) `extend`(`pushChar`$(d,c,[sf\,..\,ef],[sr\,..\,er])$,
`pushChar`$(d,c,[sf'\,..\,ef'],[sr'\,..\,er']),P',d')$
(12) **if** $(\# \in C')$ **then**
(13) **if** $(d=\mathsf{left})$ **then** $P' \leftarrow \#P$ **else** $P' \leftarrow P\#$
(14) `extend`$([sf\,..\,ef],[sr\,..\,er],$`pushChar`$(d,\#,[sf'\,..\,ef'],[sr'\,..\,er']),P',d')$

Algorithm `search`(P)
(15) $[sf\,..\,ef],[sr\,..\,er] \leftarrow [1\,..\,N],[1\,..\,N]$ {*Forward and reverse text index*}
(16) $[sf'\,..\,ef'],[sr'\,..\,er'] \leftarrow [1\,..\,M],[1\,..\,M]$ {*Forward and reverse pattern index*}
(17) **for** $i \in [1\,..\,|P|]$ **do**
(18) $[sf\,..\,ef],[sr\,..\,er] \leftarrow$ `pushChar`$(\mathsf{right},P[i],[sf\,..\,ef],[sr\,..\,er])$
(19) $[sf'\,..\,ef'],[sr'\,..\,er'] \leftarrow$ `pushChar`$(\mathsf{right},P[i],[sf'\,..\,ef'],[sr'\,..\,er'])$
(20) **if** $[sf\,..\,ef]$ is non-empty **then**
(21) `extend`$([sf\,..\,ef],[sr\,..\,er],[sf'\,..\,ef'],[sr'\,..\,er'],P,\mathsf{left})$

Fig. 1. The recursive bidirectional search is initiated by calling `search`(P). The values of N and M denote the total length of text and patterns, respectively. The parameter $d \in \{\mathsf{left},\mathsf{right}\}$ denotes the direction. The intervals $[sf\,..\,ef]$, $[sr\,..\,er]$ and $[sf'\,..\,ef']$, $[sr'\,..\,er']$ correspond to the forward and reverse intervals matching P in the text and pattern indexes, respectively.

directions, over all combinations of symbols on the left and right side of subpatterns occurrences in $P_1, P_2, \ldots, P_p$. The recursion is initiated (row 21) by calling `extend`$([sf\,..\,ef],[sr\,..\,er],[sf'\,..\,ef'],[sr'\,..\,er'],P,\mathsf{left})$, where $[sf\,..\,ef]$, $[sr\,..\,er]$, $[sf'\,..\,ef']$ and $[sr'\,..\,er']$ are the intervals matching P and the first extension direction is left.

Fig. 1 gives a recursive algorithm (rows 1–14) that extends the subpattern recursively on both sides. The extension is done alternating between the directions $\{\mathsf{left},\mathsf{right}\}$ — interleaving left and right symbols during the search. The following invariant holds throughout the recursion: at each step, the SA intervals $[sf\,..\,ef]$, $[sr\,..\,er]$, $[sf'\,..\,ef']$ and $[sr'\,..\,er']$ correspond to P's SA intervals in the forward text index, reversed text index, forward pattern index and reverse pattern index, respectively. The only exception is the special separator symbol # that gets appended to the start and/or end of the pattern during the recursion;

the symbol # is never searched from the text index, it is merely a placeholder that marks the end of pattern extension to that direction.

Let us describe `extend`() in more detail. The first rows 1–3 are the end condition of the recursion: both ends of P have the separator symbol #, thus, the interval $[sf', ef']$ corresponds to some distinct subset of patterns. Occurrences of these patterns in T can now be enumerated from $[sf, ef]$. The next two rows 4–5 check if there exists a separator symbol at the start (resp. end) of the pattern. Then the next extension must be to the right (resp. left) since # marks the end of extension to that particular direction. The rows 6–7 enumerate the symbols that can be used for extension to direction d in text and pattern indexes. The actual extension is done only for symbols that are branching from both the text and pattern index (cf. the intersection of these sets at row 9). The row 8 flips the direction for the next recursion step, and row 10 constructs the new extended pattern. The next recursion step is called at row 11, having the new intervals of extended pattern as parameters. The rows 12–14 check if the pattern can be extended with the separator symbol.

Let us now analyze the number of steps required by `extend`(). Let $x = lsize(I, P)$ and $y = rsize(I, P)$ denote the search space size, in the worst case scenario of text containing occurrences of all the patterns, using bidirectional index I when extending P *only* to the left and *only* to the right, respectively. The worst case number of steps taken by the bidirectional recursive search can be written as a recursion:

$$f(x, y) = \sum_c g(x_c - 1, y - 1), \quad \texttt{where} \quad \sum_c x_c = x;$$
$$g(x, y) = \sum_c f(x - 1, y_c - 1), \quad \texttt{where} \quad \sum_c y_c = x,$$

with initialization $f(x, 1) = g(x, 1) = x$ for all x and $f(1, y) = g(1, y) = y$ for all y (and otherwise 0). Here $f(x, y)$ and $g(x, y)$ denote the number steps required when extending pattern to the left and right, respectively, with all possible c. The recursion follows from the fact that, in the worst case, extending to one direction splits to one or more independent cases where the search space in the other direction is only decreased by one. It is easy to see that $f(x, y) < x * y$ and $g(x, y) < x * y$, and hence the worst case number of steps by `extend`() is $lsize(I, P) * rsize(I, P)$. This is also a strict upper bound as can be seen as follows. Let $ldepth(I, P)$ and $rdepth(I, P)$ denote the maximum depth of the search space when extending P only to the left and only to the right, respectively. Then one can construct an instance where search space to the left starts with a unary path of length $rdepth(I, P)$ and hence, after $rdepth(I, P)$ steps identifying $x \leq \min(p, rsize(I, P))$ leaves, one still has to explore x times search space of size $lsize(I, P) - rdepth(I, P) = O(lsize(I, P))$.

A lower bound for the number of steps required by `extend`(), in the worst case scenario of text containing occurrences of all patterns, is also of interest. This is easily seen to be $lsize(I, P) + rsize(I, P)$.

3.2 Hardness of Subpattern Selection

In the above, we developed a multi-pattern search algorithm that can exploit given set of subpatterns that cover all patterns. Let us now study how to find a good set of such subpatterns.

First, it is easy to see that selecting minimum size set of subpatterns to cover all the patterns is NP-hard, by a reduction from set cover:

Problem 1 (Set cover). Given a universe U of n elements, and a collection of subsets of U, $\mathbb{S} = \{S_1, \ldots, S_k\}$, find a minimum size subcollection of $\mathbb{S}$ that covers all elements of U.

From a set cover instance we create the following set of patterns for $1 \leq i \leq n$:

$$P_i = \prod_{S_j, u_i \in S_j} \#_j \#_{k+i}, \tag{1}$$

where each $\#_x$ is a distinct symbol, and $\prod$ denotes catenation. Patterns $\mathbb{P} = P_1, P_2, \ldots, P_n$ are now from alphabet $\{\#_1, \#_2, \ldots, \#_{k+n}\}$. Any subpattern occurring in more than one P_i is necessarily a single symbol corresponding to a subset in $\mathbb{S}$. Finding minimum size collection of subpatterns covering $\mathbb{P}$ solves the set cover instance, and the problem of selecting the best subpatterns is hence NP-hard.

We can generalize the reduction to a more realistic version of the problem. Let $w(P) = |P| + lsize(I, P) + rsize(I, P)$, that is the lower bound (and hence an optimistic estimate) for the total number of steps taken by the recursive bidirectional search with a worst case text.

The cost of a set $\mathbb{S}'$ of subpatterns is $c(\mathbb{S}') = \sum_{P \in \mathbb{S}'} |P| + lsize(I, P) + rsize(I, P)$. We wish to find minimum cost set $\mathbb{S}'$. This problem is also NP-hard by a slight modification of the above reduction. Notice that with the current reduction $2 \leq |\#_j| + lsize(I, \#_j) + rsize(I, \#_j) \leq \sum_{u_i \in S_j} \sum_{S_{j'}: u_i \in S_{j'}} 2 \leq 2n^2$. Let us modify the generated pattern set to

$$P_i = \left(\prod_{S_j, u_i \in S_j} \#_j \#_{k+i} \right) \underbrace{1 \cdots 1}_{2n^3}. \tag{2}$$

Then one can verify that minimum cost cover with $k' + 1$ subpatterns costs more that maximum cost cover with k' subpatterns. Hence, finding minimum cost subpattern cover solves the set cover problem, and hence the former is also NP-hard.

A positive connection to set cover also exists; an algorithm analogous to the well-known greedy approximation algorithm for weighted set cover [25] [Chapter 2] can be used to compute a good subpattern cover: Choose first pattern P that minimizes $\frac{|P| + steps(I,P)}{m(\mathbb{P},P)}$, where $m(\mathbb{P}, P)$ denotes the number of patterns in $\mathbb{P}$ which contain P as subpattern. Set $\mathbb{P} = \mathbb{P} \setminus \mathbb{P}'$, where $\mathbb{P}'$ denotes the set of patterns covered by P. Iterate the process until $\mathbb{P}$ is empty. The set cover analysis

[25][Chapter 2] can be used verbatim to see that the process results in a set of subpatterns with cost at most $\log p$ times the optimal, where p is the size of $\mathbb{P}$. Notice that here we do not know value $steps(I, P)$ exactly for any pattern, so we will only obtain approximation with respect to our estimate on $steps(I, P)$; the estimation error can be arbitrarily more than the $\log p$ factor from the set cover approximation.

3.3 Subpattern Selection Using Affix Trees

The above approximation algorithm can be computed using *affix trees* [24,18]. Here we assume that $steps(I, P)$ is estimated as a function of $lsize(I, P)$ and $rsize(I, P)$, without fixing the exact formula (it could be the lower bound, the upper bound, or some function whose parameters are estimated with training data). Let us start by describing *suffix trees* [26]. The suffix tree of a string $T[1..n]$ represents all substrings of T in a rooted, directed tree where each internal node has at least two children and at most one outgoing edge label that starts with each $c \in \Sigma$. Edge labels are encoded as a reference to T, e.g. a pair of starting and ending text positions. The suffix tree has $O(n)$ nodes and, if T is terminated with a special symbol $\$ \notin \Sigma$, the resulting suffix tree has exactly n leaf nodes, one for each suffix of T. The affix tree of a string T incorporates the suffix tree of both T and its reversed string T^R. An internal node in the affix tree can have both suffix and prefix descendants: the outgoing *suffix edges* (resp. *prefix edges*) point to the descendants of the corresponding node in the suffix tree of T (resp. T^R). For each node v in the suffix tree of T (resp. T^R), there exists a corresponding node in the affix tree having the upward suffix edge (resp. prefix edge) labels equal to $\texttt{path}(v)$. The total number of nodes and edges is $O(n)$. Affix trees can be constructed in linear time and space [18].

The greedy approximation algorithm requires us to compute $lsize(I, P)$, $rsize(I, P)$, and $m(\mathbb{P}, P)$ values. The latter values can be computed with the *color set size* algorithm [11]. It stores, for all nodes v in the suffix tree of $P_1\$P\$\cdots P_p\$$, the number of patterns in $\mathbb{P}$ which have $\texttt{path}(v)$ as a subpattern. The algorithm requires linear time and space — we omit the technical details. To compute $lsize(I, P)$ and $rsize(I, P)$ values, we first build an affix tree for the concatenated string $T = \#P_1\#P_2\#\cdots\#P_p\#$, where $\#$ is a special separator symbol, $\# \notin \Sigma$. Then, we construct a bitvector $B[1, |T|]$ that marks the separators' positions, i.e. $B[i] = 1$ iff $T[i] = \#$. With $o(|T|)$ bits of extra space we can support constant time *rank* and *select* operations [12,2] on the bitvector: a $rank_1(B, i)$ query returns the number of 1-bits in $B[1..i]$, and $select_1(B, i)$ returns the position of the i-th 1-bit in B.

The goal is to compute the sum of edge label lengths, down to the first terminator symbol, in subtrees consisting either of prefix or suffix descendants. Let $size(\overleftarrow{v})$ and $size(\overrightarrow{v})$ denote the size of the subtree of v consisting either of prefix or suffix descendants, respectively; notice that the former equals $lsize(I, \texttt{path}(v))$ and the latter equals $rsize(I, \texttt{path}(v))$. To compute $size(\overleftarrow{v})$ and $size(\overrightarrow{v})$ values for each affix tree node v, we do two depth-first traversals over the affix tree: one traversal over all the prefix edges and one over the suffix

edges. Let us describe the latter, i.e. how to compute $size(\overrightarrow{v})$. We start from the root node and traverse through suffix edges until we encounter either a leaf node or an edge label that contains the separator symbol #. The latter condition can be checked in constant time by taking the starting and ending position of the edge label, say $[s\,..\,e]$, and calling $rank_1(B, e)$ and $rank_1(B, s)$. If the two rankings are different, there exists at least one 1-bit in $B[s\,..\,e]$, the first one occurring at the position $select_1(B, rank_1(B, s-1)+1)$. If v is a leaf node, we set $size(\overrightarrow{v}) = 0$, and if the incoming edge label of v has the separator symbol, we set $size(\overrightarrow{v}) = select_1(B, rank_1(B, s-1)+1)-s$, that is, label's length up to the first # symbol. Finally, if v is a internal node, $size(\overrightarrow{v})$ is set to be the sum over the sizes of its suffix descendants. After the traversal we have stored, for each node v in the affix tree, the correct $size(\overrightarrow{v})$ for the subpattern $P = \texttt{path}(v)$. We can repeat the same traversal over the prefix edges to compute $size(\overleftarrow{v})$.

The above algorithm can be used to choose the first pattern P that minimizes $\frac{|P|+steps'(I,P)}{m(\mathbb{P},P)}$, where $steps'(I, P)$ is our estimator on $steps(I, P)$ based on $lsize(I, P)$ and $rsize(I, P)$. The algorithm needs to be iterated to find the subpattern cover, each iteration requiring linear time and space. Notice that only the $m(\mathbb{P}, P)$ values need to be updated on each iteration for each new set $\mathbb{P} = \mathbb{P} \setminus \mathbb{P}'$, where $\mathbb{P}'$ denotes the set of patterns covered by P.

3.4 Subpattern Selection Using Bidirectional Compressed Suffix Trees

Affix tree occupies $O(M \log M)$ bits for the pattern set of total length M. We aim for $O(M \log \sigma)$ bits solution. To achieve this, we use compressed suffix trees [22,21,6], one for $T = \#P_1\#P_2\#\cdots\#P_p\#$ and one for T^R (i.e. latter being prefix tree). Let us denote these two compressed suffix trees $\mathcal{S}$ and $\mathcal{P}$ (standing for suffix and prefix). For what follows, it is sufficient to know that the compressed suffix tree $\mathcal{S}$ (or identically $\mathcal{P}$) supports, in addition to tree navigation operations, the following operations: $j = \mathcal{S}.SA(i)$ for retrieving i-th suffix array value i.e. the starting position of the lexicographically i-th suffix of the text indexed by $\mathcal{S}$, $i = \mathcal{S}.SA^{-1}(j)$ for retrieving the lexicographic rank i of the suffix starting at position j in the text indexed by $\mathcal{S}$, $v = \mathcal{S}.lca(l, r)$ for retrieving the lowest common ancestor node v of leaves denoted by suffix array indices l and r, $[l, r] = \mathcal{S}.range(v)$ for retrieving range $[l, r]$ such that l (r) is the left-most (right-most) leaf under v, and $\mathcal{S}.sdepth(v)$ for retrieving the *string depth* of node v i.e. the total length of labels from root to v: $|\mathcal{S}.path(v)|$.

Recall the affix tree algorithm to compute $size(\overleftarrow{v})$ and $size(\overrightarrow{v})$. With $\mathcal{S}$ and $\mathcal{P}$ we can independently count values $size(\overrightarrow{v})$ and $size(\overrightarrow{w})$ for internal nodes v of $\mathcal{S}$ and internal nodes w of $\mathcal{P}$, respectively, but we do not anymore have the explicit linking between v and w, that is, to find w such that $size(\overrightarrow{w}) = size(\overleftarrow{v})$. Moreover, $\mathcal{S}.path(v)^R$ may not spell a path in $\mathcal{P}$ that ends in an internal node w, and vice versa. We say that v in $\mathcal{S}$ *is linked to* w in $\mathcal{P}$ if $\mathcal{S}.path(v)^R$ is a prefix of $\mathcal{P}.path(w)$ and w has smallest string depth among nodes where this condition holds. If $\mathcal{S}.sdepth(v) < \mathcal{P}.sdepth(w)$, then $\mathcal{S}.path(v)^R$ ends at the

incoming edge to w. Hence, knowing v is linked to w, we have the connection $size(\overrightarrow{w}) + \mathcal{P}.sdepth(w) - \mathcal{S}.sdepth(v) = size(\overleftarrow{v})$. It remains to see how to implicitly restore the linking between v and w.

Let $A[i] = \mathcal{P}.SA^{-1}(n - \mathcal{S}.SA(i))$ for $1 \leq i \leq n$, where $n = M + p + 1$ is the length of T. We preprocess $A[1 \ldots n]$ for range minimum and range maximum queries: $i = rminq(A, l, r)$ returns i such that $A[i] \leq A[j]$ for all $l \leq j \leq r$, and $i = rmaxq(A, l, r)$ returns i such that $A[i] \geq A[j]$ for all $l \leq j \leq r$. For each query type, $2n + o(n)$ bits data structure is enough for answering the corresponding query in constant time [5]; array A does not need to be constructed explicitly as access to it is only required during construction time. Consider v in $\mathcal{S}$ and its suffix array range $[l, r] = \mathcal{S}.range(v)$. We compute the corresponding suffix array range $[l', r']$ in $\mathcal{P}$ by

$$l' \leftarrow \mathcal{P}.SA^{-1}(n - (\mathcal{S}.SA(rminq(A, l, r)) + \mathcal{S}.sdepth(v))) \text{ and}$$
$$r' \leftarrow \mathcal{P}.SA^{-1}(n - (\mathcal{S}.SA(rmaxq(A, l, r)) + \mathcal{S}.sdepth(v))).$$

We claim that $w = \mathcal{P}.lca(l', r')$ is the node linked to v. To see this, notice that $\mathcal{S}.SA(l), \mathcal{S}.SA(l+1), \ldots, \mathcal{S}.SA(r)$ are the only starting positions of prefixes of T followed by $\mathcal{S}.path(v)$. That is, $\mathcal{S}.SA(l) + \mathcal{S}.depth(v), \mathcal{S}.SA(l+1) + \mathcal{S}.depth(v), \ldots, \mathcal{S}.SA(r) + \mathcal{S}.depth(v)$ are the only starting positions of prefixes of T ending with $\mathcal{S}.path(v)$. The relative lexicographic order of the prefixes in this range does not change when the starting positions are increased by $\mathcal{S}.depth(v)$ and hence we can select the lexicographically smallest and largest prefixes before shifting the start positions. As all the suffixes $\mathcal{P}.SA(l'), \mathcal{P}.SA(l'+1), \ldots, \mathcal{P}.SA(r')$ share the same prefix $\mathcal{S}.path(v)^R$, it must hold for $w = \mathcal{P}.lca(l', r')$ that $\mathcal{S}.path(v)^R$ is a prefix of $\mathcal{P}.path(w)$ and w has the smallest string depth among nodes with this property.

Finally, the space bottleneck in the computation is the storage of values $size(\overrightarrow{v})$ in $\mathcal{S}$ and $size(\overrightarrow{w})$ in $\mathcal{P}$. The values in $\mathcal{S}$ can be computed during depth-first traversal and need not be stored, but one may still need to maintain $O(n)$ values in stack each occupying $O(\log n)$ bits; this can be improved to $O(n)$ bits by maintaining a dynamic partial sums data structures both for the stack and for the values following almost verbatim the algorithm in [7].

Storage of values $size(\overrightarrow{w})$ in $\mathcal{P}$ requires a different approach, namely, sampling: Partition the leaves of $\mathcal{P}$ from left-to-right to blocks of length $\log n$. Compute the lowest common ancestor node for each block and mark it sampled. On every path in the tree from root towards leaves, mark all nodes sampled until encountering the first node marked in the first step. For all marked nodes w compute the values $size(\overrightarrow{w})$ using depth-first traversal similarly as for $\mathcal{S}$ using $O(n)$ bits during computation in addition to the sampled values. There are $O(n/\log n)$ marked nodes, so storage of the samples takes $O(n)$ bits. Each unmarked node w has the property that in its $O(\log n)$ size subtree, each path ends with a leaf or with a marked node. Hence, one can compute in $O(\log n)$ time (assuming constant navigation time) the value $size(\overrightarrow{w})$.

The running time for computing the linking depends on the chosen compressed suffix tree, but $O(n \log n)$ time can be achieved e.g. using [22].

4 Future Work

We implemented the scheme presented here almost verbatim and noticed that it needs considerable engineering to be able to compete with the standard approach of aligning one read at a time. However, this engineering pays off. We are able to get a 3-fold speed-up on a realistic metagenomics scenario [14]. This is obtained by (i) modifying the bidirectional search to use different data structures than the ones presented here and (ii) fitting a polynomial to model the actual number of steps required by the bidirectional search so that subpattern selection optimizes better the total running time.

The biggest issue for practicality of the approach is that supporting exact search will be sufficient only when cutting the reads to smaller pieces. One can support approximate search using general backtracking mechanism inside the bidirectional search, but to do this efficiently the existing pruning mechanisms (like in [16,17,3,19]) need to be modified or new ones introduced to work within our search scheme. Also, subpattern cover needs to be refined in order to guarantee that all approximate occurrences are found.

References

1. Burrows, M., Wheeler, D.: A block sorting lossless data compression algorithm. Technical Report Technical Report 124, Digital Equipment Corporation (1994)
2. Clark, D.R.: Compact pat trees. PhD thesis, Waterloo, Ont., Canada, Canada (1998)
3. Li, R., et al.: Soap2. Bioinformatics 25(15), 1966–1967 (2009)
4. Ferragina, P., Manzini, G.: Indexing compressed texts. Journal of the ACM 52(4), 552–581 (2005)
5. Fischer, J., Heun, V.: Space-efficient preprocessing schemes for range minimum queries on static arrays. SIAM J. Comput. 40(2), 465–492 (2011)
6. Fischer, J., Mäkinen, V., Navarro, G.: Faster entropy-bounded compressed suffix trees. Theor. Comput. Sci. 410(51), 5354–5364 (2009)
7. Fischer, J., Mäkinen, V., Välimäki, N.: Space efficient string mining under frequency constraints. In: ICDM, pp. 193–202 (2008)
8. Gagie, T., Karhu, K., Kärkkäinen, J., Mäkinen, V., Salmela, L., Tarhio, J.: Indexed Multi-pattern Matching. In: Fernández-Baca, D. (ed.) LATIN 2012. LNCS, vol. 7256, pp. 399–407. Springer, Heidelberg (2012)
9. Gagie, T., Puglisi, S.J., Turpin, A.: Range Quantile Queries: Another Virtue of Wavelet Trees. In: Karlgren, J., Tarhio, J., Hyyrö, H. (eds.) SPIRE 2009. LNCS, vol. 5721, pp. 1–6. Springer, Heidelberg (2009)
10. Handelsman, J., Rondon, M.R., Brady, S.F., Clardy, J., Goodman, R.: Molecular biological access to the chemistry of unknown soil microbes: a new frontier for natural products. Chemistry & Biology 5, 245–249 (1998)
11. Hui, L.C.K.: Color set size problem with application to string matching. In: Proc. 3rd Annual Symposium on Combinatorial Pattern Matching, pp. 230–243. Springer, London (1992)
12. Jacobson, G.: Succinct Static Data Structures. PhD thesis. Carnegie–Mellon University, CMU-CS-89-112 (1989)

13. Karhu, K.: Improving exact search of multiple patterns from a compressed suffix array. In: Holub, J., Žďárek, J. (eds.) Proceedings of the Prague Stringology Conference 2011, pp. 226–231. Czech Technical University in Prague, Czech Republic (2011)
14. Karhu, K., Mäkinen, V.: Practical multi-pattern matching with bidirectional indexes. Submitted manuscript (2012)
15. Lam, T.W., Li, R., Tam, A., Wong, S., Wu, E., Yiu, S.M.: High throughput short read alignment via bi-directional BWT. In: IEEE International Conference on Bioinformatics and Biomedicine, vol. 0, pp. 31–36 (2009)
16. Langmead, B., Trapnell, C., Pop, M., Salzberg, S.L.: Ultrafast and memory-efficient alignment of short dna sequences to the human genome. Genome Biology 10(3), R25 (2009)
17. Li, H., Durbin, R.: Fast and accurate short read alignment with burrows-wheeler transform. Bioinformatics 25(14), 1754–1760 (2009)
18. Maaß, M.G.: Linear bidirectional on-line construction of affix trees. Algorithmica 37(1), 43–74 (2003)
19. Mäkinen, V., Välimäki, N., Laaksonen, A., Katainen, R.: Unified View of Backward Backtracking in Short Read Mapping. In: Elomaa, T., Mannila, H., Orponen, P. (eds.) Ukkonen Festschrift 2010. LNCS, vol. 6060, pp. 182–195. Springer, Heidelberg (2010)
20. Navarro, G., Mäkinen, V.: Compressed full-text indexes. ACM Computing Surveys 39(1), article 2 (2007)
21. Russo, L.M.S., Navarro, G., Oliveira, A.L.: Fully compressed suffix trees. ACM Trans. Algorithms 7, 53:1–53:34 (2011)
22. Sadakane, K.: Compressed suffix trees with full functionality. Theor. Comp. Sys. 41, 589–607 (2007)
23. Schnattinger, T., Ohlebusch, E., Gog, S.: Bidirectional Search in a String with Wavelet Trees. In: Amir, A., Parida, L. (eds.) CPM 2010. LNCS, vol. 6129, pp. 40–50. Springer, Heidelberg (2010)
24. Stoye, J.: Affix trees. Technical Report 2000-04, Faculty of Technology, Bielefeld University (2000), http://www.techfak.uni-bielefeld.de/~stoye/rpublications/report00-04.pdf
25. Vazirani, V.V.: Approximation Algorithms. Springer (2001)
26. Weiner, P.: Linear pattern matching algorithm. In: Proc. 14th Annual IEEE Symposium on Switching and Automata Theory, pp. 1–11 (1973)

Succinct Representations of Binary Trees for Range Minimum Queries

Pooya Davoodi[1,⋆], Rajeev Raman[2], and Srinivasa Rao Satti[3]

[1] Polytechnic Institute of New York University, United States
pooyadavoodi@gmail.com
[2] University of Leicester, United Kingdom
r.raman@leicester.ac.uk
[3] Seoul National University, South Korea
ssrao@cse.snu.ac.kr

Abstract. We provide two succinct representations of binary trees that can be used to represent the Cartesian tree of an array A of size n. Both the representations take the optimal $2n + o(n)$ bits of space in the worst case and support range minimum queries (RMQs) in $O(1)$ time. The first one is a modification of the representation of Farzan and Munro (SWAT 2008); a consequence of this result is that we can represent the Cartesian tree of a random permutation in $1.92n + o(n)$ bits in expectation. The second one uses a well-known transformation between binary trees and ordinal trees, and ordinal tree operations to effect operations on the Cartesian tree. This provides an alternative, and more natural, way to view the 2D-Min-Heap of Fischer and Huen (SICOMP 2011). Furthermore, we show that the pre-processing needed to output the data structure can be performed in linear time using $o(n)$ bits of extra working space, improving the result of Fischer and Heun who use $n + o(n)$ bits working space.

1 Introduction

Given an array $A[1 \cdots n]$ of totally ordered values, the *range minimum query* (RMQ) problem is to preprocess A into a data structure such that given two indexes $1 \leq i \leq j \leq n$, we return the index of the minimum value in $A[i \cdots j]$; the aim is to minimize the time and space requirements of both the preprocessing and the data structure. This problem finds a variety of applications that deal with huge datasets, thus highly space-efficient solutions are of great interest. We consider the problem in the word RAM model with word size $\Theta(\log n)$ bits.

A standard approach to solve the RMQ problem is to use the *Cartesian tree* [20]. The Cartesian tree of an array $A[1 \cdots n]$ is a binary tree with nodes labeled by the indexes of A. The root has label i, where $A[i]$ is the minimum

⋆ Part of this research was done while the first author was a PhD student at MADALGO (supported by the Danish National Research Foundation), Aarhus University, Denmark.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 396–407, 2012.

element in A. The left subtree of the root is the Cartesian tree of the subarray $A[1 \cdots i-1]$, and the right subtree of the root is the Cartesian tree of the subarray $A[i+1 \cdots n]$. Thus the answer for the query range $[i \cdots j]$ is the label of the lowest common ancestor (LCA) of the nodes labeled by i and j. The Cartesian tree of A can be constructed in $O(n)$ time [6]. A data structure that uses $O(n)$ words of space and finds the LCA of two nodes in a tree of size n in $O(1)$ time can be constructed in $O(n)$ time [10]—an apparently optimal solution.

In fact, the Cartesian tree of an array A completely characterizes A with respect to RMQs: the Cartesian tree of two arrays have different topology, iff there exists at least one query that has different answers over the two arrays. Since the information-theoretic lower bound for representing a binary tree on n nodes is $2n - \Theta(\log n)$ bits, there is a significant gap between this lower bound and the space usage of [6], which is $O(n)$ words, or $O(n \log n)$ bits. A fair amount of effort has gone into closing this gap, particularly since in several applications the array A need not be kept once we have enough information to answer RMQs.

It is known that binary trees can be represented *succinctly*, i.e. using space within a lower-order term of the information-theoretic lower bound. Specifically, an n-node binary tree can be represented in $2n+o(n)$ bits to support a number of operations, including LCA, in $O(1)$ time [12,1,2,3]. Unfortunately, we cannot use these representations to solve the RMQ problem. The difficulty is that the label of a node in the Cartesian tree (the index of the corresponding array-element) is its rank in the *inorder* traversal of the Cartesian tree. However, succinct tree representations cannot label nodes in an arbitrary manner without blowing up the space usage, and support only a few numbering schemes, including level-order [12], preorder [3] and others, but *not* inorder.

In parallel, many succinct representations of *ordinal* trees (arbitrary rooted trees where the order of children matters) were developed. These take $2n + o(n)$ bits to represent n-node ordinal trees, and support a wide variety of operations including LCA queries (see e.g. [11,2,19]). However, ordinal trees do not distinguish between left and right children in a binary tree, so the structure of the Cartesian tree cannot be represented. Also, as above, we need to find a way to translate between the array indexes and the numbering scheme of the ordinal tree representation. To get around these problems, Sadakane [18] adds a new leaf to each node in a Cartesian tree of n nodes, and views the resulting tree of $n' = 2n$ nodes as an ordinal tree. He notes that array index i corresponds to the i-th leaf in the ordinal tree in preorder. Representing this ordinal tree succinctly, he answers RMQs in $O(1)$ time, but the space usage is $2n' + o(n') = 4n + o(n)$ bits—twice the optimal. A solution using $2n + o(n)$ bits that answers RMQs in $O(1)$ time was proposed by Fischer [4,5], who defined the *2D-Min-Heap* of an array $A[1 \cdots n]$, an ordinal tree with $n + 1$ nodes that stores information on prefix minima of sub-arrays of A, and represents this ordinal tree succinctly.

Our Results. We present two different techniques to represent Cartesian trees using $2n + o(n)$ bits, and both the representations support the inorder numbering scheme and LCA queries. An immediate consequence of each of these representations is a data structure of size $2n + o(n)$ bits that supports RMQs in

$O(1)$ time. Although we do not improve the upper bounds of [5] for the RMQ problem in the worst case (as they were already optimal), we introduce more natural ways or simpler approaches to solve the RMQ problem.

In Section 2, we provide a new representation of binary trees that is a modification of the representation of [2], and to support the operations of converting between its numbering system and inorder numbering (so-called *node-rank*$_{\text{inorder}}$ and *node-select*$_{\text{inorder}}$) in $O(1)$ time. A consequence of this result is that we can represent the Cartesian tree of a *random* permutation of A in $1.92n + o(n)$ bits in expectation [9], and perform RMQs in $O(1)$ time.

In Section 3, we recall that there is a well-known transformation between binary trees and ordinal trees, which essentially converts inorder numbers in the binary tree to preoreder/postoreder numbers in the ordinal tree, and preserves the preorder/postorder numbers of the binary tree. Using this, we show another method to represent Cartesian trees: transform a Cartesian tree (a binary tree) into an ordinal tree, and then represent the ordinal tree. Using ordinal tree operations on the resulting tree, it is possible to represent the Cartesian tree in optimal space and perform RMQs in constant time. This provides an alternative and more natural way to view the 2D-Min-Heap of [5], as essentially the result of the above transformation of the Cartesian trees into ordinal trees. We also observe a connection between the above transformation and Jacobson's binary tree representation [12].

Finally, in Section 4, we show that constructing the data structure of Section 3 (outputting the data structure given an input array) can be done in linear time using only $O(\sqrt{n}\log n)$ bits of space, "improving" the result of [5] where $n + o(n)$ working space is used (the accounting of space is slightly different). This improvement is useful if the preprocessing and subsequent deployment of the data structure are done on the same machine.

Preliminaries. Given a bit vector, *rank*(i) returns the number of 1s up to the position i in the bit vector, and *select*(i) returns the position of the ith 1 in the bit vector. We use the following succinct representations of bit vectors.

Lemma 1. [16] *Given a bit vector of size m with n 1s, one can construct*

(a) an indexable dictionary *that uses* $\log \binom{m}{n} + o(n) + O(\log\log m)$ *bits, and supports rank, for only those positions where there is a 1 in the bit vector, and select queries in constant time, and*

(b) a fully indexable dictionary *that uses* $\log \binom{m}{n} + o(m)$ *bits, and supports rank and select queries in constant time.*

Given a sequence of balanced parentheses, we define the following operations: *find-close*(i) returns the position of the closing parenthesis that matches the open parenthesis at position i of the sequence (the *find-open*(i) opreation is analogous); *excess*(i) returns the difference between the number of open and closing parentheses from the beginning of the sequence up to position i. The operation *double-enclose*(i, j) returns the position of the pair of matching parentheses that

tightly encloses two non-overlapping pairs of parentheses whose open parentheses respectively appear at positions i and j in the sequence. It is known that in an ordinal tree represented by its *balanced parenthesis representation (BP)*, the LCA of two nodes, whose open (closing) parentheses are in positions i and j, is equivalent to *double-enclose*(i, j) [14].

Lemma 2. **[14,17,13]** *Given a sequence of balanced parentheses of size n, there exists a data structure of size $n + o(n)$ bits that supports the operations* $\text{rank}_($, $\text{select}_($, $\text{rank}_)$, $\text{select}_)$, find-close, find-open, excess, *and* double-enclose *operations on the sequence all in $O(1)$ time.*

2 Representation Based on Tree Decomposition

We show a succinct representation of binary trees that supports multiple numberings (preorder, postorder, DFUDS order and inorder) on the nodes of the tree, plus a comprehensive list of operations suggested by [11,2]. This data structure is essentially the same as the k-ary (cardinal) tree representation of Farzan and Munro [2] for the case when $k = 2$, with the additional support for two more operations, $\text{node-rank}_{\text{inorder}}$ and $\text{node-select}_{\text{inorder}}$. The first operation returns the inorder number of a node given its preorder number, and the second operation performs the inverse. We use the preorder numbers of the nodes to refer to them. Since we can support the *node-rank* and *node-select* operations with respect to inorder, postorder and DFUDS order, we can also use the numberings of the nodes in any of these three orders to refer to them in the operations.

We begin by outlining the succinct representation of Farzan and Munro [2]. Like the representations of [8,11,15], the representation of Farzan and Munro recursively decomposes the tree into sub-trees. A prominent property of their decomposition method is that each sub-tree, aside from its root, has at most one *boundary node* that connects the sub-tree to other sub-trees, and furthermore the boundary node has at most one child outside of the sub-tree. The following lemma states the result of the decomposition:

Lemma 3. **[2, Theorem 1]** *A tree with n nodes can be decomposed into $\Theta(n/L)$ subtrees, each of size at most L. The subtrees are disjoint aside from their roots. Moreover, aside from edges leaving root of subtrees, there is at most one edge in each subtree that connects a node of the subtree to its child in another subtree.*

The ordinal tree is first decomposed using Lemma 3 into $O(n/\log^2 n)$ *mini-trees* each of size $O(\log^2 n)$. Each mini-tree is further decomposed into $O(\log n)$ *micro-trees* of size at most $\lceil \frac{\log n}{2} \rceil$. Each micro-tree is represented with its size, and an index to a lookup table of size $o(n)$ bits, which stores answers of all queries asked within the micro-trees. The sum of the sizes of the representations of all the micro-trees (i.e., the total space for storing all the indexes to the lookup table) is $2n + o(n)$ bits in total, which is the dominating part of the space. Each mini-tree is represented by the explicitly stored list of pointers between the micro-trees within the mini-tree, where each pointer uses only $O(\log \log n)$ bits. The roots of

micro-trees are represented using indexable dictionary structure of Lemma 1(a). The original tree that contains the mini-trees is represented analogously to the representation of mini-trees, by storing the list of pointers between the mini-trees. All the parts together take $2n + o(n)$ bits. The data structure can support the full set of navigational operations and queries in binary trees.

Lemma 4. [2] *A binary tree with n nodes can be represented using $2n + o(n)$ bits of space, while a full set of operations in [2, Table 2] including* LCA *can be supported in $O(1)$ time.*

We now show the main theorem of this section:

Theorem 1. *Given a binary tree with n nodes, there exists a data structure of size $2n + o(n)$ bits, that supports the following operations in $O(1)$ time:* node-rank$_{\text{inorder}}$, node-select$_{\text{inorder}}$, *and all the operations supported by the ordinal tree representation of Farzan and Munro [2] for Cardinal trees, including* LCA.

Proof. As mentioned above, in our data structure, to perform any operation on a node, except node-select operations, we need to give the preorder number of the node to the operation, and the operation also returns a node in the form of its preorder number (this is not the case if the operation does not return a node at all such as depth). Thus, in circumstances in which we are asked to perform an operation on a node referred to by its inorder number, we first need to compute the preorder number of the node and then perform the operation as usual. Similarly, when we are asked to return the result of an operation in the form of an inorder number, we need to compute the inorder number of the node from the preorder number returned by the operation. These two tasks are performed by the operations node-select$_{\text{inorder}}$ and node-rank$_{\text{inorder}}$ respectively.

node-rank$_{\text{inorder}}$. For a node v, we want to compute the inorder number of v, given its preorder number. We count the following; c_1: the number of nodes that are visited before v in inorder traversal but visited after v in preorder traversal; c_2: the number of nodes that are visited after v in inorder traversal but visited before v in preorder traversal. It is not hard to see that the inorder number of v is equal to its preorder number $+ c_1 - c_2$.

The nodes counted in c_1 are all the nodes located in the left subtree of v, which can be counted by subtree size of the left child of v. The nodes counted in c_2 are all the ancestors of v of which left child is on the v-to-root path. We compute c_2 in a way similar to computing the depth of a node as follows. At the root of each mini-tree, we store c_2 of that root, which requires $O((n/\log^2 n)\log n) = o(n)$ bits. At the root r_μ of each micro-tree, we store the local-c_2 of r_μ, that is, the number of ancestors of r_μ, only up to the root of the mini-tree containing r_μ, where their left child is on the r_μ-to-root path. The local-c_2 of v is analogously defined for the ancestors of v within its micro-tree, which can be computed using table lookup. To calculate c_2 of v, we clearly take the sum of the following: c_2 of the root of the mini-tree containing v, local-c_2 of the root of the micro-tree containing v, and local-c_2 of v, all computed in $O(1)$ time.

node-select$_{\text{inorder}}$. For a node v, we want to compute the preorder number of v, given its inorder number. Notice that a node that is visited before v in preorder traversal is the root r_m of the mini-tree containing v. The preorder number of v can be expressed as the sum of two quantities: (1) preorder number of r_m; and (2) the number of nodes that are visited after r_m and before v in preorder traversal, which may include nodes both within and outside the mini-tree. In the following, we explain how to compute these two quantities.

(1) The preorder number of r_m is stored with the mini-tree representation, and thus we only need to find the mini-tree containing the node v. We number all the mini-trees in some arbitrary order, counting from zero up to $n_m - 1$, where $n_m = O(n/\log^2 n)$ is the number of mini-trees. We call these numbers, *names* of the mini-trees. Starting with an empty bit vector A, we traverse the tree in inorder, and after visiting each new node, we append a bit to A as follows: during the traversal when we enter a mini-tree from another mini-tree, we append a 1 to A, and while we are traversing within a mini-tree we append a 0 to A. During the traversal when we enter a mini-tree from another mini-tree, we also write down the name of the current mini-tree in another array B. At the beginning of A we write 1 corresponding to the first visited node (the root), and at the beginning of B, we write the name of the first visited mini-tree (containing the root). Thus, at the end of the traversal, A is a bit vector of length n. We observe that the i-th node in the inorder traversal of the tree belongs to the mini-tree with name $B[j]$ where j to be the number of 1s before $A[i+1]$ (i.e., $j = rank_A(i+1)$).

We store B explicitly as it only requires $O(n_m \cdot \log n) = o(n)$ bits due to the following. The length of B is at most $2 \cdot n_m$ because the traversal can enter a mini-tree at most two times (each mini-tree has at most one edge leaving the mini-tree aside from its root; see Lemma 3), and thus its name can be written in B at most two times. For the same reason, the number of 1s in A is at most $2 \cdot n_m$. We represent A using the FID structure of Lemma 1(b) which uses $O(\log \binom{n}{n_m}) = o(n)$ bits and supports rank operation on A in constant time.

(2) The number of nodes that are visited after r_m and before v in preorder, is computed by taking the sum of the following quantities: (i) the number of such nodes that are outside the mini-tree; (ii) the number of such nodes that are within the micro-tree t_μ containing v; and (iii) the number of such nodes that are within the mini-tree and outside t_μ (visited after r_m and before the root of t_μ). To compute these three quantities, we first need to find t_μ among the other micro-trees within the same mini-tree. We utilize the same method as we used in (1) as follows Each mini-tree plays the role of the original tree in (1) and its micro-trees play the role of mini-trees in (1). That is, we give a name to each micro-tree in the mini-tree; we traverse the mini-tree in inorder; we make the arrays A and B; and we use an FID to encode A. Applying the same analysis provides $o(n)$ bits space.

The nodes in (i) only exist if the mini-tree has a boundary node which is visited before the root of t_μ. The nodes in (i) are in fact all the nodes in a subtree of such a boundary node, and thus the subtree size of the child of the boundary node which is outside of the mini-tree determines the quantity in (i).

The quantity in (ii) is computed using table lookup. The number in (iii) is the local-preorder number of the root of t_μ. We store the local-preorder number of the root of each micro-tree in $O(\log \log n)$ bits which requires $o(n)$ bits in total. This completes the proof of Theorem 1. □

The following theorem gives a slight generalization of Theorem 1, which uses entropy coding to exploit any differences in frequency between the four node types (Theorem 1 corresponds to choosing all the α_is to be 1/4):

Theorem 2. *For any positive constants $\alpha_0, \alpha_L, \alpha_R$ and α_2, such that $\alpha_0 + \alpha_L + \alpha_R + \alpha_2 = 1$, a binary tree with n_0 leaves, n_L (n_R) nodes with only a left (right) child and n_2 nodes with both children can be represented using $\left(\sum_{i\in\{0,L,R,2\}} n_i \log_2 1/\alpha_i\right) + o(n)$ bits of space, while a full set of operations [2, Table 2] including* LCA *can be supported in $O(1)$ time.*

Proof. We proceed as in the proof of Theorem 1, but if $\alpha = \min_{i\in\{0,L,R,2\}} \alpha_i$, we choose the size of the micro-trees to be at most $\mu = \frac{\log n}{2\log_2(1/\alpha)}$. Then, given a micro-tree with μ_i nodes of type i, for $i \in \{0, L, R, 2\}$ we encode it by writing the node types in level order (cf. [12]) and encoding this string using arithmetic coding with the probability of a node of type i taken to be α_i. The size of this micro tree is $\left\lceil \sum_{i\in\{0,L,R,2\}} \mu_i \log_2 1/\alpha_i \right\rceil$, from which the theorem follows. □

Corollary 1. *If A is a random permutation over $\{1, \ldots, n\}$, then RMQ queries on A can be answered using $1.92n + o(n)$ bits in expectation.*

Proof. Choose $\alpha_0 = \alpha_2 = 1/3$ and $\alpha_R = \alpha_L = 1/6$. The claim follows from [9, Theorem 1]. □

3 Transforming Binary Trees into Ordinal Trees

We now give a succinct representation of binary trees based upon a well-known transformation between binary trees and ordinal trees. We show that this transformation not only supports inorder numbering, but also permits navigational and LCA operations by using the relevant operations on the ordinal tree.

Theorem 3. *A binary tree on n nodes can be represented in $2n + o(n)$ bits to support* left-child, right-child, parent, subtree-size *and* LCA *in $O(1)$ time, where the nodes are referred to by any of the inorder, preorder, or postorder numbers.*

Proof. We first describe two (related) transformations between binary trees and ordinal trees, and then describe how binary tree operations can be performed. Let t_b be a binary tree with n nodes that we want to transform to an ordinal tree, and let t_1 and t_2 be the ordinal trees resulting from the first and second transformation respectively. Each of t_1 and t_2 has $n + 1$ nodes, where each node corresponds to a node in t_b, except the root which is dummy. In the first transformation, the root of t_b corresponds to the first child of the dummy root of t_1;

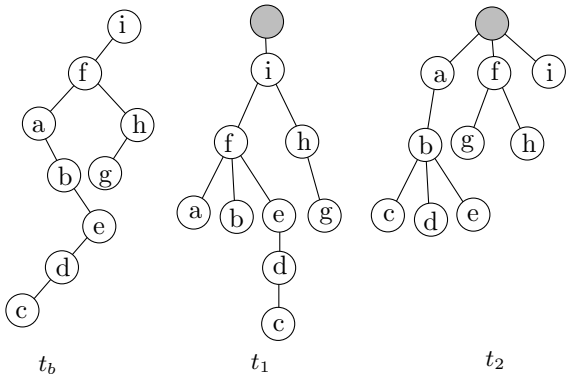


Fig. 1. An example for the transformations: t_b is a binary tree, t_1 and t_2 are the ordinal trees obtained by applying the first and second transformations respectively to t_b. The gray nodes are dummy and do not correspond to any node in t_b.

the left child of a node in t_b corresponds to the first child of the corresponding node in t_1; and the right child of a node in t_b corresponds to the next sibling of the corresponding node in t_1. In the second transformation, the root of t_b corresponds to the last child of the dummy root of t_2; the left child of a node in t_b corresponds to the previous sibling of the corresponding node in t_2; and the right child of a node in t_b corresponds to the last child of the corresponding node in t_2 (see Fig. 1).

These two transformations have a useful property which allows us to use the inorder number as the interface of the operations. In the first transformation, the inorder number of a node in t_b is equal to the postorder number of its corresponding node in t_1. In the second transformation, the inorder number of a node in t_b is equal to the preorder number of its corresponding node in t_2. Furthermore, the preorder number of a node in t_b is equal to the preorder number of its corresponding node in t_1, and the postorder number of a node in t_b is equal to the postorder number of its corresponding node in t_2.

The first and second transformations can be modified to make a third and fourth transformation, respectively, by reversing the order of all siblings in the ordinal tree. That is, if some node is the ith child out of the k children of its parent in the ordinal tree, then in the reverse order, it will be the $k - i + 1$th child of its parent (the new tree can be seen as the mirror image of the original ordinal tree). The third transformation is used in Section 4.

Taking advantage of the transformations and using the known ordinal tree representations that use preorder or postorder numbers as the interface of the operations, we obtain binary trees representations that use the inorder numbers as the interface of the operations. To represent a binary tree, we transform it into an ordinal tree using either of the transformations, and then we represent the ordinal tree by utilizing one of the known succinct representations that supports at least the operations ith-child, parent, next-sibling, previous-sibling, subtree-size, leftmost leaf, rightmost leaf, LCA, level-ancestor, and depth. In the following,

we only show how to support the operations in the binary tree using the first transformation. Supporting the operations on the second transformation is analogous. Given a node v in a binary tree t_b, let v_{t_1} denote the corresponding node in t_1, the transformed binary tree using the first transformation.

left-child, right-child, parent. The left child of a node v in t_b is the first child of the node v_{t_1}, which can be determined using the operation ith-child on t_1. The right child of a node v in t_b is the next sibling of v_{t_1}, which can be determined by the operation next-sibling on t_1. For the parent of a node v in t_b there are two cases: 1) if v_{t_1} is the first child of its parent, then the answer is the parent; 2) if v_{t_1} is not the first child of its parent, then the answer is the previous sibling of v_{t_1}. These also can be determined using the operations ith-child, parent, and previous-sibling.

subtree-size. It is not difficult to see that the subtree size of v is equal to the sum of the subtree size of v_{t_1} and the subtree sizes of all the siblings to the right of v_{t_1}. Let ℓ be the right-most leaf in the subtree of the parent of v_{t_1}. To obtain the above sum, we only subtract the preorder number of v_{t_1} from the preorder number of ℓ.

LCA. Let w be the LCA of two nodes u and v in t_b that we want to compute. Notice that the LCA of u_{t_1} and v_{t_1} is a node z_{t_1}, that is a child of w_{t_1} and an ancestor of u_{t_1}, assuming that u is to the left of v in t_b. Thus, we only need to find the ancestor of u_{t_1} at level i, where $i-1$ is the depth of z_{t_1}. To compute this, we utilize the operations LCA, depth, and level-ancestor on t_1. □

We now observe an interesting connection between the above transformation and the binary tree representation of Jacobson [12]. Given a binary tree, we first add external nodes wherever there is a missing child, and label the internal nodes with an open parenthesis, and the external nodes with a closing parenthesis. We then traverse the tree in preorder and write down the labels of the nodes visited in the traversal order (this is similar to Jacobson's [12] encoding, except that he visits the tree in level-order). If the original tree has n nodes, the sequence so obtained has length $2n+1$ (as $n+1$ external nodes are added to the tree). It is easy to show that by adding an extra open parenthesis at the beginning, we get a balanced parenthesis sequence S of length $2n+2$. See Fig. 2 for an example. Note that in the depth-first search, if we switch the order in which the children of a node are visited (i.e., visit the right child before the left child), then the resulting sequence obtained is the balanced parenthesis sequence of the tree obtained by applying the first transformation to the given binary tree.

Furthermore, each open parenthesis in S, except the extra parenthesis that is added, and its matching closing parenthesis, are (conceptually) associated with a node in the given tree that was visited when the parenthesis is added to the sequence. It is easy to verify that the open parentheses in S from left to right correspond to the nodes in preorder, and the closing parentheses from left to right correspond to the nodes in inorder.

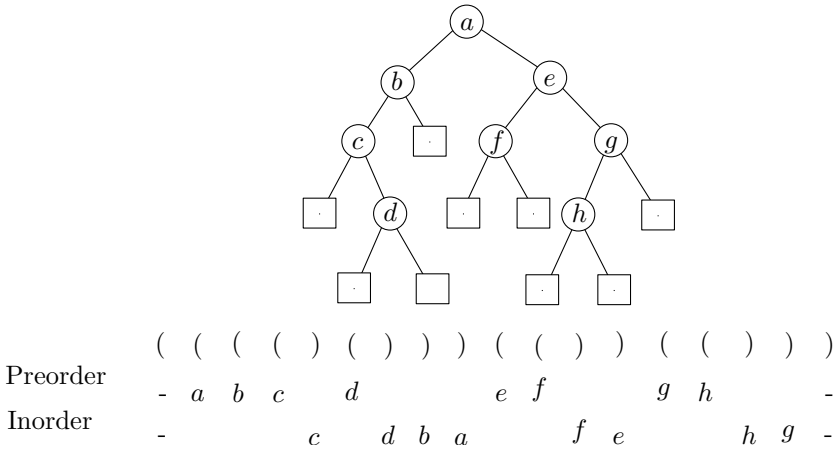


Fig. 2. Illustrating the connection between Jacobson's approach to representing binary trees and the representation of Theorem 3 to a binary tree. Note that the open/closing parentheses from left to right are in the same order as a preorder/inorder traversal of the nodes respectively.

4 Cartesian Tree Construction in $o(n)$ Working Space

We show how to construct the succinct representation of Section 3, using only $o(n)$ bits during the construction. A straightforward way to construct a succinct representation of a Cartesian tree is to construct the standard pointer-based representation of the Cartesian tree from the given array in linear time [6], and then construct the succinct representation using the pointer-based representation. The drawback of this approach is that the space used during the construction is $O(n \log n)$ bits, although the final structure uses only $O(n)$ bits. Fischer and Heun [5] show that the construction space can be reduced to $n + o(n)$ bits. In this section, we show how to improve the construction space to $o(n)$ bits.

Theorem 4. *Given an array A of n values, we can build a $2n + o(n)$-bit representation of its Cartesian tree in $O(n)$ time using $o(n)$ bits of auxiliary space.*

Proof. The proof assumes that the array A is present in read-only memory and it is possible to randomly access A. The algorithm reads A from left to right, and outputs a parenthesis sequence as follows: if, having completed the preprocessing for $A[1], \ldots, A[i]$, for some $i \geq 0$, when processing the $A[i+1]$, we compare $A[i+1]$ with all the suffix minima of $A[1..i]$—if $A[i+1]$ is smaller than $j \geq 0$ suffix minima, then we output the string $)^j($. This is so far a restatement of the algorithm of [6] for constructing a Cartesian tree, and it is not hard to see that the string output is balanced, by adding j closing parentheses to the end, where j is number of suffix minima of $A[1..n]$. This sequence is in fact the reverse of the DFUDS sequence of the ordinal tree obtained by applying the third transformation of Section 3 to the Cartesian tree. While the straightforward

approach would be to maintain a linked list of the locations of the current suffix minima, this list could contain $\Theta(n)$ locations and could take $\Theta(n \log n)$ bits.

Our approach is to use the output string itself to encode the positions of the suffix minima. It is not hard to see that if the output string is created by the above process, it will be of the form $b_0(b_1(...(b_k($ where each b_i is a (possibly empty) maximal balanced parenthesis string – the remaining parentheses are called *unmatched*. It is not hard to see that the unmatched parentheses encode the positions of the suffix minima in the sense that if the unmatched parentheses are the $i_1, i_2 \ldots, i_k$-th opening parentheses in the current output sequence then the position $i_1, \ldots, i_k$ are precisely the suffix minima positions. Our task is therefore to sequentially access the next unmatched parenthesis, starting from the end, when adding the new element $A[i+1]$. We conceptually break the string into blocks of size $\lfloor \sqrt{n} \rfloor$. For each block that contains at least one unmatched parenthesis, store the following info:

- it's block number (in the original paren string) and the total number of open parenthesis in the current output string before the start of the block.
- the position p of the rightmost paren in the block, and the number of open parentheses before it in the block.
- a pointer to the next block with at least one unmatched parenthesis.

This takes $O(\log n)$ bits per block, which is $O(\sqrt{n} \log n)$ bits.

- For the rightmost block (in which we add the new parens), keep positions of all the unmatched parens: the space for this is also $O(\sqrt{n} \log n)$ bits.

When we process the next element of A, we compare it with unmatched parens in the rightmost block, which takes $O(1)$ time per unmatched paren that we compared the new element with, as in the algorithm of [6]. Updating the last block is also trivial. Suppose we have compared $A[i+1]$ and found it smaller than all suffix maxima in the rightmost block. Then, using the linked list, we find the rightmost unmatched paren (say at position p) in the next block in the list, which takes $O(1)$ time, and compare with it (this is also $O(1)$ time). If $A[i+1]$ is smaller, then sequentially scan this block leftwards starting at position p, skipping over a maximal BP sequence to find the next unmatched paren in that block. The time for this sequential scan is $O(n)$ overall, since we never sequentially scan the same paren twice. Updating the blocks is straightforward. Thus, the creation of the output string can be done in linear time using $O(\sqrt{n} \log n)$ bits. For constructing the auxiliary structures for the DFUDS in linear time see [7]. □

References

1. Benoit, D., Demaine, E.D., Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Representing trees of higher degree. Algorithmica 43(4), 275–292 (2005)
2. Farzan, A., Munro, J.I.: A Uniform Approach Towards Succinct Representation of Trees. In: Gudmundsson, J. (ed.) SWAT 2008. LNCS, vol. 5124, pp. 173–184. Springer, Heidelberg (2008)

3. Farzan, A., Raman, R., Rao, S.S.: Universal Succinct Representations of Trees? In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 451–462. Springer, Heidelberg (2009)
4. Fischer, J.: Optimal Succinctness for Range Minimum Queries. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 158–169. Springer, Heidelberg (2010)
5. Fischer, J., Heun, V.: Space-efficient preprocessing schemes for range minimum queries on static arrays. SIAM Journal on Computing 40(2), 465–492 (2011)
6. Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and related techniques for geometry problems. In: Proc. 16th Annual ACM Symposium on Theory of Computing, pp. 135–143. ACM Press (1984)
7. Geary, R.F., Rahman, N., Raman, R., Raman, V.: A simple optimal representation for balanced parentheses. Theoretical Computer Science 368(3), 231–246 (2006)
8. Geary, R.F., Raman, R., Raman, V.: Succinct ordinal trees with level-ancestor queries. ACM Transactions on Algorithms 2(4), 510–534 (2006)
9. Golin, M., Iacono, J., Krizanc, D., Raman, R., Rao, S.S.: Encoding 2D Range Maximum Queries. In: Asano, T., Nakano, S.-i., Okamoto, Y., Watanabe, O. (eds.) ISAAC 2011. LNCS, vol. 7074, pp. 180–189. Springer, Heidelberg (2011)
10. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM Journal on Computing 13(2), 338–355 (1984)
11. He, M., Munro, J.I.J., Rao, S.S.: Succinct Ordinal Trees Based on Tree Covering. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 509–520. Springer, Heidelberg (2007)
12. Jacobson, G.: Succinct Static Data Structures. PhD thesis. Carnegie Mellon University, Pittsburgh, PA, USA (1989)
13. Lu, H.-I., Yeh, C.-C.: Balanced parentheses strike back. ACM Transactions on Algorithms 4(3) (2008)
14. Munro, J.I., Raman, V.: Succinct representation of balanced parentheses and static trees. SIAM Journal on Computing 31(3), 762–776 (2001)
15. Munro, J.I., Raman, V., Storm, A.J.: Representing dynamic binary trees succinctly. In: Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 529–536. SIAM (2001)
16. Raman, R., Raman, V., Satti, S.R.: Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. ACM Transactions on Algorithms 3(4) (2007)
17. Sadakane, K.: Succinct representations of lcp information and improvements in the compressed suffix arrays. In: Proc. 13th Symposium on Discrete Algorithms, pp. 225–232 (2002)
18. Sadakane, K.: Succinct data structures for flexible text retrieval systems. Journal of Discrete Algorithms 5(1), 12–22 (2007)
19. Sadakane, K., Navarro, G.: Fully-functional succinct trees. In: Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 134–149. SIAM (2010)
20. Vuillemin, J.: A unifying look at data structures. Communications of the ACM 23(4), 229–239 (1980)

Lower Bounds against Weakly Uniform Circuits

Ruiwen Chen and Valentine Kabanets

School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada
ruiwenc@sfu.ca, kabanets@cs.sfu.ca

Abstract. A family of Boolean circuits $\{C_n\}_{n\geqslant 0}$ is called $\gamma(n)$-*weakly uniform* if there is a polynomial-time algorithm for deciding the direct-connection language of every C_n, given *advice* of size $\gamma(n)$. This is a relaxation of the usual notion of uniformity, which allows one to interpolate between complete uniformity (when $\gamma(n) = 0$) and complete non-uniformity (when $\gamma(n) > |C_n|$). Weak uniformity is essentially equivalent to *succinctness* introduced by Jansen and Santhanam [12].

Our main result is that PERMANENT is not computable by polynomial-size $n^{o(1)}$-weakly uniform TC^0 circuits. This strengthens the results by Allender [2] (for *uniform* TC^0) and by Jansen and Santhanam [12] (for weakly uniform *arithmetic* circuits of constant depth). Our approach is quite general, and can be used to extend to the "weakly uniform" setting all currently known circuit lower bounds proved for the "uniform" setting. For example, we show that PERMANENT is not computable by polynomial-size $(\log n)^{O(1)}$-weakly uniform threshold circuits of depth $o(\log \log n)$, generalizing the result by Koiran and Perifel [16].

Keywords: advice complexity classes, alternating Turing machines, counting hierarchy, permanent, succinct circuits, threshold circuits, uniform circuit lower bounds, weakly uniform circuits.

1 Introduction

Understanding the power and limitation of efficient algorithms is the major goal of complexity theory, with the "P vs. NP" problem being the most famous open question in the area. While proving that no NP-complete problem has a uniform polynomial-time algorithm would suffice for separating P and NP, a considerable amount of effort was put into the more ambitious goal of trying to show that no NP-complete problem can be decided by even a *nonuniform* family of polynomial-size Boolean circuits.

More generally, an important goal in complexity theory has been to prove strong (exponential or super-polynomial) circuit lower bounds for "natural" computational problems that may come from complexity classes larger than NP, e.g., the class NEXP of languages decidable in nondeterministic exponential time. By the counting argument of Shannon [23], a randomly chosen n-variate Boolean function requires circuits of exponential size. However, the best currently known circuit lower bounds for *explicit* problems are only linear for NP problems [17,11], and polynomial for problems in the polynomial-time hierarchy PH [14].

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 408–419, 2012.

To make progress, researchers introduced various restrictions on the circuit classes. In particular, for Boolean circuits of *constant* depth, with NOT and unbounded fan-in AND and OR gates (AC^0 circuits), exponential lower bounds are known for the PARITY function [8,29,9]. For constant-depth circuits that additionally have (unbounded fan-in) MOD_p gates, one also needs exponential size to compute the MOD_q function, for any distinct primes p and q [20,24]. With little progress for decades, Williams [28] has recently shown that a problem in NEXP is not computable by polynomial-size ACC^0 circuits, which are constant-depth circuits with NOT gates and unbounded fan-in AND, OR and MOD_m gates, for any integer $m > 1$. However, no lower bounds are known for the class TC^0 of constant-depth threshold circuits with unbounded fan-in majority gates.[1]

To make more progress, another restriction has been added: *uniformity* of circuits. Roughly speaking, a circuit family is called uniform if there is an efficient algorithm that can construct any circuit from the family. There are two natural variations of this idea. One can ask for an algorithm that outputs the entire circuit in time polynomial in the circuit size; this notion of uniformity is known as P-uniformity. In the more restricted notion, one asks for an algorithm that describes the local structure of the circuit: given two gate names, such an algorithm determines if one gate is the input to the other gate, as well as determines the types of the gates, in time linear (or polynomial) in the input size (which is logarithmic or polylogarithmic time in the size of the circuit described by the algorithm); such an algorithm is said to decide the *direct-connection language* of the given circuit. This restricted notion is called DLOGTIME- (or POLYLOGTIME-) uniformity [22,5,3]. We will use the notion of POLYLOGTIME-uniformity by default, and, for brevity, will omit the word POLYLOGTIME.

It is easy to show (by diagonalization) that, for any fixed exponential function $s(n) = 2^{n^c}$ for a constant $c \geqslant 1$, there is a language in EXP (deterministic exponential time) that is not computable by a uniform (even P-uniform) family of Boolean $s(n)$-size circuits.[2] Similarly, as observed in [2], a PSPACE-complete language requires exponential-size uniform TC^0 circuits. For the smaller complexity class $\#\mathsf{P} \subseteq \mathsf{PSPACE}$, Allender and Gore [3] showed PERMANENT (which is complete for $\#\mathsf{P}$ [26]) is not computable by uniform ACC^0 circuits of subexponential size. Later, Allender [2] proved that PERMANENT cannot be computed by uniform TC^0 circuits of size $s(n)$ for any function s such that, for all k, $s^{(k)}(n) = o(2^n)$ (where $s^{(k)}$ means the function s composed with itself k times). Finally, Koiran and Perifel [16] extended this result to show that PERMANENT is not computed by polynomial-size uniform threshold circuits of depth $o(\log \log n)$.

Recently, Jansen and Santhanam [12] have proposed a natural relaxation of uniformity, termed *succinctness*, which allows one to interpolate between nonuniformity and uniformity. According to [12], a family of $s(n)$-size circuits $\{C_n\}$

[1] A plausible explanation of this "barrier" is given by the "natural proofs" framework of [21], who argue it is hard to prove lower bounds against the circuit classes that are powerful enough to implement cryptography.

[2] Unlike the nonuniform setting, where every n-variate Boolean function is computable by a circuit of size about $2^n/n$ [18], *uniform* circuit lower bounds can be $> 2^n$.

is succinct if the direct-connection language of C_n is decided by some circuit of size $s(n)^{o(1)}$. In other words, while there may not be an efficient algorithm for describing the local structure of a given $s(n)$-size circuit C_n, the local structure of C_n can be described by a *non-uniform* circuit of size $s(n)^{o(1)}$. Note that if we allow the non-uniform circuit to be of size $s(n)$, then the family of circuits $\{C_n\}$ would be completely non-uniform. So, intuitively, the restriction to the size $s(n)^{o(1)}$ makes the notion of succinctness close to that of non-uniformity.

The main result of [12] is that PERMANENT does not have succinct polynomial-size *arithmetic* circuits of constant depth, where arithmetic circuits have unbounded fan-in addition and multiplication gates and operate over integers. While relaxing the notion of uniformity, [12] were only able to prove a lower bound for the *weaker* circuit class, as polynomial-size constant-depth arithmetic circuits can be simulated by polynomial-size TC^0 circuits. A natural next step was to prove a super-polynomial lower bound for PERMANENT against succinct TC^0 circuits. This is achieved in the present paper.

1.1 Our Main Results

We improve upon [12] by showing that PERMANENT does not have succinct polynomial-size TC^0 circuits. In addition to strengthening the main result from [12], we also give a simpler proof. Our argument is quite general and allows us to extend to the "succinct" setting all previously known uniform circuit lower bounds of [3,2,16].

Recall that the direct-connection language for a circuit describes the local structure of the circuit; more precise definitions will be given in the next section. For a function $\alpha : \mathbb{N} \to \mathbb{N}$, we say that a circuit family $\{C_n\}$ of size $s(n)$ is α-*weakly uniform* if the direct-connection language L_{dc} of $\{C_n\}$ is decided by a polynomial-time algorithm that, in addition to the input of L_{dc} of size $m \in O(\log s(n))$, has an advice string of size $\alpha(m)$; the advice string just depends on the input size m. The notion of α-weakly uniform is essentially equivalent to the notion of α-succinct introduced in [12]; see the next section for details.

We will call a circuit family subexp-*weakly uniform* if it is α-weakly uniform for $\alpha(m) \in 2^{o(m)}$. Similarly, we call a circuit family poly-*weakly uniform* if it is α-weakly uniform for $\alpha(m) \in m^{O(1)}$. Observe that for $m = O(\log s)$, we have $2^{o(m)} = s^{o(1)}$ and $m^{O(1)} = \mathsf{poly} \log s$.

Our main results are as below. First, we strengthen the lower bound of [12].

Theorem 1. PERMANENT *is not computable by* subexp-*weakly uniform* poly-*size* TC^0 *circuits.*

Let us call a function $s(n)$ *sub-subexponential* if, for any constant $k > 0$, we have that the k-wise composition $s^{(k)}(n) \leqslant 2^{n^{o(1)}}$. We use $\mathsf{subsubexp}$ to denote the class of all sub-subexponential functions $s(n)$. We extend a result of Allender [2] to the "weakly-uniform" setting.

Theorem 2. PERMANENT *is not computable by* poly-*weakly uniform* $\mathsf{subsubexp}$-*size* TC^0 *circuits.*

Finally, we extend the result of [16].

Theorem 3. PERMANENT *is not computable by* poly*-weakly uniform* poly*-size threshold circuits of depth* $o(\log \log n)$.

1.2 Our Techniques

At the high level, we use the method of *indirect diagonalization*:

- assuming PERMANENT is easy and using diagonalization, we first show the existence of a "hard" language in a certain complexity class $\mathcal{C}$ (the counting hierarchy, to be defined below);
- assuming PERMANENT is easy, we show that the above "hard" language is actually "easy" (as the easiness of PERMANENT collapses the counting hierarchy), which is a contradiction.

In more detail, we first extend the well-known correspondence between uniform TC^0 and alternating polylog-time Turing machines (that use majority states) to the weakly uniform setting, by considering alternating Turing machines with *advice*. To construct the desired "hard" language, we use diagonalization against such machines with advice. The assumed easiness of PERMANENT is used to argue two things about the constructed "hard" language L_{hard}:

1. L_{hard} is in fact "hard" for a much more powerful class $\mathcal{A}$ of algorithms;
2. L_{hard} is decided by a "simple" algorithm A.

The contradiction ensues since algorithm A turns out to be from the class $\mathcal{A}$.

1.3 Relation to the Previous Work

A similar indirect-diagonalization strategy was used (explicitly or implicitly) in all previous papers showing uniform or weakly uniform circuit lower bounds for PERMANENT [3,2,16,12]. Our approach is most closely related to that of [2,16]. The main difference is that we work in the weakly uniform setting, which means that we need to handle a certain amount of non-uniform advice. To that end, we have adapted the method of indirect diagonalization, making it modular (as outlined above) and sufficiently general to work also in the setting with advice. Due to this generality of our proof argument, we are able to extend the aforementioned lower bounds from the uniform setting to the weakly uniform setting.

The approach adopted by [12] goes via the well-known connection between derandomization and circuit lower bounds (cf. [10,13,1]). Since the authors of [12] work with the algebraic problem of Polynomial Identity Testing (given an arithmetic circuit computing some polynomial over integers, decide if the polynomial is identically zero), their final lower bounds are also in the algebraic setting: for weakly uniform arithmetic constant-depth circuits. By making the diagonalization arguments in [12] more explicit (along the lines of [2]), we are able to get the lower bound for weakly uniform Boolean (TC^0) circuits, thereby both strengthening the results and simplifying the proofs from [12].

2 Preliminaries

We refer to [4] for the basic complexity notions.

2.1 Weakly Uniform Circuit Families

Following [22,3], we define the *direct connection language* of a circuit family $\{C_n\}$ as $L_{dc} = \{(n, g, h)\colon g = h$ and g is a gate in C_n, or $g \neq h$ and h is an input to $g\}$, where n is in binary representation, and g and h are binary strings encoding the gate types and names. The *type* of a gate could be constant 0 or 1, Boolean logic gate NOT, AND, or OR, majority gate MAJ, modulo gate MOD_m for some integer m, or input $x_1, x_2, \ldots, x_n$. For a circuit family of size $s(n)$, we need $c_0 \log s(n)$ bits to encode (n, g, h), where c_0 is a small constant at most 4.

A circuit family $\{C_n\}$ is *uniform* [5,3] if its direct connection language is decidable in time polynomial in its input length $|(n, g, h)|$; this was referred to as POLYLOGTIME-uniformity in [3].

We say a function $f(n)$ is *constructible* if there is a deterministic TM that computes $f(n)$ in binary in time $O(f(n))$, when given n in binary as the input[3].

Following [12], for a constructible function $\alpha : \mathbb{N} \to \mathbb{N}$, we say that a circuit family $\{C_n\}$ of size $s(n)$ is *α-succinct* if its direct connection language L_{dc} is in $\mathsf{SIZE}(\alpha)$; i.e., L_{dc} has (non-uniform) Boolean circuits of size $\alpha(m)$, where $m = c_0 \log s(n)$ is the input size for L_{dc}. Trivially, for $\alpha(m) \geqslant 2^m$, every circuit family is α-succinct. The notion becomes nontrivial when $\alpha(m) \ll 2^m/m$. We will use $\alpha(m) = 2^{o(m)}$ (slightly succinct) and $\alpha(m) = m^{O(1)}$ (highly succinct).

We recall the definition of Turing machines with advice from [15]. Given functions $t\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and $\alpha\colon \mathbb{N} \to \mathbb{N}$, we say that a language L is in $\mathsf{DTIME}(t)/\alpha$, if there is a deterministic Turing machine M and a sequence of advice strings $\{a_n\}$ of length $\alpha(n)$ such that, for any $x \in \{0,1\}^n$, machine M on inputs (x, a_n) decides whether $x \in L$ in time $t(n, \alpha(n))$. If the function $t(n, m)$ is upper-bounded by a polynomial in $n + m$, we say that $L \in \mathsf{P}/\alpha$.

Definition 1. *A circuit family $\{C_n\}$ of size $s(n)$ is* α-weakly uniform *if its direct connection language is decided in P/α; recall that the input size for the direct-connection language describing C_n is $m = c_0 \log s(n)$, and so the size of the advice string needed in this case is $\alpha(c_0 \log s(n))$.*

The two notions are closely related.

Lemma 1. *In the notation above, $\alpha(m)$-succinctness implies $\alpha(m) \log \alpha(m)$-weak uniformity, and conversely, $\alpha(m)$-weak uniformity implies $(\alpha(m)+m)^{O(1)}$-succinctness.*

Proof (sketch). A Boolean circuit of size s can be represented by a binary string of size $O(s \log s)$; and a Turing machine running in time t can be simulated by a circuit family of size $O(t \log t)$. □

[3] We note that $f(n)$ is constructible in our sense if and only if $2^{f(n)}$ is constructible according to Allender's definition in [2].

The notion of weak uniformity (succinctness) interpolates between full uniformity on one end and full non-uniformity on the other end. For example, 0-weak uniformity is the same as uniformity. On the other hand, α-weak uniformity for $\alpha(m) \geqslant 2^m$ is the same as non-uniformity. For that reason, we will assume that the function α in "α-weakly uniform" is such that $0 \leqslant \alpha(m) \leqslant 2^m$.

Definition 2. *We say a circuit family* $\{C_n\}$ *is* subexp-weakly uniform *if it is* α*-weakly uniform for* $\alpha(m) \in 2^{o(m)}$*; similarly, we say* $\{C_n\}$ *is* poly-weakly uniform *if it is* α*-weakly uniform for* $\alpha(m) \in m^{O(1)}$*.*

2.2 Weak Uniformity vs. Alternating Turing Machines with Advice

Following [7,19,3], a *threshold Turing machine* is an alternating TM (ATM) with majority (MAJ) states; a configuration in *majority state* may have an unbounded number of successors, and it is accepting iff more than half of its successors are accepting. We denote by $\mathsf{Th}_{d(n)}\mathsf{TIME}(t(n))$ the class of languages accepted by threshold TMs having at most $d(n)$ alternations and running in time $O(t(n))$.

The *counting hierarchy* [27,25] is defined as $\mathsf{CH} = \cup_{d \geqslant 0}\mathsf{CH}_d$, where $\mathsf{CH}_0 = \mathsf{P}$ and $\mathsf{CH}_{d+1} = \mathsf{PP}^{\mathsf{CH}_d}$. The counting hierarchy can be equivalently defined via threshold Turing machines: $\mathsf{CH}_d = \mathsf{Th}_d\mathsf{TIME}(n^{O(1)})$.

It is well-known that uniform $\mathsf{AC}^0(2^{\mathsf{poly}(n)})$ corresponds to the polynomial-time hierarchy PH [8]. Similarly, the correspondence exists between uniform $\mathsf{TC}^0(2^{\mathsf{poly}(n)})$ and the counting hierarchy CH [19,5,2]. For constructible $t(n)$ such that $t(n) = \Omega(\log n)$, we have $\cup_{d \geqslant 0}\mathsf{Th}_d\mathsf{TIME}(\mathsf{poly}(t(n)))$ is precisely the class of languages decided by uniform $\mathsf{TC}^0(2^{\mathsf{poly}(t(n))})$.

The following lemma gives the correspondence between weakly uniform threshold circuits and threshold TMs with advice. The proof follows from [3], and is left to the full version [6].

Lemma 2. *Let* L *be any language decided by a family of* α*-weakly uniform* $d(n)$*-depth threshold circuits of size* $s(n)$*. Then* L *is decidable by a threshold Turing machine with* $d'(n) = 3d(n) + 2$ *alternations, taking advice of length* $\alpha(m)$ *for* $m = c_0 \log s(n)$*, and running in time* $t(n) = d'(n) \cdot \mathsf{poly}(m + \alpha(m))$*.*

3 Indirect Diagonalization

Here we establish the components needed for our indirect diagonalization, as outlined in Section 1.2. First, in Section 3.1, we give a diagonalization argument against alternating Turing machines with advice, getting a language in the counting hierarchy CH that is "hard" against weakly uniform TC^0 circuits of certain size. Then, in Section 3.2, using the assumption that a canonical P-complete problem has small weakly uniform TC^0 circuits, we conclude that the "hard" language given by our diagonalization step is actually hard for a stronger class of algorithms: weakly uniform Boolean circuits of some size s' *without any depth restriction*. Finally, in Section 3.3, using the assumption that PERMANENT

has small weakly uniform TC^0 circuits, we show that CH collapses, and our assumed hard language is in fact decidable by weakly uniform s'-size Boolean circuits, which is a contradiction. (Our actual argument is more general: we consider threshold circuits of not necessarily constant depth $d(n)$, and non-constant levels of the counting hierarchy.)

3.1 Diagonalization against ATMs with Advice

Lemma 3. *For any constructible functions $\alpha, d, t, T : \mathbb{N} \to \mathbb{N}$ such that $\alpha(n) \in o(n)$ and $t(n) \log t(n) = o(T(n))$, there is a language $D \in \mathsf{Th}_{d(n)}\mathsf{TIME}(T(n))$ which is not decided by threshold Turing machines with $d(n)$ alternations running in time $t(n)$ and taking advice of length $\alpha(n)$.*

Proof. Define the language D consisting of those inputs x of length n that have the form $x = (M, y)$ (using some pairing function) such that the threshold TM M with advice y, where $|y| = \alpha(n)$, rejects input (M, y) in time $t(n)$ using at most $d(n)$ alternations. Language D is decided in $\mathsf{Th}_{d(n)}\mathsf{TIME}(T(n))$ by simulating M and flipping the result[4].

For contradiction, suppose that D is decided by some threshold Turing machine M_0 with $d(n)$ alternations taking advice $\{a_n\}$ of size $\alpha(n)$. Consider the input (M_0, a_n) with $|M_0| = n - \alpha(n)$; we assume that each TM has infinitely many equivalent descriptions (by padding), and so for large enough n, there must exist such a description of size $n - \alpha(n)$. By the definition of D, we have (M_0, a_n) is in D iff M_0 with advice a_n rejects it; but this contradicts the assumption that M_0 with advice $\{a_n\}$ decides D. □

3.2 If P Is Easy

Let L_0 be a P-complete language under uniform projections (functions computable by uniform Boolean circuits with NOT gates only). For example, the standard P-complete set $\{(M, x, 1^t) : M \text{ accepts } x \text{ in time } t\}$ works.

Lemma 4. *Suppose L_0 is decided by a family of α-weakly uniform $d(n)$-depth threshold circuits of size $s(n)$. Then, for any constructible function $t(n) \geqslant n$ and $0 \leqslant \beta(m) \leqslant 2^m$, every language L in β-weakly uniform $\mathsf{SIZE}(t(n))$ is decided by $\mu(n)$-weakly uniform $d(\mathsf{poly}(t(n)))$-depth threshold circuits of size $s'(n) = s(\mathsf{poly}(t(n)))$ on n inputs, where $\mu(n) = \alpha(c_0 \log s'(n)) + \beta(c_0 \log t(n))$.*

Proof. Let U be an advice-taking algorithm deciding the direct-connection language for the $t(n)$-size circuits for L. For any string y of length $\beta(m)$ for

[4] $\mathsf{Th}_{d(n)}\mathsf{TIME}(T(n))$ is closed under complement since the negation of MAJ is MAJ of negated inputs when MAJ has an odd number of inputs; the latter is easy to achieve by replacing $MAJ(x_1, \ldots, x_k)$ with $MAJ(x_1, x_1, \ldots, x_k, x_k, 0)$. Allender [2] uses a lazy diagonalization argument [30] for nondeterministic TMs. However, that argument seems incapable of handling the amount of advice we need. Fortunately, the basic diagonalization argument we use here is sufficient for our purposes.

$m = c_0 \log t(n)$, we can run U with the advice y to construct some circuit C^y of size $t(n)$ on n inputs. We can construct the circuit C^y in time at most $\mathsf{poly}(t(n))$, and then evaluate it in time $\mathsf{poly}(t(n))$ on any given input of size n.

Consider the language $L' = \{(x, y, 1^{t(n)}) \mid |x| = n,\ |y| = \beta(m),\ C^y(x) = 1\}$. By the above, we have $L' \in \mathsf{P}$. Hence, by assumption, L' is decided by an α-weakly uniform $d(l)$-depth threshold circuits of size $s(l)$, where $l = |(x, y, 1^{t(n)})| \leqslant \mathsf{poly}(t(n))$. To get a circuit for L, we simply use as y the advice of size $\beta(m)$ needed for the direct-connection language of the $t(n)$-size circuits for L. Overall, we need $\alpha(c_0 \log s(l)) + \beta(m)$ amount of advice to decide L by weakly uniform $d(\mathsf{poly}(t(n)))$-depth threshold circuits of size $s(\mathsf{poly}(t(n)))$. □

3.3 If Permanent Is Easy

Since PERMANENT is hard for the first level of the counting hierarchy CH, assuming that PERMANENT is "easy" implies the collapse of CH (see, e.g., [2]). It was observed in [16] that it is also possible to collapse super-constant levels of CH, under the same assumption. Below we argue the collapse of super-constant levels of CH by assuming that PERMANENT has "small" weakly uniform circuits.

We use the notation $f \circ g$ to denote the composition of the functions f and g, and the notation $f^{(i)}$ is used to denote the composition of f with itself for i times; we use the convention that $f^{(0)}$ is the identity function.

Lemma 5. *Suppose that* PERMANENT *is in γ-weakly uniform $\mathsf{SIZE}(s(n))$, for some $\gamma(m) \leqslant 2^{o(m)}$. For every $d(n) \leqslant n^{o(1)}$, every language A in $\mathsf{Th}_{d(n)}\mathsf{TIME}(\mathsf{poly})$ is also in $(2d(n) \cdot \gamma)$-weakly uniform $\mathsf{SIZE}((s \circ q)^{(d(n)+1)}(n))$, for some polynomial q dependent on A.*

Proof. The language A is computable by a uniform threshold circuit family $\{C_n\}$ of depth $d(n)$ and size $\mathsf{poly}(n)$. Let M be a polynomial-time TM deciding the direct-connection language of $\{C_n\}$. More precisely, we identify the gates of the circuit with the configurations of the given threshold TM for A; the output gate is the initial configuration; leaf (input) gates are halting configurations; deciding if one gate is an input to the other gate is deciding if one configuration follows from the other according to our threshold TM, and so can be done in polynomial time (dependent on A); finally, given a halting configuration, we can decide if it is accepting or rejecting also in polynomial time (dependent on A).

Consider an arbitrary n. Let $d = d(n)$. For a gate g of C, we denote by C_g the subcircuit of C that determines the value of the gate g. We say that g is at depth i, for $1 \leqslant i \leqslant d$, if the circuit C_g is of depth i. Note that each gate at depth $i \geqslant 1$ is a majority gate.

For every $0 \leqslant i \leqslant d$, let B_i be a circuit that, given $x \in \{0,1\}^n$ and a gate g at depth i, outputs the value $C_g(x)$.

Claim. There are polynomials q and q' dependent on A such that, for each $0 \leqslant i \leqslant d$, there are $2i\gamma$-weakly uniform circuits B_i of size $(s \circ q)^{(i)} \circ q'$.

Proof. We argue by induction on i. For $i = 0$, to compute $B_0(x, g)$, we need to decide if the halting configuration g of our threshold TM for A on input x is accepting or not; by definition, this can be done by the TM M in deterministic polynomial time. Hence, B_0 can be decided by a completely uniform circuit of size at most $q'(n)$ for some polynomial q' dependent on the running time of M.

Assume we have the claim for i. Let s' be the size of the γ'-weakly uniform circuit B_i, where $s' \leqslant (s \circ q)^{(i)} \circ q'$ and $\gamma' \leqslant 2i\gamma$. Consider the following TM N:

> "On input $z = (x, g, U, y, 1^{s'^2})$, where $|x| = n$, g is a gate of C, $|U| = \gamma(c_0 \log s')$, $|y| = \gamma'(c_0 \log s')$, interpret U as a Turing machine that takes advice y to decide the direct-connection language of some circuit D of size s' on inputs of length $|(x, g)|$. Construct the circuit D using U and y, where to evaluate U on a given input we simulate U for at most s' steps. Enter the MAJ state. Nondeterministically guess a gate h of C and a bit $b \in \{0, 1\}$. If h is not an input gate for g, then accept if $b = 1$ and reject if $b = 0$; otherwise, accept if $D(x, h) = 1$ and reject if $D(x, h) = 0$."

We will be interested in the case where U is a polynomial-time TM. For any such U, the running time on any input is bounded by $\mathsf{poly}(c_0 \log s' + \gamma'(c_0 \log s'))$, which is less than s' by our assumptions that $\gamma(m) \leqslant 2^{o(m)}$ and $d \leqslant (s')^{o(1)}$. Thus, to evaluate U on a particular input, it suffices to simulate U for at most s' steps, which is independent of what the actual polynomial time bound of U is. It follows that we can construct the circuit D (given U and y) in time $p(s')$, where p is a polynomial that does not depend on U. Also, to decide if h is an input gate to g, we use the polynomial-time TM M. We conclude that N is a PP machine which runs in some polynomial time (dependent on A). Since PERMANENT is PP-hard [26,31], we have a uniform reduction mapping z (an input to N) to an instance of PERMANENT of size $q(|z|)$, for some polynomial q (dependent on A).

By our assumption on the easiness of PERMANENT, we get that the language of N is decided by γ-weakly uniform circuits C_N of size at most $s'' = s(q(s'))$. If we plug in for U and y the actual TM description and the advice needed to decide the direct-connection language of B_i, we get from C_N the circuit B_{i+1}. Note that the direct-connection language of this circuit B_{i+1} is decided in polynomial time (using the algorithm for direct-connection language of C_N) given the advice needed for C_N plus the advice needed to describe U and y. The total advice size is at most $\gamma(c_0 \log s'') + \gamma(c_0 \log s'') + \gamma'(c_0 \log s') \leqslant 2(i + 1)\gamma(c_0 \log s'')$. □

Finally, we take the circuit B_d and use it to evaluate $A(x)$ by computing the value $B_d(x, g)$ where g is the output gate of C, which can be efficiently constructed (since this is just the initial configuration of our threshold TM for A on input x). By fixing g to be the output gate of C, we get the circuit for A which is $2d\gamma$-weakly uniform of size at most $(s \circ q)^{(d)}(r(n))$, where the polynomial r depends on the language A. Upper-bounding r by $(s \circ q)$ yields the result. □

4 Proofs of the Main Results

Here we use the technical tools from the previous section in order to prove our main results. Recall that L_0 is the P-complete language defined earlier.

4.1 Proof of Theorem 1

First, assuming L_0 is easy, we construct a hard language in CH.

Lemma 6. *Suppose L_0 is in* subexp*-weakly uniform TC^0 of depth d. Then, for a constant d' dependent on d, there is a language $L_{diag} \in \mathsf{CH}_{d'}$ which is not in* subexp*-weakly uniform* $\mathsf{SIZE}(\mathsf{poly})$.

Proof. Let $\alpha(m) \in 2^{o(m)}$ be such that L_0 is in α-weakly uniform TC^0 of depth d. Consider an arbitrary language L in β-weakly uniform $\mathsf{SIZE}(\mathsf{poly})$, for an arbitrary $\beta(m) \in 2^{o(m)}$. By Lemma 4, L has $\mu(n)$-weakly uniform threshold circuits of depth d and polynomial size, where $\mu(n) = \alpha(O(\log n)) + \beta(O(\log n)) \leqslant n^{o(1)}$. By Lemma 2, we have that L is decided by a threshold Turing machine with $d' = O(d)$ alternations, taking advice of length $\mu(n) \leqslant n^{o(1)} \leqslant n/\log^2 n$, and running in time $d' \cdot \mathsf{poly}(O(\log n) + n^{o(1)}) \leqslant n^{o(1)} \leqslant n/\log^2 n$. We conclude that *every* language in subexp-weakly uniform $\mathsf{SIZE}(\mathsf{poly})$ is also decided by some threshold TM in time $n/\log^2 n$, using d' alternations and advice of size $n/\log^2 n$.

Using Lemma 3, define L_{diag} to be the language in $\mathsf{Th}_{d'}\mathsf{TIME}(n)$ which is not decidable by any threshold Turing machine in time $n/\log^2 n$, using d' alternations and advice of size $n/\log^2 n$. It follows that L_{diag} is different from every language in subexp-weakly uniform $\mathsf{SIZE}(\mathsf{poly})$. □

Next, assuming PERMANENT is easy, we have that every language in CH is easy. The proof is immediate by Lemma 5.

Lemma 7. *If* PERMANENT *is in* subexp*-weakly uniform* $\mathsf{SIZE}(\mathsf{poly})$*, then every language in* CH *is in* subexp*-weakly uniform* $\mathsf{SIZE}(\mathsf{poly})$.

We now show that L_0 and PERMANENT cannot both be easy. The proof is immediate by Lemmas 6 and 7.

Theorem 4. *At least one of the following must be false:*

1. *L_0 is in* subexp*-weakly uniform TC^0;*
2. PERMANENT *is in* subexp*-weakly uniform* $\mathsf{SIZE}(\mathsf{poly})$.

To unify the two items in Theorem 4, we use the next lemma and its corollary.

Lemma 8 ([26,3]). *For every language $L \in \mathsf{P}$, there are uniform AC^0-computable function M (mapping a binary string to a poly-size Boolean matrix) and Boolean function f such that, for every x, we have $x \in L$ iff $f(\text{PERMANENT}(M(x)) = 1$.*

This lemma immediately yields the following.

Corollary 1. *If* PERMANENT *has α-weakly uniform $d(n)$-depth threshold circuits of size $s(n)$, then L_0 has α-weakly uniform $(d(n^{O(1)})+O(1))$-depth threshold circuits of size $s(n^{O(1)})$.*

Now we prove Theorem 1, which we re-state below.

Theorem 5. PERMANENT *is not in* subexp*-weakly uniform* TC^0.

Proof. Otherwise by Corollary 1, both claims in Theorem 4 would hold, which is impossible. □

4.2 Proofs of Theorem 2 and Theorem 3

The following two lemmas are similar to Lemma 6, and are used to prove Theorems 2 and 3; the proofs can be found in the full version [6].

Lemma 9. *Suppose L_0 is in* poly*-weakly uniform* $\mathsf{TC}^0(\mathsf{subsubexp})$ *of depth d. Then, for a constant $d' = O(d)$, there is a language $L_{diag} \in \mathsf{CH}_{d'}$ which is not in* poly*-weakly uniform* $\mathsf{SIZE}(\mathsf{subsubexp})$.

Lemma 10. *Suppose L_0 is computable by* poly*-weakly uniform polynomial-size threshold circuits of depth $o(\log\log n)$. Then there exists a language $L_{diag} \in \mathsf{Th}_{\log\log n}\mathsf{TIME}(n)$ which is not in* poly*-weakly uniform* $\mathsf{SIZE}(n^{\mathsf{poly}(\log n)})$.

5 Other Lower Bounds

Using similar indirect diagonalization, we are also able to show the following; the proofs are left to the full version [6] of this paper.

Theorem 6. PERMANENT *is not in* poly*-weakly uniform* $\mathsf{ACC}^0(2^{n^{o(1)}})$.

Theorem 7. EXP *is not in* poly*-weakly uniform* $\mathsf{SIZE}(2^{n^{o(1)}})$.

Theorem 8. PSPACE *is not computable by* poly*-weakly uniform Boolean formulas of size* $O(2^{n^{o(1)}})$.

6 Conclusion

We have shown how to use indirect diagonalization to prove lower bounds against weakly uniform circuit classes. In particular, we have proved that PERMANENT cannot be computed by polynomial-size TC^0 circuits that are only slightly uniform (whose direct-connection language can be efficiently computed using sublinear amount of advice). We have also extended to the weakly uniform setting other circuit lower bounds that were previously known for the uniform case.

One obvious open problem is to improve the TC^0 circuit lower bound for PERMANENT to be exponential, which is not known even for the uniform case. Another problem is to get super-polynomial uniform TC^0 lower bounds for a language from a complexity class below #P (e.g., PH). Strongly exponential lower bounds even against uniform AC^0 would be very interesting. One natural problem is to prove a better lower bound against *uniform* AC^0 (say for PERMANENT) than the known non-uniform AC^0 lower bound for PARITY.

References

1. Agrawal, M.: Proving Lower Bounds Via Pseudo-random Generators. In: Sarukkai, S., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 92–105. Springer, Heidelberg (2005)
2. Allender, E.: The permanent requires large uniform threshold circuits. Chicago Journal of Theoretical Computer Science (1999)

3. Allender, E., Gore, V.: A uniform circuit lower bound for the permanent. SIAM Journal on Computing 23(5), 1026–1049 (1994)
4. Arora, S., Barak, B.: Complexity theory: a modern approach. CUP, NY (2009)
5. Barrington, D.A.M., Immerman, N., Straubing, H.: On uniformity within NC^1. JCSS 41, 274–306 (1990)
6. Chen, R., Kabanets, V.: Lower bounds against weakly uniform circuits. In: ECCC, vol. 19, p. 7 (2012)
7. Chandra, A., Kozen, D., Stockmeyer, L.: Alternation. JACM 28(1), 114 (1981)
8. Furst, M., Saxe, J.B., Sipser, M.: Parity, circuits, and the polynomial-time hierarchy. Mathematical Systems Theory 17(1), 13–27 (1984)
9. Håstad, J.: Almost optimal lower bounds for small depth circuits. In: STOC 1986 (1986)
10. Heintz, J., Schnorr, C.-P.: Testing polynomials which are easy to compute. L'Enseignement Mathématique 30, 237–254 (1982)
11. Iwama, K., Morizumi, H.: An Explicit Lower Bound of $5n$-$o(n)$ for Boolean Circuits. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 353–364. Springer, Heidelberg (2002)
12. Jansen, M., Santhanam, R.: Permanent Does Not Have Succinct Polynomial Size Arithmetic Circuits of Constant Depth. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6755, pp. 724–735. Springer, Heidelberg (2011)
13. Kabanets, V., Impagliazzo, R.: Derandomizing polynomial identity tests means proving circuit lower bounds. Computational Complexity 13(1–2), 1–46 (2004)
14. Kannan, R.: Circuit-size lower bounds and non-reducibility to sparse sets. Information and Control 55, 40–56 (1982)
15. Karp, R.M., Lipton, R.J.: Turing machines that take advice. L'Enseignement Mathématique 28(3-4), 191–209 (1982)
16. Koiran, P., Perifel, S.: A superpolynomial lower bound on the size of uniform nonconstant-depth threshold circuits for the permanent. In: CCC (2009)
17. Lachish, O., Raz, R.: Explicit lower bound of $4.5n - o(n)$ for boolean circuits. In: Proc. of the Thirty-Third ACM Symp. on Theory of Computing, pp. 399–408 (2001)
18. Lupanov, O.B.: On the synthesis of switching circuits. Doklady Akademii Nauk SSSR 119(1), 23–26 (1958); English translation in Soviet Mathematics Doklady
19. Parberry, I., Schnitger, G.: Parallel computation with threshold functions. In: Proc. of the First IEEE Conf. on Structure in Complexity Theory, pp. 272–290 (1986)
20. Razborov, A.A.: Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. Mathematical Notes 41, 333–338 (1987)
21. Razborov, A.A., Rudich, S.: Natural proofs. JCSS 55, 24–35 (1997)
22. Ruzzo, W.L.: On uniform circuit complexity. JCSS 22(3), 365–383 (1981)
23. Shannon, C.E.: The synthesis of two-terminal switching circuits. Bell System Technical Journal 28(1), 59–98 (1949)
24. Smolensky, R.: Algebraic methods in the theory of lower bounds for boolean circuit complexity. In: Proc. of the Nineteenth ACM STOC, pp. 77–82 (1987)
25. Torán, J.: Complexity classes defined by counting quantifiers. JACM 38, 752 (1991)
26. Valiant, L.: The complexity of computing the permanent. TCS 8, 189–201 (1979)
27. Wagner, K.W.: The complexity of combinatorial problems with succinct input representation. Acta Informatica 23, 325–356 (1986)
28. Williams, R.: Non-uniform ACC circuit lower bounds. In: CCC (2011)
29. Yao, A.C.: Separating the polynomial-time hierarchy by oracles. In: FOCS (1985)
30. Zak, S.: A Turing machine hierarchy. TCS 26, 327–333 (1983)
31. Zanko, V.: #P-Completeness via Many-One Reductions. IJFCS 1, 77 (1991)

On TC^0 Lower Bounds for the Permanent

Jeff Kinne

Indiana State University
jkinne@cs.indstate.edu

Abstract. In this paper we consider the problem of proving lower bounds for the permanent. An ongoing line of research has shown super-polynomial lower bounds for slightly-non-uniform small-depth threshold and arithmetic circuits [1,2,3,4]. We prove a new parameterized lower bound that includes each of the previous results as sub-cases. Our main result implies that the permanent does not have Boolean threshold circuits of the following kinds.

1. Depth $O(1)$, poly-log(n) bits of non-uniformity, and size $s(n)$ such that for all constants c, $s^{(c)}(n) < 2^n$. The size s must satisfy another technical condition that is true of functions normally dealt with (such as compositions of polynomials, logarithms, and exponentials).
2. Depth $o(\log \log n)$, poly-log(n) bits of non-uniformity, and size $n^{O(1)}$.
3. Depth $O(1)$, $n^{o(1)}$ bits of non-uniformity, and size $n^{O(1)}$.

Our proof yields a new "either or" hardness result. One instantiation is that either NP does not have polynomial-size constant-depth threshold circuits that use $n^{o(1)}$ bits of non-uniformity, or the permanent does not have polynomial-size general circuits.

1 Introduction

The Search for Hard Problems Computational complexity aims to determine the computational costs of solving important problems, requiring both upper bounds and lower bounds. Though many types of lower bounds have been difficult to prove, conjectured lower bounds have become central across complexity theory. As an example, consider the area of derandomization – the task of converting randomized algorithms into deterministic algorithms with as little loss in efficiency as possible. Work initiated by Nisan and Wigderson [5] gives conditions for derandomizing BPP, the set of problems decided by bounded-error randomized polynomial-time algorithms. If E, deterministic linear-exponential time, contains a problem requiring super-polynomial circuits then BPP is contained in subexponential time (SUBEXP) [6]. If E contains a problem requiring circuits of size $2^{\epsilon n}$ for some positive constant ϵ then BPP = P [7]. These results are called hardness versus randomness tradeoffs because randomness can be more efficiently removed by using stronger hardness assumptions.

The lower bound results that have been proved so far have generally been for very high complexity classes (e.g., exponential-time Merlin-Arthur protocols

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 420–432, 2012.

require super-polynomial size circuits [8,9]) or for restricted models of computation (e.g., lower bounds for parity on constant-depth circuits, lower bounds on monotone circuits).

The long-term goal in proving lower bounds for restricted models is to prove lower bounds for increasingly more general models. The lower bounds for constant-depth circuits could be improved by allowing more powerful gates than the standard AND, OR, NOT. [10] showed that non-uniform ACC^0 circuits – constant-depth circuits that may include MOD_m gates for arbitrary modulus m – of polynomial size cannot compute languages that are complete for non-deterministic exponential time (NEXP). Because MOD_m gates can be viewed as specialized majority gates, the next step in this direction is to prove lower bounds for constant-depth circuits with majority gates (TC^0 circuits).

Uniform TC^0 *Lower Bounds.* The discussion above refers to proving lower bounds on *non-uniform* circuits – circuit families that consist of a different circuit for each input length n, with no requirement that the circuits are related to each other in any way. The task of proving super-polynomial lower bounds for non-uniform TC^0 circuits for languages in NEXP remains open. Progress has been made in proving lower bounds for *uniform* TC^0 circuits – circuit families where there exists a single Turing machine that can be used to reconstruct the circuit for any input length. The standard definition of uniform TC^0 requires a family of threshold circuits to be *Dlogtime uniform* – there exists a Turing machine that correctly answers queries about connections in the circuits in time that is logarithmic in the size of the circuit. We say a TC^0 circuit is uniform if it is Dlogtime uniform.

Polynomial-size uniform TC^0 circuits can be simulated in logarithmic space, so the space hierarchy theorem implies that PSPACE-complete languages cannot be computed by uniform TC^0 circuits of subexponential size. [1] showed that the hard language can be lowered from PSPACE to the first level of the counting hierarchy at the expense of a slight loss in the size: languages complete for PP, such as language versions of the permanent, cannot be computed by uniform TC^0 circuits of size $s(n)$ if s is time-constructible and $s^{(c)}(n) < 2^n$ for all constants c, where $s^{(c)}$ denotes s composed with itself c times. [2] show a lower bound for threshold circuits with super-constant depth: languages complete for PP cannot be computed by uniform threshold circuits of depth $o(\log \log n)$ and polynomial size.

Non-Uniform TC^0 *Lower Bounds.* While there has been progress on proving lower bounds for uniform TC^0 circuits, the ultimate question of proving super-polynomial lower bounds for non-uniform TC^0 circuits remains open. It has been observed that for every constant $k > 0$ the counting hierarchy contains languages that require general circuits of size n^k (see, e.g., [11,12,4]), and this applies to TC^0 circuits as well. Thus we have two types of lower bounds for TC^0 circuits in the counting hierarchy: fixed-polynomial size and fixed-polynomial non-uniformity, and large size but uniform. [3,4] show a lower bound that is intermediate between these two but for arithmetic circuits rather than Boolean

threshold circuits: constant-depth constant-free arithmetic circuits of polynomial size and $n^{o(1)}$-succinctness cannot compute the permanent. [3] introduces the notion of succinctness as a way to interpolate between fully uniform and fully non-uniform circuits. A circuit family is $a(n)$-*succinct* if questions about connections in the circuit can be answered by a non-uniform circuit of size $a(n)$. Note that Dlogtime uniform polynomial-size circuits are necessarily poly-log-succinct, and non-uniform polynomial-size circuits are poly-succinct.

Permanent and Polynomial Identity Testing The line of research on TC^0 circuit lower bounds [1,2,3,4] has focused on the permanent as the hard language. The primary property of the permanent that is used is the PP-completeness of the language version of the permanent. Beyond the fact that the proofs work for the permanent, proving lower bounds for the permanent is of interest because of connections to derandomization and the polynomial identity testing problem.

As mentioned already, strong enough lower bounds imply derandomization of BPP, but such lower bounds have been difficult to prove. To make progress, research has focused on derandomizing specific problems. In particular, much work has been done for polynomial identity testing (PIT) – for which there is a simple and efficient randomized algorithm while the best deterministic algorithms known require exponential time. Some results have given fast deterministic algorithms for restricted versions of PIT (for example, testing identities encoded by depth two or three arithmetic circuits), while other results have shown that lower bounds for restricted types of circuits yield deterministic identity testing procedures for restricted versions of PIT.

Of particular note to the current discussion is the result that exponential lower bounds for the permanent on constant-depth arithmetic circuits imply deterministic quasi-polynomial time algorithms for testing identities on constant-depth circuits that have degree poly-log(n) in each variable [13]. Thus strong enough TC^0 lower bounds for the permanent directly imply improved PIT algorithms. This provides motivation to continue working towards non-uniform TC^0 lower bounds for the permanent, beyond the fact that TC^0 lower bounds are a logical next step after the ACC^0 lower bounds of [10].

1.1 Our Results

Lower Bounds for Permanent Our main result is a new lower bound for slightly non-uniform small-depth threshold circuits computing the permanent, Thm. 1.1. Let L be any PP-hard language such that any language decidable in PPTIME$(t(n))$ can be reduced to an instance of L of size $t(n) \cdot \text{poly-log}(t(n))$ by a quasi-linear size uniform TC^0 reduction. By [14], a paddable version of the 0-1 permanent satisfies this property. Let $s'^{(d)}$ denote a function s' composed with itself d times.

Theorem 1.1. *Let $s(n)$, $a(n)$, and $d(n)$ be non-decreasing functions with $s(n)$ time-constructible and $d(n), a(n) \leq s(n)$ for all n. Let $s'(n) = s(n^{O(1)})$, and $m = s(2^{O(n)})$. If $s'^{(d(m))}(n + a(m) + \text{poly-log}(s(m))) < 2^n$ then L does not have depth $d(n) - O(1)$, $(a(n) - \text{poly-log}(s(n)))$-succinct threshold circuits of size $s(n)/\text{poly-log}(s(n))$.*

Each of the constants in the big-O and polylog terms in Thm. 1.1 are absolute constants independent of the functions s, a, and d.

We have the following corollary for the extreme cases of maximizing each of the three parameters – size, depth, and amount of non-uniformity – in Thm. 1.1.

Corollary 1.1. *The permanent does not have threshold circuits of the following kinds.*

1. *Depth* $O(1)$, poly-log$(s(n))$*-succinct, and size* $s(n)$ *such that* $s(n)$ *is non-decreasing, time-constructible and for all constants* c, $s^{(c)}(n) < 2^n$ *and* $\log(s(s(2^{c \cdot n}))) = s^{(O(1))}(n)$.
2. *Depth* $o(\log \log n)$, poly-log(n)*-succinct, and size* $n^{O(1)}$.
3. *Depth* $O(1)$, $n^{o(1)}$*-succinct, and size* $n^{O(1)}$.

Corollary 1.1 capture the main results from the previous works in this area as sub-cases. The main results of [1] and [2] for the permanent correspond to Items 1[1] and 2 of Cor. 1.1 but for uniform circuits. The main unconditional lower bounds for the permanent in [4] have the same parameters as Item 3 except they are for constant-depth constant-free arithmetic circuits rather than Boolean threshold circuits.

Lower Bounds for Permanent or Satisfiability

We prove that Thm. 1.1 can be strengthened to imply that either NP is hard for succinct small-depth threshold circuits, or the permanent is hard for general circuits. In Thm. 1.2, SAT is the NP-complete language Boolean formula satisfiability. Any of the standard NP-complete languages could be used instead.

Theorem 1.2. *Let* $s(n)$, $a(n)$, *and* $d(n)$ *be non-decreasing functions with* $s(n)$ *time-constructible and* $d(n), a(n) \leq s(n)$ *for all* n. *Let* $s'(n) = s(n^{O(1)})$, *and* $m = s(s(2^{O(n)}))$. *If* $s'^{(d(m))}(n + a(m) + \text{poly-log}(s(m))) < 2^n$ *then either*

- SAT *does not have depth* $d(n) - O(1)$, $(a(n) - \text{poly-log}(s(n)))$*-succinct threshold circuits of size* $s(n)/\text{poly-log}(s(n))$, *or*
- *The permanent does not have non-uniform general circuits of size* $s(n)/\text{poly-log}(s(n))$.

Each item of Cor. 1.1 can also be stated in a similar way.

More Direct Proof of Known Result. Our proof of Thm. 1.1 uses the fact that the exponential-time counting hierarchy contains a language that requires circuits of exponential size. The result has been stated for the setting of the standard counting hierarchy and circuits of polynomial size in a number of works – including [11], [12] and [4] – where it is derived by combining Toda's Theorem [15] and the fact that the polynomial hierarchy contains languages that require fixed-polynomial size circuits [16].

[1] The last condition on s in Item 1 holds for "normal" functions, those composed of logarithms, exponentials, and polynomials. The corresponding result of [1] does not require this condition.

We give a more direct proof of the result and state it for a wider range of parameters. The argument can be used to obtain a language with hardness up to the minimum of $2^n - 1$ and the maximal circuit complexity. For our definition of circuit size (string length of the circuit's description), the maximal circuit complexity is at least 2^n.

Theorem 1.3. *Let $h(n)$ be a time-constructible function such that for all n, $n \leq h(n) < 2^n$. There is a language L_{hard} in $\mathrm{DTIME}^{\mathrm{PP}}(\mathrm{poly}(h(n)))$ that does not have circuits of size $h(n)$.*

Since separations for high resources imply separations for low resources, it will be optimal to set $h(n)$ as large as possible. We use the following in the proof of Theorem 1.1.

Corollary 1.2. *There is a language L_{hard} in $\mathrm{DTIME}^{\mathrm{PP}}(2^{O(n)})$ that does not have circuits of size $2^n - 1$.*

We point out that our direct proof of Thm. 1.3 obviates the need to use Toda's theorem in a result of [12]. [12] gives an alternate proof of the result of [17] that if polynomial identity testing can be derandomized, then either Boolean circuit lower bounds for NEXP or arithmetic circuit lower bounds for the permanent must follow. A number of proofs are known for these types of results [18], and the proof of [12] gives the best-known tradeoff between the parameters. The proof of [12] is more direct than other proofs, and using our proof of Thm. 1.3 simplifies their proof further.

Theorem 1.3 can be used to simplify the proof of the result of [19] that for every constant $k > 0$ there is a language in PP requiring circuits of size n^k; for more details see the full version of this paper on the author's website.

1.2 Techniques

To see the structure of the proof of Thm. 1.1, we first give an outline for proving that constant-depth uniform threshold circuits of polynomial size cannot compute the permanent. We assume the permanent has uniform poly-size constant-depth threshold circuits and aim for a contradiction. We achieve the contradiction in two parts.

(i) We use the assumed easiness of the permanent to conclude that a non-uniformly hard language can be solved by large uniform small-depth threshold circuits. In particular, by the PP-completeness of the permanent and under the assumed easiness of the permanent, L_{hard} of Cor. 1.2, which is in E^{PP}, has uniform constant-depth threshold circuits of size $2^{O(n)}$.

(ii) Let C_{hard} be the circuit for L_{hard} at input length n from (i). By viewing the threshold gates within C_{hard} as questions about the permanent, we shrink the circuit as follows. The first level of threshold gates closest to the inputs in C_{hard} can be viewed as PP questions of size $\mathrm{poly}(n)$; using the assumed easiness of the permanent a circuit C_1 of size $\mathrm{poly}(n)$ can be used in place of the threshold gates on the first level. A similar argument shows that the second

level of threshold gates reduce to PP questions of size poly($|C_1|$), which can be replaced by a circuit of size poly(poly($|C_1|$)) using the assumed easiness of the permanent. This process is repeated for each level of threshold gates in C_{hard}. If C_{hard} has depth d, we obtain a circuit of size $p^{(d)}(n)$ for some polynomial p after iterating for each level of threshold gates in C_{hard}.

The conclusion of (ii) is a contradiction – we have constructed a circuit of size poly(n) for computing C_{hard} although it should require size 2^n.

Parameterized Proof Theorem 1.1 follows the same strategy but with the size, depth, and succinctness of the assumed circuits for the permanent parameterized as $s(n)$, $d(n)$, and $a(n)$ respectively. Then the circuit C_{hard} is of size, depth, and succinctness $s(m)$, $d(m)$, and $O(a(m) + \text{poly-log}(s(m)))$, for $m = s(2^{O(n)})$. The size of C_{hard} is $s(s(2^{O(n)}))$ rather than just $s(2^{O(n)})$ because the assumed easiness of the permanent is used twice to reduce E^{PP} to a threshold circuit. The term $a(m) + \text{poly-log}(s(m))$ appears in the inequality of Thm. 1.1 because the PP questions in (ii) for the threshold gates must refer to a particular gate in C_{hard} – which requires the $O(a(m)) + \text{poly-log}(s(m))$ bits of succinctness for constructing C_{hard}. The circuit size s is composed with itself $d(m)$ times in the inequality because the process is iterated for each level of threshold gates in C_{hard}.

Permanent or NP. For Thm. 1.2, we look more closely at how the easiness of the permanent is used. In (i) we use the fact that there is a hard language in E^{PP}. For Thm. 1.2 we instead use a hard language in E^{Σ_2}, meaning assuming NP is easy is enough to obtain the large threshold circuit C_{hard}. In (ii) we only use the assumption that the permanent has a small circuit – the depth and amount of succinctness do not matter. These two observations give Thm. 1.2.

Hardness of E^{PP}. We give an argument for Thm. 1.3 that is more direct than arguments that have previously been given, showing that there is a language in $\mathrm{DTIME}^{\mathrm{PP}}(\text{poly}(h(n)))$ that does not have circuits of size $h(n)$ for $n \leq h(n) < 2^n$. Consider input length n. The main idea is to pick an input, compute the output of all size $h(n)$ circuits on this input, and choose the output to differ from at least half; then repeat this on a new input, differing from at least half of the remaining size $h(n)$ circuits; continue for $h(n) + 1$ iterations to differ from all circuits of size $h(n)$. $h(n) + 1$ iterations are enough because for the definition of circuit size that we use (string length of the circuit's description) there are at most $2^{h(n)}$ circuits of size $h(n)$. The diagonalizing machine only needs to be able to determine the majority answer of $2^{h(n)}$ computations. In other words, the power of counting is needed, so that the appropriate output can be chosen using a PP oracle.

We point out that this diagonalization strategy has been used before, e.g., in [20] to show that for every constant $k > 0$, EXP contains languages that require more than 2^{n^k} time and n^k bits of non-uniform advice.

Comparison with Previous Work. Each of the previous works proving super-polynomial lower bounds for the permanent on small-depth threshold or arithmetic circuits [1,2,4] includes a component similar to step (ii) above – the

assumed easiness of the permanent is used to iteratively shrink a large threshold circuit. [1] and [4] phrase that portion of their argument as collapsing the counting hierarchy under the assumed easiness of the permanent. This is equivalent to collapsing a large threshold circuit due to the equivalence between exponential-size uniform constant-depth threshold circuits and the counting hierarchy.

[1] shows unconditionally that for s satisfying $s^{(O(1))} < 2^n$ the counting hierarchy contains a language that does not have uniform constant-depth threshold circuits of size $s(n)$. If the permanent has uniform constant-depth threshold circuits of size s, then the counting hierarchy collapses (in a way similar to our step (ii)) to size $s^{(O(1))}(n)$ uniform constant-depth threshold circuits – a contradiction if $s^{(O(1))}(n) < 2^n$.

[4] uses the collapse of the counting hierarchy under the assumed easiness of the permanent within a framework involving hitting sets for polynomials whose coefficients are computed by constant-depth arithmetic circuits. If the permanent is easy then there is a polynomial that avoids the hitting set and has coefficients computable in the counting hierarchy. The collapse of the counting hierarchy under the assumed easiness of the permanent then shows that the coefficients of the polynomial can be computed more efficiently than should be possible. The complete proof also uses machinery to translate between arithmetic and threshold circuits.

[2] uses an outline that is very similar to ours. If the permanent is easy then E has small-depth threshold circuits of size $2^{O(n)}$ and depth $o(\log n)$. These threshold circuits are then collapsed in a way that is similar to our step (ii) above, reaching a contradiction that E can be computed by subexponential size uniform threshold circuits (and thus in subexponential time).

Each of the earlier works uses a step similar to our step (ii) to contradict a known separation. Each work differs in the known separation that is contradicted, and the choice of separation to base the argument on effects some portions of the argument. The separations used by [1] and [2] are uniform separations, meaning care must be taken to keep track of the uniformity of the circuit that results from step (ii). By using a non-uniform separation, we do not need to keep track of the uniformity, resulting in a simpler argument. Using a non-uniform separation is also required for obtaining hardness against non-uniform circuits.

We have also stated our result as a tradeoff between the different parameters – size, depth, and non-uniformity – which previous works have not done.

1.3 Alternate Proof of Our Results

After completing our work, we learned that results equivalent to Cor. 1.1 were obtained independently by Chen and Kabanets [21]. An examination of the statement of results in [21] shows that our Thm. 1.1 implies Theorems 1.1, 1.2, and 1.3 of [21]. Not only are our results the same as [21], but the overall proof structure is similar. The main difference is that [21] uses L_{hard} resulting from the time hierarchy for threshold Turing machines with $o(n)$ bits of advice, whereas we use a non-uniformly hard language in the exponential time counting hierarchy.

Using the different hard languages results in some differences between the two proofs.

2 Preliminaries

We assume the reader is familiar with standard complexity classes and notions such as Turing machines, Boolean circuits, PP, PPTIME, TC^0, AC^0, P, EXP, and DTIME. We refer to standard complexity theory texts or the complexity zoo website for precise definitions and background.

In our results and proofs, all circuits are Boolean circuits. A *threshold circuit* may have, in addition to AND, OR, and NOT gates, majority gates of arbitrary fan-in. Majority gates can be used in place of AND and OR gates, and NOT gates can be pushed to the inputs. Thus without loss of generality a threshold circuit takes all input bits and negations of input bits as the inputs, and all remaining gates are majority gates.

We use the convention that the *size of a circuit* is the string length of its description. Thus the number of circuits of size n is at most 2^n. This makes the analysis cleaner than using the number of gates or wires and only effects results by polylogarithmic factors. Because there are exactly 2^{2^n} Boolean functions on n bits, it is immediate that there exists a language that requires circuits of size at least 2^n.

Succinct Circuits. Our results concern a notion of non-uniformity termed *succinctness* that was introduced in [3]. Succinctness is a natural notion of non-uniformity for circuit classes that are only slightly non-uniform. A circuit family $\{C_i\}_{i\in\mathbb{N}}$ is $a(n)$-*succinct* if for each n, there is a circuit Q_n that is of size $a(n)$ and correctly answers queries about the connections in C_n. The standard notion of uniform constant-depth circuits is Dlogtime uniformity; a circuit family is *Dlogtime uniform* if there is a Turing machine that correctly answers queries about the connections in C_n and runs in time linear in its input length (and thus logarithmic in the size of the circuit). Note that Dlogtime uniform circuits are necessarily poly-log-succinct.

Permanent and PP. The only property of the permanent needed for our results is PP-hardness. [14], building on [22], implies that any language in PPTIME(n) reduces to the 0-1 permanent with a quasi-linear size uniform AC^0 reduction, where quasi-linear means $n \cdot \text{poly-log}(n)$.

For the main part of the proof of Thm. 1.1, we use a different PP-complete language, $L_{\text{PP}} = \{x, M, 1^t |$ the probabilistic machine M runs in time at most t on input x and the majority of computation paths are accepting$\}$. The advantage of this language is that any PPTIME(n) language reduces to L_{PP} in linear time, so we can avoid polylog factors in the analysis by only reducing to the permanent at the very end of our proof. L_{PP} is contained in quasi-linear time PP, and by the result mentioned above reduces to instances of the permanent of quasi-linear size.

3 Lower Bounds for Permanent

In this section we prove our main result, Thm. 1.1. Further details on the proofs of Thms. 1.2 and 1.3 are contained in the full version of this paper on the author's website.

Proof of Thm. 1.1. To prove Thm. 1.1, we combine the hard language L_{hard} resulting from Cor. 1.2 with the following two claims. Let L_{PP} be the PP-complete language defined in Sect. 2.

Claim 1. *Let $s(n)$, $a(n)$, and $d(n)$ be non-decreasing functions with $s(n)$ time-constructible and $d(n), a(n) \leq s(n)$ for all n. If L_{PP} has $a(n)$-succinct threshold circuits of depth $d(n)$ and size $s(n)$ then L_{hard} has threshold circuits of size $s(m)$, depth $d(m)$, and is $O(a(m)) + \text{poly-log}(s(m)))$-succinct, for $m = s(2^{O(n)})$.*

Proving Clm. 1 amounts to plugging in the assumed circuit for L_{PP} into the E^{PP} computation of L_{hard}.

Claim 2. *Let $s(n)$, $a(n)$, and $d(n)$ be non-decreasing functions with $s(n)$ time-constructible and $d(n), a(n) \leq s(n)$ for all n. If L_{PP} has $a(n)$-succinct threshold circuits of depth $d(n)$ and size $s(n)$ then L_{hard} has size $s'^{(d(m))}(n + a(m) + \text{poly-log}(s(m)))$ circuits for $s'(n) = s(n^{O(1)})$ and $m = s(2^{O(n)})$.*

To prove Clm. 2, we use the threshold circuit from Clm. 1 and shrink it by using the easiness of L_{PP} to collapse the threshold gates iteratively. This is the step that is at the heart of all previous papers [1,2,3,4] proving lower bounds for the permanent on small-depth threshold or arithmetic circuits.

If the size of the circuit for L_{hard} in Clm. 2 is less than 2^n, we conclude that L_{PP} cannot have $a(n)$-succinct threshold circuits of depth $d(n)$ and size $s(n)$. The statement of Thm. 1.1 follows by the quasi-linear size uniform AC^0 reduction from L_{PP} to the permanent: if the permanent has depth $d(n)$ threshold circuits of size $s(n)$ and succinctness $a(n)$, then L_{PP} has depth $d(n) + O(1)$ threshold circuits of size $s(n)\,\text{poly-log}(s(n))$ and succinctness $a(n) + \text{poly-log}(s(n))$.

All that remains is to prove the claims.

Proof of Clm. 1. We take the E^{PP} computation of L_{hard} of Cor. 1.2. First consider the PP oracle from the definition of L_{hard}. The oracle O in the proof of Thm. 1.3 is computable in polynomial PPTIME, and the instances we need are of size $O(2^n)$. The oracle queries can thus be translated to queries to L_{PP} of size $N = 2^{O(n)}$ [2]. Given the assumed threshold circuits for L_{PP}, the oracle queries can be decided by a depth $d(N)$ threshold circuit C_{PP} of size $s(N)$ and succinctness $a(N) + \text{poly-log}(s(N))$.

Deciding membership in L_{hard} amounts to querying the oracle O on at most 2^n inputs. This gives an oracle circuit that makes exponentially many adaptive queries to O. In this circuit we replace each oracle gate with the circuit C_{PP},

[2] We can assume all queries are the same size because L_{PP} is paddable – queries of smaller length can be made longer to match the longest query.

obtaining a single circuit deciding L_{hard} that is of size $\text{poly}(2^n \cdot s(N))$ that requires $a(N) + \text{poly-log}(s(N))$ bits of succinctness. This circuit can be viewed as a circuit value problem of size $m' = \text{poly}(2^n \cdot s(N))$. Because P $\subseteq$ PP and L_{PP} is complete for PP, this circuit value problem reduces to an instance of L_{PP} of size $m = O(m')$. Using the assumed easiness of L_{PP}, such instances can be solved by threshold circuits of depth $d(m)$ and size $s(m)$. The amount of succinctness needed throughout the reductions is $a(N) + \text{poly-log}(s(N)) + a(m) + \text{poly-log}(s(m))$, which is $O(a(m) + \text{poly-log}(s(m)))$. The statement of Clm. 1 results from simplifying the expression for m to $s(2^{O(n)})$ using the fact that $s(n) \geq n$ and both s and a are non-decreasing.

Proof of Clm. 2. Main Idea. For each input length n, we aim to build a circuit for L_{hard} at input length n. With the assumed easiness of L_{PP} and using Clm. 1, we have a threshold circuit C_{hard} for L_{hard} with size $s(m)$, depth $d(m)$, and succinctness $O(a(m) + \text{poly-log}(s(m)))$, for $m = s(2^{O(n)})$. The plan is to shrink C_{hard} by viewing the threshold gates as small PP questions and using the assumed easiness of L_{PP} to collapse the gates. We do this iteratively level by level in the circuit. The proof consists mostly of keeping track of the size of the circuit produced as a result of this process.

Iterative Shrinking of C_{hard}. For each i, we define the following language: $L_i = \{(x, j)|$ on input x of length n, gate j in C_{hard} is at depth i and outputs $1\}$. The value j is padded to n bits to ensure that each input length of L_i regards a single value of n. We iteratively construct circuits for input length $2n = |(x, y)|$ for $L_1, L_2, ..., L_{d(m)}$. For each i, we use the circuit constructed for L_i to build the circuit for L_{i+1}. The final circuit for $L_{d(m)}$ at length $2n$ corresponds to the output gate of C_{hard} for inputs x of length n.
First level of Threshold Gates First consider L_1, corresponding to the first level of threshold gates in C_{hard}. Given input (x, j) with x of length n, a PP machine determines the output of gate j as follows.

1. Use $O(a(m) + \text{poly-log}(s(m)))$ bits of advice as the succinctness for C_{hard} to verify that j is a gate at depth 1, and if not split into a rejecting and an accepting state.
2. Nondeterministically guess an input label k and use the advice from 1. as the succinctness for C_{hard} to verify k is an input to gate j; if not split into a rejecting and an accepting state.
3. Accept iff the input bit labeled by k is 1.

This PP computation has a majority of accepting computation paths iff j is a gate at depth 1 and the majority of the inputs to gate j are 1. The amount of time for the computation is $\text{poly}(n + a(m) + \text{poly-log}(s(m)))$, and we have used $O(a(m) + \text{poly-log}(s(m)))$ bits of non-uniformity. This PP computation can be reduced to an instance of L_{PP} of size $\text{poly}(n + a(m) + \text{poly-log}(s(m)))$. By the assumed easiness of L_{PP}, and including the $O(a(m) + \text{poly-log}(s(m)))$ bits of non-uniformity, these instances are solved by a threshold circuit C_1 of size

$$S_1 = s(\text{poly}(n + a(m) + \text{poly-log}(s(m)))) + O(a(m) + \text{poly-log}(s(m))).$$

Because our ultimate goal is a non-uniform, arbitrary-depth circuit for L_{hard} we do not need to keep track of the depth and amount of succinctness in this circuit.

Level $i+1$ of Threshold Gates. Now consider L_{i+1} assuming we have a circuit C_i for L_i. Given an input (x, j), a PP machine can determine the correct output of gate j as follows.

1. Use $O(a(m) + \text{poly-log}(s(m)))$ bits of advice as the succinctness for C_{hard} to verify that j is a gate at depth $i+1$, and if not split into a rejecting and an accepting state.
2. Nondeterministically guess a gate label k and use the advice from 1. as the succinctness for C_{hard} to verify k is an input to gate j; if not split into a rejecting and an accepting state.
3. Accept iff C_i indicates that k outputs 1, namely if $C_i(x, k) = 1$.

This PP computation computes L_{i+1} just as in the case for L_1 above. The amount of time for the computation is $\text{poly}(n + a(m) + \text{poly-log}(s(m)) + |C_i|)$, and we have used $O(a(m) + \text{poly-log}(s(m))) + |C_i|$ bits of non-uniformity. This PP computation can be reduced to an instance of L_{PP} of size $\text{poly}(n + a(m) + \text{poly-log}(s(m)) + |C_i|)$. By the assumed easiness of L_{PP}, letting $S_i = |C_i|$, and including the $O(a(m) + \text{poly-log}(s(m))) + |C_i|$ bits of non-uniformity, we have a circuit C_{i+1} for L_{i+1} that is of size

$$S_{i+1} = s(\text{poly}(n + a(m) + \text{poly-log}(s(m)) + S_i))$$

$$+O(a(m) + \text{poly-log}(s(m))) + S_i.$$

Simplifying the Expression for the Circuit Size. Let us simplify the formula for S_i. First, S_1 can be simplified as $S_1 = s(\text{poly}(n + a(m) + \text{poly-log}(s(m))))$ using the fact that s and a are non-decreasing and $s(M) \geq a(M)$. For similar reasons, S_{i+1} can be written as $s(\text{poly}(n + a(m) + \text{poly-log}(s(m) + S_i)))$. Since s and a are non-decreasing, $S_1, S_2, ..., S_i$ is non-decreasing so that $n + a(m) + \text{poly-log}(s(m)) \leq S_i$ for each i. We can thus rewrite S_{i+1} as $s(\text{poly}(S_i))$. Letting $s'(M) = s(M^c)$ for large enough constant c, we have that $S_{i+1} = s'^{(i)}(n + a(m) + \text{poly-log}(s(m)))$. C_{hard} is computed at level $d(m)$, so by a circuit of size $s'^{(d(m))}(n + a(m) + \text{poly-log}(s(m)))$.

Acknowledgments. This research was partially supported by Indiana State University, University Research Council grants #11-07 and #12-18. We thank Matt Anderson, Dieter van Melkebeek, and Dalibor Zelený for discussions that began this project, continued discussions since, and comments on early drafts of this work. We thank Matt in particular for observations that refined the statement of Thm. 1.2. We also thank the reviewers for comments and suggestions that improved the exposition of the paper.

References

1. Allender, E.: The permanent requires large uniform threshold circuits. Chicago Journal of Theoretical Computer Science (1999)
2. Koiran, P., Perifel, S.: A superpolynomial lower bound on the size of uniform nonconstant-depth threshold circuits for the permanent. In: Proceedings of the IEEE Conference on Computational Complexity (CCC), pp. 35–40 (2009)
3. Jansen, M., Santhanam, R.: Permanent Does Not Have Succinct Polynomial Size Arithmetic Circuits of Constant Depth. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6755, pp. 724–735. Springer, Heidelberg (2011)
4. Jansen, M., Santhanam, R.: Marginal hitting sets imply super-polynomial lower bounds for permanent. In: Innovations in Theoretical Computer Science (2012)
5. Nisan, N., Wigderson, A.: Hardness vs. randomness. Journal of Computer and System Sciences 49(2), 149–167 (1994)
6. Babai, L., Fortnow, L., Nisan, N., Wigderson, A.: BPP has subexponential time simulations unless EXPTIME has publishable proofs. Computational Complexity 3, 307–318 (1993)
7. Impagliazzo, R., Wigderson, A.: P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In: Proceedings of the ACM Symposium on Theory of Computing (STOC), pp. 220–229 (1997)
8. Buhrman, H., Fortnow, L., Thierauf, T.: Nonrelativizing separations. In: Proceedings of the IEEE Conference on Computational Complexity (CCC), pp. 8–12 (1998)
9. Miltersen, P.B., Vinodchandran, N.V., Watanabe, O.: Super-Polynomial Versus Half-Exponential Circuit Size in the Exponential Hierarchy. In: Asano, T., Imai, H., Lee, D.T., Nakano, S.-i., Tokuyama, T. (eds.) COCOON 1999. LNCS, vol. 1627, pp. 210–220. Springer, Heidelberg (1999)
10. Williams, R.: Non-uniform ACC circuit lower bounds. In: Proceedings of the IEEE Conference on Computational Complexity (CCC), pp. 115–125 (2011)
11. Allender, E.: Circuit complexity before the dawn of the new millennium. In: Proceedings of the Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), pp. 1–18 (1996)
12. Kinne, J., van Melkebeek, D., Shaltiel, R.: Pseudorandom Generators and Typically-Correct Derandomization. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX and RANDOM 2009. LNCS, vol. 5687, pp. 574–587. Springer, Heidelberg (2009)
13. Dvir, Z., Shpilka, A., Yehudayoff, A.: Hardness-randomness tradeoffs for bounded depth arithmetic circuits. SIAM Journal on Computing 39(4), 1279–1293 (2009)
14. Zanko, V.: #P-completeness via many-one reductions. International Journal of Foundations of Computer Science 2, 77–82 (1991)
15. Toda, S.: PP is as hard as the polynomial-time hierarchy. SIAM Journal on Computing 20, 865–877 (1991)
16. Kannan, R.: Circuit-size lower bounds and nonreducibility to sparse sets. Information and Control 55, 40–56 (1982)
17. Kabanets, V., Impagliazzo, R.: Derandomizing polynomial identity tests means proving circuit lower bounds. Computational Complexity 13(1/2), 1–46 (2004)
18. Aaronson, S., van Melkebeek, D.: A note on circuit lower bounds from derandomization. Electronic Colloquium on Computational Complexity 17 (2010)
19. Vinodchandran, N.V.: A note on the circuit complexity of PP. Theoretical Computer Science 347(1-2), 415–418 (2005)

20. Impagliazzo, R., Kabanets, V., Wigderson, A.: In search of an easy witness: exponential time vs. probabilistic polynomial time. Journal of Computer and System Sciences 65(4), 672–694 (2002)
21. Chen, R., Kabanets, V.: Lower bounds against weakly uniform circuits. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 408–419. Springer, Heidelberg (2012)
22. Valiant, L.G.: The complexity of computing the permanent. Theoretical Computer Science 8, 189–201 (1979)

Formula Complexity of Ternary Majorities

Kenya Ueno

The Hakubi Center for Advanced Research
and Graduate School of Informatics,
Kyoto University
kenya@kuis.kyoto-u.ac.jp

Abstract. It is known that any self-dual Boolean function can be decomposed into compositions of 3-bit majority functions. In this paper, we define a notion of a ternary majority formula, which is a ternary tree composed of nodes labeled by 3-bit majority functions and leaves labeled by literals. We study their complexity in terms of formula size. In particular, we prove upper and lower bounds for ternary majority formula size of several Boolean functions. To devise a general method to prove the ternary majority formula size lower bounds, we give an upper bound for the largest separation between ternary majority formula size and DeMorgan formula size.

1 Introduction

The parity and majority functions are the most basic Boolean functions studied in the literature. When the number of input bits is odd, both of them are invariant under negations of all the input variables and the output (i.e., self-dual). The class of self-dual Boolean functions is closed under compositions. Therefore the recursive majority function defined by compositions of the 3-bit majority function is also self-dual.

A class of Boolean functions closed under compositions is called a Boolean clone. There are systematic studies on the relationship among Boolean clones known as Post's lattice [16]. (See also a survey [3] on Post's lattice with its applications.) According to the theory of Post's lattice, any monotone self-dual Boolean function can be decomposed into compositions of 3-bit majority functions. In other words, the 3-bit majority function is the universal gate for the class of monotone self-dual Boolean functions. On the other hand, the 3-bit Boolean function denoted by $(x_1 \wedge \neg x_2) \vee (\neg x_2 \wedge \neg x_3) \vee (\neg x_3 \wedge x_1)$ is the universal gate for the class of self-dual Boolean functions. It is also representable by the 3-bit majority function with negations. Therefore any self-dual Boolean function can be also decomposed into compositions of 3-bit majority functions with negations.

Ibaraki and Kameda [8] developed a decomposition theory of monotone self-dual Boolean functions for the data structure called coteries which realize mutual exclusions in distributed systems. The theory was further investigated for self-dual Boolean functions in general by Bioch and Ibaraki [2], who gave the

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 433–444, 2012.

decomposition scheme of the 3-bit parity function into compositions of 3-bit majority functions. We will fully utilize this decomposition scheme in our results.

There are two kinds of formula models whose nodes are labeled by 2-bit Boolean functions, known as $\mathbf{U}_2$-formula (DeMorgan formula) and its extension $\mathbf{B}_2$-formula (full binary basis formula). Studies on formula complexity of $\mathbf{U}_2$-formulas and $\mathbf{B}_2$-formulas have a long period of history. Most of lower bound methods for $\mathbf{U}_2$-formula size are regarded as extensions of Khrapchenko [10] proving the $\Theta(n^2)$ matching bound for the parity function. However, there are some hard limitation against $\mathbf{U}_2$-formula complexity around $\Omega(n^2)$ revealed in [7,9,11,12]. In the case of $\mathbf{B}_2$-formula, the lower bound technique introduced by Nechiporuk [13] $\Omega(n^2/\log n)$ is the most classical and still the strongest method.

Independently from any choice of formula models, proving formula size lower bounds is one of the most important problems in computational complexity theory as a weaker version of the circuit size lower bound problem and $\mathbf{P} \neq \mathbf{NP}$. A super-polynomial formula size lower bound for a function in some complexity class (e.g., $\mathbf{NP}$) including $\mathbf{NC}^1$ implies a separation between the two complexity classes (e.g., $\mathbf{NC}^1 \neq \mathbf{NP}$). The complexity class $\mathbf{NC}^1$ is defined in terms of logarithm circuit depth, which turns out to be equivalent to polynomial formula size. Therefore, the effect of the basis for formula complexity is also significant from the viewpoint of logical circuit design. With all the effort, it is extremely hard to give a slight improvement for the formula size problem even for a basic Boolean functions such as the majority function. Therefore there are fewer achievements in recent years concerned with formula complexity in spite of its importance.

In this paper, we consider a formula model $\mathbf{MAJ}_3$-formula (ternary majority formula) besides $\mathbf{U}_2$-formula and $\mathbf{B}_2$-formula. Every node of a $\mathbf{MAJ}_3$-formula is labeled by the 3-bit majority function while every node of a $\mathbf{U}_2$-formula and $\mathbf{B}_2$-formula is labeled by a 2-bit Boolean function. We will prove the $\mathbf{MAJ}_3$-formula size lower and upper bounds in Section 4 and 5, respectively. To prove the lower bounds, we will show that the largest separation between $\mathbf{MAJ}_3$-formula and $\mathbf{U}_2$-formula complexity is at most $O(n^{\log_2 3})$ in Section 3. It can be regarded as analogue of Pratt's result [17], which showed the largest separation between $\mathbf{B}_2$-formula complexity and $\mathbf{U}_2$-formula complexity is at most $O(n^{\log_3 10})$.

Our work is intended as a basis towards further studies on $\mathbf{MAJ}_3$-formula and any similar kinds of circuits and formula models. Since $\mathbf{MAJ}_3$-formula can be seen as the most simplified form of threshold circuits as well as neural networks, there are possibilities to utilize related techniques. We hope that developing a new stream of studies on $\mathbf{MAJ}_3$-formulas will contribute a new progress revealing the complexity of itself as well as other existing formula models.

2 Definitions

In this section, we summarize definitions concerned with Boolean functions and formula size. We assume that the readers are familiar with the basics of these concepts together with the notations of O, o, Ω, ω and Θ.

2.1 Boolean Functions

In this paper, we consider the following Boolean functions. Through the paper, n means the number of input bits.

Definition 1 (Boolean Functions). *The parity function* $\mathbf{PAR}_n : \{0,1\}^n \mapsto \{0,1\}$ *is defined by*

$$\mathbf{PAR}_n(x_1, \cdots, x_n) = \begin{cases} 1 & (\sum_{i=1}^{n} x_i \equiv 1 \mod 2), \\ 0 & (\sum_{i=1}^{n} x_i \equiv 0 \mod 2). \end{cases}$$

The majority function $\mathbf{MAJ}_{2l+1} : \{0,1\}^{2l+1} \mapsto \{0,1\}$ *on odd number of input bits is defined by*

$$\mathbf{MAJ}_{2l+1}(x_1, \cdots, x_n) = \begin{cases} 1 & (\sum_{i=1}^{n} x_i \geq l+1), \\ 0 & (\sum_{i=1}^{n} x_i \leq l). \end{cases}$$

The recursive majority function $\mathbf{RecMAJ}_3^h : \{0,1\}^{3^h} \mapsto \{0,1\}$ *is defined by*

$$\begin{aligned} \mathbf{RecMAJ}_3^h(x_1, \cdots, x_{3^h}) = \mathbf{MAJ}_3(&\mathbf{RecMAJ}_3^{h-1}(x_1, \cdots, x_{3^{h-1}}), \\ &\mathbf{RecMAJ}_3^{h-1}(x_{3^{h-1}+1}, \cdots, x_{2\cdot 3^{h-1}}), \\ &\mathbf{RecMAJ}_3^{h-1}(x_{2\cdot 3^{h-1}+1}, \cdots, x_{3^h})) \end{aligned}$$

with $\mathbf{RecMAJ}_3^1 = \mathbf{MAJ}_3$.

We will define another Boolean function right before it will appear. The notions of monotone and self-dual for Boolean function are defined as follows.

Definition 2 (Monotone and Self-Dual). *For Boolean vectors* $\boldsymbol{x} = (x_1, \cdots, x_n)$ *and* $\boldsymbol{y} = (y_1, \cdots, y_n)$*, we define* $\boldsymbol{x} \leq \boldsymbol{y}$ *if* $x_i \leq y_i$ *for all* $i \in \{1, \cdots n\}$*. A Boolean function* f *is called monotone if* $\boldsymbol{x} \leq \boldsymbol{y}$ *implies* $f(\boldsymbol{x}) \leq f(\boldsymbol{y})$ *for any* $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^n$*. A Boolean function* f *is called self-dual if* $f(x_1, \cdots, x_n) = \neg f(\neg x_1, \cdots, \neg x_n)$ *where* $\neg$ *denotes the negation, which flips* 1 *to* 0*, and* 0 *to* 1*.*

2.2 Formula Size

In this paper, we consider the following three formula models. For each model, a literal means either a variable x_i or the a negated variable $\neg x_i$ for some index i. Each formula is called monotone if it does not have negated variables. In the definition, the nodes $\wedge$ and $\vee$ mean the logical conjunction and disjunction, respectively.

Definition 3 (Formula Models). *A* $\mathbf{U}_2$*-formula is a binary tree with leaves labeled by literals and internal nodes labeled by* $\wedge$ *and* $\vee$*. A* $\mathbf{B}_2$ *-formula is a binary tree with leaves labeled by literals and internal nodes labeled by any of 2-bit Boolean functions such as* $\wedge$*,* $\vee$ *and* $\mathbf{PAR}_2$*. A* $\mathbf{MAJ}_3$*-formula is a ternary tree with leaves labeled by literals and internal nodes labeled by* $\mathbf{MAJ}_3$*.*

If we allow 0 and 1 in leaves along with literals, $\mathbf{MAJ}_3$-formulas can compute all the Boolean functions because $\mathbf{MAJ}_3(x_1, x_2, 0) = x_1 \wedge x_2$ and $\mathbf{MAJ}_3(x_1, x_2, 1) = x_1 \vee x_2$. So the 3-bit majority function with 0 and 1 can be regarded as a kind of the universal gate for all the Boolean functions. In this sense, $\mathbf{MAJ}_3$-formula is yet another natural extension of $\mathbf{U}_2$-formula like $\mathbf{B}_2$-formula. Even if we do not allow 0 and 1 in leaves $\mathbf{MAJ}_3$-formulas can compute all the self-dual Boolean functions. Furthermore, even if we allow only variables without negations, they can compute all the monotone self-dual Boolean functions.

The formula size for each formula model is defined as follows. For the convenience, we will not distinguish a Boolean function f and a formula computing f. Note that $L_{\mathbf{MAJ}_3}(f)$ is defined only for self-dual Boolean functions while $L_{\mathbf{B}_2}(f)$ and $L_{\mathbf{U}_2}(f)$ are defined for all Boolean functions.

Definition 4 (Formula Size). *The size of a formula is its number of leaves for any formula model. We define the formula size of a Boolean function* f *as the size of the smallest formula computing* f*. We denote the size of* $\mathbf{U}_2$*-formula,* $\mathbf{B}_2$*-formula and* $\mathbf{MAJ}_3$*-formula of a Boolean function* f *by* $L_{\mathbf{B}_2}(f)$*,* $L_{\mathbf{U}_2}(f)$ *and* $L_{\mathbf{MAJ}_3}(f)$*, respectively. We will sometimes abbreviate* $L_{\mathbf{U}_2}(f)$ *to* $L(f)$ *for simplicity.*

3 Translation from Ternary Majority Formulas to DeMorgan Formulas

In this section, we analyze the relation between $\mathbf{MAJ}_3$-formula complexity and $\mathbf{U}_2$-formula complexity. The results in this section will be useful to derive a $\mathbf{MAJ}_3$-formula size lower bound from a $\mathbf{U}_2$-formula size lower bound for the same function as shown in Section 4. We begin with the following simple proposition.

Proposition 1. $L_{\mathbf{MAJ}_3}(\mathbf{RecMAJ}_3^h) = 3^h$.

Proof. The upper bound $L_{\mathbf{MAJ}_3}(\mathbf{RecMAJ}_3^h) \leq 3^h$ follows from the same construction as the definition. The lower bound $L_{\mathbf{MAJ}_3}(\mathbf{RecMAJ}_3^h) \geq 3^h$ is also immediate because it depends on all the variables. □

From a majority formula $L(\mathbf{MAJ}_3) \leq 5$, we can recursively construct a formula for the recursive majority function whose size is 5^h. Therefore we have an upper bound $L(\mathbf{RecMAJ}_3^h) \leq 5^h$, i.e., $L(\mathbf{RecMAJ}_3^h) \in O(L_{\mathbf{MAJ}_3}(f)^{1.4650})$. Similarly, the best upper bound we know for $\mathbf{B}_2$-formula is also $L_{\mathbf{B}_2}(\mathbf{RecMAJ}_3^h) \leq 5^h$. The quantum adversary bound [11], which is useful to prove $\mathbf{U}_2$-formula size lower bounds, has a nice composition property written as $\mathbf{ADV}(f \cdot g) \geq$

$\mathbf{ADV}(f) \cdot \mathbf{ADV}(g)$. It implies a formula size lower bound $4^h \leq L(\mathbf{RecMAJ}_3^h)$, i.e. $L(\mathbf{RecMAJ}_3^h) \in \Omega(L_{\mathbf{MAJ}_3}(f)^{1.2618})$.

We call the value γ an expansion factor from a $\mathbf{MAJ}_3$-formula into $\mathbf{U}_2$-formula for an arbitrary self-dual Boolean function f if $L(f) \in O((L_{\mathbf{MAJ}_3}(f))^\gamma)$. In the case of the recursive majority function, we can prove $\gamma \geq \log_3 5$ by solving $5 \cdot a^\gamma \leq (3a)^\gamma$ where $L_{\mathbf{MAJ}_3}(f_1) = L_{\mathbf{MAJ}_3}(f_2) = L_{\mathbf{MAJ}_3}(f_3) = a$. At first glance, the recursive majority function seems to have the largest expansion factor $\log_3 5$ from a $\mathbf{MAJ}_3$-formula into a $\mathbf{U}_2$-formula among all the $\mathbf{MAJ}_3$-formulas. Surprisingly, this is not true as we prove in the next lemma.

Lemma 1. *For any self-dual Boolean function f,*

$$L(f) \in O(L_{\mathbf{MAJ}_3}(f)^{\log_2 3}) \subseteq O(L_{\mathbf{MAJ}_3}(f)^{1.5850}).$$

Proof. We are looking for the largest formula expansion from a $\mathbf{MAJ}_3$-formula into a $\mathbf{U}_2$-formula. Differently from the recursive majority function, the same variable might appear more than once in a ternary majority formula for an arbitrary Boolean function f. In this case, the expanded $\mathbf{U}_2$-formula can shrink more. So we can concentrate on the case in which all the variable appear exactly once. That is, $L_{\mathbf{MAJ}_3}(f) = n$.

We assume that $L(f) \leq \beta \cdot L_{\mathbf{MAJ}_3}(f)^\gamma$ for any self-dual Boolean function and consider an inductive argument. The expansion factor γ must satisfy an inequality

$$\begin{aligned} L(f) &\leq 2 \cdot \beta \cdot (L_{\mathbf{MAJ}_3}(f_1))^\gamma + 2 \cdot \beta \cdot (L_{\mathbf{MAJ}_3}(f_2))^\gamma + \beta \cdot (L_{\mathbf{MAJ}_3}(f_3))^\gamma \\ &\leq \beta \cdot (L_{\mathbf{MAJ}_3}(f))^\gamma \end{aligned}$$

by looking at a formula expansion from a $\mathbf{MAJ}_3$-formula $f = \mathbf{MAJ}_3(f_1, f_2, f_3)$ into a $\mathbf{U}_2$-formula $f = (f_1 \wedge f_2) \vee ((f_1 \vee f_2) \wedge f_3)$. This expansion is processed from leaves to the root in a recursive way.

We can assume that $L_{\mathbf{MAJ}_3}(f_1) \leq L_{\mathbf{MAJ}_3}(f_2) \leq L_{\mathbf{MAJ}_3}(f_3)$ without loss of generality. We set $L_{\mathbf{MAJ}_3}(f_1) = a-b$, $L_{\mathbf{MAJ}_3}(f_2) = a+b$ and $L_{\mathbf{MAJ}_3}(f_3) = a+c$ where $a > b \geq 0$ and $c \geq b \geq 0$. In this case, we need to find the minimum value of γ which always satisfies

$$2 \cdot (a-b)^\gamma + 2 \cdot (a+b)^\gamma + (a+c)^\gamma \leq (3a+c)^\gamma.$$

So we set

$$p(a, b, c, \gamma) = (3a+c)^\gamma - (a+c)^\gamma - 2 \cdot (a+b)^\gamma - 2 \cdot (a-b)^\gamma$$

and seek the minimum value of γ such that $p(a, b, c, \gamma) \geq 0$ for any $a > b \geq 0$ and $c \geq b > 0$.

First we fix a, b and γ and consider

$$q(\alpha) = (3+\alpha)^\gamma - (1+\alpha)^\gamma$$

where $\alpha = \frac{c}{a}$ $(0 < \alpha)$. By the derivative $y' = \gamma \cdot x^{\gamma-1}$ of $y = x^\gamma$, $(3+\alpha)^\gamma$ increases more than $(1+\alpha)^\gamma$ whenever α slightly increases. So $q(\alpha)$ monotonically increases

as α increases. To minimize $p(a, b, c, \gamma)$ for fixed γ, we would like to minimize $q(\alpha)$ and had better α be as small as possible. Hence we set $c = b$ because $c \geq b$.

Next we consider

$$r(\alpha, \gamma) = \frac{p(a, b, b, \gamma)}{a^\gamma} = (3 + \alpha)^\gamma - 3 \cdot (1 + \alpha)^\gamma - 2 \cdot (1 - \alpha)^\gamma$$

where $\alpha = \frac{b}{a}$ $(0 \leq \alpha < 1)$. Since $r(\alpha, \gamma) \geq 0$ for any α $(0 \leq \alpha < 1)$ implies $p(a, b, c, \gamma) \geq 0$ for any a, b and c, it suffices to seek the minimum γ which satisfies this condition.

It is easy to see that $\log_3 5$ is not the largest expansion factor because

$$r(1, \log_3 5) \approx -0.660928 < 0$$

while

$$r(1, \log_3 5) = 0.$$

On the other hand, $\log_2 3$ seems to be a good candidate which is very near to the largest expansion factor because

$$r(0, \log_2 3) \approx 0.704522 > 0$$

and

$$r(1, \log_2 3) \approx 1.77636 \times 10^{-15} > 0.$$

To confirm $r(0, \log_2 3) \geq 0$ for $0 \leq \alpha < 1$, it is sufficient to draw the graph of $r(\alpha, \log_2 3)$ $(0 \leq \alpha < 1)$ as shown in Figure 1. (Strictly speaking, it requires a rigorous analysis on $r(\alpha, \log_2 3)$, but we omit it in this paper.)

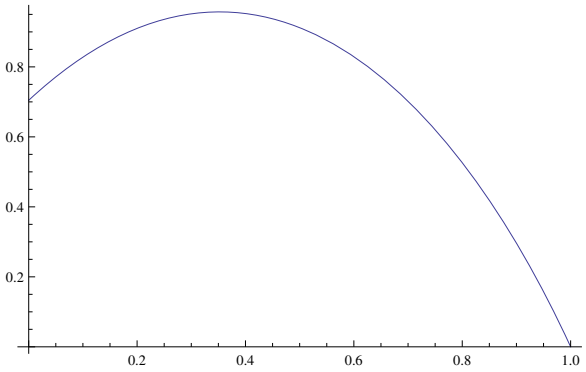


Fig. 1. $r(\alpha, \log_2 3) = (3 + \alpha)^{\log_2 3} - 3 \cdot (1 + \alpha)^{\log_2 3} - 2 \cdot (1 - \alpha)^{\log_2 3}$ $(0 \leq \alpha < 1)$

Therefore the largest expansion factor is at most $\log_2 3$, which is given for a $\mathbf{MAJ}_3$-formula with $a - 1 = b = c$, i.e., $L_{\mathbf{MAJ}_3}(f_1) = 1$ and $L_{\mathbf{MAJ}_3}(f_2) = L_{\mathbf{MAJ}_3}(f_3)$ for each subtree. □

Pratt [17] has proved

$$L_{\mathbf{U}_2}(f) \in O((L_{\mathbf{B}_2}(f))^{\log_3 10}) \subseteq O((L_{\mathbf{B}_2}(f))^{2.096}).$$

The exponent $\log_3 10$ is derived from the $\mathbf{U}_2$-formula size of 10 for the 3-bit parity function. The above lemma can be seen as an analogue of Pratt's bound [17] for the relation between $\mathbf{MAJ}_3$-formulas and $\mathbf{U}_2$-formulas.

4 Ternary Majority Formula Size Lower Bounds

In this section, we devise a general method to prove ternary majority formula size lower bounds. In general, we can derive a $\mathbf{MAJ}_3$-formula size lower bound for an arbitrary Boolean function from a $\mathbf{U}_2$-formula size lower bound of the same function using Lemma 1 as follows.

Theorem 1. *For any self-dual Boolean function f such that $L(f) \in \Omega(n^c)$, $L_{\mathbf{MAJ}_3}(f) \in \Omega(n^{c/\log_2 3})$.*

Proof. By Lemma 1, an upper bound for $\mathbf{U}_2$-formula size expanded from a $\mathbf{MAJ}_3$-formula of size N is at most $O(N^{\log_2 3})$. This size must be not smaller than the formula size lower bound $L(f) \in \Omega(n^c)$. Therefore we have obtained the theorem. □

From $\mathbf{U}_2$-formula size lower bounds of $L(\mathbf{PAR}_n) \in \Omega(n^2)$ and $L(\mathbf{MAJ}_n) \in \Omega(n^2)$ by Khrapchenko [10], we have the following corollaries. After completion of our work, we have noticed that the lower bound for the parity function is weaker than 1.33 of Chokler and Zwick [4] using the random restriction technique. Still, our lower bound method has merit in the sense that it can be applied for any Boolean function.

Corollary 1. *For any $n = 2l + 1$, we have $L_{\mathbf{MAJ}_3}(\mathbf{PAR}_{2l+1}) \in \Omega(n^{1.2618})$ and $L_{\mathbf{MAJ}_3}(\mathbf{MAJ}_{2l+1}) \in \Omega(n^{1.2618})$.*

Since $2/\log_2 3 = \log_3 4$, these lower bounds are equal to the $\mathbf{U}_2$-formula size lower bound for the recursive majority function accidentally. It seems to be difficult to give a matching $\mathbf{MAJ}_3$-formula size upper and lower bounds even for the parity function while we can obtain those for $\mathbf{U}_2$-formula and $\mathbf{B}_2$-formula. Both of them seem to be not tight and have room for further improvements.

The current best $\mathbf{B}_2$-formula size lower bound is $\Omega(n^2/\log n)$ shown by Nechiporuk [13] for the element distinctness function. We should note that Pratt's bound [17] $L_{\mathbf{U}_2}(f) \in O((L_{\mathbf{B}_2}(f))^{2.096})$ is not sufficient to give a substantial lower bound larger than n differently from the case of $\mathbf{MAJ}_3$-formula. This is because $\mathbf{U}_2$-formula size lower bounds have got stuck the barrier around $\Omega(n^2)$ [7,9,11,12] for almost all explicitly defined Boolean functions except the Andreev function discussed from now on.

The current best $\mathbf{U}_2$-formula size lower bound is $\Omega(n^{3-o(1)})$ by Håstad [6] for the Andreev function [1]. We can define the Andreev function so that it is self-dual.

Definition 5. *The Andreev function* $\mathbf{A}_{2n}$ *is composed of* $2n$ *input bits which are divided into* 2 *parts. The first part consists of* n *bits represent the truth table of a Boolean function* f *on* $\log n$ *bits. The second part consists of* n *bits which are also divided into* $\log n$ *blocks of* $n/\log n$ *bits. First, the function computes* $\log n$ *parity function on* $n/\log n$ *input bits from the second part of the input bits. and obtain* $\log n$ *output bits. Here we assume that the number of input bits for all the parity function is even. That is,* $n/\log n$ *is even. Then it computes the Boolean function* f *represented by the first part with the obtained* $\log n$ *output bits as input bits.*

With the slightly modification of the Andreev function from its original definition version, we can confirm that it is self-dual as follows.

Lemma 2. *The Andreev function* $\mathbf{A}_{2n}$ *defined as above is self-dual.*

Proof. We consider the situation in which we flip all the input bits for the Andreev function. Because we defined the Andreev function so that the number of input bits for all the parity functions inside the Andreev function is even, the output bits are invariant when we flip all the input bits. In other words, the parity function with even number of input bits is anti-dual, i.e.,

$$\mathbf{PAR}_{n/\log n}(x_1, \cdots, x_{n/\log n}) = \mathbf{PAR}_{n/\log n}(\neg x_1, \cdots, \neg x_{n/\log n}).$$

Therefore the Andreev function outputs a bit in the same position in the first part of the input bits after flipping all the input bits. On the other hand, this output bit has been also flipped. Hence the output bit of the Andreev function is also flipped after we flip all the input bits. □

So our $\mathbf{MAJ}_3$-formula size lower bound for the Andreev function is given as follows.

Theorem 2. $L_{\mathbf{MAJ}_3}(\mathbf{A}_{2n}) \in \Omega(n^{1.8927})$

Proof. Since we have defined the Andreev function so that it is self-dual, it can be represented by a $\mathbf{MAJ}_3$-formula from the theory of Post [16]. Moreover, the modification of the Andreev function does not affect the $\mathbf{U}_2$-formula size lower bound of $\Omega(n^{3-o(1)})$ by Håstad [6]. Thus we can apply Theorem 1 and obtain the lower bound by $3/\log_2 3 \approx 1.89279$. □

5 Ternary Majority Formula Size Upper Bounds

In this section, we prove $\mathbf{MAJ}_3$-formula size upper bounds of the parity and majority function. In both cases, the upper bounds are shown by utilizing the decomposition scheme of Bioch and Ibaraki [2] for the 3-bit parity function as

$$\mathbf{PAR}_3(x_1, x_2, x_3) = [1, [\bar{1}, \bar{2}, \bar{3}], [\bar{1}, 2, 3]]$$

where we use notations $[i, j, k] = \mathbf{MAJ}_3(x_i, x_j, x_k)$, $i = x_i$ and $\bar{i} = \neg x_i$. From the decomposition scheme, we obtain $L_{\mathbf{MAJ}_3}(\mathbf{PAR}_3) \leq 7$. We show that $\mathbf{MAJ}_3$-formula complexity is intermediate between $\mathbf{B}_2$-formula complexity and $\mathbf{U}_2$-complexity for both functions.

5.1 The Parity Function

In the case of $\mathbf{U}_2$-formula, we can construct a 2-bit parity formula $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$. By a recursive construction, we can prove an upper bound $L(\mathbf{PAR}_n) \leq n^2$ where $n = 2^h$ from a recursive inequality $L(\mathbf{PAR}_{2n}) \leq 4 \cdot L(\mathbf{PAR}_n)$.

In the case of $\mathbf{MAJ}_3$-formula, we can decompose the 3^h-bits parity function into a composition of a 3-bit parity function and three 3^{h-1}-bits parity functions. Thus we have a recursive inequality $L_{\mathbf{MAJ}_3}(\mathbf{PAR}_{3^h}) \leq 7 \cdot L_{\mathbf{MAJ}_3}(\mathbf{PAR}_{3^{h-1}})$ from the decomposition scheme of the 3-bit parity function. Solving this inequality straightforwardly, we can show an upper bound $L_{\mathbf{MAJ}_3}(\mathbf{PAR}_{3^h}) \in O(n^{\log_3 7}) \subseteq O(n^{1.7712})$. Actually we can give a better upper bound as follows.

Theorem 3 (See also [4]). *$L_{\mathbf{MAJ}_3}(\mathbf{PAR}_{2l+1}) \in O(n^{1.7329})$ where $n = 2l + 1$.*

Proof. For some constant α, we consider decomposition of the $(2\alpha + 1) \cdot m$-bit parity function into a composition of a 3-bit parity function with a m-bit parity function and two $\alpha \cdot m$-bit parity functions as follows.

$$\begin{aligned}\mathbf{PAR}_{(2\alpha+1)\cdot m}(x_1, \cdots, x_{(2\alpha+1)\cdot m}) = \mathbf{PAR}_3(&\mathbf{PAR}_m(x_1, \cdots, x_m),\\ &\mathbf{PAR}_{\alpha\cdot m}(x_{m+1}, \cdots, x_{(\alpha+1)\cdot m}),\\ &\mathbf{PAR}_{\alpha\cdot m}(x_{(\alpha+1)\cdot m+1}, \cdots, x_{(2\alpha+1)\cdot m})).\end{aligned}$$

Here we can assume that $\alpha \cdot m$ is an odd integer by increasing or decreasing it at most 1. In this case, $(2\alpha + 1) \cdot m$ becomes also an odd integer if m is odd.

Let $S(n) = L_{\mathbf{MAJ}_3}(\mathbf{PAR}_n)$ and assume $S(n) \leq \beta \cdot n^\gamma$ for some constants $\beta, \gamma > 0$ for any odd number n. By increasing the value of β, the slight modification which makes $\alpha \cdot m$ be an odd integer can be ignored for the following estimation of γ. By using decomposition scheme of the 3-bit parity function,

$$\begin{aligned}S((1 + 2\alpha) \cdot m) &\leq 3 \cdot S(m) + 2 \cdot S(\alpha \cdot m) + 2 \cdot S(\alpha \cdot m)\\ &\leq (3 + 4 \cdot \alpha^\gamma) \cdot \beta \cdot n^\gamma.\end{aligned}$$

It suffices to show that the last expression is bounded by $(1 + 2\alpha)^\gamma \cdot \beta \cdot n^\gamma$. Therefore we consider the minimum value of γ which satisfies

$$3 + 4 \cdot \alpha^\gamma \leq (1 + 2\alpha)^\gamma$$

by eliminating $\beta \cdot m^\gamma$ from both sides. We can verify that this inequality is satisfied when $\alpha = 1.73896$ and $\gamma = 1.73282$. □

We have possibilities to improve the upper bound by analysis of the parity function with larger number of input bits. In this case, the proof will become much more complicated as the number of input bits increases. For example, we can construct $\mathbf{MAJ}_3$-formula of size 21 for the 5-bit parity function as

$$\begin{aligned}&\mathbf{PAR}_5(x_1, x_2, x_3, x_4, x_5) =\\ &\quad [1, [\bar{1}, \bar{2}, [\bar{3}, [3, 4, 5], [3, \bar{4}, \bar{5}]]], [\bar{1}, 2, [\bar{1}, 2, [3, [\bar{3}, \bar{4}, \bar{5}], [\bar{3}, 4, 5]]]]].\end{aligned}$$

From the construction, we can obtain the upper bound of $O(n^{\log_5 21}) = O(n^{1.8917})$. This is far from the above upper bound. To obtain a better bound, we need much more succinct construction of a $\mathbf{MAJ}_3$-formula for the 5-bit parity function of size as close to $5^{1.7328} = 16.262...$ as possible.

5.2 The Majority Function

Our $\mathbf{MAJ}_3$-formula size upper bound for the majority function essentially relies on the general theory established by Paterson, Pippenger and Zwick [14]. Their idea is based on construction of a carry save adder from a full adder of fixed size as building blocks. Here we consider a full adder $\mathbf{FA}_3$ from 3 bits to 2 bits. The first and second output bits y_1, y_2 of $\mathbf{FA}_3$ are the 3-bit parity and majority function, respectively.

In the case of $\mathbf{U}_2$-formula, the full adder $\mathbf{FA}_3$ can be constructed by

$$y_1 = (x_1 \wedge ((\neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3))) \vee (\neg x_1 \wedge ((x_2 \wedge \neg x_3) \vee (\neg x_2 \wedge x_3))),$$

and

$$y_2 = (x_1 \wedge x_2) \vee ((x_1 \vee x_2) \wedge x_3).$$

They defined the notion of the occurrence matrix. It summarizes the information of the number of occurrence in the formula. For example, the occurrence matrix of the above case is $M = \begin{pmatrix} 2\,4\,4 \\ 1\,2\,2 \end{pmatrix}$. In the first and second row of the matrix, each entry counts the number of occurrence of each variable in the first and second formula, respectively.

From the construction of an arbitrary fixed size full adder and its corresponding occurrence matrix, Paterson, Pippenger and Zwick [14] gave the following general upper bound method.

Theorem 4 ([14]). *Let M be an occurrence matrix of some full adder for some fixed basis and some Boolean function f. Let $\epsilon(M)$ be the maximum value of $\frac{1}{\gamma}$ such that $\|\boldsymbol{x}\|_\gamma \leq \|M \cdot \boldsymbol{x}\|_\gamma$ for any vector $\boldsymbol{x} \geq \mathbf{0}$ where $\|x\|_\gamma = (\sum_i |x_i|^\gamma)^{1/\gamma}$. Then $O(n^{\epsilon(M)+o(1)})$ gives a formula size upper bound for f on the fixed basis.*

By the theorem, we can derive a $\mathbf{U}_2$-formula size upper bound of $O(n^{4.70})$. Paterson and Zwick [15] gave a construction of the full adder from 11 bits to 4 bits and an improved upper bound of $O(n^{4.57})$.

In the case of $\mathbf{B}_2$-formula, Paterson, Pippenger and Zwick [14] proved a $\mathbf{B}_2$-formula size upper bound of $O(n^{3.21})$ improved to $O(n^{3.13})$ by Paterson and Zwick [15].

In the case of $\mathbf{MAJ}_3$-formula, the full adder $\mathbf{FA}_3$ can be constructed by $y_1 = [1, [\bar{1}, \bar{2}, \bar{3}], [\bar{1}, 2, 3]]$ and $y_2 = [1, 2, 3]$. So the corresponding occurrence matrix is $M = \begin{pmatrix} 3\,2\,2 \\ 1\,1\,1 \end{pmatrix}$. From this, we can obtain the following $\mathbf{MAJ}_3$-formula size upper bound for the majority function.

Theorem 5. $L_{\mathbf{MAJ}_3}(\mathbf{MAJ}_{2l+1}) \in O(n^{3.7925})$ *where* $n = 2l + 1$.

Proof. For the occurrence matrix, $M = \begin{pmatrix} 3\,2\,2 \\ 1\,1\,1 \end{pmatrix}$, the inequality $\|\boldsymbol{x}\|_\gamma \leq \|M \cdot \boldsymbol{x}\|_\gamma$ which appears in the theorem of Paterson, Pippenger and Zwick [14] can be interpreted as

$$p(a, b, c, \gamma) = (3 \cdot a + 2 \cdot b + 2 \cdot c)^\gamma + (a + b + c)^\gamma - a^\gamma - b^\gamma - c^\gamma \geq 0.$$

If $L_{\mathbf{MAJ}_3}(\mathbf{MAJ}_n) \in O(n^\gamma)$, there exists $a, b, c > 0$ such that $p(a, b, c, \gamma) < 0$. We set $a = 0.729608$, $b = c = 1$ and $1/\gamma = 3.7925$. Then, we have

$$p(a, b, c, \gamma) \approx -0.0000256657 < 0.$$

This certifies that the maximum value of $1/\gamma$ which satisfies $\|\boldsymbol{x}\|_\gamma \leq \|M \cdot \boldsymbol{x}\|_\gamma$ for any vectors $\boldsymbol{x} \geq \mathbf{0}$ is less than 3.7925. (The optimality of the value γ can be confirmed by numerical analysis. That is, the minimum value of $p(a, b, c, \gamma) > 0$ for $\gamma = 3.7924$.) Thus we have obtained the upper bound. □

The best monotone $\mathbf{U}_2$-formula size upper bound for the majority function is $O(n^{5.3})$ by a probabilistic construction of Valiant [18]. Following the analysis of Valiant's construction replaced by balanced compositions of the 3-bit majority function with random variables, we can construct a monotone $\mathbf{MAJ}_3$-formula whose size is $O(n^{4.2945})$ ($\supseteq O(n^{\log_{3/2} 3 + \log_2 3})$). The size of its conversion into a monotone $\mathbf{U}_2$-formula is $O(n^{6.2913})$ ($\supseteq O(n^{\log_{3/2} 5 + \log_2 5})$) and larger than Valiant's bound.

6 Concluding Remarks

In this paper, we have introduced the notion of $\mathbf{MAJ}_3$-formula and have shown the upper and lower bounds for $\mathbf{MAJ}_3$-formula size of several Boolean functions. The results shown in this paper are summarized in Figure 2. The figure also shows comparison with $\mathbf{U}_2$-formula complexity and $\mathbf{B}_2$-formula complexity.

	$\mathbf{B}_2$-formula	$\mathbf{MAJ}_3$-formula	$\mathbf{U}_2$-formula
Parity	$\Theta(n)$	$O(n^{1.7329})$ [4]	$\Theta(n^2)$ [10]
		$\Omega(n^{1.2618})$, $O(n^{1.3333})$ [4]	
Majority	$O(n^{3.13})$ [14,15]	$O(n^{3.7925})$	$O(n^{4.57})$ [14,15]
	$\Omega(n \log n)$ [5]	$\Omega(n^{1.2618})$	$\Omega(n^2)$ [10]
Recursive Majority		$\Theta(n)$	$O(n^{1.4650})$ [11]
			$\Omega(n^{1.2618})$ [11]
Andreev		$\Omega(n^{1.8927})$	$\Theta(n^{3-o(1)})$ [6]

Fig. 2. Formula Size Upper and Lower Bounds

There are still large gaps between the upper and lower bounds even for the parity function while we have its matching bounds for $\mathbf{U}_2$-formula and $\mathbf{B}_2$-formula. The obvious open questions are how to close these gaps. We hope that a new technical discovery to clarify $\mathbf{MAJ}_3$-formula complexity will also shed light on resolving the stiff barrier against formula complexity of the existing models.

Acknowledgment. This research is supported by the Kyoto University Hakubi Project and Grants-in-Aid for Scientific Research from the Japan Society for the Promotion of Science.

References

1. Andreev, A.E.: On a method for obtaining more than quadratic effective lower bounds for the complexity of π-scheme. Moscow University Mathematics Bulletin 42(1), 63–66 (1987)
2. Bioch, J.C., Ibaraki, T.: Decompositions of positive self-dual boolean functions. Discrete Mathematics 140(1-3), 23–46 (1995)
3. Böhler, E., Creignou, N., Reith, S., Vollmer, H.: Playing with boolean blocks, part I: Post's lattice with applications to complexity theory. ACM SIGACT News 34(4), 38–52 (2003)
4. Chockler, H., Zwick, U.: Which bases admit non-trivial shrinkage of formulae? Computational Complexity 10(1), 28–40 (2001)
5. Fischer, M.J., Meyer, A.R., Paterson, M.S.: $\Omega(nlogn)$ lower bounds on length of Boolean formulas. SIAM Journal on Computing 11(3), 416–427 (1982)
6. Håstad, J.: The shrinkage exponent of De Morgan formulas is 2. SIAM Journal on Computing 27(1), 48–64 (1998)
7. Hrubeš, P., Jukna, S., Kulikov, A., Pudlák, P.: On convex complexity measures. Theoretical Computer Science 411, 1842–1854 (2010)
8. Ibaraki, T., Kameda, T.: A theory of coteries: Mutual exclusion in distributed systems. IEEE Transactions on Parallel and Distributed Computing PDS-4(7), 779–794 (1993)
9. Karchmer, M., Kushilevitz, E., Nisan, N.: Fractional covers and communication complexity. SIAM Journal on Discrete Mathematics 8(1), 76–92 (1995)
10. Khrapchenko, V.M.: Complexity of the realization of a linear function in the case of π-circuits. Mathematical Notes 9, 21–23 (1971)
11. Laplante, S., Lee, T., Szegedy, M.: The quantum adversary method and classical formula size lower bounds. Computational Complexity 15(2), 163–196 (2006)
12. Lee, T.: A New Rank Technique for Formula Size Lower Bounds. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 145–156. Springer, Heidelberg (2007)
13. Neciporuk, E.I.: A boolean function. DOKLADY: Russian Academy of Sciences Doklady. Mathematics (formerly Soviet Mathematics–Doklady) 7, 999–1000 (1966)
14. Paterson, M.S., Pippenger, N., Zwick, U.: Optimal carry save networks. In: Boolean Function Complexity. London Mathematical Society Lecture Note Series, vol. 169, pp. 174–201. Cambridge University Press (1992)
15. Paterson, M.S., Zwick, U.: Shallow circuits and concise formulae for multiple addition and multiplication. Computational Complexity 3(3), 262–291 (1993)
16. Post, E.L.: The two-valued iterative systems of mathematical logic. Annals Mathematical Studies, vol. 5. Princeton University Press (1941)
17. Pratt, V.R.: The effect of basis on size of Boolean expressions. In: Proceedings of the 16th Annual Symposium on Foundations of Computer Science (FOCS 1975), October 13-15, pp. 119–121. IEEE (1975)
18. Valiant, L.G.: Short monotone formulae for the majority function. Journal of Algorithms 5(3), 363–366 (1984)

On the Kernelization Complexity of Problems on Graphs without Long Odd Cycles

Fahad Panolan and Ashutosh Rai

The Institute of Mathematical Sciences, Chennai, India
{fahad,ashutosh}@imsc.res.in

Abstract. Several NP-hard problems, like Maximum Independent Set, Coloring, and Max-Cut are polynomial time solvable on bipartite graphs. An equivalent characterization of bipartite graphs is that it is the set of all graphs that do not contain any odd length cycle. Thus, a natural question here is what happens to the complexity of these problems if we know that the length of the longest odd cycle is bounded by k? Let $\mathcal{O}_k$ denote the set of all graphs G such that the length of the longest odd cycle is upper bounded by k. Hsu, Ikura and Nemhauser [*Math. Programming*, 1981] studied the effect of avoiding long odd cycle for the Maximum Independent Set problem and showed that a maximum sized independent set on a graph $G \in \mathcal{O}_k$ on n vertices can be found in time $n^{O(k)}$. Later, Grötschel and Nemhauser [*Math. Programming*, 1984] did a similar study for Max-Cut and obtained an algorithm with running time $n^{O(k)}$ on a graph $G \in \mathcal{O}_k$ on n vertices.

In this paper, we revisit these problems together with q-Coloring and observe that all of these problems admit algorithms with running time $O(c^k n^{O(1)})$ on a graph $G \in \mathcal{O}_k$ on n vertices. Thus, showing that all these problems are fixed parameter tractable when parameterized by the length of the longest odd cycle of the input graph. However, following the recent trend in parameterized complexity, we also study the kernelization complexity of these problems. We show that Maximum Independent Set, q-Coloring for some fixed $q \geq 3$ and Max-Cut do not admit a polynomial kernel unless co-NP $\subseteq$ NP/poly, when parameterized by k, the length of the longest odd cycle.

1 Introduction

One of the most common ways to solve NP-hard problems is to restrict the inputs. For an example, most NP-hard graph problems become polynomial time solvable on trees. There are also some NP-hard problems, like Maximum Independent Set, q-Coloring, and Max-Cut, that remain polynomial time solvable on bipartite graphs. A graph $G = (V, E)$ is called *bipartite* if the vertex set V can be partitioned into two parts V_1 and V_2 such that every edge in E has one end-point in V_1 and the other in V_2. An equivalent characterization of bipartite graphs is that it is the set of all graphs that do not contain any odd length cycle. Thus, a natural question here is what happens to the complexity

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 445–457, 2012.

of these problems if we know that the the length of the longest odd cycle is bounded by k?

Let $\mathcal{O}_k$ denote the set of all graphs G such that the length of the longest odd cycle is upper bounded by k. Hsu, Ikura and Nemhauser [16] initiated a study of NP-hard optimization problems on $\mathcal{O}_k$. In particular, they studied the effect of avoiding long odd cycle for the Maximum Independent Set problem and showed that a maximum sized independent set on a graph $G \in \mathcal{O}_k$ on n vertices can be found in time $n^{O(k)}$. Later, Grötschel and Nemhauser [14] did a similar study for Max-Cut and obtained an algorithm with running time $n^{O(k)}$ on a graph $G \in \mathcal{O}_k$ on n vertices. In the modern language of parameterized complexity, it means that these problems admit XP algorithms parameterized by the length of the longest odd cycle of the input graph.

The goal of parameterized complexity is to find ways of solving NP-hard problems more efficiently than brute force: here aim is to restrict the combinatorial explosion to a parameter that is hopefully much smaller than the input size. Formally, a *parameterization* of a problem is assigning an integer k to each input instance and we say that a parameterized problem is *fixed-parameter tractable (FPT)* if there is an algorithm that solves the problem in time $f(k)\cdot|I|^{O(1)}$, where $|I|$ is the size of the input and f is an arbitrary computable function depending on the parameter k only. There is a long list of NP-hard problems that are FPT under various parameterizations: finding a vertex cover of size k, finding a cycle of length k, finding a maximum independent set in a graph of treewidth at most k, etc. There is also a theory of hardness that allows us to show that certain parametrized problem is not amenable to this approach. XP is the class of all problems that are solvable in time $O(n^{g(k)})$. For more background, the reader is referred to the monographs [10,11,21].

In this paper we study the following parameterized problems in the realm of parameterized complexity. A subset S of vertices of a graph $G = (V, E)$ is called *independent* if there is no edge in E with both end-points in S.

Maximum Independent Set (MIS)

Instance: An undirected graph $G = (V, E) \in \mathcal{O}_k$ and a positive integer t.
Parameter: k.
Problem: Does G has an independent set of size t?

Let $q \geq 3$ be a fixed positive integer. We say that a graph $G = (V, E)$ is q-*colorable* if there is function $f : V \to \{1, \ldots, q\}$ such that for every edge $(u, v) \in E$, $f(u) \neq f(v)$.

q-coloring

Instance: An undirected graph $G = (V, E) \in \mathcal{O}_k$
Parameter: k
Problem: Does there exist a proper q-coloring of G?

We say that a graph $G = (V, E)$ has a *cut* of size t if the vertex set V can be partitioned into two parts V_1 and V_2 such that the number of edges in E that have one end-point in V_1 and the other in V_2 is at least t.

MAX-CUT

Instance: An undirected graph $G = (V, E) \in \mathcal{O}_k$ and a positive integer t
Parameter: k
Problem: Does there exist a cut of size at least t in G?

We observe that MIS, q-COLORING and MAX-CUT are FPT by giving algorithms that have running time of the form $O(c^k n^{O(1)})$ on a graph $G \in \mathcal{O}_k$ on n vertices. However, we go further and study the kernelization complexity of these problems. A parameterized problem is said to admit a *polynomial kernel* if every instance (I, k) can be reduced in polynomial time to an equivalent instance with both size and parameter value bounded by a polynomial in k. The study of kernelization is a major research frontier of Parameterized Complexity and many important recent advances in the area are on kernelization. These include general results showing that certain classes of parameterized problems have polynomial kernels [2,6,12] or randomized kernels [20]. The recent development of a framework for ruling out polynomial kernels under certain complexity-theoretic assumptions [5,9,13] has added a new dimension to the field and strengthened its connections to classical complexity. For overviews of kernelization we refer to surveys [4,15] and to the corresponding chapters in books on Parameterized Complexity [11,21].

We show that MIS, q-COLORING for some fixed $q \geq 3$ and MAX-CUT do not admit a polynomial kernel unless co-NP $\subseteq$ NP/poly, when parameterized by k, the length of the longest odd cycle. These are our main results. Our results fit into the recent study of problems parameterized by other structural parameters. See, for example ODD CYCLE TRANSVERSAL parameterized by various structural parameters [19] or TREEWIDTH parameterized by vertex cover [7] or VERTEX COVER parameterized by feedback vertex set [17], q-COLORING parameterized by the number of vertex-deletions needed to make the graph chordal [18]. The effect of length of cycles on parameterized complexity of various problems has also been studied [22].

2 Preliminaries

The notions of kernelization is formally defined as follows.

Definition 1. *A kernelization algorithm, or in short, a kernel for a parameterized problem $\Pi \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, outputs in time polynomial in $|x| + k$ a pair $(x', k') \in \Sigma^* \times \mathbb{N}$ such that*

(a) $(x, k) \in \Pi$ if and only if $(x', k') \in \Pi$
(b) $|x'| + k' \leq g(k)$, where g is an arbitrary computable function.

The function g is referred to as the size of the kernel. If g is a polynomial function then we say that Π admits a polynomial kernel.

Let $G = (V, E)$ be a graph and $A \subseteq V$, then by $G[A]$, we denote the subgraph induced on A.

Definition 2. *Let G be a graph. A* tree decomposition *of a graph $G = (V, E)$ is a pair $(T = (V(T), E(T)), \mathcal{X} = \{X_t\}_{t \in V(T)})$ such that*

- $\cup_{t \in V(T)} X_t = V$,
- *for every edge $(x, y) \in E$, there is a $t \in V(T)$ such that $\{x, y\} \subseteq X_t$, and*
- *for every vertex $v \in V$, the subgraph of T induced by the set $\{t \mid v \in X_t\}$ is connected.*

The width *of a tree decomposition is $\left(\max_{t \in V(T)} |X_t|\right) - 1$ and the* treewidth *of G is the minimum width over all tree decompositions of G.*

Definition 3. *Let $G = (V, E)$ be a connected graph. A vertex $v \in V$ is called an articulation point of G if the graph obtained from G, after deleting vertex v, is disconnected.*

Definition 4. *A connected graph on 3 or more vertices is 2-connected if it has no articulation point. A subgraph that has no articulation point and is maximal with this property is called a block. The block decomposition of a graph is just the set of all the blocks of the graph.*

Clearly, each block of a connected graph is either 2-connected (if it contains 3 or more vertices) or 2 vertices joined by an edge. The block decomposition of a connected graph can be found in $O(|E|)$ time [1].

Observation 5. *A connected graph with more than one block contains at least two blocks with at most one articulation point. Also, two blocks can share at most one vertex.*

Each tree with more than one node has at least two leaves. Since the blocks impose a tree like structure on the graph, if there are more than one block, at least two of them are at leaf level, and have only one articulation point. Also, we see that the blocks are maximal 2-connected components, so if two blocks share more than one vertex, the maximality condition gets violated.

3 FPT Algorithms

In this section we give FPT algorithms for MIS, q-Coloring and Max-Cut by improving on block composition algorithms developed in [14,16], and using some facts about 2-connected graphs. For MIS and Max-Cut we revisit the algorithms developed in [14,16] and speed their main step, using the FPT algorithms for MIS and Max-Cut when parameterized by treewidth, to obtain FPT algorithms when the input graph does not contain long odd cycle. Our algorithm for q-Coloring is new. For our algorithms, we also need the following two theorems.

Theorem 6 ([14]). *Let G be a 2-connected, non-bipartite graph whose longest odd cycle has length l_0. Then the longest cycle of G has length at most $2(l_0 - 1)$.*

Theorem 7 ([3]). *If the size of the longest cycle of a graph is bounded by t, then its treewidth is at most $(t - 1)$*

Combining the above two, we get that each block in a connected graph $G \in \mathcal{O}_k$ is either 2-connected, non-bipartite subgraph with its treewidth at most $(2(k-1)-1) < 2k$, or it is a bipartite subgraph.

For all three FPT algorithms, we assume that the input is a graph G along with a block decomposition of G. Since it can be computed in $O(|E|)$ time, it does not affect the running time asymptotically, as all the algorithms already have a multiplicative factor of $n^{O(1)}$.

3.1 Maximum Independent Set (MIS)

We know that the maximum sized independent set can be found in time $2^{tw}n^{O(1)}$ on graphs of treewidth tw [21]. Also, it is well known that it can be solved in time $n^{O(1)}$ on bipartite graphs. Thus, for each block, the maximum sized independent set can be computed in time $4^k n^{O(1)}$, since treewidth of each block is bounded by $2k$. Now, we give the main lemma on which the algorithm is based.

Lemma 8 (Corollary 1, [16]). *Let $G = (V, E)$ be a graph with at least two blocks. Let $G[S]$ be a block of G with vertex set $S \subseteq V$ and with only one articulation vertex v. For any $X \subseteq V$, let P_X be the maximum sized independent set on $G[X]$*

1. *If $|P_S| = |P_{S\setminus\{v\}}|$, then $P_{S\setminus\{v\}} \cup P_{V\setminus S}$ is a maximum sized independent set on G.*
2. *If $|P_S| = |P_{S\setminus\{v\}}| + 1$, then $(P_S \setminus \{v\}) \cup P_{(V\setminus S)\cup\{v\}}$ is a maximum sized independent set on G.*

Now, we solve the MIS problem in the following way.

Case 1: G contains only one block. If the block is bipartite, then we can find a maximum sized independent set in $n^{O(1)}$ time. Otherwise, the block has treewidth at most $2k$ and the bounded treewidth algorithm works. So, we can find a maximum sized independent set in time $4^k n^{O(1)}$ for a single block.

Case 2: G contains at least two blocks. We first select a block $G[S]$ with only one articulation point, say v (such a block is guaranteed by Observation 5). Now, we find a maximum sized independent sets P_S and $P_{S\setminus\{v\}}$. If $|P_S| = |P_{S\setminus\{v\}}|$, then we delete S from the graph to get $G' = G \setminus S$ and find a maximum sized independent set on G' recursively. Finally, we obtain a maximum sized independent set of the original graph G as $P_{S\setminus\{v\}} \cup P_{V\setminus S}$.

Else, we delete $S \setminus \{v\}$ from the graph to get G' and find a maximum sized independent set on G' recursively. Then, we obtain the maximum sized independent set of the original graph G as $(P_S \setminus \{v\}) \cup P_{(V\setminus S)\cup\{v\}}$.

Correctness of the algorithm follows from Lemma 8. For the running time analysis, observe that in each step, the number of blocks decreases by one. Also, in each step, the subroutine for finding a maximum sized independent set on a block is called at most twice. Combining the two solutions takes $O(n)$ time. So, in total, the algorithm makes at most n recursive calls, each of them taking at most $O(4^k n^{O(1)})$ time. Finally, we compare the solution size with the given number t and answer appropriately to solve the decision version. Thus, we get the following theorem.

Theorem 9. MIS *can be solved in time* $4^k n^{O(1)}$.

3.2 q-Coloring

For our FPT algorithm we will use the well known result that one can test whether a graph of treewidth tw is q-colorable or not in time $q^{tw} n^{O(1)}$. For bipartite graphs, we know that they are 2-colorable. We first prove the following lemma in order to show q-Coloring is FPT.

Lemma 10. *A graph G is q-colorable iff all its blocks are q-colorable.*

Proof. The forward direction is trivial, since all the blocks are induced subgraphs of G. We prove the backward direction by induction on the number of blocks in G. Clearly, when G has only one block (G itself), the statement of the lemma is true. Now, we take the number of blocks to be $n > 1$, which are all q-colorable. We find a block B with only one articulation point, say v (such a block is guaranteed by Observation 5). We delete $(B \setminus \{v\})$ to get a graph G' which has $(n-1)$ blocks. Now, since all these blocks are q-colorable, G' is q-colorable by induction hypothesis. We also know that B is q-colorable. If the vertex v has got the same color in both the colorings, i.e. in B and G', then we just take the union of them to be the q-coloring of the graph G and we are done. We argue that we can always get such a coloring. If not, we permute the colors in B so that v has the same color as it has in G'. This concludes the proof of the lemma. □

The lemma immediately gives rise to an algorithm which checks the q-colorability of all the blocks and outputs G is q-colorable iff all its blocks are q-colorable. As mentioned earlier, the q-colorability of a block can be checked in time $q^{2k} n^{O(1)}$, and there are at most n blocks, hence the algorithm runs in time $q^{2k} n^{O(1)}$. Thus we get the following result.

Theorem 11. q-Coloring *for fixed $q \geq 3$ can be solved in time* $q^{2k} n^{O(1)}$.

Along the lines of Theorems 9 and 11 we get the following Theorem 12.

Theorem 12. [∗][1] Max-Cut *can be solved in time* $4^k n^{O(1)}$.

[1] Proofs of results marked [∗] will appear in the full version of the paper.

4 Kernelization Lower Bounds

In this section, we show that even though MIS, q-Coloring, and Max-Cut admit algorithms with running time $4^k n^{O(1)}$, $q^{2k} n^{O(1)}$ and $4^k n^{O(1)}$ respectively on $\mathcal{O}_k$, they do not admit polynomial kernels. We first set up the known machinery to show the kernelization lower bounds and then apply them to these problems.

4.1 Lower Bound Machinery

In this section, we state some of the known techniques developed for showing that problems do not admit polynomial kernels.

Definition 13 (Composition [5]). *A composition algorithm (also called OR-composition algorithm) for a parameterized problem $\Pi \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that receives as input a sequence $((x_1, k), ..., (x_t, k))$, with $(x_i, k) \in \Sigma^* \times \mathbb{N}$ for each $1 \leq i \leq t$, uses time polynomial in $\sum_{i=1}^{t} |x_i| + k$, and outputs $(y, k') \in \Sigma^* \times \mathbb{N}$ with (a) $(y, k') \in \Pi \iff (x_i, k) \in \Pi$ for some $1 \leq i \leq t$ and (b) k' is polynomial in k. A parameterized problem is compositional (or OR-compositional) if there is a composition algorithm for it.*

We define the notion of the *unparameterized version* of a parameterized problem Π. The mapping of parameterized problems to unparameterized problems is done by mapping (x, k) to the string $x\#1^k$, where $\# \in \Sigma$ denotes the blank letter and 1 is an arbitrary letter in Σ. In this way, the unparameterized version of a parameterized problem Π is the language $\tilde{\Pi} = \{x\#1^k | (x, k) \in \Pi\}$. The following theorem yields the desired connection between the two notions.

Theorem 14 ([5,13]). *Let Π be a compositional parameterized problem whose unparameterized version $\tilde{\Pi}$ is NP-complete. Then, if Π has a polynomial kernel then* co-NP $\subseteq$ NP/*poly.*

For some problems, obtaining a composition algorithm directly is a difficult task. Instead, we can give a reduction from a problem that provably has no polynomial kernel unless co-NP $\subseteq$ NP/poly to the problem in question such that a polynomial kernel for the problem considered would give a kernel for the problem we reduced from. We now define the notion of polynomial parameter transformations.

Definition 15 ([8]). *Let P and Q be parameterized problems. We say that P is polynomial parameter reducible to Q, written $P \leq_{ppt} Q$, if there exists a polynomial time computable function $f : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ and a polynomial p, such that for all $(x, k) \in \Sigma^* \times \mathbb{N}$ (a) $(x, k) \in P \iff (x', k') = f(x, k) \in Q$ and (b) $k' \leq p(k)$. The function f is called polynomial parameter transformation.*

Proposition 16 ([8]). *Let P and Q be the parameterized problems and $\tilde{P}$ and $\tilde{Q}$ be the unparameterized versions of P and Q respectively. Suppose that $\tilde{P}$ is NP-complete and $\tilde{Q}$ is in NP. Furthermore if $P \leq_{ppt} Q$, then if Q has a polynomial kernel then P also has a polynomial kernel.*

4.2 MIS

In this section, we prove that MIS on $\mathcal{O}_k$ does not admit polynomial kernel unless co-NP $\subseteq$ NP/poly. To prove this, we assume MIS has a polynomial kernel in parameter k. Then we will show that MIS is OR-compositional. We know MIS is NP–complete. Due to Theorem 14, this implies that co-NP $\subseteq$ NP/poly. Hence we will conclude that MIS on $\mathcal{O}_k$ does not admit polynomial kernel unless co-NP $\subseteq$ NP/poly.

Let $((x, l), k)$ denote an instance of MIS, where we want to find whether given graph $x \in \mathcal{O}_k$, has an independent set of size l. Here the parameter we consider is k. Now we construct an OR-composition algorithm assuming MIS has polynomial kernel as follows. The OR-composition algorithm receives as input a sequence $(((x_1, l_1), k), ..., ((x_t, l_t), k))$, where each $((x_i, l_i), k)$ is an instance of MIS. Following are the steps of OR-composition algorithm.

1. Kernelize each instance $((x_i, l_i), k)$ and get output $((y_i, l'_i), k'_i)$.
2. For each $((y_i, l'_i), k'_i)$ add $l'_i - 1$ more vertices and add edges from all newly added vertices to all vertices in y_i to get the instance $((z_i, l'_i), k''_i)$, where $k''_i = k'_i(l'_i - 1)$.
3. Output $(y, 1 + \sum_{i=1}^{t}(l'_i - 1), \max(k''_i))$, where y is the disjoint union of $z_i s$.

Lemma 17. $(y, 1 + \sum_{i=1}^{t}(l'_i - 1), \max(k''_i))$ *is an* Yes *instance of* MIS *if and only if there exists* i *such that* $((x_i, l_i), k_i)$ *is an* Yes *instance of* MIS

Proof. Suppose there exists i such that $((x_i, l_i), k_i)$ is an Yes instance of MIS. Then $((y_i, l'_i), k'_i)$ is an Yes instance of MIS. The l'_i vertices of the independent set of y_i together with all newly added $l'_j - 1$ vertices (in step 2) to y_j for all $j \neq i$ form an independent set of y of size $1 + \sum_{i=1}^{t}(l'_i - 1)$. Suppose for each i, $((x_i, l_i), k_i)$ is No instance of MIS. Then for each i, $((y_i, l'_i), k'_i)$ is No instance of MIS. Any independent set of z_i will be either completely from the vertices of y_i or completely from the newly added $l'_i - 1$ vertices in step 2, because all vertices in y_i are adjacent to all newly added $l'_i - 1$ vertices in step 2. The maximum independent set size in each z_i will be $l'_i - 1$. Hence $(y, 1 + \sum_{i=1}^{t}(l'_i - 1), \max(k''_i))$ is a No instance of MIS. □

Lemma 18. $\max(k''_i)$ *is bounded by polynomial in* k.

Proof. In the first step of the algorithm we kernelize each instance $((x_i, l_i), k)$ and get output $((y_i, l'_i), k'_i)$. Hence each y_i, l'_i and k'_i is bounded by polynomial in k. Since in the second step, for each $((y_i, l'_i), k'_i)$ we add $l'_i - 1$ more vertices to y_i to get $((z_i, l'_i), k''_i)$, $k''_i = k'_i(l'_i - 1)$ is bounded by polynomial in k. Hence $\max(k''_i)$ is polynomial in k. □

Hence we have the following theorem.

Theorem 19. MIS *does not admit any polynomial kernel unless* co-NP $\subseteq$ NP/*poly.*

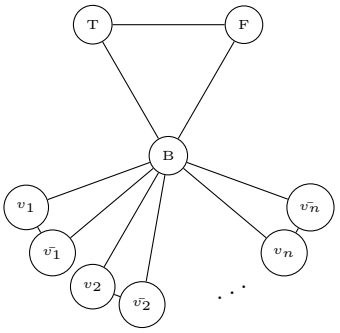


Fig. 1.

4.3 *q*-Coloring

In this section, we first show that 3-coloring on $\mathcal{O}_k$ does not admit polynomial kernel unless co-NP $\subseteq$ NP/poly using Proposition 16 and then extend the proof to q-coloring. Towards our goal, we give a polynomial parameter transformation from Satisfiability (cnf-sat) parameterized by the number of variables n to 3-coloring. It is known that CNF-SAT parameterized by the number of variables does not admit polynomial kernel unless co-NP $\subseteq$ NP/poly [13].

To show polynomial parameter transformation from cnf-sat to 3-coloring, from a formula ϕ in conjunctive normal form having n variables and m clauses, we construct a graph G in which odd cycle length is bounded by polynomial in n with the property that ϕ is satisfiable if and only if G is 3-colorable.

Construction. We define nodes v_i and $\bar{v_i}$ corresponding to each variable x_i and $\bar{x_i}$. We also define three special nodes T, F and B which we refer to as *True,False* and *Base*. We join each $v_i, \bar{v_i}$ and B to form a triangle. We also join T, F and B to form a triangle. The graph we created so far will look like in Fig.1.

Now for each clause we will construct a gadget. For a clause having odd number of literals we construct a gadget as shown in the Fig.2. Let $C_i = v_1 \vee v_2 \vee ... \vee v_l$ is clause having odd number of literels. Then corresponding clause gadget will look like in the Fig.2. Here a_i-vertices and b_i-vertices are new vertices and, T and v_i s are the existing vertices in Fig.1. Similar way we can construct a clause gadget corresponding to a clause C_i' having even number of literals (say $C_i' = v_1 \vee v_2 \vee ... \vee v_r$) as shown in the Fig.3. Here note that each $a_i s, b_i s$ and c are different for different clause gadgets.

In any proper coloring of G, we intrepret colors of vertices B, T and F as *base color, true color* and *false color* respectively.

Lemma 20. *A clause gadget corresponding to a clause C is 3-colorable if and only if one vertex corresponding to a literal in C is colored with true color.*

Proof. Let $C = v_1 \vee v_2 \vee ... \vee v_t$ is a clause. Suppose all vertices corresponding to literals in the clause C are colored with *false color*, then all $a_i s$ should get *base color* and b_1 can be colored with *true color* or *false color*. If b_1 is colored *true*,

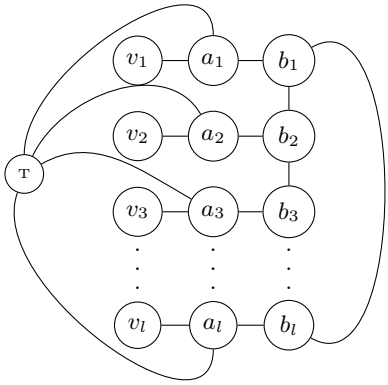


Fig. 2. Gadget corresponding to a clause having odd number of literals

b_2 should be colored *false*, b_3 should be colored *true* and finally b_t will get color *true* (in the case when t is odd) or vertex c will get color *true* (in the case when t is even), which is not a proper coloring. So to have a proper coloring vertices corresponding to the literals should not be all colored *false*. If at least one of the vertices corresponding to a literal (say v_j) in the clause C is colored true, then a_j can be colored *false* and b_j can be colored *base*. So we can properly color vertices $b_i s$ with *true* and *false* colors. □

Lemma 21. *ϕ is satisfiable if and only if G is 3-colorable.*

Proof. Suppose ϕ is satisfiable, we can color vertices corresponding literals which are true in the satisfying assignment with *true color* and the vertices corresponding to literals which are false in the satisfying assignment with *false color*. Since ϕ is satisfiable for each clause gadget at least one vertex corresponding to a literal in the clause gadget colored with *true color*. Hence by Lemma 20, every clause gadget is 3-colorable. Since each v_i and $\bar{v_i}$ gets different colors, G is 3-colorable. Conversely, if there exist a 3-coloring of G then we can assign each variable x_i is true or false depending on the color of v_i. By Lemma 20, a clause gadget is 3-colorable implies at least one vertex corresponding to a literal in the clause gets color *true*. Since G is 3-colorable every clause gadget has one vertex corresponding to a literal in the clause which is colored *true* and each v_i and $\bar{v_i}$ gets different colors. Hence the constructed satisfying assignment satisfies ϕ. □

Lemma 22. *The largest odd cycle length in G is bounded by polynomial in n.*

Proof. Each clause gadget is connected to other clause gadget only via vertices B,T and $v_i s$ which is bounded by $O(n)$. Number of vertices in a clause gadget is bounded by $O(n)$. Hence largest cycle length in G is bounded by $O(n^2)$. □

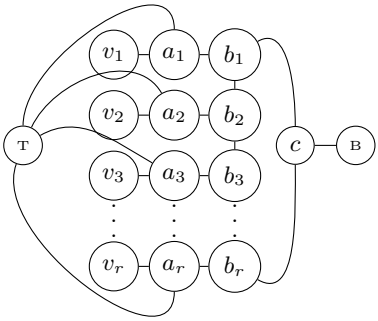


Fig. 3. Gadget corresponding to a clause having even number of literals

Hence we have the following theorem.

Theorem 23. 3-COLORING *does not admit polynomial kernel unless* CO-NP $\subseteq$ NP/*poly.*

Now we explain how to extend the proof of Theorem 23 to show that CNF-SAT is polynomial parameter reducible to q-COLORING. From the given instance ϕ with n variables of CNF-SAT, we first construct a graph G as described in the proof of Theorem 23. Now construct a graph G' by adding a complete graph on $q-3$ vertices (K_{q-3}) to G and adding edges between vertices from K_{q-3} and vertices from G. ϕ is satisfiable if and only if G is 3-colorable. Now it is easy to see that G is 3-colorable if and only if G' is q-colorable. Since each clause gadget in G' is connected with another clause gadget only via $B, T, v_i s$ and K_{q-3}, length of largest cycle in G' is bounded by $O(n(n+q))$. Hence we have the following theorem.

Theorem 24. q-COLORING *for a fixed* q*, does not admit polynomial kernel unless* CO-NP $\subseteq$ NP/*poly.*

Theorem 25. [$*$] MAX-CUT *does not admit polynomial kernel unless* CO-NP $\subseteq$ NP/*poly.*

5 Conclusion

In this paper we studied several NP-hard problems parameterized by the length of the longest odd cycle. We showed that MIS, q-COLORING and MAX-CUT are FPT and do not admit polynomial kernels under certain complexity theory assumptions when parameterized by the length of the longest odd cycle. We also observe that our parameterized algorithms also work for weighted version of MIS and MAX-CUT. It would be interesting to obtain meta-theorems in the realm of kernelization and parameterized complexity for graph problems on $\mathcal{O}_k$.

Acknowledgements. We thank Saket Saurabh for the insightful discussions which led to this work, and the anonymous reviewers for the numerous comments.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley (1974)
2. Alon, N., Gutin, G., Kim, E.J., Szeider, S., Yeo, A.: Solving MAX-r-SAT above a tight lower bound. In: SODA, pp. 511–517 (2010)
3. Birmele, E.: Tree-width and circumference of graphs. Journal of Graph Theory 43(1), 24–25 (2003)
4. Bodlaender, H.L.: Kernelization: New Upper and Lower Bound Techniques. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 17–37. Springer, Heidelberg (2009)
5. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. J. Comput. Syst. Sci. 75(8), 423–434 (2009)
6. Bodlaender, H.L., Fomin, F.V., Lokshtanov, D., Penninkx, E., Saurabh, S., Thilikos, D.M.: (meta) kernelization. In: FOCS, pp. 629–638 (2009)
7. Bodlaender, H.L., Jansen, B.M.P., Kratsch, S.: Preprocessing for Treewidth: A Combinatorial Analysis through Kernelization. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6755, pp. 437–448. Springer, Heidelberg (2011)
8. Bodlaender, H.L., Thomassé, S., Yeo, A.: Analysis of data reduction: Transformations give evidence for non-existence of polynomial kernels. Technical report (2008)
9. Dell, H., van Melkebeek, D.: Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In: STOC, pp. 251–260 (2010)
10. Downey, R.G., Fellows, M.R.: Parameterized Complexity, 530 p. Springer (1999)
11. Flum, J., Grohe, M.: Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series). Springer-Verlag New York, Inc., Secaucus (2006)
12. Fomin, F.V., Lokshtanov, D., Saurabh, S., Thilikos, D.M.: Bidimensionality and kernels. In: SODA, pp. 503–510 (2010)
13. Fortnow, L., Santhanam, R.: Infeasibility of instance compression and succinct PCPs for NP. In: STOC, pp. 133–142 (2008)
14. Grötschel, M., Nemhauser, G.L.: A polynomial algorithm for the max-cut problem on graphs without long odd cycles. Math. Programming 29(1), 28–40 (1984)
15. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. SIGACT News 38(1), 31–45 (2007)
16. Hsu, W.L., Ikura, Y., Nemhauser, G.L.: A polynomial algorithm for maximum weighted vertex packings on graphs without long odd cycles. Math. Programming 20(2), 225–232 (1981)
17. Jansen, B.M.P., Bodlaender, H.L.: Vertex cover kernelization revisited: Upper and lower bounds for a refined parameter. In: STACS, pp. 177–188 (2011)
18. Jansen, B.M.P., Kratsch, S.: Data Reduction for Graph Coloring Problems. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 90–101. Springer, Heidelberg (2011)
19. Jansen, B.M.P., Kratsch, S.: On Polynomial Kernels for Structural Parameterizations of Odd Cycle Transversal. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 132–144. Springer, Heidelberg (2012)

20. Kratsch, S., Wahlström, M.: Compression via matroids: a randomized polynomial kernel for odd cycle transversal. In: SODA, pp. 94–103 (2012)
21. Niedermeier, R.: Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications). Oxford University Press, USA (2006)
22. Raman, V., Saurabh, S.: Short cycles make W-hard problems hard: FPT algorithms for W-hard problems in graphs with no short cycles. Algorithmica 52(2), 203–225 (2008)

The Complexity of Unary Subset Sum

Nutan Limaye[1], Meena Mahajan[2], and Karteek Sreenivasaiah[2]

[1] Indian Institute of Technology, Bombay, India
nutan@cse.iitb.ac.in.
[2] The Institute of Mathematical Sciences, Chennai, India
{meena,karteek}@imsc.res.in.

Abstract. Given a stream of n numbers and a number B, the subset sum problem deals with checking whether there exists a subset of the stream that adds to exactly B. The unary subset sum problem, USS, is the same problem when the input is encoded in unary. We prove that any p-pass randomized algorithm computing USS with error at most $1/3$ must use space $\Omega(\frac{B}{p})$. For $p \leq B$, we give a randomized p-pass algorithm that computes USS with error at most $1/3$ using space $\tilde{O}(\frac{nB}{p})$. We give a deterministic one-pass algorithm which given an input stream and two parameters B, ϵ, decides whether there exist a subset of the input stream that adds to a value in the range $[(1-\epsilon)B, (1+\epsilon)B]$ using space $O\left(\frac{\log B}{\epsilon}\right)$. We observe that USS is monotone (under a suitable encoding) and give a monotone NC^2 circuit for USS. We also show that any circuit using ε-approximator gates for USS under this encoding needs $\Omega(n/\log n)$ gates to compute the Disjointness function.

1 Introduction

The Subset Sum problem is defined as follows: Given a number $B \in \mathbb{N}$ and a sequence of numbers $a_1, \ldots, a_n \in \mathbb{N}$, decide whether there is a subset $S \subseteq [n]$ such that $\sum_{i \in S} a_i = S$. This problem is one of the earliest problems shown to be NP-complete and can be found in [GJ79].

The Unary Subset Sum problem (USS) is the same problem, but with the input numbers given in unary (for instance $1^B 0 1^{a_1} 0 \ldots 1^{a_n}$.)

USS is known to be in P. In [EJT10], Elberfeld, Jakoby and Tantau showed a powerful meta-theorem for obtaining logspace upper bounds, and used it to conclude that USS is even in LogSpace. In [Kan10] Daniel Kane gave a considerably simplified logspace algorithm. Improving upon this further, in [EJT12] it is shown that under appropriate encodings USS has polynomial-sized formulas and hence is in NC^1. They also show that USS has polynomial-sized constant-depth formulas using MAJORITY and NOT gates and hence is in TC^0. On the other hand, it can be shown easily that MAJORITY reduces to computing USS and hence USS is TC^0-hard. Thus USS is TC^0-complete.

A natural question to ask at this point is: Is it crucial to have access to the entire input at any time in order to be able to solve USS in LogSpace? In other words: how hard, with regard to space, is USS when the inputs are, say, read

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 458–469, 2012.

in a stream? We study the complexity of USS and related questions in different models from this perspective.

In Section 2, we consider the space complexity of USS in the streaming world. The input numbers arrive in a stream, and we want to design a small space algorithm that makes one, or maybe a few, passes over the input stream and decides the instance. We assume that B is the first number in the stream. We assume that the stream contains integers in the range $1, \ldots, B$. We use lower bounds from communication complexity to show that any randomized p-pass streaming algorithm for USS that makes error bounded by $1/3$ must use $\Omega(\frac{B}{p})$ space (Theorem 1). Also, by modifying the algorithm from [Kan10], we obtain, for each $p \leq B$, a randomized streaming algorithm for USS that makes p passes over the input, uses space $O((\frac{nB}{p}) \log^2(Bn))$, and on each input errs with probability at most $1/3$ (Theorem 2).

In Section 3, we consider the complexity of approximating USS. We say that an algorithm ϵ-approximates USS if it outputs Yes exactly when there is a subset $S \subseteq [n]$ such that $|B - \sum_{i \in S} a_i| < \epsilon B$. Note that this problem is not necessarily easier than the exact version of USS since the exact version is not an optimization problem. i.e., if the answer to exact-USS on an input instance is NO, this does not mean that the answer to the approximate version is a NO. However, there is a fully polynomial time approximation scheme (FPTAS) for approximate subset sum, where the goal is to find a subset S such that $B - \epsilon B \leq \sum_{i \in S} a_i \leq B$ (see for instance [CLRS09]). But this is not efficient in terms of streaming space, even when the input is given in unary. We give a simple deterministic 1-pass streaming algorithm that takes input $\epsilon, B, \tilde{a}$ and ϵ-approximates USS on the stream $\tilde{a}$ using space $O(\frac{\log B}{\epsilon})$ (Theorem 3). We also show that this is almost tight (Lemma 3).

In Section 4, we consider the monotone circuit complexity of USS. Note that USS is naturally monotone in the following sense: if the number of occurrences of a number i in the stream is increased, a Yes instance remains a Yes instance. To model this monotonicity, we consider the following encoding of USS: For each positive integer B, the input consists of the frequency of each number in the stream in unary. That is, an instance consists of B blocks of B bits each, where the ith block w_i has as many 1s as the number of occurrences m_i of the number i in the stream. Thus, the input records the multiplicity of each number in $[B]$ (without loss of generality, no multiplicity exceeds B). Call this problem the multiplicity-USS problem, mUSS. We show, by a monotone reduction to reachability, that this problem has monotone circuits of polynomial size and $O(\log^2 B)$ depth (Theorem 5). The circuit we construct can also be used to solve the approximate version of USS.

A related question is: How powerful are ε-approximators when used as gate primitives in circuits? We explore this direction in section 5. We observe that ε-approximators for mUSS (we call them ApproxUSS gates) are at least as powerful as threshold gates. Using a technique introduced by Nisan in [Nis94], we also show that any circuit computing the Disjointness function using ε-mUSS gates requires $\Omega(n/\log n)$ gates. However we have not been able to compare ApproxUSS gates explicitly with Linear Threshold gates.

2 Exact USS in Streaming Model

In the communication problem corresponding to USS, both Alice and Bob are given an integer B. Further, each of them has a multiset of numbers and they have to determine if there is a sub-multiset of numbers among the union of their multisets that adds to B. The goal is to minimize the number of bits exchanged between Alice and Bob. Additionally, there may be constraints on how often the communication exchange changes direction (the number of rounds).

A standard lower bound technique (see [AMS99]) shows that a p-pass space $O(s)$ streaming algorithm yields a protocol with communication complexity $O(ps)$ and $2p-1$ rounds. Thus a communication complexity lower bound yields a streaming space lower bound. We use this technique to show that any 1-pass streaming algorithm for USS needs $\Omega(B)$ space.

Lemma 1. *Any deterministic or randomized 1-pass streaming algorithm for USS uses space $\Omega(B)$.*

Proof. We reduce the INDEX problem to USS. The INDEX_n function is defined as follows: Alice has $x \in \{0,1\}^n$ and Bob has an index $k \in [n]$. The goal is to find x_k. Alice can send one message to Bob, after which Bob should announce what he believes is the value of x_k. It is known that the 1-way randomized communication complexity of INDEX_n is $\Theta(n)$ (see [BYJKS02] or [KNR95]).

The reduction from INDEX_n to USS is as follows: Let $B = 2n$. Alice creates a set $S = \{2n - i | x_i = 1\}$. Bob creates the set $T = \{k\}$. Notice that each number in S is at least n. And so any subset of S that has two or more numbers would have a sum strictly greater than $2n$. Hence any subset of $S \cup T$ that has a sum of $2n$ can have at most one number from S. Now it is easy to see that if $x_k = 1$, the subset $\{(2n-k), k\}$ has sum $2n$. And if $x_k = 0$, there is no subset of $S \cup T$ that has sum $2n$. Thus a protocol that correctly decides the USS instance where $B = 2n$, Alice has S and Bob has T with communication cost c also correctly decides $\text{INDEX}_n(x,k)$ with communication cost c.

Assume that there is a space $s(B)$ 1-pass streaming algorithm for USS. Then there is a cost $s(B)$ protocol for USS, and hence by the above reduction, a cost $s(2n)$ protocol for $\text{INDEX}_n(x,k)$. By the lower bound for INDEX, $s(2n) \in \Omega(n)$, and so $s(B) \in \Omega(B)$. □

A generalization of the above proof gives a space lower bound for streaming USS that depends on the number of passes.

Theorem 1. *Any deterministic or randomized p-pass streaming algorithm for USS uses space $\Omega(B/p)$.*

Proof. We give a reduction from $\overline{\text{DISJ}_n}$ to USS. The Disjointness problem DISJ_n is defined as follows: for $x, y, \in \{0,1\}^n$, $\text{DISJ}_n(x,y) = \wedge_{i=1}^n \neg(x_i \wedge y_i)$. (That is, if x and y are characteristic vectors of sets $X, Y \subseteq [n]$, then $\text{DISJ}_n(x,y) = 1$ if and only if $X \cap Y = \emptyset$.) Its complement $\overline{\text{DISJ}_n}$ is the intersection problem. Alice and Bob are given $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$ respectively. The goal is to determine if there exists an $i \in [n]$ such that $x_i = y_1 = 1$. It is known

[KS92, Raz92, BYJKS04] that any randomized protocol for $\overline{\text{DISJ}_n}$, with any number of rounds, must exchange $\Omega(n)$ bits to bound error probability by $1/3$.

The reduction from $\overline{\text{DISJ}}$ to USS is as follows: We set $B = 12n - 1$. Alice constructs the set $S = \{8n - 2i | x_i = 1\}$. Bob constructs the set $T = \{4n + 2i - 1 | y_i = 1\}$. Notice that all numbers in S are greater than $B/2$, and that all numbers in T lie in the interval $(B/3, B/2)$. Further note that each number in S is even and that each number in T is odd. We claim that $\overline{\text{DISJ}_n} = 1$ exactly when $S \cup T$ has a subset adding to B. To see why, first observe that

1. Using numbers only from S cannot give a sum of B since B itself does not appear in S, and the sum of even two numbers from S exceeds B.
2. Using numbers only from T cannot give a sum of B since (1) B does not appear in T; (2) Any two numbers in T add to an even number greater than $B/2$, but B is odd; and (3) adding three or more numbers from T gives a sum greater than B.

Thus we see that if any subset of $S \cup T$ adds to B, then it must contain exactly one number from S and one from T. That is, it must be of the form $\{8n-2i, 4n+2j-1\}$. To add up to $12n - 1$, it must be the case that $i = j$. Hence such a subset exists if and only if there exists an $i \in [n]$ such that $x_i = y_i = 1$.

Now, as in Lemma 1, assume that there is a space $s(B)$ p-pass streaming algorithm for USS. Then there is a cost $(2p - 1)s(B)$ protocol for USS with p rounds, and hence by the above reduction, a cost $(2p - 1)s(12n - 1)$ protocol for $\overline{\text{DISJ}_n}$. By the lower bound for $\overline{\text{DISJ}_n}$, $(2p - 1)s(12n - 1) \in \Omega(n)$, and so $s(B) \in \Omega(B/p)$. □

We now show a space upper bound for USS, for large number of passes.

Theorem 2. *For every $s \leq B$, there is a randomized streaming algorithm for USS that makes s passes over the input, uses space $O(\frac{nB \log^2(nB)}{s})$, and on each input errs with probability at most $1/3$.*

Proof. The idea is to use the algorithm of [Kan10] for just one prime p. We will pick this prime randomly from a large enough range to ensure that the probability of success is high. We first briefly recapitulate Kane's algorithm.

Let $a_1, \ldots, a_n$ be the given set of numbers, and let A be the number of subsets of $\{a_1, \ldots, a_n\}$ that add to B. We want to determine whether $A = 0$. If $A = 0$, then $A = 0 \pmod p$ for all primes p. If $A \neq 0$, then $A \neq 0 \pmod p$ for all primes that do not divide A; the number of primes p such that $A = 0 \pmod p$ is fewer than $\log A \leq \log 2^n = n$.

The algorithm from [Kan10] proceeds as follows: Define

$$C = |B| + \sum_{i=1}^{n} |a_i| + 1.$$

Let $\mathcal{P}$ be the set of the first n primes beyond C. Compute $A \pmod p$ for each $p \in \mathcal{P}$. Clearly, $A = 0 \iff \forall p \in \mathcal{P} : A = 0 \pmod p$. So it suffices to show how to compute $A \pmod p$ for $p \in \mathcal{P}$. To do this, Kane establishes the following key lemma, which we also use.

Lemma 2 (Lemma 1 from [Kan10]). *For any prime $p > C$:*

$$\sum_{x=1}^{p-1} x^{-B} \prod_{i=1}^{n} (1 + x^{a_i}) \equiv -A \pmod{p}$$

This gives a space-efficient way of computing $A \pmod{p}$, for any fixed p: compute the left-hand-side above modulo p by sequentially accumulating the contributions from each $x \in \{1, \ldots, p-1\}$. This yields the logspace algorithm of [Kan10].

However, this approach seems to require multiple passes over the input, since for each $x \in \{1, \ldots, p-1\}$ we need all the input numbers, and furthermore, we need to compute $A \pmod{p}$ for each $p \in \mathcal{P}$.

To handle the second problem, we choose a single prime p *uniformly at random* from the first $3n$ primes beyond C. More precisely, we choose D such that there are at least $3n$ primes between C and D. Let $\mathcal{Q}$ be the set of primes between C and D. Now we pick a prime $p \in \mathcal{Q}$ uniformly at random. If $A = 0$, then $A = 0 \pmod{p}$. If $A > 0$, then $A = 0 \pmod{p}$ for at most n distinct primes p. Hence the probability that our randomly chosen prime p yields $A = 0 \pmod{p}$ is at most $1/3$. Thus it suffices to compute the left-hand-side of the expression in Lemma 2 for a single randomly chosen prime p. We are left with the problem of dealing with sequential accumulation, each x requiring all inputs.

Assume that p has been chosen. Define

$$f(x) = x^{-B} \prod_{i=1}^{n} (1 + x^{a_i}) \pmod{p}$$

$$\sigma(i, j) = \sum_{x=i+1}^{j} f(x) \pmod{p}$$

$$-A \equiv \sigma(0, p-1) \pmod{p}$$

$f(x)$ can be computed in 1 pass using $O(\log p)$ space. Hence $\sigma(i, j)$ can be computed in $j - i$ passes with $O(\log p)$ space. But it can also be computed in 1 pass with $O((j-i)\log p)$ space, by computing $f(x)$ for each $x \in [i+1, j]$ in parallel. In fact, we have a trade-off: for any $1 \leq s \leq j - i$, if s passes are allowed, then $\sigma(i, j)$ can be computed in $\frac{(j-i)}{s} \log p$ space.

We use this trade-off to compute $\sigma(0, p-1)$ in s passes. We first compute $K = \lceil \frac{p-1}{s} \rceil$. We then compute $\sigma(0, K)$ in the first pass, $\sigma(K, 2K)$ and hence $\sigma(0, 2K)$ in the second pass and so on. In s passes, we can obtain $\sigma(0, p-1)$, and we use $O(K \log p)$ space throughout. This works provided $s \leq p - 1$; since $p > B$, it works fo all $s \leq B$.

The kth prime is roughly $k \ln k \in O(k \log k)$. The prime we use, p, is at most as large as the $(C+3n)$th prime. Since $C \in O(nB)$, we see that $p \in O(nB \log(nB))$. Hence the space used is $O(\frac{nB \log^2(nB)}{s})$. □

3 Approximate USS in Streaming Model

As computing exact USS is provably hard (Theorem 1), the next natural question to ask is: can it be approximated? There is a classical approximation algorithm for the following approximation version of the Subset Sum problem: Given a set of numbers and a target B, let B^* be the largest value *smaller than* B expressible as a sum of a subset of the given numbers. Find a subset with sum in the range $[(1-\epsilon)B^*, B^*]$, for a given ϵ. (Note that B^* itself is not explicitly known.) It is known that this problem has a fully polynomial time approximation scheme (an algorithm with run time polynomial in n, B, $1/\epsilon$); see for instance [CLRS09]. This algorithm is one-pass and works even if the input data is given in binary. However, the space used is $O(n)$ even if the input is given in unary. We wish to approximate USS using a small number of passes on the input and using space polylogarithmic in the length of the input. We consider the following variant: For any ϵ and B and input stream $\tilde{a} = a_1, \ldots, a_n$ where each $a_i \in [B]$, we say that set $S \subseteq [n]$ is an *ϵ-approximator of B in $\tilde{a}$* if $\left(\sum_{i \in S} a_i\right) \in [B(1-\epsilon), B(1+\epsilon)]$. Given $\epsilon, B, \tilde{a}$, we want to decide whether there is an ϵ-approximator of B in $\tilde{a}$. We prove the following theorem:

Theorem 3. *There is a deterministic 1-pass streaming algorithm that on an input stream $\epsilon, B, \tilde{a}$, uses space $O(\frac{\log B}{\epsilon})$ and outputs 1 if and only if there exists an ϵ-approximator for B in the stream $\tilde{a}$.*

Proof. Consider the following algorithm $\mathcal{A}$:

Maintain a set of intervals T.
Initialise: $T \leftarrow \{[B(1-\epsilon), B(1+\epsilon)]\}$.
while End of stream not reached **do**
  $a \leftarrow$ Next number in stream.
  if $\exists$ interval $[\alpha, \beta] \in T$ such that $a \in [\alpha, \beta]$ **then**
    Output YES and halt.
  else
    $T' \leftarrow \{[\alpha, \beta], [\alpha - a, \beta - a] \mid [\alpha, \beta] \in T\}$;
    $T \leftarrow T'$.
    Merge overlapping intervals in T to get a set of pairwise disjoint intervals.
    (If $[a, b], [c, d] \in T$ and $a \leq c \leq b \leq d$, remove $[a, b], [c, d]$ and add $[a, d]$.)
  end if
end while

Before seeing why the algorithm is correct, we first consider the space analysis. Note that at the beginning of each iteration, T has a set of disjoint intervals and each interval has size at least $2B\epsilon$. The space required to store the endpoints of each interval is $O(\log B)$. There can be at most $B/(2B\epsilon)$ disjoint intervals from 1 to B, so at any given time, $|T| \leq \frac{1}{\epsilon}$. Since T' has two intervals for each interval of T, $|T'|$ is also $O(\frac{1}{\epsilon})$. So the space used is $O(\frac{\log B}{\epsilon})$.

We now show that $\mathcal{A}$ is correct; that is, $\mathcal{A}$ outputs YES if and only if there exists a subset of the input numbers that has sum in $[l, r]$. The intuition behind

the correctness is the following: We maintain the set of intervals T such that if any number in the union of the intervals in T is seen as input, then there indeed exists a subset that generates B. This is true in the beginning by the way we initialize T. When a number m is read, a copy of each interval in T is shifted down by m to create a new interval. So if a number in any of these new intervals is seen, then it can be combined with m to give a number in one of the older intervals. (The original intervals are also retained, so we can also not use m in creating a subset.) And this property is maintained by updating T with every number seen. Note that no interval in T gets deleted. Intervals in T only get merged into other intervals to become larger intervals and this does not affect the invariant property.

We now describe the proof more formally: For a set of intervals T, define $R(T) = \{a \mid \exists [\alpha, \beta] \in T : a \in [\alpha, \beta]\}$; $R(T)$ is the union of all the intervals in T. Let $l = B(1-\epsilon)$ and $r = B(1+\epsilon)$. Initially, $R(T) = \{a \mid l \leq a \leq r\}$.

$\Rightarrow$*:* Assume that $\mathcal{A}$ outputs YES. Let T_k denote the collection of intervals after reading k numbers from the stream. $\mathcal{A}$ accepts at a stage k when it reads a number $a_k \in R(T_{k-1})$. We show below, by induction on k, that if $a \in R(T_k)$, then there is a subset of $\{a_1, \ldots, a_k\} \cup \{a\}$ with sum in $[l, r]$. This establishes that the YES answers are correct.

In the beginning, T_0 is initialized to $\{[l, r]\}$. Thus $a \in R(T_0) \Rightarrow a \in [l, r]$.

Now assume that the property holds after reading $k-1$ numbers. That is, if $a \in R(T_{k-1})$, then there is a subset of $\{a_1, \ldots, a_{k-1}\} \cup \{a\}$ with sum in $[l, r]$.

If $a_k \in R(T_{k-1})$, the algorithm terminates here and there is nothing more to prove. Otherwise, $a_k \notin R(T_{k-1})$, and the algorithm goes on to construct T_k. The update sets $R(T_k)$ to contain all of $R(T_{k-1})$ as well as all numbers b such that $a_k + b \in R(T_{k-1})$. Now consider an $a \in R(T_k)$. If it also holds that $a \in R(T_{k-1})$, then we can pretend that a_k was not read at all, and using induction, pull out a subset of $\{a_1, \ldots, a_{k-1}\} \cup \{a\}$ with sum in $[l, r]$. If $a \notin R(T_{k-1})$, then $a_k + a \in R(T_{k-1})$. By induction, we have a subset of $\{a_1, \ldots, a_{k-1}\} \cup \{a_k + a\}$ with sum in $[l, r]$. Hence we have a subset of $\{a_1, \ldots, a_{k-1}, a_k\} \cup \{a\}$ with sum in $[l, r]$, as desired.

$\Leftarrow$*:* Let S, $|S| = k$, be the first subset of numbers in the input stream that has sum in $[l, r]$. That is,

- $S = \{a_{i_1}, a_{i_2}, \ldots, a_{i_k}\}$ for some $i_1 < i_2 < \ldots < i_k$,
- $\sum_{j=1}^{k} a_{i_j} = B - B\lambda$ for some $|\lambda| < |\epsilon|$, and
- there is no such subset in $a_1, \ldots a_{i_k - 1}$.

We will show that $\mathcal{A}$ outputs YES on reading a_{i_k}.

To simplify notation, let s_j denote a_{i_j}.

Observe that if a number a enters $R(T)$ at any stage, then it remains in $R(T)$ until the end of the algorithm. This is because an interval is deleted only when an interval containing it is added.

Now we observe the way T gets updated. After reading s_1, $R(T)$ will contain the intervals $\{[l, r], [l - s_1, r - s_1]\}$. (It may contain more numbers too, but

that is irrelevant.) After reading s_2, $R(T)$ will contain $\{[l, r], [l - s_1, r - s_1], [l - s_2, r - s_2], [l - s_1 - s_2, r - s_1 - s_2]\}$. Proceeding in this way, and using the above observation that $R(T)$ never shrinks, after reading $s_1, s_2, \cdots, s_{k-1}$, $R(T)$ will contain $[l - (s_1 + \cdots + s_{k-1}), r - (s_1 + s_2 + \cdots + s_{k-1})]$. But this interval is the following:

$$\begin{aligned}
&[(B(1-\epsilon) - (s_1 + \cdots + s_{k-1})), (B(1+\epsilon) - (s_1 + \cdots + s_{k-1}))] \\
&= [(B(1-\epsilon) - (B - B\lambda - s_k)), (B(1+\epsilon) - (B - B\lambda - s_k))] \\
&= [(s_k + B\lambda - B\epsilon), (s_k + B\lambda + B\epsilon)] \\
&= [(s_k - B(\epsilon - \lambda)), (s_k + B(\epsilon + \lambda))]
\end{aligned}$$

Since $\epsilon > 0$ and $|\lambda| < |\epsilon|$, $s_k \in [(s_k + B(\lambda - \epsilon)), (s_k + B(\lambda + \epsilon))]$. Hence $\mathcal{A}$ will output YES when s_k is read. □

The following lemma shows that the simple streaming algorithm discussed above is pretty much tight.

Lemma 3. *Let f be any real-valued function. If $f(2x) \log x \in o(x)$, then there is no randomized 1-pass streaming algorithm that ϵ-approximates USS and uses only $O(f(\frac{1}{\epsilon}) \log B)$ space.*

Proof. Assume to the contrary that $\mathcal{A}$ is a randomized 1-pass algorithm that ϵ-approximates USS and uses space $O(f(\frac{1}{\epsilon}) \log B)$. Choose $\epsilon = \frac{1}{2B}$. Now for this value of ϵ, and for every stream $\tilde{a}$, $\mathcal{A}$ will behave like an exact algorithm for USS. The lower bound from Lemma 1 now implies that $f\left(\frac{1}{\epsilon}\right) \log B \in \Omega(B)$, and hence $f(2B) \log B \in \Omega(B)$. But the last relation cannot hold if $f(2x) \log x \in o(x)$. □

4 Multiplicity USS (mUSS) and Monotone Circuits

In this section, we consider the monotone circuit complexity of USS. Without the monotone restrictions, it is known that USS is complete for the circuit class TC^0 ([EJT12]). However, in a very natural sense, Subset Sum is a monotone problem, and so we can consider monotone circuits for it. The encoding of the input becomes crucial for achieving monotonicty. We choose the following encoding:

For each positive integer B, the input consists of the frequency of each number in the stream in unary. An instance $w \in \{0,1\}^{B^2}$ consists of B blocks of B bits each. For each $k \in [B]$, if k occurs in the stream m_k times, then the kth block w_k has exactly m_k 1s; that is, $\sum_{j=1}^{B} w_{kj} = m_k$. Thus the input records the multiplicity of each number in $[B]$ (we assume that no multiplicity exceeds B).

Define the transitive relation $\preceq$: For $u = (u_1, u_2, \ldots, u_B), v = (v_1, v_2, \ldots, v_B)$ with $u_k, v_k \in \{0,1\}^B$, $u \preceq v$ if and only if $\forall k \in [B]$, $\sum_{j=1}^{B} u_{kj} \leq \sum_{j=1}^{B} v_{kj}$.

We define the multiplicity-USS problem, denoted as mUSS, and its approximation variant ϵ-mUSS, as follows.

$$\begin{aligned}
\mathsf{mUSS}(w,B)=1 \iff & \exists y=(y_1,y_2,\ldots,y_B): \\
& y_k \in \{0,1\}^B \ \ \forall k \in [B], \ \ y \preceq w, \text{ and} \\
& B = \left(\sum_{k=1}^{B} k \left(\sum_{j=1}^{B} y_{kj}\right)\right) \\
\epsilon\text{-}\mathsf{mUSS}(w,B)=1 \iff & \exists y=(y_1,y_2,\ldots,y_B): \\
& y_k \in \{0,1\}^B \ \ \forall k \in [B], \ \ y \preceq w, \text{ and} \\
& B(1-\epsilon) \leq \left(\sum_{k=1}^{B} k \left(\sum_{j=1}^{B} y_{kj}\right)\right) \leq B(1+\epsilon)
\end{aligned}$$

We call such a y a *witness* for (w, B). The vector y represents a subset of the multi-set represented by w such that the elements in y sum to B (or to a number within ϵ of B, respectively).

For example, for $B = 4$, the stream 1 3 2 2 1 4 3 can be encoded by any of the following strings (and by many more): 1100 1100 1100 1000, 1010 0101 0011 0010. Some witnesses for this instance are 1100 1000 0000 0000 (use two 1s and a 2), 0100 0000 0001 0000 (use a 1 and a 3), 0000 0000 000 1000 (use the 4).

Fact 4. mUSS *is a monotone function, i.e. for each positive integer B, and for each $u = (u_1, u_2, \ldots, u_B)$, if* $\mathsf{mUSS}(u, B)= 1$*, and if $v = (v_1, v_2, \ldots, v_B)$ is obtained from u by changing some* 0*s to* 1*s, then* $\mathsf{mUSS}(v, B)= 1$*.*
Similarly, for each ϵ and B, ϵ-mUSS*is a monotone function.*

It has been known for over three decades ([MS80]) that USS is in nondeterministic logspace; hence USS reduces to the problem Reach defined below:

> Given: a layered directed acyclic graph G, two designated nodes s, t
> Output: 1 if there is a path from s to t in G, 0 otherwise.

It is well-known that Reach has monotone circuits of depth $O(\log^2 n)$, where n is the number of vertices in the input instance. (This follows from the construction of [Sav70]. See for example [AB09].) We show that with the encoding described above, exact and approximate versions of mUSS reduce to Reach via monotone projections, and hence have small depth monotone circuits.

Theorem 5. *For every positive integer B,* $\mathsf{mUSS}(\cdot, B)$ *and* ϵ-$\mathsf{mUSS}(\cdot, B)$ *have monotone circuits of depth $O(\log^2 B)$.*

Proof. (Sketch) We prove this by reducing an instance of mUSS into an instance of Reach via a monotone projection.

For every integer B, and given $w \in \{0,1\}^{B^2} = (w_1, w_2, \ldots, w_B)$ we create a graph with $B^2 + 1$ layers. The zero-th layer consists of the source vertex and the other B^2 layers have $(B + 1)$ vertices each. We further partition B^2 layers into B blocks of B consecutive layers each.

Let $v^i_{j,k}$ denote the i-th vertex in the layer j in the block k. Intuitively, each layer corresponds to a bit position in the input string. We add edges in order to

ensure that a vertex $v^i_{k,j}$ is reachable from the source vertex if and only if the stream corresponding to the first $k-1$ blocks of w and j bits from the kth block has a subset that adds to i.

If after reading l bits of w there is a subset that adds to i then this subset continues to exist even after reading more bits. To capture this phenomenon, we add *horizontal edges* from every vertex v in layer l to the copy of v in layer $l+1$.

If the bit $w_{kj} = 1$, then using this copy of k, for each existing subset sum s, the subset sum $s + k$ can also be created. To capture this, we include *slanted* edges from each $v^i_{j,k}$ to $v^{i+k}_{j+1,k}$.

Thus, there is a path form the source to vertex $v^i_{B,B}$ exactly when there is a subset that sums to i. By connecting $v^i_{B,B}$ for appropriate i to a new target node t, we reduce mUSS or ϵ-mUSS to Reach. □

5 Circuits with ε-Approximators for mUSS as Gates

We now examine the power of ϵ-approximators for mUSS when used as a primitive to compute other functions. In [Nis94], Nisan showed that any circuit for DISJ_n using linear threshold gates requires $\Omega(n/\log n)$ gates. We introduce a new kind of gate, an ApproxUSS gate, that we show is at least as powerful as a Threshold or Majority gate, and show that any circuit that uses ApproxUSS gates to compute Disjointness needs size $\Omega(n/\log n)$. However, we do not know whether linear threshold gates can simulate ApproxUSS gates with at most sub-logarithmic blowup in the number of gates or vice versa.

We define approximate USS gates, denoted ApproxUSS, as gates that solve the ϵ-mUSS problem defined in Section 4. An $\mathsf{ApproxUSS}_{\epsilon,B}$ gate takes a bit string x of length B^2 as input, and outputs 1 exactly when $\epsilon\text{-mUSS}(x, B) = 1$.

While it is trivial to see that majority can be computed with a single call to an oracle for mUSS, it is not immediately clear that oracle access to ϵ-mUSS when $\epsilon > 0$ is also sufficient. We show that this is indeed the case, by showing that ApproxUSS gates are at least as powerful as standard threshold gates. Specifically, we show that an ApproxUSS gate can simulate majority with only a polynomial blowup in the number of wires.

Lemma 4. *The MAJ_{2n+1} function can be computed by an $\mathsf{ApproxUSS}_{\varepsilon,B}$ gate with $B = O(n^3)$ and a suitable non-zero value for ε.*

On the other hand, it is not known whether ApproxUSS, for $\epsilon \neq 0$, can be decided with a single oracle call to majority. It is conceivable that demanding a YES answer for a wider range of inputs (the approximation) makes the problem harder. It is therefore interesting to examine the power of circuits using ApproxUSS gates. We follow this thread below.

The communication problem ccApproxUSS corresponding to ApproxUSS can be described as follows. Let $S \subseteq [B^2]$. Both Alice and Bob know S, B and ϵ. Alice knows the bits x_i for $i \in S$, and Bob knows the remaining bits of x. They must decide whether $\mathsf{ApproxUSS}_{\epsilon,B}(x) = 1$, that is, whether $\epsilon\text{-mUSS}(x, B) = 1$.

In Theorem 3 we proved that for every ϵ, there is a one-pass streaming algorithm that ϵ-approximates USS using space $O((\log B)/\epsilon)$. The algorithm works for every possible ordering of the numbers in the input stream. This implies that there is a $O((\log B)/\epsilon)$ bit one-round protocol for ccApproxUSS for worst case partitioning of the input (for every $S \subseteq [B^2]$). (The string x determines the multi-set of numbers. For any partition of x, Alice and Bob can construct a stream of numbers forming this multi-set, where Alice has the initial part of the stream and Bob has the latter part. Treat the indices in B^2 as pairs k, j where k is the block number and j is the index within the block. Alice includes in her stream a copy of k for each bit $x_{kj} = 1$ in her part. Bob does the same.) Therefore, using an argument similar to that of [Nis94], we can prove the following lemma:

Lemma 5. *Let C be a circuit that computes $DISJ_n$ using s $\mathsf{ApproxUSS}_{\varepsilon,B}$ gates, where $\epsilon \in \Theta(1)$, and the value of B at each $\mathsf{ApproxUSS}_{\varepsilon,B}$ is bounded above by a polynomial in n. Then $s \in \Omega(n/\log n)$.*

Proof. Let C be such a circuit, with s ApproxUSS gates. Let t denote the maximum value of B in any of the gates. We use C to obtain a protocol for DISJ_n as follows. Alice and Bob evaluate C bottom-up, reaching a gate only after all its children have been evaluated. At each ApproxUSS gate, we know that $\log B \in O(\log t) \subseteq O(\log n)$. When an ApproxUSS gate has to be evaluated, an efficient protocol of $O((\log t)/\epsilon)$ bits for ccApproxUSS is invoked with the appropriate partition of the inputs of the gate. As there are s ApproxUSS gates, the entire protocol for computing DISJ_n uses $O(((\log t)/\epsilon) \times s)$ bits of communication. However, we know that any protocol for DISJ_n requires $\Omega(n)$ bits of communication ([KS92, Raz92, BYJKS04]). Hence, $s \log t = \Omega(\epsilon n)$. By assumption, $\epsilon \in \Theta(1)$, and $\log t = O(\log n)$. Hence $s = \Omega(n/\log n)$. □

6 Discussion

We now discuss a few aspects of our results and some open questions.

- The upper and lower bounds from Theorems 1 and 2, for s-pass streaming algorithms for USS, do not quite match. There is a gap of $n \log^2(nB)$. Closing this gap will be interesting.
- If the multiplicities of all numbers are restricted to be between $\{0, 1\}$, then the problem does not become easier. In fact, our lower bound proof for USS in Section 2 (Theorem 1) generates such instances.
- We know that USS is in TC^0 [EJT12] and we have proved that mUSS is in monotone NC^2. It is known that there exists a monotone formula of polynomial size which cannot be computed by constant depth polynomial sized monotone threshold circuits [Yao89]. However, the question of whether monotone TC^0 is contained in monotone NC^1 is open (see for instance [Ser04]). Majority is known to be in monotone NC^1 [Val84]. And it is easy to observe that majority reduces to USS. In the light of these results, the question of whether mUSS is contained in monotone NC^1 is interesting.

References

[AB09] Arora, S., Barak, B.: Computational Complexity - A Modern Approach. Cambridge University Press (2009)

[AMS99] Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. Journal of Computer and System Sciences 58(1), 137–147 (1999)

[BYJKS02] Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D.: Information theory methods in communication complexity. In: Proceedings of the 17th Annual IEEE Conference on Computational Complexity, pp. 93–102 (2002)

[BYJKS04] Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D.: An information statistics approach to data stream and communication complexity. Journal of Computer and System Sciences 68(4), 702–732 (2004); (preliminary verion in FOCS 2002)

[CLRS09] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press (2009)

[EJT10] Elberfeld, M., Jakoby, A., Tantau, T.: Logspace versions of the theorems of Bodlaender and Courcelle. In: FOCS, pp. 143–152, (2010); ECCC TR 62 (2010)

[EJT12] Elberfeld, M., Jakoby, A., Tantau, T.: Algorithmic meta theorems for circuit classes of constant and logarithmic depth. In: 29th STACS: Leibniz International Proceedings in Informatics (LIPIcs), vol. 14, pp. 66–77 (2012); See also ECCC TR 128 (2011)

[GJ79] Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)

[Kan10] Kane, D.M.: Unary subset-sum is in logspace. CoRR (arxiv), abs/1012.1336 (2010)

[KNR95] Kremer, I., Nisan, N., Ron, D.: On randomized one-round communication complexity. Computational Complexity 8, 596–605 (1995)

[KS92] Kalyanasundaram, B., Schnitger, G.: The probabilistic communication complexity of set intersection. SIAM Journal on Discrete Mathematics 5(4), 545–557 (1992)

[MS80] Monien, B., Sudborough, I.H.: The interface between language theory and complexity theory. In: Book, R.V. (ed.) Formal Language Theory. Academic Press (1980)

[Nis94] Nisan, N.: The communication complexity of threshold gates. In: Proceedings of Combinatorics, Paul Erdos is Eighty, pp. 301–315 (1994)

[Raz92] Razborov, A.A.: On the distributional complexity of disjointness. Theoretical Computer Science 106(2), 385–390 (1992)

[Sav70] Savitch, W.: Relationships between nondeterministic and deterministic tape complexities. Journal of Computer and System Sciences 4(2), 177–192 (1970)

[Ser04] Servedio, R.A.: Monotone boolean formulas can approximate monotone linear threshold functions. Discrete Applied Mathematics 142(1-3), 181–187 (2004)

[Val84] Valiant, L.G.: Short monotone formulae for the majority function. Journal of Algorithms 5(3), 363–366 (1984)

[Yao89] Yao, A.C.: Circuits and local computation. In: Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing, STOC, pp. 186–196. ACM, NY (1989)

On the Advice Complexity of Tournaments

Sebastian Ben Daniel*

Dept. of Computer Science, Ben-Gurion University, Beer Sheva, Israel

Abstract. Advice complexity, introduced by Karp and Lipton, asks how many bits of "help" suffice to accept a given language. This is a notion that contains aspects both of informational and computational complexity, and captures non-uniform complexity. We are concerned with the connection between this notion and P-selective sets. The main question we study in our paper is how complex should the advice be as a function of the power of the interpreter, from the standpoint of average-case complexity. In the deterministic case, Ko proved that quadratic advice suffices, and Hemaspaandra and Torenvliet showed that linear advice is required. A long standing open problem is the question how to close this gap. We prove that in the probabilistic case linear size advice is enough, as long as this advice depends on the randomness. This is the first sub-quadratic result for the class P-sel for bounded-error probabilistic machines. As a consequence, several Karp-Lipton type theorems are obtained. Our methods are based on several fundamental concepts of theoretical computer science, as hardness amplification and Von Neumann's *Minimax* theorem, and demonstrate surprising connections between them and the seemingly unrelated notion of selectivity.

1 Introduction

Can we accept more languages in a certain model of computation when given somewhat more of a certain resource? This fundamental question has challenged many complexity theorists since Hartmanis and Stearns [19] seminal paper regarding deterministic Turing machines. Time and space hierarchy theorems have long been the main tools for separation of classes in Complexity theory. We address this question, where the resource is the non-uniformity needed for certain complexity classes. Can we give lower bounds against this resource? Can we reduce the amount of non-uniformity needed in respect to the trivial results? We emphasize the importance of both questions: strong answers to the former can yield separation between complexity classes, while to the latter can facilitate the decision procedures required for the sets in question.

1.1 Selectivity and Membership-Comparability

Partial information classes relate to the complexity of approximating the decisions problems. If we consider membership for tuples of words, the meaning of

* The author is partially supported by the Frankel Center for Computer Science.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 470–481, 2012.

approximate can be described as reducing the number of exponentially (in the size of the tuple) many possible outcomes for the characteristic function of the set. Algorithms that get a set of problem instances arise both in theory and in practice. Even if the problem itself is not one of multiple instances, they often arise via the use of divide-and-conquer methods or self-reductions.

A P-selective set is a set which can be decided in polynomial time by some algorithm A, which takes as input a pair (x, y) of strings and selects one of them as "more likely" to belong to the set, in the sense that if exactly one of the two strings belongs to the set, then it is guaranteed that this one is selected by A. Selman [37] introduced P-selective sets as the polynomial time analog of semi-recursive sets [26] and used them to study polynomial time reducibilities on NP. Since Selman's first paper on P-selective sets, these sets and their nondeterministic counterparts have found application in several areas of complexity theory. They have played an important role in understanding the structure of NP, and were studied in [38, 41, 12]. The interested reader is referred to Hemaspaandra and Torenvliet's book [24] for an extensive introduction.

Ogihara [36] introduced the notion of membership comparable sets, as an extension of P-selective languages. Informally, for a membership comparable set there exists a polynomial time algorithm that when given as input a collection of sufficiently many strings, excludes one of the incorrect possible values of the characteristic function. Ogiahara [36] and Amir, Beigel, and Gasarch [2] proved that, for constant number of queries, the class is a subset of P/poly. They also showed that for polynomial number of queries, the class is a subset of Σ_2/poly. Beigel [6], extending on the first result, has shown that in the logarithmic case, the class has polynomial size circuits.

These properties encourage the study of various questions: how simple are P-selective sets? How robust are P-selective sets? In particular, the following question has received a lot of attention in literature [17, 16, 20, 21, 23, 22, 28], and is the starting point of our paper:

Question 1. *What is the advice complexity of a (deterministically or nondeterministically) polynomial time selective language?*

Ko [28] proved that P-selective sets have quadratic advice complexity: for each P-selective set L, there is a polynomial time Turing machine that accepts L with only n^2 bits of advice (for input of length n). Can Ko's quadratic-advice be replaced by a linear-size advice (possibly by increasing the power of the interpreter)? This was first achieved by Hemaspaandra et al. [20], who showed that with a PP interpreter, linear advice is enough. Hemaspaandra and Torenvliet [23] extend this result to NP interpreters.In fact, they prove that P-sel $\subseteq$ NP/$n+1$, yet P-sel $\not\subseteq$ NP/n. Thus, it follows that P-sel $\not\subseteq$ P/n. Also, if P-sel $\notin$ P/linear, then P $\neq$ NP. Note that Hemaspaandra and Torenvliet's upper and lower bounds (for NP) on the advice complexity for sets is tight. However, as discussed above, the bounds on the advice complexity for sets in P-sel when the advice interpreter is deterministic are not: quadratic versus linear. Thus, understanding the advice complexity of P-sel, might shed light on the P vs. NP question. Thakur [40] has shown that no relativizable proof can show linear advice in the deterministic

case. Since showing that linear advice is not enough separates P from NP, such proof cannot be relativizable as well [39].

Our Approach. The main essence of our technique is a series of *quid pro quo* arguments, exchanging complexity measures to reduce the required resources. First, we give a simple proof which shows that for **every** distribution on the inputs, the average-case advice complexity of P-sel is linear. Then, we show that for the uniform distribution and balanced sets, we can significantly increase the success probability of the algorithm. Finally, we argue that there exists a **single** algorithm which works for **every** distribution, in the expense of the advice being dependent on the randomness of the algorithm.

1.2 Results

P-Selective and Membership-Comparable Sets. We show that P-selective sets have probabilistic polynomial time algorithms with linear advice, which depends on the randomness. In the case of average-case complexity, we show that the success probability can be increased to $1-1/\log n$ and still have linear advice, if we restrict ourselves to the uniform distribution and balanced sets. If a set L is in k-mc, then it can be decided with a deterministic polynomial time algorithm (in n and 2^k), with advice of length $\mathcal{O}(n^2 \cdot 2^k)$, which extends the results of [36], giving a continuous trade-off for $k > \log n$. For probabilistic machines, we can achieve linear advice and bounded error probability in polynomial time (provided we have random access to the advice), which extends the result of [23].

BPP-Selective Sets. We define a new classes of problems, BPP-sel, k-BPPmc, which are probabilistic analogs of P-sel and k-mc respectively. A "natural" property of this classes is that they extended the well known connection between P-selective and tournaments to generalized tournaments [32]. For formal definitions and basic theorem about this kind of tournaments, we refer the reader to Moon's book [33]. We show them to have polynomial size circuits. We also show that if SAT is BPP-selective then NP = RP.

Overview. In Section 3, we analyze the average-case complexity of the sets. Next, in Section 4, we building on the previous section, we show that membership-comparable sets have low probabilistic advice complexity. We conclude the paper with some suggested extension to our work.

2 Preliminaries

We assume basic familiarity with complexity classes such as P, BPP, NP, and their exponential-time versions. When we write that a language $L \subseteq \Sigma^*$ has a DTime(f(n)) machine M, we mean that there exists a deterministic Turing machine such that for every input x, halts in $f(|x|)$ steps and correctly decides L. We denote the characteristic function of a language $L \subseteq \Sigma^*$ by χ_L. It is

convenient to often refer to languages via their characteristic function, if it is not clear from the context, we explicitly mention this. We say that a complexity class C is **closed** under reduction $\leq$ if $\forall B \in \mathcal{C}$ and $\forall A \subseteq \Sigma^*$ the following holds: $A \leq B \Rightarrow A \in \mathsf{C}$.

Definition 1 ([37]). *A set L is P-selective if there exists a polynomial time computable function f such that for each $x, y \in \Sigma^*$*

1. *$f(x, y) \in \{x, y\}$ and*
2. *$\{x, y\} \cap L \neq \emptyset \Rightarrow f(x, y) \in L$.*

The class of all P-selective sets is denoted by P-sel.

Definition 2. *A set L is k-membership-comparable if there exists a polynomial time function f s.t. for all $x_1, \ldots, x_k \in \Sigma^*$, the function $f(x_1, \ldots, x_k)$ outputs $(b_1, \ldots, b_k) \in \{0, 1\}^k$ such that $b_i \neq \chi_L(x_i)$ for at least one $1 \leq i \leq k$. The class of all k-membership comparable sets is denoted by k-mc.*[1]

Wang [46] extended the notion of selectivity to include probabilistic unbounded-error selector functions. Following this same path we include bounded-error probabilistic selector functions.

Definition 3 (BPP-selective). *A set L is BPP-selective if there exists a probabilistic polynomial time computable function f such that, for every $x, y \in \Sigma^*$*

1. *$f(x, y) \in \{x, y\}$ with probability one, and*
2. *if $\{x, y\} \cap L \neq \emptyset$ then $\Pr[f(x, y) \in L] \geq 2/3$ (the probability is taken over the random coins of f).*

The class of all BPP-selective sets is denoted by BPP-sel.

By a BPP algorithm we mean a polynomial time probabilistic Turing machine that accepts every input with probability at least $2/3$ or at most $1/3$. For a BPP algorithm A, we will denote by $L(A)$ the language decided by A, i.e., $L(A) = \{x | \Pr[A \text{ accepts } x] > 2/3\}$.

We define what it means for a language to be computable in BPP with advice. The model of advice we use is the model defined by Trevisan and Vadhan [44], where the advice depends on the randomness. We say $L \in \mathsf{BPP}//\mathrm{a(n)}$, if there is a probabilistic machine M such that for any random string r used by the machine on inputs of length n, there is a string $f(r)$, where $|f(r)| = a(n)$, such that given as advice on that random string, satisfies the following: for each x of length n, if $x \in L$, M accepts with probability at least $2/3$ over r given $f(r)$; when $x \notin L$, M rejects with probability at least $2/3$ over r given $f(r)$.

An average case complexity class consists of pairs, called distributional problems. Each such pair consists of a decision problem and a probability distribution on the problem instances. It is convenient to refer to the probability measure as an ensemble of probabilities over $\{0, 1\}^n$. The uniform distribution will play an important role, and we denote it by $\mathcal{U}$.

[1] It is easy to verify that P-sel $\subseteq$ 2-mc: if w.l.o.g the selector return x_1, then $(0, 1)$ is an impossible answer.

Definition 4 (Distributional Problem). *A distributional decision problem is a pair (L, μ), where $L : \{0,1\}^* \to \{0,1\}$ and μ is an ensemble of distributions $\{\mu_n\}_{n\in\mathbb{N}}$ over $\{0,1\}^n$.*

Definition 5 (Average-case Complexity). *Let C be a class of algorithms (e.g., polynomial time Turing machines) and (L, μ) be a distributional problem. We say that $(L, \mu) \in Avg_\epsilon\mathsf{C}$ if there is an algorithm $g \in \mathsf{C}$ such that for every large enough n, $\Pr_{x\sim\mu_n}[L(x) \neq g(x)] \leq \epsilon$.*

3 Average-Case Complexity of Membership Comparable Sets

In this section, we show that given a distribution ensemble, for any membership-comparable set, there a exists an polynomial algorithm and an advice of size proportional to the arity of the comparability function, which computes the set on average. Following ideas from [8] we are able to show a simple proof for the next theorem.

Theorem 1. *Let $L \in Avg_\epsilon$ k-mc. For every distribution ensemble $\mu = \{\mu_n\}_{n\in\mathbb{N}}$ there exists a deterministic polynomial time algorithm which decides L, with probability $(2^{k-1})(1-\epsilon)/(2^k-1)$ over μ_n, and takes $(k-1)\cdot n + \log k + \mathcal{O}(1)$ bits of advice. Furthermore, If $L \in$ P-sel the algorithm never errs.*

Proof. Let $\mathcal{A}$ be the algorithm which witnesses that $L \in Avg_\epsilon$ k-mc. For every distribution μ_n we construct $\mathcal{A}_\mu$ such that the probability according to μ_n that $\mathcal{A}_\mu$ agrees with L is at least $2^{k-1}\epsilon/(2^k-1) = 1-\epsilon'$. For the construction of $\mathcal{A}_\mu$ we use constants $q_0, \ldots, q_k$ such that $1 = q_0 \geq q_1 \geq \cdots \geq q_k$. These constants will be specified later. We also use the following random variables $Z_1, \ldots, Z_k$ defined as follows. We pick inputs $x_1, \ldots, x_k$ according to the distribution μ_n^k. Let $\sigma_1, \ldots, \sigma_k$ be the output of $\mathcal{A}(x_1, \ldots, x_k)$. Finally, for $1 \leq i \leq k$, we define $Z_i = T$ if $\sigma_i = L(x_i)$ and $Z_i = F$ otherwise.

By the correctness of $\mathcal{A}$, $\Pr[Z_1 = \cdots = Z_k = T] \leq \epsilon = q_k$, where the probability is taken over the choice of the inputs according to the distribution μ_n^k. Since $\Pr[Z_1 = \cdots = Z_k = T] = \prod_{i=1}^k \Pr[Z_i = T | Z_1 = \cdots = Z_{i-1} = T]$, thus, there must be an index i, where $1 \leq i \leq k$, such that $\Pr[Z_i = T | Z_1 = \cdots = Z_{i-1} = T] \leq q_i/q_{i-1}$. Let i be the least such index. Let $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k)$ be a fixed tuple which achieves this probability, that, together with their membership in L, i, and the most frequent answer for this coordinate, will be the advice of length $1 + \log k + (n+1)\cdot(k-1) = \mathcal{O}(nk)$ for $\mathcal{A}_\mu$. That is, $\Pr[Z_j = T | Z_1 = \cdots = Z_{j-1} = T] \geq q_j/q_{j-1}$ for $1 \leq j \leq i-1$. In particular, $p \stackrel{\text{def}}{=} \Pr[Z_1 = \cdots = Z_{i-1} = T] \geq (q_1/q_0)\cdot(q_2/q_1)\cdot\ldots\cdot(q_{i-1}/q_{i-2}) = q_{i-1}$.

Intuitively, $\mathcal{A}_\mu$ will use the fact that with a noticeable probability it can take the i-th output of $\mathcal{A}$ and invert it to obtain a correct output for the i-th input (assuming they are distributed according to μ_n).

Construction: Given x, $\mathcal{A}_\mu$ executes $\mathcal{A}(x_1,\ldots,x_{i-1},x,x_{i+1},\ldots,x_k)$=$\sigma_1,\ldots,\sigma_k$. If $\sigma_j = L(x_j)$ for $1 \le j \le i-1$, then $\mathcal{A}_\mu$ outputs $\overline{\sigma}_i$. Otherwise, $\mathcal{A}_\mu$ outputs the most frequent answer for this coordinate (which is also a part of the advice).

Next we claim that if the input x is distributed according to μ_n, then the error of $\mathcal{A}_\mu$ is at most $1-\epsilon'$. Algorithm $\mathcal{A}_\mu$ is correct in two cases: (1) $\sigma_j = L(x_j)$ for $1 \le j \le i-1$ and $\sigma_i \neq L(x)$; by our choice of i, this happens with probability at least $p \cdot (1 - q_i/q_{i-1})$ and (2) $\sigma_j \neq f(x_j)$ for some $1 \le j \le i-1$ and $b = L(x)$; this happens with probability $0.5(1-p)$. All together, the success probability of $\mathcal{A}_\mu$ is $p \cdot (1 - q_i/q_{i-1}) + 0.5(1-p)$, where the probability is taken over the choice of the inputs x according to the distribution μ_n, the choice of the other $k-1$ inputs for $\mathcal{A}$ according to μ_n^{k-1}, and the choice of b. Since $p \ge q_{i-1}$ and by choosing q_i, q_{i-1} such that $q_i/q_{i-1} \le 0.5$, the success probability is at least

$$q_{i-1} \cdot (1 - q_i/q_{i-1}) + 0.5(1 - q_{i-1}) = 0.5 + 0.5q_{i-1} - q_i. \tag{1}$$

We choose $q_0 \stackrel{\text{def}}{=} 1$ and $q_i \stackrel{\text{def}}{=} \frac{2^i-1}{2^{i-1}}\epsilon' - (1 - \frac{1}{2^{i-1}})$ for $1 \le i \le k$. Notice that $0.5 + 0.5q_{i-1} - q_i = 1 - \epsilon'$ for $1 \le i \le k$, and that

$$q_k = \frac{2^k-1}{2^{k-1}}\left(\frac{1}{2} - \frac{1/2 - 2^{k-1}}{2^k - 1}\right) - (1 - \frac{1}{2^{k-1}}) = \epsilon.$$

With our choice for q_i and (1) we have that $q_i/q_{i-1} = \frac{(2^{i-1}-0.5)\epsilon' - 2^{i-2} + 0.5}{2((2^{i-2}-0.5)\epsilon' - 2^{i-3}+0.5)} \le 0.5$ for $\epsilon \le 0.5$, thus the success probability of $\mathcal{A}_\mu$ is at least $1-\epsilon'$. The proof of the furthermore part appears in the final version of the paper.

3.1 Hardness Amplification

In this part, we show how to amplify the constant probability algorithm we showed to exist, to succeed with probability $1 - 1/\log n$ on $\mathcal{U}$ for balanced sets, while increasing the advice by a constant factor. In order to achieve this, we implement hardness amplification techniques that have been previously showed to work within NP, but in the opposite direction.

The standard technique for hardness amplification is the Yao's famous XOR lemma; this method cannot work for P-selective sets because the class is not closed under XOR as proved in [25]. However, P-sel is closed under positive reductions [11], which allows us to rely on the work initiated by O'Donnel [35], and later improved by [42, 43, 13], on hardness amplification within NP. As in these papers, we want to achieve hardness against semi-uniform machines like BPP//linear, i.e., uniform classes with advice where the non-uniformity is small.

Lemma 1 (advice efficient version of the Hard-core lemma [42]). *Let $\mathcal{C}$ be a distribution of circuits samplable in time t, $f : [N] \to \{0,1\}$ be a function. and let γ, ϵ, δ be such that for every subset $H \subseteq [N], |H| = \delta N$, we have $\Pr_{C\sim\mathcal{C}}[\Pr_{x\in H}[f(x) = C(x)] \ge \frac{1}{2} + \epsilon] \ge \gamma$. Then, there is a distribution $\mathcal{C}'$ samplable in time $t \cdot \text{poly}(1/\epsilon, 1/\delta)$ such that $\Pr_{C\sim\mathcal{C}'}[\Pr_{x\in[N]}[f(x) = C(x)] \ge 1-\delta] \ge \gamma^{\text{poly}(1/\epsilon,1/\delta)}$.*

The conclusion of this lemma can formulated in the following form: There exists a probabilistic algorithm which produces a list of $(1/\gamma)^{\mathrm{poly}(1/\epsilon,1/\delta)}$ circuits such that with high probability one of them solves f on $(1-\delta)$ fraction of the inputs.

Theorem 2 ([35]). *For every $\epsilon, \delta > 0$ there is a $k = \mathrm{poly}(1/\epsilon, 1/\delta)$ and a function $g : \{0,1\}^k \to \{0,1\}$ such that the following holds. Let $f : \{0,1\}^n \to \{0,1\}$ be a function ϵ/k closed to balanced[2], $H \subset \{0,1\}^n$ be of density δ, $b_1^f, \ldots, b_k^f : \{0,1\}^n \to \{0,1\}$ be independent random functions such that $b_i^f(x) = f(x)$ for $x \notin H$ and $b_i^f(x)$ outputs a random bit if $x \in H$. Then the distributions $x_1, \ldots, x_k, g(b_1^f(x_1), \ldots, b_k^f(x_k))$ and $x_1, \ldots, x_k, r$ have statistical distance at most 2ϵ, where the x_i are uniform in $\{0,1\}^n$ and r is uniform in $\{0,1\}$.*

The second step is the following statement, which is the reduction to the information theoretic case, and is based on the existence of an appropriate function we can use instead of the XOR function Theorem 2 .

Lemma 2 ([42]). *let $g : \{0,1\}^k \to \{0,1\}$ be as in Theorem 2, $f : \{0,1\}^n \to \{0,1\}$ be ϵ/k close to balanced and A be a polynomial time algorithm such that $\Pr[A(x_1, \ldots, x_k) = g(f(x_1), \ldots, f(x_k))] \geq \frac{1}{2} + 3\epsilon$. Then there is a polynomial time samplable distribution of circuits $\mathcal{C}$ such that for every set $H \subset \{0,1\}^n$ of density δ we have $\Pr_{C \sim \mathcal{C}}[\Pr_{x \in H}[C(x) = f(x)] \geq \frac{\epsilon}{2\delta k}] \geq \frac{\epsilon}{\delta k^2 \cdot 2^{2k-1}}$.*

In the proof of Lemma 2, Trevisan treats the algorithm A, as an oracle solving f, i.e., if A is a probabilistic algorithm with advice, the only difference will be that the resulting algorithm solving f on a significant larger fraction of the input will also require that advice. Finally, putting it all together,

Lemma 3. *Let $g : \{0,1\}^k \to \{0,1\}$ be as in Theorem 2, $f : \{0,1\}^n \to \{0,1\}$ be ϵ/k close to balanced and A be a polynomial time algorithm with ℓ length advice, such that $\Pr[A(x_1, \ldots, x_k) = g(f(x_1), \ldots, f(x_k))] \geq \frac{1}{2} + 3\epsilon$. Then there is a polynomial time samplable distribution of circuits $\mathcal{C}$ such that $\Pr_{C \in \mathcal{C}}[\Pr_{x \in H}[C(x) = f(x)] \geq 1 - \delta] \geq 2^{-\mathrm{poly}(1/\epsilon, 1/\delta)}$.*

Putting all together we get the following

Theorem 3. *There exists a constant $c > 0$ such that every balanced language L in* P-sel*, can be decided on average with a probabilistic polynomial time, (regular) linear length advice, and logarithmic length advice depending on the randomness. Thus algorithm which succeeds on a $1 - 1/\log^c n$ fraction of the inputs.*

4 Games, Strategies, and Advice-Efficient Version of the Minimax Theorem

Building on the average-case result from Section 3, in this section we construct a randomized algorithm with advice for every set in P-sel and k-mc. The natural

[2] Closed to balanced refers to the absolute distance from 1/2 of (YES instances/NO instances).

path to take is to describe a random strategy using the "hard" direction of the minmax principle. However, this usually gives an exponential-size strategy (in the length of the input). This is useful in models for which the size of the program can grow fast and the complexity measure is not affected by this growth, e.g; communication complexity. Our complexity measure is a polynomial time machines with linear advice, which allows only linear-size "growth" in the program length. In order to overcome this, we will use an advice-efficient version of the minmax theorem, introduced in [1, 31] for general measures.

Theorem 4 (Minimax [34]). $\min_p \max_j \sum_i p(i)M_{ij} = \max_q \min_i \sum_j q(i)M_{i,j}$.

Definition 6. *A mixed strategy is ℓ-uniform if it chooses uniformly from a multiset of ℓ pure strategies.*

Lipton and Young [29] showed that for ℓ proportional to the logarithm of the number of pure strategies available to the opponent, each player has a near-optimal ℓ-uniform strategy.

Definition 7 ([29]). *Fix a finite class P of programs, a finite class I of inputs, and a function $M : P \times I \to \{0, 1\}$ (where $M_{i,j}$ is 1 if the computation of machine i on input j errs and 0 otherwise). The (unlimited) randomized complexity of M is $\min_p \max_{j \in I} \sum_i p(i)M_{i,j}$, where p ranges over the probability distributions on P, i.e., the cost of the best program on the worst input. The (unlimited) distributional complexity of machine M is $\max_q \min_{i \in I} \sum_j q(j)M_{i,j}$, where q ranges over the probability distributions on I, i.e., the worst distribution on the best program. The program/input game for M is the two-player zero-sum game given by $M_{i,j}$ for $i \in P$ and $j \in I$.*

As Yao [47] observed, von Neumann's theorem applied to the program/input game implies that the unlimited randomized complexity and the unlimited distributional complexity are equal to $\nu(M)$.

Corollary 1 ([29]). *For any $c > 0$ and $k > \ln(|I|)/2\epsilon^2$, the k-uniform randomized complexity of M exceeds the unlimited randomized complexity by less than ϵ.*

Recall that our complexity measure is polynomial time Turing machines, with linear advice. Applying the above corollary to our measure, we obtain the following,

Theorem 5. *If $L \in$ 2-mc, then $L \in$ BPP//linear. Furthermore, the machine needs only $\log n + \mathcal{O}(1)$ random coins on an n-bit input. If $L \in$ P-sel then the probabilistic machine never errs.*

Proof. By Theorem 1, for every distribution μ there exists a polynomial time deterministic algorithm A_μ which computes L correctly with probability at least $2/3$ over the inputs according to μ. Yao's version of the Minmax theorem states that we have a mixed strategy over programs which achieves the same success

probability. We have $2^{n+\mathcal{O}(1)}$ different programs because we have $2^{n+\mathcal{O}(1)}$ different advice, with the same algorithm for all advice.

Corollary 1, with $\epsilon = 0.1$, gives an ℓ-uniform strategy over $\mathcal{O}(\ln(2^{n+2})) = \mathcal{O}(n)$ programs with error $\leq 1/3+0.1$. This strategy gives the desired algorithm, where the advice (of linear size) is the appropriate program we need to execute. The $\mathcal{O}(n)$-uniform strategy can be viewed as a probabilistic algorithm and a set of $\mathcal{O}(n)$ advice, from which it chooses one uniformly. The error probability is taken over the coins and is independent of the input distribution. We can run the algorithm $\mathcal{O}(1)$ times, and take the majority vote, in order to reduce the error to the desired error probability. We need only $\mathcal{O}(\log n)$ random coins to choose uniformly from the $\mathcal{O}(n)$ different programs. Our strategy essentially uses the distributional algorithms (the pure strategies) as oracles, which means that if $L \in \mathsf{P\text{-}sel}$ then by the furthermore part of Theorem 1 the algorithm never errs, and returns "?" with small probability.

Corollary 2. *If* $L \in \mathsf{k\text{-}mc}$*, then* $L \in \mathsf{BPP}//\mathcal{O}(n \cdot 2^{2k})$*, with error probability at most* $(2^{k-1}-1)/(2^k-1)$*. Thus,* $L \in \mathsf{DTime}(\mathrm{poly}(\mathsf{n}) \cdot 2^{2\mathsf{k}})/\mathcal{O}(n \cdot 2^k)$.

4.1 Karp-Lipton Type Results

A few interesting consequences of our results, are several Karp-Lipton [27] type of results. Very informally, we refer to results of the type "if SAT $\in \mathsf{P}/\mathrm{poly}$ then PH collapses to some low level of the hierarchy", see [5, 4, 14, 15, 27] for further studies on this type of questions. We show that if we have some additional information about the language, e.g., it can be decided in exponential time, or it is complete for some class with an instance-checker ([3, 30]), we can remove the advice completely. Intuitively, if L has an instance checker, then machine C, given an input x and an oracle L' that suppose to compute L, with high probability will be able to verify the validity of the oracle on x by comparing $L'(x)$ to $C^{L'}(x)$. The following result uses this in a similar way that the self-reducibility of SAT is used to prove that $\mathsf{NP} \subset \mathsf{P}/\log \Rightarrow \mathsf{P} = \mathsf{NP}$.

Theorem 6 ([44]). *If* $L \in \mathsf{BPP}//a(n)$ *admitting an instance checker with queries of length* $\ell(n)$ *then* $L \in \mathsf{BPTime}(\mathrm{poly}(\mathsf{n}) \cdot 2^{\mathsf{a}(\ell(\mathsf{n}))})$.

Combining theorems 2, 6, and the fact that every problem that is complete for $\mathsf{EXP}, \mathsf{PSpace}$, or $\mathsf{P}^{\#\mathsf{P}}$ has an instance checker [3, 30], we conclude the following, which can be viewed as probabilistic extension to the fact that P-selective sets which are also self-reducible are in P [10].

Corollary 3. *If* $L \in \mathsf{k\text{-}mc}$ *is complete for one of* $\{\mathsf{EXP}, \mathsf{PSpace}, \mathsf{P}^{\#\mathsf{P}}\}$ *then* $L \in \mathsf{BPTime}(\mathrm{poly}(\mathsf{n})2^{2^{2\mathsf{k}}\mathsf{n}^{\mathsf{d}}})$*, for some constant* $d > 0$.

Corollary 4. *Let* $f(n) \geq \log n$ *be any time constructible function. There exists a constant* $c > 0$ *s.t. if* $\mathrm{SAT} \in \mathsf{f(n)\text{-}mc}$ *then* $\Sigma_3^n \subseteq \Sigma_2^{2^{f(n^c)c}}$.

Theorem 7. *If* SAT *is* BPP*-selective then* $\mathsf{NP} = \mathsf{RP}$.

5 Concluding Remarks

Prior to our paper there where no indications that P-selective sets have sub-quadratic advice-complexity. In fact, the only conclusion of the negation of this claim, was that $\mathsf{P} \neq \mathsf{NP}$ which is widely accepted. We argue that our results justifies giving a fresh look to this classes, together with the evident importance of small and especially optimal circuits. We suggest a few possible paths for further extending our work.

Improving the Probabilistic Algorithm. In the main theorem of Section 3, we only use the fact that the sets are $\mathcal{O}(1)$-mc, and we know that this is a strictly greater class as proved in [2]. Furthermore, [7] showed that there exists a 2-mc which is not n^c non-adaptive reducible to a P-selective set, and that it is essential to use the selectiveness between different lengths. We showed that these sets have low randomized complexity, and it was already known that they also have low non-deterministic complexity (with advice). It can be proven that in such situations the uniform assumptions required for derandomizing the algorithm can be relaxed, e.g, $\mathsf{ZPP} \neq \mathsf{EXP}$ (instead of $\mathsf{BPP} \neq \mathsf{EXP}$) is sufficient for sub-exponential simulation.

Better Parameters for Hardness Amplification. In our proof we apply methods which were previously used to show such results for NP. However, there are evidences [18, 45] that NP may be a problematic companion, because it will be hard to improve the amplification for NP, and as a result, in our case. The bottleneck in our proof is the list size in Trevisan's proof. Better derandomized uniform direct product lemma's can give better parameters. Since we have an errorless algorithm for P-selective sets, an interesting direction can be to achieve an errorless hardness amplification result in our setting as in [9].

References

1. Althfer, I.: On sparse approximations to randomized strategies and convex combinations. Linear Algebra and its Applications 199(Suppl. 1), 339–355 (1994); Special Issue Honoring Ingram Olkin
2. Amir, A., Beigel, R., Gasarch, W.I.: Some connections between bounded query classes and non-uniform complexity. ECCC 7(024) (2000)
3. Babai, L., Fortnow, L., Lund, C.: Non-deterministic exponential time has two-prover interactive protocols. In: Proc. of the 31st IEEE Symp. on Foundations of Computer Science, pp. 16–25 (1990)
4. Balcázar, J., Book, R.V., Schöning, U.: Sparse sets lowness and highness. SIAM J. Comput. 15, 739–747 (1986)
5. Balcázar, J.L., Book, R.V., Schöning, U.: The polynomial-time hierarchy and sparse oracles. J. ACM 33, 603–617 (1986)
6. Beigel, R.: Personal communication. Weak Approximation, Help Bits, and the Complexity of Optimization Problems (2005)
7. Beigel, R., Fortnow, L., Pavan, A.: Membership comparable and P-selective sets. Technical Report 2002-006N. NEC Research Institute (2008)

8. Beimel, A., Ben Daniel, S.A., Kushilevitz, E., Weinreb, E.: Choosing, Agreeing, and Eliminating in Communication Complexity. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 451–462. Springer, Heidelberg (2010)
9. Bogdanov, A., Safra, S.: Hardness amplification for errorless heuristics. In: Annual IEEE Symposium on Foundations of Computer Science, vol. 0, pp. 418–426 (2007)
10. Buhrman, H., Torenvliet, L.: P-selective self-reducible sets: A new characterization of p. J. Comput. Syst. Sci. 53(2), 210–217 (1996)
11. Buhrman, H., Torenvliet, L., van Emde Boas, P.: Twenty questions to a P-selector. Inf. Process. Lett. 48(4), 201–204 (1993)
12. Buhrman, H., van Helden, P., Torenvliet, L.: P-selective self-reducibles sets: a new characterization of p. In: Proceedings of the Eighth Annual Structure in Complexity Theory Conference 1993, pp. 44–51, 18–21 (1993)
13. Buresh-Oppenheim, J., Kabanets, V., Santhanam, R.: Uniform hardness amplification in NP via monotone codes. Electronic Colloquium on Computational Complexity (ECCC) 13(154) (2006)
14. Cai, J., Chakaravarthy, V., Hemaspaandra, L., Ogihara, M.: Some Karp-Lipton type theorems based on S_2 Technical Report TR-819, Department of Computer Science, University of Rochester, Rochester, NY (2001)
15. Cook, S.A., Krajícek, J.: Consequences of the provability of NP subset of or equal to P/poly. J. Symb. Log. 72(4), 1353–1371 (2007)
16. Faliszewski, P., Hemaspaandra, L.: Open questions in the theory of semifeasible computation. SIGACT News 37, 47–65 (2006)
17. Faliszewski, P., Hemaspaandra, L.A.: Advice for semifeasible sets and the complexity-theoretic cost(lessness) of algebraic properties. Int. J. Found. Comput. Sci. 16(5), 913–928 (2005)
18. Gopalan, P., Guruswami, V.: Hardness amplification within NP against deterministic algorithms. In: Annual IEEE Conference on Computational Complexity, vol. 0, pp. 19–30 (2008)
19. Hartmanis, J., Stearns, R.E.: On the computational complexity of algorithms. Transactions of the American Mathematical Society 117, 285–306 (1965)
20. Hemaspaandra, E., Naik, A.V., Ogihara, M., Selman, A.L.: P-selective sets, and reducing search to decision vs. self-reducibility (1994)
21. Hemaspaandra, L.A., Naik, A.V., Ogihara, M., Selman, A.L.: Computing solutions uniquely collapses the polynomial hierarchy. SIAM J. Comput. 25(4), 697–708 (1996)
22. Hemaspaandra, L.A., Nasipak, C., Parkins, K.: A note on linear nondeterminism, linear-sized, Karp-Lipton advice for the P-selective sets. JJUCS 4(8), 670–674 (1998)
23. Hemaspaandra, L.A., Torenvliet, L.: Optimal advice. Theoretical Computer Science 154(2), 367–377 (1996)
24. Hemaspaandra, L.A., Torenvliet, L.: Theory of Semi-Feasible Algorithms. Springer, New York (2003)
25. Hemaspaandra, L.A., Zhigen, J.: P-selectivity: Intersections and indices. Theoretical Computer Science 145(1-2), 371–380 (1995)
26. Jockusch, C.: Semirecursive sets and positive reducibility. Transactions of the American Mathematical Society 131(2), 420–436 (1968)
27. Karp, R.M., Lipton, R.J.: Some connections between nonuniform and uniform complexity classes. In: STOC, pp. 302–309 (1980)
28. Ko, K.: On self-reducibility and weak P-selectivity. JCSS 26(2), 209–221 (1983)

29. Lipton, R., Young, N.E.: Simple strategies for large zero-sum games with applications to complexity theory. In: Proceedings of ACM Symposium on Theory of Computing, pp. 734–740 (1994)
30. Lund, C., Fortnow, L., Karloff, H.: Algebraic methods for interactive proof systems. J. ACM 39(4), 859–868 (1992)
31. Lund, C., Yannakakis, M.: On the hardness of approximating minimization problems. J. of the ACM 41(5), 960–981 (1994)
32. Moon, J.W.: An extension of Landau's theorem on tournaments. Pacific J. Math. 13, 1343–1345 (1963)
33. Moon, J.W.: Topics on tournaments. Rinehart and Winston, New York (1968)
34. Von Neumann, J.: Zur theorie der gesellschaftsspiele. Mathematische Annalen 100(1), 295–320 (1928)
35. O'Donnell, R.: Hardness amplification within NP. In: STOC 2002: Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing, pp. 751–760. ACM, New York (2002)
36. Ogihara, M.: Polynomial-time membership comparable sets. SIAM J. Comput. 24(5), 1068–1081 (1995)
37. Selman, A.L.: P-Selective Sets, Tally Languages, and the Behavior of Polynomial Time Reducibilities on NP. In: Maurer, H.A. (ed.) ICALP 1979. LNCS, vol. 71, pp. 546–555. Springer, Heidelberg (1979)
38. Selman, A.L.: Analogues of semicursive sets and effective reducibilities to the study of NP complexity. Information and Control 52(1), 36–51 (1982)
39. Baker, T., Gill, J., Solovay, R.: Relativizations of the $\mathcal{P} =? \mathcal{NP}$ question. SIAM Journal on Computing 4(4), 431–442 (1975)
40. Thakur, M.: On optimal advice for P-selective sets. Technical Report TR-819, Department of Computer Science. University of Rochester, Rochester, NY (2003)
41. Toda, S.: On polynomial-time truth-table reducibility of intractable sets to P-selective sets. Theory of Computing Systems 24, 69–82 (1991), doi:10.1007/BF02090391
42. Trevisan, L.: List-decoding using the XOR lemma. In: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2003, pp. 126–135. IEEE Computer Society, Washington, DC (2003)
43. Trevisan, L.: On uniform amplification of hardness in NP. In: STOC 2005: Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing, pp. 31–38. ACM, New York (2005)
44. Trevisan, L., Vadhan, S.P.: Pseudorandomness and average-case complexity via uniform reductions. Computational Complexity 16(4), 331–364 (2007)
45. Viola, E.: The complexity of constructing pseudorandom generators from hard functions. Computational Complexity 13, 147–188 (2005), doi:10.1007/s00037-004-0187-1
46. Wang, J.: Some results on selectivity and self-reducibility. Inf. Proc. Letters 55(2), 81–87 (1995)
47. Yao, A.C.: Some complexity questions related to distributed computing. In: Proc. of the 11th ACM Symp. on the Theory of Computing, pp. 209–213 (1979)

A Remark on One-Wayness versus Pseudorandomness*

Periklis A. Papakonstantinou and Guang Yang

Institute for Theoretical Computer Science,
Institute for Interdisciplinary Information Sciences, Tsinghua University,
Beijing 100084, China
papakons@tsingua.edu.cn, yangguang10@mails.tsinghua.edu.cn

Abstract. Every pseudorandom generator is in particular a one-way function. If we only consider part of the output of the pseudorandom generator is this still one-way? Here is a general setting formalizing this question. Suppose $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ is a pseudorandom generator with stretch $\ell(n)$. Let $M_R \in \{0,1\}^{m(n)\times\ell(n)}$ be a linear operator computable in polynomial time given randomness R. Consider the function

$$F(x,R) = \big(M_R G(x), R\big)$$

We obtain the following results.

- There exists a pseudorandom generator s.t. for every positive constant $\mu < 1$ and for an arbitrary polynomial time computable $M_R \in \{0,1\}^{(1-\mu)n\times\ell(n)}$, F is not one-way.

  Furthermore, our construction yields a tradeoff between the hardness of the pseudorandom generator and the output length $m(n)$. For example, given $\alpha = \alpha(n)$ and a 2^{cn}-hard pseudorandom generator we construct a $2^{\alpha cn}$-hard pseudorandom generator such that F is not one-way, where $m(n) \leq \beta n$ and $\alpha + \beta = 1 - o(1)$.
- We show this tradeoff to be tight for 1-1 pseudorandom generators. That is, for any G which is a $2^{\alpha n}$-hard 1-1 pseudorandom generator, if $\alpha + \beta = 1 + \epsilon$ then there is $M_R \in \{0,1\}^{\beta n\times\ell(n)}$ such that F is a $\Omega(2^{\epsilon n})$-hard one-way function.

Keywords: cryptographic hardness, one-way function, pseudorandom generator.

1 Introduction

A one-way function is a function easy to compute but hard to invert. A pseudorandom generator is an efficient deterministic algorithm that stretches a short random seed to a longer one which is hard to distinguish from random. They are both fundamental primitives in private-key cryptography.

* This work was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61061130540, 61073174, 61150110582.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 482–494, 2012.

We tend to believe that one-wayness is a weaker notion than pseudorandomness. One reason is that every pseudorandom generator is in particular a one-way function, but the other direction fails dramatically. In this paper we consider the effect on the one-wayness of a pseudorandom generator when "hashing" its output. A natural way to formalize this is to consider the application of an efficiently sampleable linear operator, which also captures (but a minor issue[1]) universal families of hash functions and certain randomness extractors. Formally, let $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$, $\ell(n) > n$ be a pseudorandom generator, and fix an arbitrary polynomial time algorithm that on input R it outputs a matrix $M_R \in \{0,1\}^{m(n)\times\ell(n)}$. Consider the following "hashing method":

$$F^G(x,R) = \big(M_R G(x), R\big)$$

We study the effect of the size of $m(n)$ on the one-wayness of F^G. In fact, all of our results hold for affine $\mathbf{F}(\mathbf{x},\mathbf{R}) = \big(\mathbf{M_R G(x)} + \mathbf{b_R}, \mathbf{R}\big)$ as well.

1.1 Previous Work and Motivation

Studying relations among basic cryptographic primitives is fundamental for cryptography. Since the seminal work of Håstad-Impagliazzo-Levin-Luby [HILL89], the first to construct a pseudorandom generator from any one-way function, there is a line of excellent works (e.g. [HRV10, HHR06a, HHR06b]) improving its efficiency. Questions regarding the other direction have so far been neglected[2].

Instead of asking whether one-wayness is preserved when hashing the output of every pseudorandom generator, we can ask the weaker question of whether there exists a pseudorandom generator that has this property. Suppose that it was possible to apply a simple length-shrinking hash (e.g. a projection) on the output of an NC^0 pseudorandom generator, then via the work of Applebaum-Ishai-Kushilevitch [AIK04, AIK05] we can build several cryptographic primitives in a streaming fashion. *Streaming Cryptography* [KGY89, BJP11], not to be confused with stream ciphers, concerns the computation of cryptographic primitives with a device that has small working memory, e.g. logarithmic or sub-linear, and it makes a small number of passes, e.g. poly-logarithimic, over its input. Our results rule out a natural class of constructions in Streaming Cryptography.

1.2 Our Results

We have obtained both negative and positive results. We show that there exists a pseudorandom generator where if we apply a length-shrinking, even by a constant factor, linear operator on its output then this *is not* a one-way function. Our construction (Theorem 1) yields a tradeoff between the hardness of this generator and the shrinkage factor. Theorem 1 is also, in particular, about universal families

[1] Applying a random linear operator does not exactly yield a universal family of hash functions just because of its value at $\mathbf{0}$.

[2] This is not surprising, since a pseudorandom generator is in particular a one-way function.

of hash functions. In Theorem 2 we show that our construction is optimal, in the sense that if instead we use any generator which is a little harder, or if the shrinkage factor is a little bigger, then the resulting function *is* one-way.

Theorem 1. *Suppose G is a pseudorandom generator with hardness $s_G(\cdot)$. Then for every constant $\mu > 0$ and $\delta > 0$, and for an arbitrary polynomial $\ell(n)$, there is a pseudorandom generator*

$$G^* : \{0,1\}^n \to \{0,1\}^{\ell(n)}$$

such that $F^{G^*}(x,R) = (M_R G^*(x), R)$ is not one-way, where $M_R \in \mathbb{F}_2^{m(n)\times\ell(n)}$ is polynomial time computable using randomness R with $m(n) \leq (1-\mu)n$. Moreover, G^* preserves the injectivity of G and has hardness at least $s_G(\mu n - n^\delta)$.

The "moreover" part makes the theorem stronger. Also, preserving injectivity in this theorem finds application in explaining a subtle issue regarding the optimal output length of hash functions in the first step of [HILL89] construction (see Section 4 in [HILL89], or p.138 in [Gol01]).

A variant of Theorem 1 shows that when M_R is restricted to random projections with $m(n) = O(\frac{n}{\log(n)})$ (i.e. just sampling $m(n)$ bits from the output of G), then there exists (another) G^* s.t. F^{G^*} is invertible in non-uniform NC^2.

On the other hand, we prove that when hashing a 2^{cn}-hard pseudorandom generator to a little more than $(1-c)n$ bits then its one-wayness is preserved.

Theorem 2. *Suppose $f : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ is a 2^{cn}-hard 1-1 pseudorandom generator. Let $F := F^f(x,h) = \big(h(f(x)), h\big)$, where $h : \{0,1\}^{\ell(n)} \to \{0,1\}^{m(n)}$ is a hash function from a universal family of hash functions $S^{m(n)}_{\ell(n)}$. If $m(n) \geq (1-c+\epsilon)n$ for constant $\epsilon \in (0, \frac{c}{5})$, then F is one-way with hardness $2^{\epsilon n}$.*

In fact, the above theorem holds true if instead of a pseudorandom generator we consider f to be an injective one-way function.

1.3 Outline

In Section 2, we introduce notations, definitions, and basic facts. In Section 3, we construct G^* from a pseudorandom generator G such that F^{G^*} is not one-way when hashing down its output by a constant factor. In Section 4 we show that for every 1-1 pseudorandom generator f with hardness 2^{cn} and $m(n) \geq (1-c+\epsilon)n$, F^f preserves the one-wayness and has hardness at least $2^{\epsilon n}$. We conclude in Section 5 with some further research directions.

2 Preliminary

2.1 Notation and Definitions

Probability Notation. For probability distributions X, Y, we denote by $X \sim Y$ that X and Y are identically distributed. $x \leftarrow X$ denotes that x is sampled from X, and $x \in_R S$ denotes that x is sampled uniformly from S. U_n denotes the uniform distribution over $\{0,1\}^n$. The *statistical distance* between two distributions X and Y is defined as $\Delta(X,Y) = \frac{1}{2}\sum_z |\Pr[X=z] - \Pr[Y=z]|$.

Universal Families of Hash Functions. Let S_n^m denote a set of functions from $\{0,1\}^n$ to $\{0,1\}^m$. Let H_n^m be a random variable uniformly distributed over S_n^m. S_n^m is called a *universal family of hash functions* if following conditions hold:

- S_n^m is a pairwise independent family of mappings: for every $x \neq y$, $H_n^m(x)$ and $H_n^m(y)$ are independent and both identically to U_m.
- S_n^m has a succinct representation: $\forall h \in S_n^m$, the description of h is $\mathsf{poly}(n,m)$.
- S_n^m can be efficiently evaluated: there is a polynomial time algorithm $\mathcal{H}$ such that for every $h \in S_n^m, x \in \{0,1\}^n$, $\mathcal{H}(h,x) = h(x)$.

Specifically, $h(x) = M \cdot x + b$ is a universal family of hash functions when the matrix M and vector b are uniformly distributed. Actually, $h(x) = M \cdot x$ satisfies all above conditions except that $H_n^m(x)$ is not uniformly distributed when $x = \mathbf{0}$.

Cryptographic Primitives. Here are the definitions of one-way functions, pseudorandom generators, and k-wise independent distributions. The definitions are for uniform adversaries, however our results hold in the non-uniform setting as well (c.f. [Gol01, Vad11]).

A *one-way function* $f : \{0,1\}^* \to \{0,1\}^*$ is a polynomial time computable function where no probabilistic polynomial time algorithm A inverts f with non-negligible probability; i.e. for every k and any polynomial time algorithm A, $\Pr_{x \leftarrow U_n}[A(f(x), 1^n) \in f^{-1}(f(x))] < n^{-k}$ holds for sufficiently large n.

Furthermore, we say that f has *hardness* $s(n)$ if for every sufficiently large input of length n, f cannot be inverted with probability $\geq \frac{1}{s(n)}$ by any adversary A which runs in time $\leq s(n)$. Obviously, f is a one-way function if f has super-polynomial hardness $s(n)$.

A *pseudorandom generator* G is a polynomial time computable function which stretches every n-bit input to an output of length $\ell(n) > n$, such that no probabilistic polynomial time algorithm D can distinguish between $U_{\ell(n)}$ and $G(U_n)$; i.e. for every k and D, $|\Pr[D(G(U_n), 1^n) = 1] - \Pr[D(U_{\ell(n)}, 1^n) = 1]| < n^{-k}$ when n is sufficiently large. We call ℓ the *stretch of* G. Similar to one-way functions we define an $s(n)$-hard pseudorandom generator.

We subscript a string $\sigma \in \{0,1\}^n$ with $R \subseteq \{1,\ldots,n\}$, and we write σ_R, to denote the substring of σ keeping exactly the bits indexed by R. In this notation, a function h is called k-wise independent if for every $K \subseteq \{1,\ldots,n\}$ where $|K| = k$ we have that $h(U_n)_K \sim U_k$.

Circuit Classes. We denote by NC^2 the functions computed by *non-uniform* families of poly-size boolean circuits with *multiple outputs*, where the gates are of constant fan-in and the depth of the circuit is $O(\log^2 n)$ for input length n.

2.2 Basic Facts and Lemmas

Below is a well-known fact (implicitly shown in [LR87], also see e.g. [Gol01]).

Lemma 1. *Let G be a pseudorandom generator. Then, G is a one-way function.*

The following lemma states that a uniform randomly chosen matrix has a good chance of being row independent. In fact, more general results hold for $n \times n$ matrices (see e.g. [BKW97, Muk84]). The proof of the following lemma is an easy exercise and is omitted here.

Lemma 2. *Uniformly at random pick a $p \times q$ matrix N over $\mathbb{F}_2$; i.e. $N \in_R \mathbb{F}_2^{p \times q}$. Then, N has full row-rank with probability at least $1 - 2^{p-q}$.*

A deep result due to Mulmuley [Mul87] (which derandomizes [BvzGH82]) states that Gaussian elimination for linear systems over $\mathbb{F}_2$ can be done in uniform NC^2. Later on, when applying this lemma in our paper, we introduce non-uniformity for a different reason.

Lemma 3 ([Mul87]). *Gaussian elimination can be done in uniform NC^2.*

3 Length-Shrinking Linear Operators Destroy One-Wayness: A Shrinkage-Hardness Tradeoff

We prove Theorem 1. That is, given a pseudorandom generator G of hardness $s_G(n)$ we construct a pseudorandom generator G^* of *almost* the same hardness $s_{G^*}(n) = s_G\big((\mu - o(1))n\big)$ for some constant μ, such that an application of any efficiently sampled linear operator, which outputs $(1-\mu)n$ bits, on the output of G^* does not preserve one-wayness.

First we introduce the construction of G^*. It is easy to see that it preserves pseudorandomness and injectivity; i.e. if G is 1-1 then G^* is also 1-1.

Construction 1. *Construct G^* as*

$$G^*(x_1, x_2, x_3) = (\hat{G}(x_1) + (P_G(x_3) \cdot x_2), x_2, x_3) \tag{1}$$

$|x_1| = n_1$, $|x_2| = n_2$, $|x_3| = n_3$, $n_1 + n_2 + n_3 = n$. $\hat{G}(x_1) = G^{(z)}(x_1)|_{\{1,2,\cdots,\ell'(n)\}}$ where $G^{(z)}$ means z iterated compositions of G with itself such that $|G^{(z)}(x_1)| \geq \ell'(n) = \ell(n) - n_2 - n_3$. $P_G(x_3)$ is an $\ell'(n) \times n_2$ pseudorandom matrix whose entries are generated by iteratively applying G on x_3. All operations are over $\mathbb{F}_2$.

By definition of $\hat{G}$, $|\hat{G}(x_1)| = \ell'(n)$. That is, $|G^*(x_1, x_2, x_3)| = \ell'(n) + n_2 + n_3 = \ell(n)$. Since we XOR $\hat{G}(x_1)$ with $P_G(x_3) \cdot x_2$, then $s_G(n_1)$ lower bounds the hardness of $G^*(x)$. We can choose n_3 to be an arbitrarily small polynomial in n. The parameters n_1 and n_2 determine a tradeoff between the hardness of the pseudorandom generator G^* and the shrinking length. This tradeoff is not a minor issue. If we were to choose arbitrarily close to 1 the constants in the hardness and in the shrinking length then a modification of [HILL89] would have shown that exponentially hard pseudorandom generators, unconditionally, do not exist (this is not an immediate argument).

The following lemma is the main ingredient of the proof of Theorem 1.

Lemma 4. *Let* $F^{G^*}(x,R) = (M_R G^*(x), R)$ *and let* $G^*(x_1,x_2,x_3)$ *be as in Construction 1. Let* $M_R \in \{0,1\}^{m(n)\times \ell(n)}, m(n) < n_2$, *be computable in polynomial time given* R. *Then, there is a probabilistic polynomial time algorithm* A *s.t.*

$$\Pr_{y,R}[F^{G^*}(A(y,R)) = (y,R)] > 1 - 2^{-(n_2 - m(n))} - \mathsf{poly}(\frac{1}{s_G(n_3)})$$

Proof. Recall that $G^*(x_1,x_2,x_3) = \big(\hat{G}(x_1) + (P_G(x_3)\cdot x_2), x_2, x_3\big)$, where $x = (x_1,x_2,x_3)$ and x_1,x_2,x_3 has length n_1,n_2,n_3 respectively. Then,

$$F^{G^*}(x,R) = \big(M_R G^*(x), R\big) = \big(M_R(\hat{G}(x_1) + (P_G(x_3)\cdot x_2), x_2, x_3), R\big)$$

Therefore for the goal $F^{G^*}(x,R) = (y,R)$, it suffices to find an x such that

$$M_R\big(\hat{G}(x_1) + (P_G(x_3)\cdot x_2), x_2, x_3\big) = y \tag{2}$$

We analyze further the structure of the above matrix equation. Without loss of generality, we may assume that M_R is already in reduced row echelon form, after applying Gaussian elimination, and it has full row-rank (easy to guarantee by deleting all zero rows). To match the form of the column vector $\big(\hat{G}(x_1)+(P_G(x_3)\cdot x_2), x_2, x_3\big)$, we partition M_R into $M_R = (M_1|M_2|M_3)$ where the sub-matrices M_1, M_2, M_3 have $\ell'(n), n_2$ and n_3 columns respectively. Then

$$M_R = \begin{pmatrix} M_1 & M_2 & M_3 \end{pmatrix} = \begin{pmatrix} M_1' & M_2'' & M_3''' \\ 0 & M_2' & M_3'' \\ 0 & 0 & M_3' \end{pmatrix}$$

where M_1', M_2' and M_3' have full row-rank. Note that depending on M_R, it is possible that M_2', M_3' and M_3'' are empty (i.e. size 0, instead of having 0-entries). Equation (2) can be rewritten as a linear system in x_2,

$$\begin{cases} \big(M_1' P_G(x_3) + M_2''\big)\ x_2 = y_1 + M_3''' x_3 + M_1'\hat{G}(x_1) \\ \qquad\qquad M_2'\ \ x_2 = y_2 + M_3'' x_3 \\ \qquad\qquad\quad \mathbf{0}\ \ = y_3 + M_3' x_3 \end{cases} \tag{3}$$

Now the problem reduces to finding a solution x to (3). We present an adversary A which finds a solution to the above system.

A : INVERTING F^{G^*} (on input (y, R)):

1 Compute M_R with input R;
2 Do Gaussian elimination on the left of $(M_R|y)$;
3 Delete zero-rows and return "No answer" if detecting a row $(0, 0, \cdots, 0, 1)$;
4 Compute $M'_1, M'_2, M''_2, M'_3, M''_3, M'''_3$;
5 Set x_1 to a fixed value u, say n_1 zeros;
6 Uniformly at random pick v from $\{x_3 \big| M'_3 x_3 = y_3\} \subseteq \{0,1\}^{n_3}$ ($v \leftarrow U_{n_3}$ if M'_3 is empty);
7 Compute $P_G(v)$ and $\hat{G}(u)$;
8 Consider: $\begin{pmatrix} M'_1 P_G(v) + M''_2 \\ M'_2 \end{pmatrix} x_2 = \begin{pmatrix} y_1 + M'_1 \hat{G}(u) + M'''_3 v \\ y_2 + M''_3 v \end{pmatrix}$;
9 Solve x_2 and output $(x, R) = ((u, x_2, v), R)$. Output "Fail" if there is no solution.

It is easy to verify that A runs in polynomial time and the output is a pre-image of (y, R). Now, we analyze the probability that A succeeds. It suffices to calculate the probability that A outputs "Fail", which is upper bounded by the probability that $\mathcal{M} = \begin{pmatrix} M'_1 P_G(v) + M''_2 \\ M'_2 \end{pmatrix}$ does not have full row-rank. Let $\mathcal{M}' = \begin{pmatrix} M'_1 \cdot U_{\ell'(n) \times n_2} + M''_2 \\ M'_2 \end{pmatrix}$. Since M'_1, M'_2 have full row-rank, $\mathcal{M}' \sim \begin{pmatrix} U_{r_1 \times n_2} \\ M'_2 \end{pmatrix}$ does not have full row-rank with probability at most $\sum_{1 \le i \le r_1} \frac{2^{r_2+i-1}}{2^{n_2}} < \frac{2^{r_1+r_2}}{2^{n_2}} = 2^{-(n_2-r_1-r_2)}$ by Lemma 2, where r_1, r_2 is the number of rows in M'_1, M'_2 respectively. Moreover, the gap between the probability $\Pr[\mathcal{M}$ has full row-rank$]$ and $\Pr[\mathcal{M}'$ has full row-rank$]$ is bounded by $\mathsf{poly}(\frac{1}{s_G(n_3)})$, since otherwise there exists a polynomial time distinguisher for $P_G(v)$ and $U_{\ell'(n) \times n_2}$ with advantage $\mathsf{poly}(\frac{1}{s_G(n_3)})$. So we have

$$\begin{aligned} \Pr[\mathcal{M} \text{ has full row-rank}] &\ge \Pr[\mathcal{M}' \text{ has full row-rank}] - \mathsf{poly}(\frac{1}{s_G(n_3)}) \\ &\ge 1 - 2^{-(n_2-r_1-r_2)} - \mathsf{poly}(\frac{1}{s_G(n_3)}). \end{aligned}$$

Since M_R has $m(n)$ rows in total, which implies $r_1 + r_2 \le m(n)$,

$$\Pr_y[A \text{ succeeds}] \ge \Pr[\mathcal{M} \text{ has full row-rank}] \ge 1 - 2^{-(n_2 - m(n))} - \mathsf{poly}(\frac{1}{s_G(n_3)})$$

Thus complete our proof of Lemma 4.

Corollary 1. *If $m(n) \le n_2 - \omega(\log(n))$ and $n_3 = n^{\Omega(1)}$, then $F^{G^*}(x, R) = \big(M_R G^*(x), R\big)$ is not (even weakly) one-way.*

Let $n_1 = \mu n - n^\delta$, $n_2 = (1-\mu)n + \log^2(n)$, and $n_3 = n - n_1 - n_2 = n^\delta - \log^2(n)$ in Construction 1 and $m(n) = n_2 - \log^2(n) = (1-\mu)n$. Applying Lemma 4 and Corollary 1, we conclude the proof of Theorem 1. In general, hashing down the output of a pseudorandom generator by a constant factor does not preserve one-wayness, even if the pseudorandom generator is exponential hard.

Regarding the roles of n_1, n_2, n_3 in above argument, we first notice that n_3 is the least important one since we only need $s_G(n_3)$ super-polynomial. In most common cases of interest $s_G(\cdot)$ is monotonically increasing (hence, s_G^{-1} is well defined), it suffices to set $n_3 = s_G^{-1}(n^{\omega(1)})$ which could be as small as $\log^{O(1)}(n)$ for exponential s_G. Meanwhile, the difference $n_2 - m(n)$ is also negligible. It turns out $n_1 + m(n) = n - o(n)$. Recalling that G^* has hardness $s_G(n_1)$, there is the tradeoff between the hardness of G^* and the output length of M_R. Letting $n_1 = \alpha n, m(n) = \beta n$, we get $\alpha + \beta = 1 - o(1)$ as stated in the abstract.

Special Case of Random Projections. When M_R is a projection of length $O(\frac{n}{\log n})$ we construct a simpler pseudorandom generator G^* where F^{G^*} is invertible in NC^2. For this we combine the "strong pseudorandom" (cryptographic) object G with a "weak pseudorandom" object, a k-wise independent generator. Specifically, let $G^*(x_1, x_2) = (\hat{G}(x_1) + Hx_2)$ where H realizes a k-wise generator with $k = \Theta(\frac{n}{\log(n)})$. See Proposition 6.5 in [ABI86] and Chap. 7.6 in [MS77] for details.

Lemma 5. *Let $m(n) \leq k$, where k as above. Then,$F^{G^*}(x, R) = \big(M_R G^*(x), R\big)$ can be inverted in NC^2.*

The adversary is a modification of A which appears in the proof of Lemma 4. In particular, in Step 4, only M_1' matters since other matrices are 0-sized; in Step 6,7,8, $P_G(v)$ is replaced by H and the linear system in Step 8 becomes $M_1' H x_2 = y_1 + \hat{G}(u)$. Although $\hat{G}$ is polynomial time computable, we can non-uniformly hardwire the value of $\hat{G}$ on a constant one for each input length. Since u can be fixed, then by Lemma 3 we have that $M_1' H$ is invertible in NC^2.

4 Tightness of the Construction

Even if we assume that a pseudorandom generator of hardness $2^{0.99n}$ exists, Theorem 1 says that then there is a generator of hardness $2^{0.99\alpha n}$ such that when applying a linear map on its output shrinking it down to βn many bits then this is not one-way, for $\alpha + \beta = 1 - o(1)$. We show that this tradeoff between α and β is tight, i.e. when $\alpha + \beta = 1 + \epsilon$ and a 1-1 generator f has hardness $2^{\alpha n}$, then F^f forms a $2^{\epsilon n}$-hard one-way function.

For the proof of Theorem 2 we apply the following well-known lemma, but in a non-uniform setting.

Lemma 6 ([Gol01], also [HILL89, Sip83, GL89]). *Let $m < \ell$ be integers, S_ℓ^m be a universal family of hash functions, and b,δ be two reals such that $m \leq b \leq \ell$ and $\delta \geq 2^{-\frac{b-m}{2}}$. Suppose that X_ℓ is a random variable distributed over*

$\{0,1\}^\ell$ such that for every x, it holds $\Pr[X_n = x] \leq 2^{-b}$. Then for every $\xi \in \{0,1\}^m$ and for all but at most $2^{-(b-m)}\delta^{-2}$ fraction of the h's in S^m_ℓ, it holds that

$$\Pr_{X_\ell}[h(X_\ell) = \xi] \in (1 \pm \delta)2^{-m}$$

Proof (Proof of Theorem 2). We present the proof for a non-uniform adversary, simpler to present but already a rather involved argument. Fix one efficient construction of sampling from a universal family of hash functions (e.g. choose one from [Vad11]). Now F is well-defined for a given f. Assume that F is not a $2^{\epsilon n}$-hard one-way function. Let A be a probabilistic algorithm which runs in time $T_A = O(2^{\epsilon n})$ and inverts F with probability $p_A(n)$, i.e.

$$\Pr_{x \leftarrow U_n, h \leftarrow_R S^{m(n)}_{\ell(n)}}[A(h(f(x)), h) \in F^{-1}(h(f(x)), h)] = p_A(n) > \frac{1}{2^{\epsilon n}}$$

We show that f is not 2^{cn}-hard with oracle access to A. That is, we construct a non-uniform adversary A_f that given $y \leftarrow f(U_n)$, A_f computes x' such that $f(x') = y$ in time $O(2^{cn})$ and with probability at least $\Omega(2^{-cn})$.

A_f is defined as follows: with the non-uniform advice $h_0 \in S^{m(n)}_{\ell(n)}$, A_f first computes $(h_0(y), h_0)$, then applies A to compute x' such that $h_0(f(x')) = h_0(y)$.

Therefore, A_f runs in time $O(T_A) = O(2^{\epsilon n}) = O(2^{cn})$. In what follows we denote by $x' = x'(h(y), h)$ the output of A on input $(h(y), h)$. Now, we calculate the probability that A_f outputs x'. We will determine later how to find h_0, and in fact why h_0 exists.

$$\Pr_{y \leftarrow f(U_n)}[A_f \text{ inverts } f \text{ on } y] = \Pr_{y \leftarrow f(U_n)}[x' = A(h_0(y), h_0), f(x') = y] \tag{4}$$

$$= \Pr_{x \leftarrow U_n}[f(x') = f(x)] \tag{5}$$

where in the last equation we omit how x' is derived and its dependence.

$$\begin{aligned}
&\Pr_{x \leftarrow U_n}[f(x') = f(x)] \\
&= \sum_{z \in h_0(f(\{0,1\}^n))} \Pr_{x \leftarrow U_n}[h_0(f(x)) = z] \Pr_{x \leftarrow U_n}[f(x') = f(x) \big| h_0(f(x)) = z] \\
&= \sum_{z \in h_0(f(\{0,1\}^n))} \Pr_{x \leftarrow U_n}[h_0(f(x)) = z] \Pr_{x \in_R (h_0 \circ f)^{-1}(z)}[x = x' = x'(z, h_0)]
\end{aligned}$$

$f(x') = f(x)$ is equivalent to $x' = x$ since f is 1-1. From this point on, $x'(z, h_0)$ is uniquely defined from z and h_0. So we can take it out of the probability.

$$\begin{aligned}
&= \sum_{z \in h_0(f(\{0,1\}^n))} \frac{|(h_0 \circ f)^{-1}(z)|}{2^n} \cdot \Big(\frac{1}{|(h_0 \circ f)^{-1}(z)|} \cdot I[h_0(f(x'(z, h_0))) = z]\Big) \\
&= \frac{1}{2^n} \sum_{z \in h_0(f(\{0,1\}^n))} I[h_0(f(x')) = z] = \frac{1}{2^n} \sum_{z \in \{0,1\}^m} I[h_0(f(x')) = z]
\end{aligned} \tag{6}$$

where $I[h_0(f(x')) = z]$ is the indicator of the event "$h_0(f(x')) = z$ for $x' = A(z, h_0)$". Note that the sum $\sum_{z\in\{0,1\}^m} I[h_0(f(x')) = z]$ corresponds to the number of z's that A inverts (z, h_0).

However, when fixing h_0, the probability "A succeeds" is

$$\Pr_{x\leftarrow U_n}[A \text{ inverts } (h_0(f(x)), h_0)] = \sum_{z\in\{0,1\}^m} \Pr_{x\leftarrow U_n}[h_0(f(x)) = z] I[h_0(f(x')) = z] \tag{7}$$

Notice that (7) is the probability of "A succeeds on $\big(h_0(f(U_n)), h_0\big)$", while (6) counts the number of z's that A inverts $\big(z, h_0\big)$. These two are related in the following sense. Remember that hashing down a weak random source smooths the distribution, hence $h_0(f(U_n))$ seems close to U_m. In this sense, we make an estimation with error upper bounded by their statistical distance.

$$\begin{aligned}
&\Big|\Pr_{x\leftarrow U_n}[A \text{ inverts } (h_0(f(x)), h_0)] - \frac{1}{2^m}\sum_{z\in\{0,1\}^m} I[h_0(f(x')) = z]\Big| \\
=&\Big|\sum_{z\in\{0,1\}^m} \Pr_{x\leftarrow U_n}[h_0(f(x)) = z]\cdot I[h_0(f(x')) = z] - \sum_{z\in\{0,1\}^m} \frac{1}{2^m} I[h_0(f(x')) = z]\Big| \\
\le&\sum_{z\in\{0,1\}^m} \Big|\Pr_{x\leftarrow U_n}[h_0(f(x)) = z] - \frac{1}{2^m}\Big|\cdot I[h_0(f(x')) = z] \\
=&2\Delta\big(h_0(f(U_n)), U_m\big)
\end{aligned} \tag{8}$$

Plugging (8) into (6), it immediately leads to the lower bound

$$\begin{aligned}
&\Pr_{x\leftarrow U_n}[f(x') = f(x)] \\
\ge&2^{m-n}\Big(\Pr_{x\leftarrow U_n}[A \text{ inverts } (h_0(f(x)), h_0)] - 2\Delta\big(h_0(f(U_n)), U_m\big)\Big)
\end{aligned} \tag{9}$$

Now, our goal is to show that there exists a choice for h_0 in (9) giving the $\Omega(\frac{1}{2^{cn}})$ lower bound.

Claim. There is a (good) $h_0 \in S^{m(n)}_{\ell(n)}$ such that

- Property 1: $\Delta\big(h_0(f(U_n)), U_m\big) < 2\cdot 2^{\frac{1+\epsilon n-(n-m)}{3}}$;
- Property 2: $\Pr_{x\leftarrow U_n}[h_0(f(x')) = h_0(f(x))] \ge 2^{-(1+\epsilon n)}$.

For Property 1, it suffices for concluding the proof to have $\delta = 2^{\frac{1+\epsilon n-(n-m)}{3}}$ and

$$\Pr_{\xi\leftarrow U_m}[\Pr[h_0(f(U_n)) = \xi] \notin (1\pm\delta)\cdot 2^{-m}] < 2^{1+\epsilon n-(n-m)}\delta^{-2}$$

Let $\delta = 2^{\frac{1+\epsilon n-(n-m)}{3}}$, $b = n, m = m(n), \ell = \ell(n)$ and $X = f(U_n)$ as in Lemma 6. Since $m \le b \le \ell(n)$ and f is 1-1 ($\Pr_X[X = z] \le \frac{1}{2^n}$ for every z), we have that $\forall \xi \in \{0,1\}^m$ and for all but at most $2^{-(n-m)}\delta^{-2}$ fraction of the h's in

$S^{m(n)}_{\ell(n)}$, it holds $\Pr[h(f(U_n)) = \xi] \in (1 \pm \delta) \cdot 2^{-m}$. Let $\mathcal{B}(h, \xi)$ denote the event $\Pr[h(f(U_n)) = \xi] \notin (1 \pm \delta) \cdot 2^{-m}$, then taking probability over ξ and h,

$$\Pr_{\xi \leftarrow U_m, h \leftarrow S^{m(n)}_{\ell(n)}}[\mathcal{B}(h, \xi)] \leq 2^{-(n-m)}\delta^{-2}$$

$$\implies \Pr_{h \leftarrow S^{m(n)}_{\ell(n)}}[\Pr_{\xi \leftarrow U_m}[\mathcal{B}(h, \xi)] \geq 2^{1+\epsilon n-(n-m)}\delta^{-2}] \leq \frac{1}{2^{1+\epsilon n}} \tag{10}$$

Thus, $\Pr_{\xi \leftarrow U_m}[\Pr[h(f(U_n)) = \xi] \notin (1 \pm \delta) \cdot 2^{-m}] < 2^{1+\epsilon n-(n-m)}\delta^{-2}$ holds for at least $1 - \frac{1}{2^{1+\epsilon n}}$ fraction of the h's in $S^{m(n)}_{\ell(n)}$. In particular, Property 1 is satisfied by that many h's.

For Property 2, we lower bound the probability that A performs not so bad for a randomly chosen h, i.e. $\Pr_{h \leftarrow S^{m(n)}_{\ell(n)}}[\Pr_{x \leftarrow U_n}[h(f(x')) = h(f(x))] \geq \frac{1}{2^{1+\epsilon n}}]$. Let $\mathcal{E}_h$ denote the event that $\Pr_{x \leftarrow U_n}[h(f(x')) = h(f(x))] \geq 2^{-1-\epsilon n}$, we have

$$\begin{aligned}
2^{-\epsilon n} \leq p_A(n) &= \Pr_{h \leftarrow S^{m(n)}_{\ell(n)}, x \leftarrow U_n}[h(f(x')) = h(f(x))] \\
&= \Pr_h[\mathcal{E}_h] \Pr_x[h(f(x')) = h(f(x)) | \mathcal{E}_h] + \Pr_h[\overline{\mathcal{E}_h}] \Pr_x[h(f(x')) = h(f(x)) | \overline{\mathcal{E}_h}] \\
&\leq \Pr_{h \leftarrow S^{m(n)}_{\ell(n)}}[\mathcal{E}_h] \cdot 1 + \Pr_{h \leftarrow S^{m(n)}_{\ell(n)}}[\overline{\mathcal{E}_h}] \cdot 2^{-1-\epsilon n} < \Pr_{h \leftarrow S^{m(n)}_{\ell(n)}}[\mathcal{E}_h] + 2^{-1-\epsilon n} \\
\implies \Pr[\mathcal{E}_h] &> 2^{-1-\epsilon n}
\end{aligned}$$

Hence, we lower bound the probability of h having Property 2 as follows

$$\Pr_{h \leftarrow S^{m(n)}_{\ell(n)}}[\Pr_{x \leftarrow U_n}[h(f(x')) = h(f(x))] \geq 2^{-1-\epsilon n}] = \Pr_{h \leftarrow S^{m(n)}_{\ell(n)}}[\mathcal{E}_h] > 2^{-1-\epsilon n}$$

The following calculation shows that an h_0 as required exists.

$$\Pr_{h \leftarrow S^{m(n)}_{\ell(n)}}[h \text{ satisfies both Property 1 and 2}] > (1 - \frac{1}{2^{1+\epsilon n}}) + 2^{-1-\epsilon n} - 1 = 0$$

Using this h_0 in (9), and recalling that $m = m(n) = (1 - c + \epsilon)n$, we obtain

$$\Pr_{x \leftarrow U_n}[f(x') = f(x)] \geq 2^{-1-cn} - 2^{(7+(5\epsilon-4c)n)/3} = \Omega(2^{-cn})$$

Note that the running time of A_f is bounded by $O(2^{cn})$, contradicting that f is 2^{cn} hard. In conclusion, $F(x, h) = \big(h(f(x)), h\big)$ is one-way, and its hardness is at least $2^{\epsilon n}$.

5 Conclusions and Open Questions

We have showed that "hashing" the output of a pseudorandom generator to a constant fraction of its input length, in general, destroys its one-wayness. We

prove this in the form of a tradeoff between cryptographic hardness and output length of the hash. We also show that this tradeoff is tight.

An interesting question is whether there exists a pseudorandom generator of reasonable hardness where one-wayness is preserved when hashing its output. This question remains open. We speculate that is a difficult mathematical problem. For example, an interesting direction would be to show that this question is equivalent to constructing 2^{n^ϵ}-hard one-way functions; i.e. a problem essentially about $\Omega(2^{n^\epsilon})$ circuit lower bounds.

Acknowledgements. We would like to thank John Steinberger and Andrew Wan for the helpful remarks on a previous draft. We would also like to thank Andrej Bogdanov, Oded Goldreich, and Charles Rackoff for the helpful discussions.

References

[ABI86] Alon, N., Babai, L., Itai, A.: A fast and simple randomized parallel algorithm for the maximal independent set problem. Journal of Algorithms 7, 567–583 (1986)

[AIK05] Applebaum, B., Ishai, Y., Kushilevitz, E.: Computationally private randomizing polynomials and their applications. Computational Complexity 15(2), 115–162 (2006); also CCC 2005

[AIK04] Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography in NC^0. SIAM Journal on Computing (SICOMP) 36(4), 845–888 (2006); also FOCS 2004 (2004)

[BJP11] Bronson, J., Juma, A., Papakonstantinou, P.A.: Limits on the Stretch of Non-adaptive Constructions of Pseudo-Random Generators. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 504–521. Springer, Heidelberg (2011)

[BKW97] Blömer, J., Karp, R., Welzl, E.: The rank of sparse random matrices over finite fields. Random Structures Algorithms 10(4), 407–419 (1997)

[BvzGH82] Borodin, A., von zur Gathen, J., Hopcroft, J.: Fast parallel matrix and GCD computations. Information and Control 52(3), 241–256 (1982)

[GL89] Goldreich, O., Levin, L.A.: A hard-core predicate for all one-way functions. In: Symposium on Theory of Computing (STOC), pp. 25–32 (1989)

[Gol01] Goldreich, O.: Foundations of cryptography. Cambridge University Press, Cambridge (2001); Basic tools (vol. I)

[HHR06a] Haitner, I., Harnik, D., Reingold, O.: Efficient Pseudorandom Generators from Exponentially Hard One-Way Functions. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 228–239. Springer, Heidelberg (2006)

[HHR06b] Haitner, I., Harnik, D., Reingold, O.: On the Power of the Randomized Iterate. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 22–40. Springer, Heidelberg (2006)

[HILL89] Hastad, J., Impagliazzo, R., Levin, L.A., Luby, M.: A pseudorandom generator from any one-way function. SIAM Journal on Computing (SICOMP) 28(4), 1364–1396 (1999); also STOC 1989

[HRV10] Haitner, I., Reingold, O., Vadhan, S.: Efficiency improvements in constructing pseudorandom generators from one-way functions. In: Symposium on Theory of Computing (STOC), pp. 437–446 (2010)

[KGY89] Kharitonov, M., Goldberg, A.V., Yung, M.: Lower bounds for pseudorandom number generators. In: Foundations of Computer Science (FOCS), pp. 242–247 (1989)

[LR87] Luby, M., Rackoff, C.: A study of password security. Journal on Cryptology 1(3), 151–158 (1989); Luby, M., Rackoff, C.: A Study of Password Security. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 392–397. Springer, Heidelberg (1988)

[MS77] MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error-Correcting Codes. North-Holland (1977)

[Muk84] Mukhopadhyay, A.: On the probability that the determinant of an $n \times n$ matrix over a finite field vanishes. Discrete Math. 51(3), 311–315 (1984)

[Mul87] Mulmuley, K.: A fast parallel algorithm to compute the rank of a matrix over an arbitrary field. Combinatorica 7(1), 101–104 (1987)

[Sip83] Sipser, M.: A complexity theoretic approach to randomness. In: Symposium on Theory of Computing (STOC), pp. 330–335 (1983)

[Vad11] Vadhan, S.: Pseudorandomness (April 2011)

Integral Mixed Unit Interval Graphs

Van Bang Le[1] and Dieter Rautenbach[2]

[1] Institut für Informatik, Universität Rostock, Rostock, Germany
le@informatik.uni-rostock.de
[2] Institut für Optimierung und Operations Research, Universität Ulm, Ulm, Germany
dieter.rautenbach@uni-ulm.de

Abstract. We characterize graphs that have intersection representations using unit intervals with open or closed ends such that all ends of the intervals are integral in terms of infinitely many minimal forbidden induced subgraphs. Furthermore, we provide a quadratic-time algorithm that decides if a given interval graph admits such an intersection representation.

Keywords: intersection graph; interval graph; proper interval graph; unit interval graph.

Classification of the topic: Graph Theory, Communication Networks, and Optimization.

1 Introduction

Interval graphs and subclasses like proper interval graphs and unit interval graphs have well studied structural [2, 10] as well as algorithmic [3–5, 12, 13] properties and occur in many applications [1, 9, 11, 14–18]. Interval graphs are the intersection graphs of closed (real) intervals and unit interval graphs are the intersection graphs of closed unit intervals.

As long as intervals of different lengths are allowed, it actually does not matter in the definition of interval graphs whether the ends of the intervals are closed or open. For unit interval graphs, this is no longer true. While Frankl and Maehara [7] proved that unit interval graphs coincide with the intersection graphs of open unit intervals, the intersection graphs of the unit intervals of different types form a strict superclass of unit interval graphs.

In two previous papers we studied the classes of intersection graphs of closed and open unit intervals [19] and of mixed unit intervals [6] where for mixed unit intervals all four combinations for the two ends, namely open-open, closed-closed, open-closed, and closed-open are allowed. Partial results in [6] naturally lead to the problem of characterizing the graphs that have intersection representations using mixed unit intervals where additionally all ends of the intervals are integers.

We refer to such graphs as *integral mixed unit interval graphs*.

Our contributions in the present paper are

– a characterization of twin-free integral mixed unit interval graphs in terms of the complete list of minimal forbidden induced subgraphs, and

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 495–506, 2012.

– a quadratic-time algorithm that decides if a given interval graph is an integral mixed unit interval graph, and if so, outputs a suitable representation.

The paper is organized as follows. In Section 2 we introduce some terminology and notation, give exact definitions, and recall some previous results. In Section 3 we study the forbidden induced subgraphs. In Section 4 we derive structural properties of the maximal cliques of integral mixed unit interval graphs. Section 5 is devoted to the representation algorithm and its analysis. Finally, in Section 6 we combine all results of the earlier sections and prove our main results.

2 Preliminaries

Let $\mathcal{M}$ be a family of sets. An *$\mathcal{M}$-representation of a graph G* is a function $M : V(G) \to \mathcal{M}$ such that for every two distinct vertices u and v of G, we have $uv \in E(G)$ if and only if $M(u) \cap M(v) \neq \emptyset$. A graph is an *$\mathcal{M}$-graph* if it has a $\mathcal{M}$-representation. Two vertices u and v in a graph G are *twins*, if they have the same closed neighborhood, that is, they are adjacent and for every vertex w in $V(G) \setminus \{u, v\}$, the vertices u and w are adjacent if and only if the vertices v and w are adjacent. Note that if u and v are twins in a graph G, then G is an $\mathcal{M}$-graph if and only if $G - u$ is an $\mathcal{M}$-graph. Thus, it suffices to consider twin-free graphs when discussing graphs admitting an $\mathcal{M}$-representation.

For two real numbers x and y, the *open interval* (x, y) is $\{z \in \mathbb{R} \mid x < z < y\}$, the *closed interval* $[x, y]$ is $\{z \in \mathbb{R} \mid x \leq z \leq y\}$, the *open-closed interval* $(x, y]$ is $\{z \in \mathbb{R} \mid x < z \leq y\}$, and the *closed-open interval* $[x, y)$ is $\{z \in \mathbb{R} \mid x \leq z < y\}$. Let

$$\begin{array}{llll}
\mathcal{I}^{--} & = \{(x, y) \mid x, y \in \mathbb{R}, x < y\}, & \mathcal{U}^{--} & = \{(x, x+1) \mid x \in \mathbb{R}\}, \\
\mathcal{I}^{++} & = \{[x, y] \mid x, y \in \mathbb{R}, x \leq y\}, & \mathcal{U}^{++} & = \{[x, x+1] \mid x \in \mathbb{R}\}, \\
\mathcal{I}^{-+} & = \{(x, y] \mid x, y \in \mathbb{R}, x < y\}, & \mathcal{U}^{-+} & = \{(x, x+1] \mid x \in \mathbb{R}\}, \\
\mathcal{I}^{+-} & = \{[x, y) \mid x, y \in \mathbb{R}, x < y\}, & \mathcal{U}^{+-} & = \{[x, x+1) \mid x \in \mathbb{R}\}, \\
\mathcal{I}^{\pm} & = \mathcal{I}^{++} \cup \mathcal{I}^{--}, & \mathcal{U}^{\pm} & = \mathcal{U}^{++} \cup \mathcal{U}^{--}, \\
\mathcal{I} & = \mathcal{I}^{\pm} \cup \mathcal{I}^{-+} \cup \mathcal{I}^{+-}, & \mathcal{U} & = \mathcal{U}^{\pm} \cup \mathcal{U}^{-+} \cup \mathcal{U}^{+-}.
\end{array}$$

We allow arithmetic operations on intervals, that is, for an interval I in $\mathcal{I}$ and two real numbers x and y, we have $xI + y = \{xz + y \mid z \in I\}$.

For an interval I in $\mathcal{I}$, let $\ell(I) = \inf(I)$ and $r(I) = \sup(I)$ denote the left and right end of I, respectively.

A $\mathcal{U}$-representation I of a graph G is *integral* if $\{\ell(I(u)) \mid u \in V(G)\} \subseteq \mathbb{Z}$, that is, all ends of the intervals are integers.

Interval graphs are $\mathcal{I}^{++}$-graphs and *unit interval graphs* are $\mathcal{U}^{++}$-graphs. A graph G is a *proper interval graph* if it has a $\mathcal{I}^{++}$-representation $I : V(G) \to \mathcal{I}^{++}$ for which there are no two vertices u and v of G such that $I(u)$ is a proper subset of $I(v)$. In this case I is a *proper interval representation of G*.

A fundamental result relating these three classes of interval graphs is due to Roberts. Please refer to Figure 1 for an illustration of the Claw $K_{1,3}$.

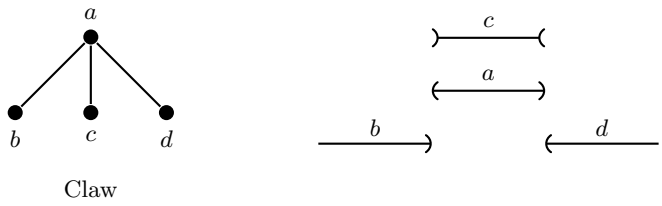


Fig. 1. The Claw $K_{1,3}$ and its four integral $\mathcal{U}$-representations

Theorem 1 (Roberts [20]). *A graph is a unit interval graph if and only if it is a proper interval graph if and only if it is a $K_{1,3}$-free interval graph.*

As mentioned in the introduction, for the definition of interval graphs, the type of the intervals does not make a difference, more precisely, if $\mathcal{M}, \mathcal{N} \in \{\mathcal{I}, \mathcal{I}^{++}, \mathcal{I}^{--}, \mathcal{I}^{+-}, \mathcal{I}^{-+}\}$, then G is a $\mathcal{M}$-graph if and only if G is a $\mathcal{N}$-graph [6,19]. Since the Claw $K_{1,3}$ has a $\mathcal{U}^{\pm}$-representation (cf. Figure 1), the situation is different for unit interval graphs. The main two results from [6,19] are the following. Please refer to Section 3 and Figure 2 for the definition and illustration of all graphs mentioned in these results.

Theorem 2 (Rautenbach and Szwarcfiter [19]). *A twin-free graph is a $\mathcal{U}^{\pm}$-graph if and only if it is a $\{R_0, Q_1, D_3, D_5\}$-free interval graph.*

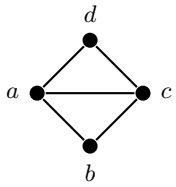


Fig. 2. The Diamond

Theorem 3 (Dourado et al. [6]). *For a diamond-free graph G, the following statements are equivalent.*

(1) *G is a $\{R_k \mid k \in \mathbb{N}_0\}$-free interval graph.*
(2) *G has an integral $\mathcal{U}$-intersection representation.*
(3) *G is a $\mathcal{U}$-graph.*

Mainly the last theorem motivated the characterization problem of the graphs that have an integral $\mathcal{U}$-representation.

3 Forbidden Induced Subgraphs

Let $\mathcal{G}$ be the class of twin-free integral mixed unit interval graphs, that is, of those twin-free graphs that have an integral $\mathcal{U}$-representation.

Let $G \in \mathcal{G}$ and let $I : V(G) \to \mathcal{U}$ be an integral $\mathcal{U}$-representation of G. For a vertex u of G, let $c(u)$ denote the number of distinct maximal cliques of G that contain u.

Since G is twin-free, the function I is necessarily injective. Hence, if H is an induced subgraph of G, then the restriction of I to $V(H)$ is an injective integral $\mathcal{U}$-representation of H, that is, even if H is not twin-free, there is an integral $\mathcal{U}$-representation of H that assigns different intervals to the vertices of H. Therefore, the minimal forbidden induced subgraphs for $\mathcal{G}$ are exactly those graphs that do not have an injective integral $\mathcal{U}$-representation while every proper induced subgraph has.

The integrality of the representation I immediately implies that every vertex u of G belongs to at most three maximal cliques of G, that is,

$(\mathcal{C}_1)$ $c(u) \leq 3$ *for every vertex* u *of* G.

where $c(u) = 3$ implies that $I(u)$ is necessarily a closed interval.

Lemma 1. *The graphs* R_0, D_1, D_2, *and* D_3 *are minimal forbidden induced subgraphs for* $\mathcal{G}$.

Proof. It follows from $(\mathcal{C}_1)$ that R_0, D_1, and D_2 are forbidden induced subgraphs for $\mathcal{G}$, because $c(a) > 3$ for each of these graphs. Suppose D_3 is an induced subgraph of some $G \in \mathcal{G}$, labelled as in Figure 4. Since G has no twins, we may assume, by symmetry, that there exists a vertex b' adjacent to a and non-adjacent to b. Now $c(a) > 3$, contradicting $(\mathcal{C}_1)$. Thus, D_3 is also a forbidden induced subgraph for $\mathcal{G}$. Finally, it is easy to verify that for each $H \in \{R_0, D_1, D_2, D_3\}$ and every $v \in V(H)$, the graph $H - v$ has an injective integral $\mathcal{U}$-representation.

Lemma 2. *The graphs* S_0, T_0, K_5, *and* D_4 *are minimal forbidden induced subgraphs for* $\mathcal{G}$.

Lemma 3. *The graph* D_5 *is a minimal forbidden induced subgraph for* $\mathcal{G}$.

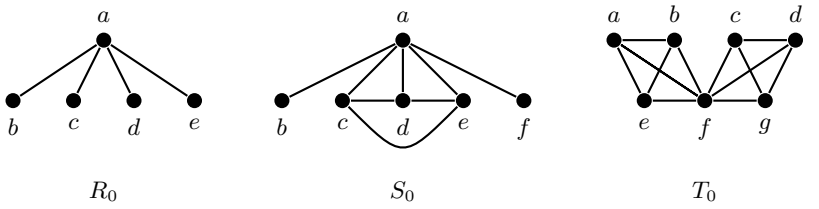


Fig. 3. The graphs R_0, S_0, and T_0

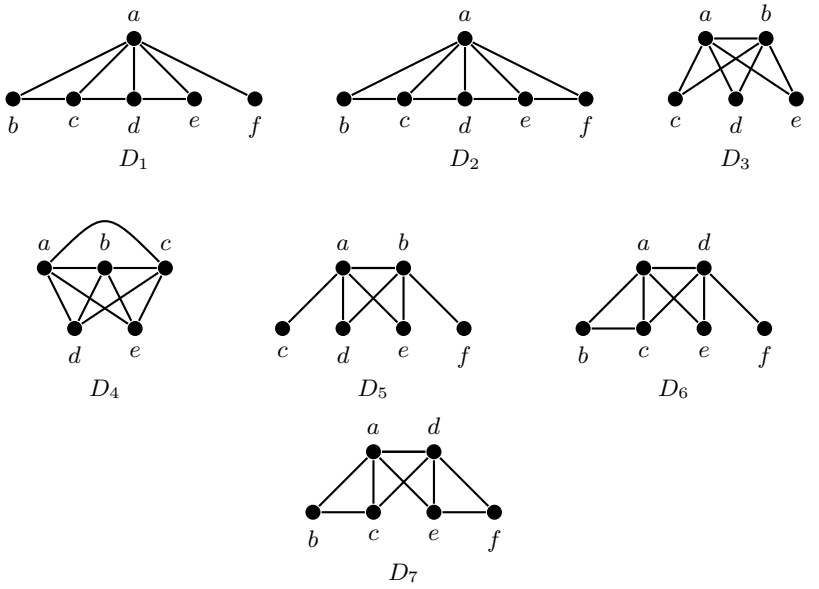


Fig. 4. The graphs D_1 to D_7

Lemma 4. *The graphs D_6 and D_7 are minimal forbidden induced subgraphs for $\mathcal{G}$.*

We now describe three infinite sequences of forbidden induced subgraphs for $\mathcal{G}$.

For $k \in \mathbb{N}$, let the graph Q_k arise from a path $a_0a_1a_2 \dots a_{k+1}$ by adding the vertices $b_1, b_2, \dots, b_{k+1}$, the edges a_ib_i for $1 \leq i \leq k+1$, and the edges $a_{i-1}b_i$ for $2 \leq i \leq k+1$. See Figure 5 for an illustration.
For $k \in \mathbb{N}$,

- let $\tilde{Q}_k$ arise from Q_k by adding a vertex b_0 and four edges a_0b_0, a_0b_1, a_1b_0, and b_0b_1,
- let R_k arise from Q_k by adding two vertices a_{k+2} and b_{k+2} and two edges $a_{k+1}a_{k+2}$ and $a_{k+1}b_{k+2}$,

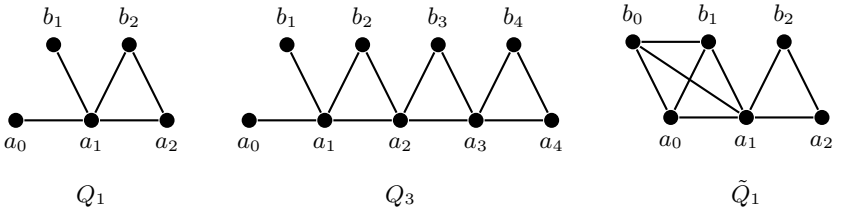


Fig. 5. The graphs Q_1, Q_3, and $\tilde{Q}_1$

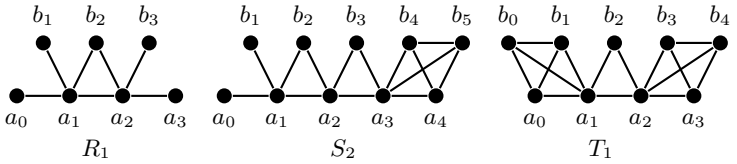


Fig. 6. The graphs R_1, S_2, and T_1

- let S_k arise from Q_k by adding three vertices a_{k+2}, b_{k+2}, and b_{k+3} and six edges $a_{k+1}a_{k+2}$, $a_{k+1}b_{k+2}$, $a_{k+1}b_{k+3}$, $a_{k+2}b_{k+2}$, $a_{k+2}b_{k+3}$, and $b_{k+2}b_{k+3}$, and
- let T_k arise from $\tilde{Q}_k$ by adding three vertices a_{k+2}, b_{k+2}, and b_{k+3} and six edges $a_{k+1}a_{k+2}$, $a_{k+1}b_{k+2}$, $a_{k+1}b_{k+3}$, $a_{k+2}b_{k+2}$, $a_{k+2}b_{k+3}$, and $b_{k+2}b_{k+3}$.

See Figures 5 and 6 for an illustration.

For $k \in \mathbb{N}$, the two vertices a_{k+1} and b_{k+1} are called the *special vertices* of Q_k and $\tilde{Q}_k$, respectively.

Lemma 5. *The graphs R_k, S_k, and T_k for $k \in \mathbb{N}$ are minimal forbidden induced subgraphs for $\mathcal{G}$.*

4 Properties of Maximal Cliques

Throughout this section, let G be a fixed connected twin-free interval graph. It is well-known [8] that there is a linear ordering of the maximal cliques of G, say $\mathcal{C} = (C_1, \ldots, C_q)$, such that every vertex of G belongs to maximal cliques that are consecutive in that ordering, that is, for every vertex u of G, there are indices $\ell(u)$ and $r(u)$ with

$$\{i \mid 1 \leq i \leq q \text{ and } u \in C_i\} = \{i \mid \ell(u) \leq i \leq r(u)\}.$$

Note that this linear ordering is unique up to reversal and that the number $c(u)$ of distinct maximal cliques of G that contain u equals $r(u) + 1 - \ell(u)$. Hence a vertex u of G is simplicial if and only if $c(u) = 1$ if and only if $\ell(u) = r(u)$.

If C and D are distinct maximal cliques of G, then $C \setminus D$ and $D \setminus C$ are both not empty, that is, for every $j \in \{1, \ldots, q\}$, there are vertices u and v such that $r(u) = \ell(v) = j$. This also implies that C_1 and C_q contain simplicial vertices.

Note that, since G is twin-free, there are no two distinct vertices u and v with $\ell(u) = \ell(v)$ and $r(u) = r(v)$.

The purpose of the present section is to derive the following structural properties of the sequence $\mathcal{C}$ that are implied by forbidding certain induced subgraphs from Section 3.

($\mathcal{C}_1$) $c(u) \leq 3$ for every vertex u of G.
($\mathcal{C}_2$) There are no two vertices u and v of G with $c(u) = c(v) = 3$ and $r(v) - r(u) = 1$.

Lemma 6. *Let G, $\mathcal{C}$, and $\ell(u)$, $r(u)$, and $c(u)$ for every vertex u of G be as above.*

(i) *If G is $\{R_0, D_1, D_2, D_3, D_4\}$-free, then ($\mathcal{C}_1$) holds.*
(ii) *If G is $\{D_5, D_6, D_7\}$-free, then ($\mathcal{C}_2$) holds.*

Proof. (i) For contradiction, we assume that $c(u_1) \geq 4$ for some vertex u_1 of G. Let $i = \ell(u_1)$. Note that $r(u_1) \geq i + 3$. Let the vertices u_2, u_3, u_4, and u_5 be such that $r(u_2) = i$, $\ell(u_3) = i + 3$, $r(u_4) = i + 1$, and $\ell(u_5) = i + 2$. Since G is R_0-free, we may assume, by symmetry, that $\ell(u_4) \leq i$. Let the vertex u_6 be such that $\ell(u_6) = i + 1$.

First, we assume that $r(u_5) = i + 2$. Since G is R_0-free, this implies $r(u_6) \geq i + 2$. If $r(u_6) = i + 2$, then $G[\{u_1, \ldots, u_6\}]$ is D_1, and, if $r(u_6) \geq i + 3$, then $G[\{u_1, u_3, \ldots, u_6\}]$ is D_3, which is a contradiction. Hence we may assume that $r(u_5) \geq i + 3$. Let the vertex u_7 be such that $r(u_7) = i + 2$.

If $r(u_6) = i + 2$, then $G[\{u_1, \ldots, u_6\}]$ is D_2, which is a contradiction. If $r(u_6) \geq i + 3$, then $G[\{u_1, u_3, u_5, u_6, u_7\}]$ is D_4, which is a contradiction. Hence we may assume that $r(u_6) = i + 1$ and, by symmetry, $\ell(u_7) = i + 2$. Now $G[\{u_1, u_2, u_3, u_6, u_7\}]$ is R_0, which is a contradiction. This completes the proof of (i).

(ii) For contradiction, we assume that the vertices u_1 and u_2 are such that $c(u_1) = c(u_2) = 3$ and $r(u_2) = r(u_1) + 1$.

Let $i = \ell(u_1)$. Let the vertices u_3, u_4, u_5, and u_6 be such that $r(u_3) = i$, $\ell(u_4) = i + 3$, $r(u_5) = i + 1$, and $\ell(u_6) = i + 2$. Now $G[\{u_1, \ldots, u_6\}]$ is one of the graphs D_5, D_6, and D_7, which is a contradiction. This completes the proof of (ii).

5 The Representation Algorithm

Throughout this section, let G be a fixed connected twin-free interval graph that is not a clique. Let $\mathcal{C} = (C_1, \ldots, C_q)$ and $\ell(u)$, $r(u)$, and $c(u)$ for every vertex u of G be as in the first paragraph of Section 4. Since G is not a clique, we have $q \geq 2$.

In this section, we describe and analyze the algorithm `IntMixUniIntRep` that, given $\mathcal{C}$ as input, produces a function $I : V(G) \to \mathcal{U}$ such that $\ell(I(u)) \in \mathbb{Z}$ for every vertex u of G. We prove that I is an integral $\mathcal{U}$-representation of G provided that $\mathcal{C}$ satisfies certain structural properties and G does not contain certain induced subgraphs. The algorithm works essentially in two phases:

- In a first phase, the algorithm determines a path $P : v_0 \ldots v_{k+1}$ in G (cf. `ClosedVertices`). To the vertices of this path it assigns the intervals $I(v_i) = [i, i + 1]$ for $i \in \{0, \ldots, k + 1\}$ (cf. line 2 of `IntMixUniIntRep`).

– In a second phase, it processes the maximal cliques of G according to the ordering given by $\mathcal{C}$ (cf. line 3 of `IntMixUniIntRep`). When it processes the maximal clique C_i, then $I(u)$ is defined for all vertices u of G with $\ell(u) = i$, that is, it specifies the unit interval for those vertices that appear in C_i for the first time and do not belong to P. (cf. line 4 of `IntMixUniIntRep`).

Recall that a vertex u is simplicial if and only if $c(u) = 1$.

Procedure `ClosedVertices`

1. let v_0 be a simplicial vertex in C_1
2. $i := 0$; $j := 1$
3. **repeat**
4. $\quad i := i + 1$
5. $\quad$ let v_i be a vertex in $C_j \setminus \{v_0, \ldots, v_{i-1}\}$ with maximum $r(v)$
6. $\quad j := r(v_i)$
7. **until** $j = q$
8. $k := i$
9. let v_{k+1} be a simplicial vertex in C_q

If $i \in \{0, \ldots, k-1\}$, then $r(v_i) < q$ and the connectivity of G implies $r(v_{i+1}) > r(v_i)$. This implies that `ClosedVertices` necessarily terminates. Clearly, by the choice of the vertices, $P : v_0 v_1 \ldots v_{k+1}$ is a path in G.

Lemma 7. *If $\mathcal{C}$ satisfies $(\mathcal{C}_1)$ and $(\mathcal{C}_2)$, then the vertices $v_0, \ldots, v_{k+1}$ selected by `ClosedVertices` satisfy the following properties.*

(i) *$r(v_i) = \ell(v_{i+1})$ for $i \in \{0, \ldots, k\}$.*
(ii) *The vertices $v_0, \ldots, v_{k+1}$ are uniquely determined.*
(iii) *Each maximal clique C_j of G contains one or two vertices from $V(P)$. Furthermore,*
- *if C_j contains only one vertex from $V(P)$, say v_i, then $j \in \{2, \ldots, q-1\}$, $\ell(v_i) = j - 1$, and $r(v_i) = j + 1$, and*
- *if C_j contains two vertices from $V(P)$, then $C_j \cap V(P) = \{v_i, v_{i+1}\}$ for some $i \in \{0, \ldots, k\}$ and $r(v_i) = \ell(v_{i+1}) = j$.*

(iv) *$c(u) \leq 2$ for every vertex u in $V(G) \setminus V(P)$.*
(v) *For every $j \in \{1, \ldots, q\}$, there are at most two vertices u with $\ell(u) = j$ that do not belong $V(P)$, that is, $|C_j \setminus (C_{j-1} \cup V(P))| \leq 2$ for every $j \in \{1, \ldots, q\}$. Furthermore, if $C_j \setminus (C_{j-1} \cup V(P))$ contains two distinct vertices u and v for some $j \in \{1, \ldots, q\}$, then $j \in \{2, \ldots, q-1\}$ and $\{c(u), c(v)\} = \{1, 2\}$.*

Proof. (i) For $i = 0$ or $i = k$, the desired statement follows easily because v_0 and v_{k+1} are simplicial vertices with $\ell(v_0) = r(v_0) = 1$ and $\ell(v_{k+1}) = r(v_{k+1}) = q$. Now let $i \in \{1, \ldots, k-1\}$. By line 5 of `ClosedVertices`, we have $r(v_i) \geq \ell(v_{i+1})$. Therefore, for contradiction, we assume that $r(v_i) > \ell(v_{i+1})$. As noted above, we have $r(v_{i+1}) > r(v_i)$. By the choice of v_i, this implies $\ell(v_i) < \ell(v_{i+1})$. By $(\mathcal{C}_1)$ and since G is twin-free, this implies that $c(v_i) = c(v_{i+1}) = 3$ and $r(v_i) = \ell(v_{i+1}) + 1$, which yields a contradiction to $(\mathcal{C}_2)$.

(ii) Since G is twin-free, each of the cliques C_1 and C_q contains exactly one simplicial vertex, which implies that v_0 and v_{k+1} are uniquely determined. If v_i has already been determined and $r(v_i) < q$, then $i \leq k$. Now part (i) and the twin-freeness of G imply that v_{i+1} is uniquely determined.

(iii) This follows immediately from part (i) and the observation $r(v_{i+1}) > r(v_i)$ for $i \in \{0, \ldots, k-1\}$.

(iv) In view of $(\mathcal{C}_1)$, we may assume, for contradiction, that $c(u) = 3$ for some $u \in V(G) \setminus V(P)$. Let $j = \ell(u)$.

If there is exactly one vertex v_i from $V(P)$ with $v_i \in C_j$, then part (iii) implies $\ell(v_i) < j < r(v_i)$. This implies $c(v_i) = 3$ and $r(u) = r(v_i) + 1$, which yields a contradiction to $(\mathcal{C}_2)$.

If there are two vertices from $V(P)$ in C_j, then part (iii) implies that $C_j \cap V(P) = \{v_i, v_{i+1}\}$ for some $i \in \{0, \ldots, k\}$ such that $j = \ell(v_{i+1})$. Since G is twin-free, this implies $c(v_{i+1}) \leq 2$, which yields a contradiction to the choice of v_{i+1}.

(v) This follows from part (iv) and the twin-freeness of G.

Lemma 8. *Let $\mathcal{C}$ satisfy $(\mathcal{C}_1)$ and $(\mathcal{C}_2)$ and let I be the function defined by `IntMixUniIntRep`.*

(i) *For every two distinct vertices u and v of G, if $\{u, v\} \cap V(P) \neq \emptyset$, then $uv \in E(G)$ if and only if $I(u) \cap I(v) \neq \emptyset$.*
(ii) *For every two distinct vertices u and v of G, if $uv \in E(G)$, then $I(u) \cap I(v) \neq \emptyset$.*

Lemma 9. *Let $\mathcal{C}$ satisfy $(\mathcal{C}_1)$ and $(\mathcal{C}_2)$. Just after an execution of line 16 of `IntMixUniIntRep` that defines $I(u)$, there is an induced subgraph H of G such that*

- *$\ell(v) < \ell(u)$ for every $v \in V(H) \setminus \{u, v_{i+1}\}$, that is, $I(v)$ is already defined for every vertex v of H,*
- *$u, v_i, v_{i+1} \in V(H)$,*
- *H is*
  - *either K_4,*
  - *or Q_k for some $k \in \mathbb{N}$ such that u and v_{i+1} are the special vertices of H,*
  - *or $\tilde{Q}_k$ for some $k \in \mathbb{N}$ such that u and v_{i+1} are the special vertices of H.*

Proof. We prove the statement by induction on j where j and i are as in the considered execution of line 16 of `IntMixUniIntRep`.

Algorithm `IntMixUniIntRep`

```
 1. run ClosedVertices to compute P : v_0 ... v_{k+1}
 2. for i := 0 to k+1 do I(v_i) := [i, i+1]
 3. for j := 1 to q do
 4.   for each u ∈ V(G) \ V(P) with ℓ(u) = j do
 5.     if |C_j ∩ V(P)| = 1 then
 6.       let C_j ∩ V(P) = {v_i} for some i ∈ {0, ..., k+1}
 7.       if u is simplicial
 8.       then I(u) := (i, i+1)
 9.       else I(u) := (i, i+1]
10.     endif
11.     if |C_j ∩ V(P)| = 2 then
12.       let C_j ∩ V(P) = {v_i, v_{i+1}} for some i ∈ {0, ..., k}
13.       if u is simplicial then
14.         if there is no v ∈ V(G) \ V(P) with ℓ(I(v)) = i
15.         then I(u) := (i, i+1]
16.         else I(u) := [i+1, i+2)
17.       else I(u) := [i+1, i+2)
18.     endif
19.   endfor
20. endfor
```

In view of line 14 of `IntMixUniIntRep`, just before the considered execution of line 16 of `IntMixUniIntRep`, there is a vertex $v \in V(G) \setminus V(P)$ with $\ell(I(v)) = i$, which implies that $\ell(v) < \ell(u)$ and $j, i \geq 1$. By Lemma 7, we have $\ell(v_i) \in \{j-1, j-2\}$. By Lemma 8, the vertex v is adjacent to v_i, that is, v and v_i both lie in a maximal clique of G. If $\ell(v) < \ell(v_i)$, then, by Lemma 7, $\ell(v_{i-1}) = \ell(v)-1$ and $\ell(v_i) = \ell(v)+1$ and in view of `IntMixUniIntRep` we obtain $I(v) = (i-1, i]$, which is a contradiction. Hence $\ell(v) \geq \ell(v_i)$. If $v \in C_j$, then $H = G[\{v_i, v_{i+1}, u, v\}]$ is K_4. Hence, we may assume that $r(v) < j = \ell(u)$.

If $\ell(v) > \ell(v_i)$, then, by Lemma 7, $\ell(v_i) = j-2$, $\ell(v) = r(v) = j-1$, and $|C_{j-1} \cap V(P)| = 1$. Now $H = G[\{v_{i-1}, v_i, v_{i+1}, v, u\}]$ is Q_1 such that u and v_{i+1} are the special vertices of H. Hence we may assume that $\ell(v) = \ell(v_i)$.

If $c(v) = 2$, then, by Lemma 7, $\ell(v_i) = j-2$, $\ell(v) = j-2$, $r(v) = j-1$, and $|C_{j-2} \cap V(P)| = 2$. Let the vertex w be such that $\ell(w) = j-1$. If $r(w) = j-1$, then $H = G[\{v_{i-1}, v_i, v_{i+1}, w, u\}]$ is Q_1 such that u and v_{i+1} are the special vertices of H. If $r(w) > j-1$, then $H = G[\{v_i, v_{i+1}, w, u\}]$ is K_4. Hence, we may assume that $c(v) = 1$.

Since $\ell(I(v)) = i$, we obtain that $I(v)$ was defined by an earlier execution of line 16 of `IntMixUniIntRep`. By induction, this implies that, just after $I(v)$ was defined, there was an induced subgraph H' of G with the desired properties. Now $H = G[\{V(H') \cup \{u, v_{i+1}\}\}]$ has the desired properties.

Lemma 10. *Let $\mathcal{C}$ satisfy $(\mathcal{C}_1)$ and $(\mathcal{C}_2)$ and let I be the function defined by `IntMixUniIntRep`.*

If G is $\{K_5\} \cup \{R_i \mid i \in \mathbb{N}\} \cup \{S_i \mid i \in \mathbb{N}_0\} \cup \{T_i \mid i \in \mathbb{N}_0\}$-free, then I is an integral $\mathcal{U}$-representation of G.

6 Harvest

In this section we prove our two main results.

Theorem 4. *If G is a twin-free connected interval graph that is not a clique, and $\mathcal{C}$ is as in the first paragraph of Section 4, then the following statements are equivalent.*

(1) *G is $\{D_1, \ldots, D_7\} \cup \{K_5\} \cup \{R_i \mid i \in \mathbb{N}_0\} \cup \{S_i \mid i \in \mathbb{N}_0\} \cup \{T_i \mid i \in \mathbb{N}_0\}$-free.*
(2) *$\mathcal{C}$ satisfies $(\mathcal{C}_1)$ and $(\mathcal{C}_2)$ and G is $\{K_5\} \cup \{R_i \mid i \in \mathbb{N}\} \cup \{S_i \mid i \in \mathbb{N}_0\} \cup \{T_i \mid i \in \mathbb{N}_0\}$-free.*
(3) *G has an integral $\mathcal{U}$-representation.*

Proof. By Lemma 6, the first statement implies the second. By Lemma 10, the second statement implies the third. Finally, by Lemmas 1, 2, 3, 4, and 5, the third statement implies the first.

It is straighforward yet tedious to derive from Theorem 4 the complete list of all minimal forbidden induced subgraphs of integral mixed unit interval graphs by considering the forbidden induced subgraphs of interval graphs and all minimal twin-free supergraphs of the graphs mentioned in (1) of Theorem 4. We leave the details to the reader.

Furthermore, Theorem 4 directly implies Theorem 3: Let G be a diamond-free graph satisfying (1) of Theorem 3. Note that we may assume that G is twin-free. This easily implies that G is K_4-free. Hence G satisfies (1) in Theorem 4, and therefore G satisfies (2) in Theorem 3. In Theorem 3, the implication (2) $\Rightarrow$ (3) is trivial and the implication (3) $\Rightarrow$ (1) follows by noting that all R_k are forbidden induced subgraphs even for the class of $\mathcal{U}$-graphs.

Theorem 5. *There is a quadratic-time algorithm that, given an interval graph G, decides if G has an integral $\mathcal{U}$-representation, and if so, outputs such a representation for G.*

Proof. Let G be an interval graph. Since all twins of G can be detected in time $O(|V(G)|^2)$, we may assume that G is twin-free. Note that a linear ordering $\mathcal{C} = (C_1, \ldots, C_q)$ of the maximal cliques of G can be computed in linear time (cf. [10]). If some C_j has more than four vertices, G does not have an integral $\mathcal{U}$-representation (cf. Lemma 2). Otherwise, we compute $c(u), \ell(u)$, and $r(u)$ for $u \in V(G)$ in linear time in an obvious way, and run `IntMixUniIntRep` to get the function I. Note that, since $|C_j| \leq 4$ for all j, `IntMixUniIntRep` has linear running time. Now we test whether I is an intersection representation of G or not by constructing the graph $H = (V(G), \{uv \mid I(u) \cap I(v) \neq \emptyset\})$ and checking if $G = H$, that is, checking $N_G(v) = N_H(v)$ for all $v \in V(G)$. This can be done in time $O(|V(G)|^2)$. If $G = H$, then I is an integral $\mathcal{U}$-representation of G. Otherwise, Lemma 10 and Theorem 4 imply that G has no such a representation.

References

1. Halldórsson, M.M., Patt-Shamir, B., Rawitz, D.: Online Scheduling with Interval Conflicts, in. In: Proceedings of the 28th Annual Conference on Theoretical Aspects of Computer Science (STACS), pp. 472–483 (2011)
2. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (1999)
3. Corneil, D.G., Kim, H., Natarajan, S., Olariu, S., Sprague, A.P.: Simple linear time recognition of unit interval graphs. Inf. Process. Lett. 55, 99–104 (1995)
4. Corneil, D.G., Olariu, S., Stewart, L.: The ultimate interval graph recognition algorithm? (Extended abstract). In: Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 175–180 (1998)
5. Corneil, D.G.: A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs. Discrete Appl. Math. 138, 371–379 (2004)
6. Dourado, M.C., Le, V.B., Protti, F., Rautenbach, D., Szwarcfiter, J.L.: Mixed unit interval graphs. manuscript (2011)
7. Frankl, P., Maehara, H.: Open-interval graphs versus closed-interval graphs. Discrete Math. 63, 97–100 (1987)
8. Fulkerson, D.R., Gross, O.A.: Incidence matrices and interval graphs. Pacific J. Math. 15, 835–855 (1965)
9. Goldberg, P.W., Golumbic, M.C., Kaplan, H., Shamir, R.: Four strikes against physical mapping of DNA. J. Comput. Biol. 2, 139–152 (1995)
10. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs, Amsterdam, The Netherlands. Annals of Discrete Mathematics, vol. 57 (2004)
11. Heggernes, P., Suchan, K., Todinca, I., Villanger, Y.: Characterizing Minimal Interval Completions. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 236–247. Springer, Heidelberg (2007)
12. Hell, P., Shamir, R., Sharan, R.: A fully dynamic algorithm for recognizing and representing proper interval graphs. SIAM J. Comput. 31, 289–305 (2001)
13. Herrera de Figueiredo, C.M., Meidanis, J., Picinin de Mello, C.: A linear-time algorithm for proper interval graph recognition. Inf. Process. Lett. 56, 179–184 (1995)
14. Kaplan, H., Shamir, R.: Pathwidth, bandwidth, and completion problems to proper interval graphs with small cliques. SIAM J. Comput. 25, 540–561 (1996)
15. Kendall, D.G.: Incidence matrices, interval graphs, and seriation in archaeology. Pacific J. Math. 28, 565–570 (1969)
16. Kratsch, D., Stewart, L.: Approximating Bandwidth by Mixing Layouts of Interval Graphs. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 248–258. Springer, Heidelberg (1999)
17. Krokhin, A.A., Jeavons, P.G., Jonsson, P.: The Complexity of Constraints on Intervals and Lengths. In: Alt, H., Ferreira, A. (eds.) STACS 2002. LNCS, vol. STACS 2002, pp. 443–454. Springer, Heidelberg (2002)
18. Papadimitriou, C.H., Yannakakis, M.: Scheduling interval-ordered tasks. SIAM J. Comput. 8, 405–409 (1979)
19. Rautenbach, D., Szwarcfiter, J.L.: Unit Interval Graphs - A Story with Open Ends. In: European Conference on Combinatorics, Graph Theory and Applications (EuroComb 2011). Electronic Notes in Discrete Mathematics, vol. 38, pp. 737–742 (2011)
20. Roberts, F.S.: Indifference graphs. In: Harary, F. (ed.) Proof Techniques in Graph Theory, pp. 139–146. Academic Press (1969)

Complementary Vertices and Adjacency Testing in Polytopes

Benjamin A. Burton*

School of Mathematics and Physics, The University of Queensland,
Brisbane QLD 4072, Australia
bab@maths.uq.edu.au

Abstract. Our main theoretical result is that, if a simple polytope has a pair of complementary vertices (i.e., two vertices with no facets in common), then it has a second such pair. Using this result, we improve adjacency testing for vertices in both simple and non-simple polytopes: given a polytope in the standard form $\{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq 0\}$ and a list of its V vertices, we describe an $O(n)$ test to identify whether any two given vertices are adjacent. For simple polytopes this test is perfect; for non-simple polytopes it may be indeterminate, and instead acts as a filter to identify non-adjacent pairs. Our test requires an $O(n^2V + nV^2)$ precomputation, which is acceptable in settings such as all-pairs adjacency testing. These results improve upon the more general $O(nV)$ combinatorial and $O(n^3)$ algebraic adjacency tests from the literature.

Keywords: polytopes, complementary vertices, disjoint facets, adjacent vertices, vertex enumeration, double description method.

1 Introduction

Two vertices of a polytope are *complementary* if they do not belong to a common facet. Complementary vertices play an important role in the theory of polytopes; for instance, they provide the setting for the d-step conjecture [9,10] (now recently disproved [15]), and in the dual setting of disjoint facets they play a role in the classification of compact hyperbolic Coxeter polytopes [6]. In game theory, Nash equilibria of bimatrix games are described by an analogous concept of complementary vertices in *pairs* of polytopes [8,16].

Our first main contribution, presented in Section 2, relates to the minimal number of complementary vertex pairs. Many polytopes have no pairs of complementary vertices at all (for instance, any neighbourly polytope). However, we prove here that if a simple polytope P of dimension $d > 1$ has at least one pair of complementary vertices, then it must have at least *two* such pairs.

The proof involves the construction of paths through a graph whose nodes represent pairs of complementary or "almost complementary" vertices of P. In this sense it is reminiscent of the Lemke-Howson algorithm for constructing

* Supported by the Australian Research Council (project DP1094516).

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 507–518, 2012.

Nash equilibria in bimatrix games [12], although our proof operates in a less well-controlled setting. We discuss this relationship further in Section 4.

Our second main contribution, presented in Section 3, is algorithmic: we use our first theorem to build a fast adjacency test. Specifically, given a polytope in the standard form $P = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq 0\}$ with V vertices, we begin with an $O(n^2V + nV^2)$ time precomputation step, after which we can test any two vertices for adjacency in $O(n)$ time. If P is simple (which the algorithm can also identify) then this test always gives a precise response; otherwise it may be indeterminate but it can still assist in identifying non-adjacent pairs.

The key idea is, for each pair of vertices $u, v \in P$, to compute the join $u \vee v$; that is, the minimal face containing both u and v. If P is simple then our theorem on complementary vertices shows that $u \vee v = u' \vee v'$ for a second pair of vertices u', v'. Our algorithm then identifies such "duplicate" joins.

Although the precomputation is significant, if we are testing all $\binom{V}{2}$ pairs of vertices for adjacency then it does not increase the overall time complexity. Our $O(n)$ test then becomes extremely fast, outperforming the standard $O(nV)$ combinatorial and $O(n^3)$ algebraic tests from the literature [7]. Even in the non-simple setting, our test can be used as a fast pre-filter to identify non-adjacent pairs of vertices, before running the more expensive standard tests on those pairs that remain.

In Section 4 we discuss these performance issues further, as well as the application of these ideas to the key problem of polytope vertex enumeration.

All time complexities are measured using the arithmetic model of computation, where we treat each arithmetical operation as constant-time.

We briefly remind the reader of the necessary terminology. Following Ziegler [17], we insist that all polytopes be bounded. A *facet* of a d-dimensional polytope P is a $(d-1)$-dimensional face, and two vertices of P are *adjacent* if they are joined by an edge. P is *simple* if every vertex belongs to precisely d facets (i.e., every vertex figure is a $(d-1)$-simplex), and P is *simplicial* if every facet contains precisely d vertices (i.e., every facet is a $(d-1)$-simplex). As before, two vertices of P are *complementary* if they do not belong to a common facet; similarly, two facets of P are *disjoint* if they do not contain a common vertex.

2 Complementary Vertices

In this section we prove the main theoretical result of this paper:

Theorem 1. *Let P be a simple polytope of dimension $d > 1$. If P has a pair of complementary vertices, then P has at least two distinct pairs of complementary vertices.*

Note that these pairs of vertices are distinct, but need not be disjoint: they might be of the form $\{u, v\}$ and $\{u, w\}$ for some vertices $u, v, w \in P$.

The proof of Theorem 1 involves an auxiliary graph, which we now describe. To avoid confusion with vertices and edges of polytopes, we describe graphs in terms of *nodes* and *arcs*. We do not allow graphs to have loops or multiple edges.

Definition 2 (Auxiliary graph). *Let P be a simple polytope of dimension $d > 1$. We construct the* auxiliary graph *$\Gamma(P)$ as follows:*

- *The* nodes *of $\Gamma(P)$ are unordered pairs of vertices $\{u, v\}$ of P where u, v have at most one facet in common. We say the node $\{u, v\}$ is of* type A *if the vertices $u, v \in P$ are complementary (they have no facets in common), or of* type B *otherwise (they have precisely one facet in common).*
- *The* arcs *of $\Gamma(P)$ join nodes of the form $\{u, x\}$ and $\{u, y\}$, where x and y are adjacent vertices of P, and where no single facet of P contains all three vertices u, x, y.*

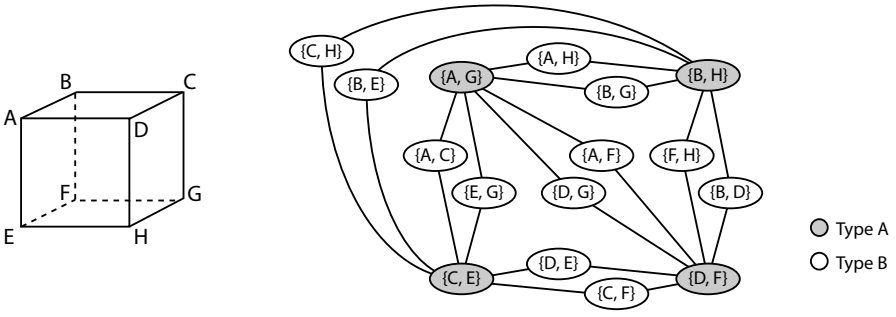


Fig. 1. A polytope P and the corresponding auxiliary graph $\Gamma(P)$

Figure 1 illustrates this graph for the case where P is a cube. Informally, each arc of $\Gamma(P)$ modifies a node by "moving" one of its two vertices along an edge of P, so that if the two vertices lie on a common facet F then this movement is away from F. More formally, we can characterise the arcs of $\Gamma(P)$ as follows:

Lemma 3. *Let $\nu = \{u, v\}$ be a node of $\Gamma(P)$ as outlined above.*

(i) If ν is of type A, there are precisely $2d$ arcs meeting ν. These include d arcs that connect ν with $\{u, x\}$ for every vertex x adjacent to v in P, and d arcs that connect ν with $\{y, v\}$ for every vertex y adjacent to u in P.

(ii) If ν is of type B, there are precisely two arcs meeting ν. Let F be the unique facet of P containing both u and v. Then these two arcs join ν with $\{u, x\}$ and $\{y, v\}$, where x is the unique vertex of P adjacent to v for which $x \notin F$, and y is the unique vertex of P adjacent to u for which $y \notin F$.

Proof. We consider the type A and B cases in turn.

(i) Let $\nu = \{u, v\}$ be of type A, and let x be any vertex of P adjacent to v. Since ν is of type A, u and v have no facets in common. Since P is simple, at most one facet of P contains x but not v. Therefore u and x have at most one facet in common, and so $\{u, x\}$ is a node of $\Gamma(P)$.

Since $\{u, x\}$ is a node of $\Gamma(P)$ and no facet contains both u and v, it follows that $\nu = \{u, v\}$ and $\{u, x\}$ are joined by an arc. A similar argument applies to nodes $\{y, v\}$ where y is any vertex of P adjacent to u. Because P is simple there are precisely d vertices adjacent to v and d vertices adjacent to u, yielding precisely $2d$ arcs of this type.

(ii) Now let $\nu = \{u, v\}$ be of type B, and let F be the unique facet of P containing both u and v. If there is an arc from ν to any node of the form $\{u, x\}$, it is clear from Definition 2 that we must have x adjacent to v and $x \notin F$. Because P is simple, there is precisely one x with these properties.

We now show that an arc from ν to $\{u, x\}$ does indeed exist. Because P is simple, at most one facet of P contains x but not v. The vertex v in turn has only the facet F in common with u; since $x \notin F$ it follows that u and x have at most one facet in common. Therefore $\{u, x\}$ is a node of $\Gamma(P)$. Because $x \notin F$ there is no facet containing all of u, v and x, and so the arc from ν to $\{u, x\}$ exists.

A similar argument applies to the arc from ν to $\{y, v\}$, yielding precisely two arcs that meet ν as described in the lemma statement.

To finish, we note that every vertex belongs to $d > 1$ facets, and so every node $\{u, v\}$ of $\Gamma(P)$ has $u \neq v$. This ensures that we do not double-count arcs in our argument; that is, the $2d$ arcs in case (i) join ν to $2d$ distinct nodes of $\Gamma(P)$, and likewise the two arcs in case (ii) join ν to two distinct nodes of $\Gamma(P)$. □

Our strategy for proving Theorem 1 is to show that, if we follow any path from a type A node of $\Gamma(P)$, we must arrive at some *different* type A node; that is, we obtain a new pair of complementary vertices. The details are as follows.

Proof of Theorem 1. To establish Theorem 1, we must prove that if $\Gamma(P)$ contains at least one type A node, then it contains at least two type A nodes.

Let ν be a type A node, and let α be any arc meeting ν. Since every type B node has degree two (by Lemma 3), this arc α begins a well-defined path through $\Gamma(P)$ that passes through zero or more type B nodes in sequence, until either (a) it arrives at a new type A node ν', or (b) it returns to the original type A node ν and becomes a cycle (see Figure 2). Case (a) gives us the desired result; our task is to prove that case (b) is impossible.

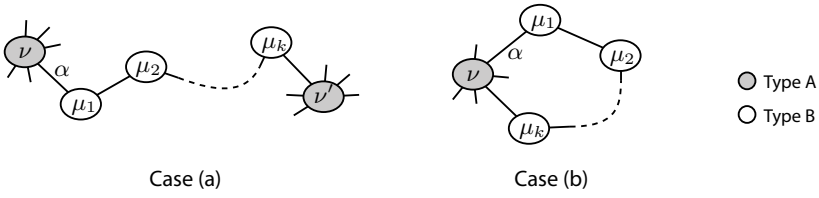


Fig. 2. Following a path from the type A node ν

Suppose then that case (b) occurs, and there is a cycle that passes through nodes $\nu, \mu_1, \mu_2, \ldots, \mu_k, \nu$ in turn, where ν is type A and $\mu_1, \ldots, \mu_k$ are type B. Since $\Gamma(P)$ is a graph with no loops or multiple edges, we have $k \geq 2$. Since $\deg(\mu_i) = 2$, the nodes $\mu_1, \ldots, \mu_k$ are all distinct.

For any node $\eta = \{u, v\}$ of $\Gamma(G)$, let $\Phi(\eta)$ denote the set of all facets of P that contain either u or v. Because P is simple, Definition 2 gives $|\Phi(\eta)| = 2d$ if η is type A, or $|\Phi(\eta)| = 2d - 1$ if η is type B.

We now show that $\Phi(\mu_1) = \Phi(\mu_2) = \ldots = \Phi(\mu_k)$. Consider two adjacent nodes $\mu_i = \{x, y\}$ and $\mu_{i+1} = \{x, z\}$ along our cycle. Because y and z are adjacent vertices of the simple polytope P, there is only one facet F for which $y \in F$ but $z \notin F$. By Lemma 3, this facet F must be the unique facet containing both x and y. Therefore every facet that contains *either* x or y must also contain either x or z, and we have $\Phi(\mu_i) \subseteq \Phi(\mu_{i+1})$. Because $|\Phi(\mu_i)| = |\Phi(\mu_{i+1})| = 2d-1$ we have $\Phi(\mu_i) = \Phi(\mu_{i+1})$, and by induction $\Phi(\mu_1) = \Phi(\mu_2) = \ldots = \Phi(\mu_k)$.

Consider now the type A node ν that begins and ends the cycle. Let $\nu = \{u, v\}$, and without loss of generality let $\mu_1 = \{u, x\}$ for some vertex $x \in P$. Because $|\Phi(\mu_1)| = 2d - 1 < 2d = |\Phi(\nu)|$, there must be some facet F for which $F \in \Phi(\nu)$ but $F \notin \Phi(\mu_1)$; it follows then that $v \in F$ but $x \notin F$.

Moving to the other end of the cycle: since μ_k is adjacent to ν in $\Gamma(P)$, we have either $\mu_k = \{u, x'\}$ where x' is adjacent to v in P, or else $\mu_k = \{x', v\}$ where x' is adjacent to u in P. The second option is not possible because $v \in F \notin \Phi(\mu_1) = \Phi(\mu_k)$; therefore $\mu_k = \{u, x'\}$ for some x' adjacent to v.

We now know that both vertices x and x' are adjacent to v; moreover, since $F \notin \Phi(\mu_1) = \Phi(\mu_k)$ we have $x, x' \notin F$. However, P is a simple polytope with $v \in F$, and so there is only one vertex adjacent to v that is not in F. Therefore $x = x'$ and $\mu_1 = \mu_k$, contradicting the earlier observations that $k \geq 2$ and the nodes $\mu_1, \ldots, \mu_k$ are all distinct. This completes the proof of Theorem 1. □

Remark. The proof of Theorem 1 is algorithmic: given a simple polytope P of dimension $d > 1$ and a pair of complementary vertices $u, v \in P$, it gives an explicit algorithm for locating a second pair of complementary vertices.

In essence, we arbitrarily replace one of the vertices u with an adjacent vertex u', and then repeatedly adjust this pair of vertices according to Lemma 3 part (ii) until we once again reach a pair of complementary vertices. For each adjustment, Lemma 3 part (ii) gives two options (corresponding to the two arcs that meet a type B node); we always choose the option that leads us "forwards" to a new pair, and not "backwards" to the pair we had immediately before.

The proof above ensures that we will eventually reach a complementary pair of vertices again, and that these will not be the same as the original pair u, v.

Passing to the dual polytope, Theorem 1 gives us an immediate corollary:

Corollary 4. *Let P be a simplicial polytope of dimension $d > 1$. If P has a pair of disjoint facets, then P has at least two distinct pairs of disjoint facets.*

We finishing by showing that the "simple" and "simplicial" conditions are necessary in Theorem 1 and Corollary 4.

Observation 5. *The triangular bipyramid (Figure 3, left) is a non-simple polytope of dimension $d = 3$ with precisely one pair of complementary vertices (the apexes at the top and bottom, shaded in the diagram).*

Its dual is the triangular prism (Figure 3, right), which is a non-simplicial polytope of dimension $d = 3$ with precisely one pair of disjoint facets (the triangles at the top and bottom, shaded in the diagram).

Fig. 3. A triangular bipyramid (left) and a triangular prism (right)

These constructions are easily generalised. For instance, we can build a non-simple bipyramid over a neighbourly polytope: the two apexes form the unique pair of complementary vertices, and all other pairs of vertices are adjacent. Felikson and Tumarkin provide further examples in the dual setting [6], involving non-simplicial Coxeter polytopes with precisely one pair of disjoint facets.

3 Adjacency Testing

In this section we prove our main algorithmic result, which uses Theorem 1 to identify simple polytopes and test for adjacent vertices. Throughout this section we work with polytopes of the form

$$P = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq 0\}, \tag{1}$$

where A is some n-column matrix. This form is standard in mathematical programming, and appears in key applications of vertex enumeration [4,5]. Note that the dimension of P is not immediately clear from (1), although $n - \operatorname{rank} A$ gives an upper bound; likewise, it is not immediately clear whether P is simple.

We recall some standard terminology from polytope theory: if F and G are faces of the polytope P, then the *join* $F \vee G$ is the unique smallest-dimensional face that contains both F and G as subfaces. For example, recall the cube from Figure 1, and consider the edges $\overline{AD}$ and $\overline{DC}$. Their meet $\overline{AD} \wedge \overline{DC}$ is the vertex D (where they intersect), and their join $\overline{AD} \vee \overline{DC}$ is the square facet $ABCD$ (the smallest face containing both edges).

Note that the join may be the entire polytope P (for instance, this happens if the faces F and G are complementary vertices).

Our main algorithmic result is the following:

Theorem 6. *Consider any polytope* $P = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq 0\}$*, and suppose we have a list of all vertices of* P*, with* V *vertices in total. Then, after a precomputation step that requires* $O(n^2V + nV^2)$ *time and* $O(nV^2)$ *space:*

- *we know immediately the dimension of* P *and whether or not* P *is simple;*
- *if* P *is simple then, given any two vertices* $u, v \in P$*, we can test whether or not* u *and* v *are adjacent in* $O(n)$ *time;*
- *if* P *is not simple then, given any two vertices* $u, v \in P$*, we may still be able to identify that* u *and* v *are non-adjacent in* $O(n)$ *time.*

We discuss the importance and implications of this result in Section 4; in the meantime, we devote the remainder of this section to proving Theorem 6.

Our overall strategy is to use our main theoretical result (Theorem 1) to characterise and identify adjacent vertices. As a first step, we describe complementary vertices in terms of joins:

Lemma 7. *Let* u *and* v *be distinct vertices of a polytope* P*. Then* u *and* v *are complementary vertices if and only if* $u \vee v = P$*.*

Proof. If $u \vee v \neq P$ then there is some facet F for which $u \vee v \subseteq F$; therefore $u, v \in F$, and u and v cannot be complementary.

On the other hand, if $u \vee v = P$ then there is no facet F for which $u, v \in F$ (otherwise we would have $u \vee v \subseteq F$). Therefore u and v are complementary. □

Using this lemma, we can now make a direct link between Theorem 1 and adjacency testing in polytopes:

Theorem 8. *Let* P *be a simple polytope, and let* F *be a face of* P*. Then* F *is an edge if and only if there is precisely one pair of distinct vertices* $u, v \in P$ *for which* $F = u \vee v$*.*

If P *is any polytope (not necessarily simple), then the forward direction still holds: if* F *is an edge then there must be precisely one pair of distinct vertices* $u, v \in P$ *for which* $F = u \vee v$*.*

Proof. The forward direction is straightforward: let F be an edge of any (simple or non-simple) polytope P, and let the endpoints of F be the vertices u and v. It is clear that $F = u \vee v$, and because F contains no other vertices it cannot be expressed as the join $x \vee y$ for any other pair of distinct vertices x, y.

We now prove the reverse direction in the case where P is a simple polytope. Let F be a face of P, and suppose that F is not an edge. If $\dim F < 1$ then F contains at most one vertex, and so there can be no pairs of distinct vertices u, v for which $F = u \vee v$.

Otherwise $\dim F > 1$; moreover, since P is a simple polytope then the face F is likewise simple when considered as a polytope of its own. Hence we can invoke Theorem 1 to finish the proof. If $F = u \vee v$ for distinct vertices u, v, then Lemma 7 shows that u, v are complementary in the "sub-polytope" F. By Theorem 1 there must be another pair of complementary vertices $\{u', v'\} \neq \{u, v\}$ in F, and by Lemma 7 we have $F = u' \vee v'$ as well. □

To make the join operation accessible to algorithms, we describe faces of polytopes using *zero sets*. Fukuda and Prodon [7] define zero sets for points in P; here we extend this concept to arbitrary faces.

Definition 9 (Zero set). *Consider any polytope of the form* $P = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq 0\}$*, and let* F *be any face of* P*. Then the* zero set *of* F*, denoted* $Z(F)$*, is the set of coordinate positions that take the value zero throughout* F*. That is,* $Z(F) = \{i \mid x_i = 0 \text{ for all } \mathbf{x} \in F\} \subseteq \{1, 2, \ldots, n\}$.

Zero sets are efficient for computation: each can be stored and manipulated using a bitmask of size n, which for moderate problems ($n \leq 64$) involves just a single machine-native integer and fast bitwise CPU instructions. Joins, equality and subface testing all have natural representations using zero sets:

Lemma 10. *Let* F *and* G *be non-empty faces of the polytope* P*. Then:*

- $F \subseteq G$ *if and only if* $Z(F) \supseteq Z(G)$*;*
- $F = G$ *if and only if* $Z(F) = Z(G)$*;*
- *the join has zero set* $Z(F \vee G) = Z(F) \cap Z(G)$.

Proof. These are all consequences of the fact that every non-empty face $F \subseteq P$ is the intersection of P with all hyperplanes of the form $x_i = 0$ where $i \in Z(F)$. See the full version of this paper for detailed proofs. □

We now define the main data structure that we build during our precomputation step in Theorem 6:

Definition 11 (Join map). *Consider any polytope of the form* $P = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq 0\}$*. The* join map *of* P*, denoted* $\mathcal{J}_P$*, is a map of the form* $\mathcal{J}_P \colon 2^{\{1,\ldots,n\}} \to \mathbb{Z}_{\geq 0}$*; that is,* $\mathcal{J}_P$ *maps subsets of* $\{1, \ldots, n\}$ *to non-negative integers. For each subset* $S \subseteq \{1, \ldots, n\}$*, we define the image* $\mathcal{J}_P(S)$ *to be the number of pairs of distinct vertices* $\{u, v\} \in P$ *for which* $Z(u \vee v) = S$.

The following result reformulates Theorem 8 in terms of the join map, and follows immediately from Theorem 8 and Lemma 10.

Corollary 12. *Let* P *be a simple polytope, and let* u, v *be vertices of* P*. Then* u *and* v *are adjacent if and only if* $\mathcal{J}_P(Z(u) \cap Z(v)) = 1$.

If P *is any polytope (not necessarily simple), then the forward direction still holds: if* u *and* v *are adjacent then we must have* $\mathcal{J}_P(Z(u) \cap Z(v)) = 1$.

As a further corollary, the join map can be used to identify precisely whether or not a polytope is simple:

Corollary 13. *Let* P *be any polytope of dimension* d*. Then* P *is simple if and only if, for every vertex* $u \in P$*, there are precisely* d *other vertices* $u' \in P$ *for which* $\mathcal{J}_P(Z(u) \cap Z(u')) = 1$.

Proof. If P is simple then, for each vertex $u \in P$, there are precisely d other vertices $u' \in P$ adjacent to u. By Corollary 12 it follows that there are precisely d other vertices $u' \in P$ for which $\mathcal{J}_P(Z(u) \cap Z(u')) = 1$.

If P is non-simple then there is some vertex $u \in P$ that belongs to $> d$ edges, and so there are $> d$ other vertices $u' \in P$ adjacent to u. By Corollary 12, we have $\mathcal{J}_P(Z(u) \cap Z(u')) = 1$ for each of these adjacent vertices. □

We now show that the join map enjoys many of the properties required for the complexity bounds in Theorem 6:

Lemma 14. *Consider any polytope* $P = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq 0\}$*, and suppose we have a list of all vertices of* P*, with* V *vertices in total. Then we can construct the join map* $\mathcal{J}_P$ *in* $O(nV^2)$ *time and* $O(nV^2)$ *space. Once it has been constructed, we can compute* $\mathcal{J}_P(S)$ *for any set* $S \subseteq \{1, \ldots, n\}$ *in* $O(n)$ *time.*

Proof. We store the join map $\mathcal{J}_P$ using a *trie* (also known as a *prefix tree*). This is a binary tree of height n. Each leaf node (at depth n) represents some set $S \subseteq \{1, \ldots, n\}$, and stores the corresponding image $\mathcal{J}_P(S)$. Each intermediate node at depth $k < n$ supports two children: a "left child" beneath which every set S has $k+1 \notin S$, and a "right child" beneath which every set S has $k+1 \in S$.

We optimise our trie by only storing leaf nodes for sets S with $\mathcal{J}_P(S) \geq 1$, and only storing intermediate nodes that have such leaves beneath them. Figure 4 illustrates the complete trie for an example map with $n = 3$.

Join map:

S	$\{\}$	$\{1\}$	$\{2\}$	$\{3\}$	$\{1,2\}$	$\{1,3\}$	$\{2,3\}$	$\{1,2,3\}$
$\mathcal{J}_P(S)$	0	0	7	5	4	0	1	1

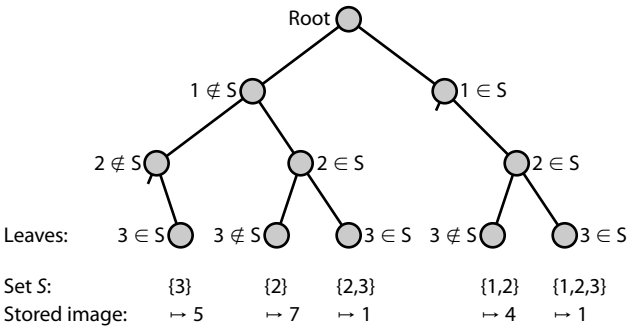


Fig. 4. An example of a trie representing a join map for $n = 3$

Tries are fast to use: for any set $S \subseteq \{1, \ldots, n\}$, the operations of looking up the value of $\mathcal{J}_P(S)$, inserting a new value of $\mathcal{J}_P(S)$, or updating an existing value of $\mathcal{J}_P(S)$ are all $O(n)$, since each operation requires us to follow a single path from the root down to level n (possibly inserting new nodes as we go). For further information on tries in general, see a standard text such as [11].

It is clear now that we can construct $\mathcal{J}_P$ in $O(nV^2)$ time: for each of the $\binom{V}{2}$ pairs of vertices $\{u, v\}$, we construct the set $S = Z(u \vee v) = Z(u) \cap Z(v)$ in $O(n)$ time (just test which coordinate positions are zero in both u and v), and then perform an $O(n)$ lookup for S in the trie. If S is present then we increment $\mathcal{J}_P(S)$; otherwise we perform an $O(n)$ insertion to store the new value $\mathcal{J}_P(S) = 1$.

It is also clear that the trie consumes at most $O(nV^2)$ space: because the construction involves $O(V^2)$ insertions we have $O(V^2)$ leaf nodes, and since the trie has depth n this gives $O(nV^2)$ nodes in total.

Finally, for any set $S \subseteq \{1, \ldots, n\}$, computing $\mathcal{J}_P(S)$ involves a simple lookup operation in the trie, which again requires $O(n)$ time. □

We are now ready to complete the proof of our main result, Theorem 6:

Proof of Theorem 6. The precomputation involves three stages:

(i) building the join map $\mathcal{J}_P$;
(ii) computing $\dim P$ using a rank computation;
(iii) iterating through the join map to determine whether P is simple.

By Lemma 14, stage (i) requires $O(nV^2)$ time and $O(nV^2)$ space.

To compute the dimension in stage (ii) we select an arbitrary vertex $v_0 \in P$, and build a $(V-1) \times n$ matrix M whose rows are of the form $v - v_0$ for each vertex $v \neq v_0$. It follows that $\dim P = \operatorname{rank} M$, and using standard Gaussian elimination we can compute this rank in $O(n^2V)$ time and $O(nV)$ space.

For stage (iii) we once again scan through all $\binom{V}{2}$ pairs of vertices u, v, compute $Z(u) \cap Z(v)$ for each in $O(n)$ time, and use an $O(n)$ lookup in $\mathcal{J}_P$ to test whether $\mathcal{J}_P(Z(u) \cap Z(v)) = 1$. We can thereby identify whether or not, for each vertex $u \in P$, there are precisely $\dim P$ other vertices $u' \in P$ for which $\mathcal{J}_P(Z(u) \cap Z(u')) = 1$; by Corollary 13 this tells us whether or not P is simple. The total running time for this stage is $O(nV^2)$.

Summing the three stages together, we find that our precomputation step requires a total of $O(n^2V + nV^2)$ time and $O(nV^2)$ space.

Once this precomputation is complete, it is clear from stages (ii) and (iii) that we know immediately the dimension of P and whether or not P is simple.

Consider now any two vertices $u, v \in P$. As before, we can compute the set $Z(u) \cap Z(v)$ in $O(n)$ time, and using Lemma 14 we can evaluate $\mathcal{J}_P(Z(u) \cap Z(v))$ in $O(n)$ time. Now Corollary 12 tells us what we need to know: if P is simple then u and v are adjacent if and only if $\mathcal{J}_P(Z(u) \cap Z(v)) = 1$, and if P is non-simple but $\mathcal{J}_P(Z(u) \cap Z(v)) \neq 1$ then we still identify that u and v are non-adjacent. □

4 Discussion

As noted in the introduction, the proof of Theorem 1 is reminiscent of the Lemke-Howson algorithm for constructing Nash equilibria [12]. The Lemke-Howson algorithm operates on a pair of simple polytopes P and Q (best response polytopes for a bimatrix game), each with precisely $f = \dim P + \dim Q$ facets labelled

$1, \ldots, f$, and locates vertices $u \in P$ and $v \in Q$ whose incident facet labels combine to give the full set $\{1, \ldots, f\}$. See [3,12] for details.

The Lemke-Howson algorithm can also be framed in terms of paths through a graph Γ, where it can be shown that these paths yield a 1-factorisation of Γ. One then obtains the corollary that the number of fully-labelled vertex pairs is even; in particular, because there is always a "trivial" pair $(\mathbf{0}, \mathbf{0})$, there must be a second pair (which gives to a Nash equilibrium).

In this paper our setting is less well controlled. We work with a single polytope P, which means we must avoid transforming the pair of vertices $\{u, v\}$ into the identical pair $\{v, u\}$. Moreover, P may have arbitrarily many facets, which makes the arcs of $\Gamma(P)$ more difficult to categorise. In particular, we do not obtain any such 1-factorisation or parity results. To illustrate this, Figure 5 shows that both parities can occur: the cube (on the left) has four complementary pairs of vertices, whereas the cube with one truncated vertex (on the right) has nine.

Fig. 5. Simple polytopes with (i) four, and (ii) nine complementary vertex pairs

Moving to Theorem 6: our $O(n)$ time adjacency test is the fastest we can hope for, since vertices require $\Omega(n)$ space to store (a consequence of the fact that there may be exponentially many vertices [13]). In contrast, standard approaches to adjacency testing use either an $O(nV)$ "combinatorial test" (where we search for a third vertex $w \in u \vee v$), or an $O(n^3)$ "algebraic test" (where we use a rank computation to determine $\dim(u \vee v)$). See [7] for details of these standard tests.

If we are testing adjacency for all pairs of vertices, our total running time including precomputation comes to $O(n^2V + nV^2)$, as opposed to $O(nV^3)$ or $O(n^3V^2)$ for the combinatorial and algebraic tests respectively. This is a significant improvement, given that V may be exponential in n.

The main drawback of our test is that it only guarantees conclusive results for simple polytopes. Nevertheless, it remains useful in the general case: it can detect when a polytope is simple, even if this is not clear from the initial representation $\{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq 0\}$, and even in the non-simple setting it gives a fast filter for eliminating non-adjacent pairs of vertices.

One application of all-pairs adjacency testing is in studying the *graph* of a polytope; that is, the graph formed from its vertices and edges. Another key application is in the *vertex enumeration* problem: given a polytope in the form $\{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq 0\}$, identify all of its vertices. This is a difficult problem, and it is still unknown whether there exists an algorithm polynomial in the combined input and output size.

The two best-known algorithms for vertex enumeration are reverse search [1,2] and the double description method [7,14], each with their own advantages and

drawbacks. The double description method, which features in several application areas such as multiobjective optimisation [4] and low-dimensional topology [5], inductively constructs a sequence of polytopes by adding constraints one at a time. Its major bottleneck is in identifying pairs of adjacent vertices in each intermediate polytope, and in this setting our fast all-pairs adjacency test can be of significant practical use.

References

1. Avis, D.: A revised implementation of the reverse search vertex enumeration algorithm. In: Polytopes—Combinatorics and Computation (Oberwolfach, 1997), DMV Sem., vol. 29, pp. 177–198. Birkhäuser, Basel (2000)
2. Avis, D., Fukuda, K.: A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. Discrete Comput. Geom. 8(3), 295–313 (1992)
3. Avis, D., Rosenberg, G.D., Savani, R., von Stengel, B.: Enumeration of Nash equilibria for two-player games. Econom. Theory 42(1), 9–37 (2010)
4. Benson, H.P.: An outer approximation algorithm for generating all efficient extreme points in the outcome set of a multiple objective linear programming problem. J. Global Optim. 13(1), 1–24 (1998)
5. Burton, B.A.: Optimizing the double description method for normal surface enumeration. Math. Comp. 79(269), 453–484 (2010)
6. Felikson, A., Tumarkin, P.: Coxeter polytopes with a unique pair of non-intersecting facets. J. Combin. Theory Ser. A 116(4), 875–902 (2009)
7. Fukuda, K., Prodon, A.: Double description method revisited. In: Deza, M., Manoussakis, I., Euler, R. (eds.) CCS 1995. LNCS, vol. 1120, pp. 91–111. Springer, Heidelberg (1996)
8. Jansen, M.J.M.: Maximal Nash subsets for bimatrix games. Naval Res. Logist. Quart. 28(1), 147–152 (1981)
9. Kim, E.D., Santos, F.: An update on the Hirsch conjecture. Jahresber. Dtsch. Math.-Ver. 112(2), 73–98 (2010)
10. Klee, V., Walkup, D.W.: The d-step conjecture for polyhedra of dimension $d < 6$. Acta Math. 117, 53–78 (1967)
11. Knuth, D.E.: The Art of Computer Programming, vol. 3: Sorting and Searching, 2nd edn. Addison-Wesley, Reading (1998)
12. Lemke, C.E., Howson Jr., J.T.: Equilibrium points of bimatrix games. J. Soc. Indust. Appl. Math. 12(2), 413–423 (1964)
13. McMullen, P.: The maximum numbers of faces of a convex polytope. Mathematika 17, 179–184 (1970)
14. Motzkin, T.S., Raiffa, H., Thompson, G.L., Thrall, R.M.: The double description method. In: Kuhn, H.W., Tucker, A.W. (eds.) Contributions to the Theory of Games, Vol. II. Annals of Mathematics Studies, vol. 28, pp. 51–73. Princeton University Press, Princeton (1953)
15. Santos, F.: A counterexample to the Hirsch conjecture. To appear in Ann. of Math. (2) (2010), arXiv: 1006.2814
16. Winkels, H.M.: An algorithm to determine all equilibrium points of a bimatrix game. In: Game Theory and Related Topics (Proc. Sem., Bonn and Hagen, 1978), pp. 137–148. North-Holland, Amsterdam (1979)
17. Ziegler, G.M.: Lectures on Polytopes. Graduate Texts in Mathematics, vol. 152. Springer, New York (1995)

Online Coloring of Bipartite Graphs with and without Advice*

Maria Paola Bianchi[1], Hans-Joachim Böckenhauer[2], Juraj Hromkovič[2], and Lucia Keller[2]

[1] Dipartimento di Informatica, Università degli Studi di Milano, Italy
[2] Department of Computer Science, ETH Zurich, Switzerland
maria.bianchi@unimi.it, {hjb,juraj.hromkovic,lucia.keller}@inf.ethz.ch

Abstract. In the online version of the well-known graph coloring problem, the vertices appear one after the other together with the edges to the already known vertices and have to be irrevocably colored immediately after their appearance. We consider this problem on bipartite, i. e., two-colorable graphs. We prove that $1.13747 \cdot \log_2 n$ colors are necessary for any deterministic online algorithm to color any bipartite graph on n vertices, thus improving on the previously known lower bound of $\log_2 n + 1$ for sufficiently large n.

Recently, the advice complexity was introduced as a method for a fine-grained analysis of the hardness of online problems. We apply this method to the online coloring problem and prove (almost) tight linear upper and lower bounds on the advice complexity of coloring a bipartite graph online optimally or using 3 colors. Moreover, we prove that $O(\sqrt{n})$ advice bits are sufficient for coloring any graph on n vertices with at most $\lceil \log_2 n \rceil$ colors.

1 Introduction

In an online problem, the input is revealed piecewise in consecutive time steps and an irrevocable part of the output has to be produced at each time step, for a detailed introduction and an overview of online problems and algorithms, see [4]. One of the most studied online scenarios is the problem of coloring a graph online. Here, the vertices of the graph are revealed one after the other, together with the edges connecting them to the already present vertices. The goal is to assign the minimum number of colors to these vertices in such a way that no two adjacent vertices get the same color. As usual in an online setting, each vertex has to be colored immediately after its appearance. The quality of an online algorithm for this problem is usually measured by the so-called *competitive ratio*, i. e., the ratio between the number of colors used by this algorithm and an optimal coloring for the resulting graph as it could be computed by an offline algorithm with unlimited computing power, knowing the whole graph in advance.

It turns out that online coloring is a very hard online problem for which no constant competitive ratio is possible [9]. For an overview of results on the

* The research is partially funded by the SNF grant 200021–141089.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 519–530, 2012.

online graph coloring problem, see, e. g., [11,12]. In particular, some bounds on the chromatic number of the class $\Gamma(k,n)$ of k-colorable graphs on n vertices have been proven: given $G \in \Gamma(k,n)$, any online coloring algorithm for G needs at least $\Omega\left(((\log n)/(4k))^{k-1}\right)$ colors [15]. On the other hand, there exists an online algorithm for coloring G with $O\left(n\frac{\log^{(2k-3)} n}{\log^{(2k-4)} n}\right)$ colors [13], where $\log^{(k)}$ is the log-function iterated k times. Even for the very restricted class of bipartite, i. e., two-colorable, graphs, any online algorithm can be forced to use at least $\lceil \log_2 n \rceil + 1$ colors for coloring a bipartite graph on n vertices [1]. On the other hand, an online algorithm coloring every bipartite graph with at most $2\log_2 n$ colors is known [13]. In the first part of this paper, we improve the lower bound for bipartite graphs to $1.13747 \cdot \log_2 n$.

The main drawback in the competitive analysis of online algorithms is that an online algorithm has a huge disadvantage compared to an offline algorithm by not knowing the future parts of the input. This seems to be a rather unfair comparison since there is no way to use an offline algorithm in an online setting. Recently, the model of advice complexity of online problems has been introduced to enable a more fine-grained analysis of the hardness of online problems. The idea here is to measure what amount of information about the yet unknown parts of the input is necessary to compute an optimal (or near-optimal) solution online [3,5,6,10]. For this, we analyze online algorithms that have access to a tape with *advice bits* that was computed by some oracle knowing the whole input in advance. The *advice complexity* of such an algorithm measures how many of these advice bits the algorithm reads during its computation. As usual, the advice complexity of an online problem is defined as the amount of advice needed by the best algorithm solving the problem.

It turns out that, for some problems, very little advice can drastically improve the competitive ratio of an online algorithm, e. g., for the simple knapsack problem, a single bit of advice is sufficient to jump from being non-competitive at all to 2-competitiveness [2]. On the other hand, many problems require a linear (or even higher) amount of advice bits for computing an optimal solution [2,3,6]. The advice complexity of a coloring problem was first investigated in [8] where linear upper and lower bounds for coloring a path were shown.

In the second part of this paper, we investigate the advice complexity of the online coloring problem on bipartite graphs. We prove almost tight upper and lower bounds on the advice complexity of computing an optimal solution, more precisely, for a graph on n vertices, $n-2$ advice bits are sufficient and $n-3$ advice bits are necessary for this. Moreover, we prove linear upper and lower bounds on the advice complexity of computing a 3-coloring, namely an upper bound of $n/2$ and a lower bound of $\frac{n}{2} - 4$. We complement these results by an algorithm that uses less than $n/\sqrt{2^{k-1}}$ advice bits for coloring a bipartite graph online with k colors.

The paper is organized as follows. In Section 2, we formally define the online coloring problem and fix our notation, in Section 3, we consider online algorithms without advice and present the improved lower bound on the number of necessary

colors for deterministic online coloring algorithms. The proof of this lower bound is contained in Section 4, while Section 5 is devoted to the advice complexity of the online coloring of bipartite graphs. Due to space limitations, some proofs are omitted.

2 Preliminaries

In this section, we fix our notation and formally define the problem we are dealing with in this paper.

Definition 1 (Coloring). *Let $G = (V, E)$ be an undirected and unweighted graph with vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set E. A* (proper) coloring of a graph G *is a function $col : V \to S$ which assigns to every vertex $v_i \in V$ a color $col(v_i) \in S$ and has the property that $col(v_i) \neq col(v_j)$, for all $i, j \in \{1, 2, \ldots, n\}$ with $\{v_i, v_j\} \in E$.*

Usually, we consider the set $S = \{1, 2, \ldots, n\} \subset \mathbb{N}^+$. Let $V' \subseteq V$, then we denote by $\mathrm{col}(V')$ the set of colors assigned to the vertices in V'. To distinguish the coloring functions used by different algorithms, we denote, for an algorithm A, its coloring function by col_A. We denote the subgraph of $G = (V, E)$ induced by a vertex subset $V' \subseteq V$ by $G[V']$, i. e., $G[V'] = (V', E')$, where $E' = \{\{v, w\} \in E \mid v, w \in V'\}$.

An instance I for the online coloring problem can be described as a sequence $I = (G_1, G_2, \ldots, G_n)$ of undirected graphs such that $V_i = \{v_1, v_2, \ldots, v_i\}$ and $G_i = G_n[V_i]$. Informally speaking, G_i is derived from G_{i-1} by adding the vertex v_i together with its edges incident to vertices from V_{i-1}. Let $\mathcal{G}_n$ denote the set of all online graph instances on n vertices. Then, $\mathcal{G}$ is the set of all possible online graph instances for the online coloring problem for all $n \in \mathbb{N}$, i. e., $\mathcal{G} = \bigcup_{n \in \mathbb{N}} \mathcal{G}_n$. With this, we can formally define the online coloring problem.

Definition 2 (Online Coloring Problem)

Input: $I = (G_1, G_2, \ldots, G_n) \in \mathcal{G}$ *for some* $n \in \mathbb{N}^+$.
Output: $(c_1, c_2, \ldots, c_n) \in \mathbb{N}^+$ *such that* $col(v_i) = c_i$ *and* $col : V_i \to \mathbb{N}^+$ *is a coloring for all* $i \in \{1, 2, \ldots, n\}$
Cost: *Number of colors used by the coloring.*
Goal: *Minimum.*

In the following, we will restrict our attention to the class of bipartite, i. e., two-colorable graphs. We denote the subproblem of the online coloring problem restricted to bipartite input graphs by BipCol. In a bipartite graph $G = (V(G), E(G))$, the vertex set $V(G)$ can be partitioned into two subsets, called *shores* and denoted by $S_1(G)$ and $S_2(G)$, with the property that the edges in $E(G)$ connect only vertices from different shores. If the graph is clear from the context, we write V, E, S_1, S_2 instead of $V(G)$, $E(G)$, $S_1(G)$, $S_2(G)$.

Given two vertices v_i and v_j, we write $v_i \leftrightarrow_t v_j$ iff there exists a path in G_t from v_i to v_j. It is always possible to partition V_t into connected components

according to the equivalence relation $\leftrightarrow_t$, and we call such components $C_t(v_i) = [v_i]_{\leftrightarrow_t}$. In the case of connected components, the shore partition is unique.

We want to analyze BipCol with giving bounds on the number of colors used in the online coloring process. These bounds will always depend on the number n of vertices in the final graph G_n. Let A be an online coloring algorithm. We denote by $F_A(G) = |\mathrm{col}_A(V_n)|$ the number of colors used by A to color graph G. Then, $F_A(n) = \max_{G \in \mathcal{G}_n} F_A(G)$ is the maximum number of colors A uses to color any online graph instance with n vertices in the final graph G_n. We say that $U : \mathbb{N} \to \mathbb{N}$ is an *upper bound* on the number of colors sufficient for online coloring, if there exists an algorithm A such that, for all $n \in \mathbb{N}$, we have $F_A(n) \leq U(n)$. Hence, it is sufficient to find a good deterministic online algorithm A to get an upper bound on the number of colors used to color an instance of $\mathcal{G}_n$. Similarly, $L : \mathbb{N} \to \mathbb{N}$ is a *lower bound* on the number of colors necessary for online coloring any graph if, for all online algorithms A for infinitely many $n \in \mathbb{N}$, we have $L(n) \leq F_A(n)$, i. e., if, for every algorithm A for infinitely many n, there is an online graph $G_A(n) \in \mathcal{G}_n$ for which A needs at least $L(n)$ colors.

3 Online Coloring without Advice

In this section, we deal with the competitive ratio of deterministic online algorithms without advice. The following upper bound is well known.

Theorem 1 (Lovász, Saks, and Trotter [13]). *There is an online algorithm requiring at most* $2 \log_2 n$ *colors for coloring any bipartite graph of* n *vertices.*

There is also a well-known lower bound which even holds for trees.

Theorem 2 (Bean [1]). *For every* $k \in \mathbb{N}^+$*, there exists a tree* T_k *on* 2^{k-1} *vertices such that for every online coloring algorithm* A*,* $\mathrm{col}_A(T_k) \geq k$*.*

Theorem 2 immediately implies that there exists an infinite number of trees (and thus of bipartite graphs) forcing any online algorithm to use at least $\log_2 n + 1$ colors on any graph on n vertices from this class. In the remainder of this section, we improve on this result by describing a graph class which forces every coloring algorithm A to use even more colors. This class is built recursively. In the proof, we will focus only on those G_i's in an instance $(G_1, G_2, \ldots, G_n) \in \mathcal{G}_n$ in which one new color has to be used by any deterministic online coloring algorithm to color v_i.

Lemma 1. *For every* $i \in \mathbb{N}^+$ *and every online coloring algorithm* A*, there exists an online graph* $G_A(i)$ *such that:*

1. $F_A(G_A(i)) \geq i$,
2. $F_A(S_1(G_A(i))) \geq i - 2$,
3. $F_A(S_2(G_A(i))) \geq i - 1$,
4. $|V(G_A(i))| =: B(i) \leq B(i-1) + B(i-2) + B(i-3) + 1$*, for* $i \geq 3$*, and* $B(0) = 0$*,* $B(1) = 1$*, and* $B(2) = 2$*.*

We will prove Lemma 1 in the following section. The recurrence given by property 4 of Lemma 1 can be resolved as follows.

Corollary 1. $B(k) \leq 1.35526 \cdot 1.83929^k - .400611$

Proof (Sketch). One can show by induction that $B(k) = \sum_{i=0}^{k+1} T(i)$ where $T(i)$ is the i-th *Tribonacci number* (see [7,14]). The number $T(n)$ can be computed as follows:

$$T(n) = 3b \cdot \frac{\left(\frac{1}{3}(a_+ + a_- + 1)\right)^n}{b^2 - 2b + 4} \leq .336229 \cdot 1.83929^n,$$

where $a_+ = (19 + 3\sqrt{33})^{\frac{1}{3}}$, $a_- = (19 - 3\sqrt{33})^{\frac{1}{3}}$, and $b = (586 + 102\sqrt{33})^{\frac{1}{3}}$. □

Theorem 3. *The lower bound on the number of colors used by any online coloring algorithm A to color any online graph instance with n vertices is*

$$1.13747 \cdot log_2(n).$$

Proof. The claim follows immediately from Corollary 1. □

4 Proof of Lemma 1

In this section, we prove Lemma 1. We proceed by an induction over k, the number of colors. For every k, we generate a class $\tilde{\mathcal{G}}(k)$ consisting of online graphs defined as

$$\tilde{\mathcal{G}}(k) = \{G_B(k) \mid B \text{ is an online coloring algorithm and properties 1 to 4 of Lemma 1 are satisfied}\}.$$

Hence, for a fixed k, we will find in $\tilde{\mathcal{G}}(k)$, for every online coloring algorithm B, an instance that forces B to use at least k colors to color $G_B(k)$. Those instances are built inductively. We will prove that we can construct, for any online coloring algorithm A, a hard instance $G_A(k)$, using three graphs $G_{k-1} \in \tilde{\mathcal{G}}(k-1)$, $G_{k-2} \in \tilde{\mathcal{G}}(k-2)$, $G_{k-3} \in \tilde{\mathcal{G}}(k-3)$, that are revealed in this order, and an additional vertex v. Let $H(k)$ be the induction hypothesis, formulated as follows:

H(k): For all $j \leq k$ and all online algorithms B, there exists a graph $G_B(j)$ with properties 1 to 4 of Lemma 1.

Assuming $H(k-1)$ holds, it is easy to show that a graph G_{k-1} exists. To show the existence of G_{k-2} and G_{k-3}, we have to take into account that the algorithm A already knows a part of the instance, and hence it behaves differently from the case where there is no part known.

In a second step, we merge the shores of G_{k-1}, G_{k-2}, and G_{k-3} in an appropriate way and, with an additional vertex v, we assure that all conditions of Lemma 1 are satisfied. We can stop the construction of this online graph also earlier, as soon as all four conditions are satisfied.

We merge two graph instances $G = (G_1, \ldots, G_l) \in \mathcal{G}$ and $G' = (G'_1, \ldots, G'_m) \in \mathcal{G}$ to an instance $M = G \circ G'$, defined as $M = (G_1, \ldots, G_l, G_l \cup G'_1, \ldots, G_l \cup G'_m) \in \mathcal{G}$.

(a) Case $k = 1$ (b) Case $k = 2$

Fig. 1. Base cases: W.l.o.g., the vertices are colored as indicated. The indices of the vertices indicate their order of appearance.

4.1 Base Cases ($k \leq 3$)

For $k = 0, 1, 2$, it is easy to show that the hypothesis is satisfied (see Figure 1).

In the case $k = 3$, for every online coloring algorithm A, $G_A(3)$ can be constructed recursively using two graphs $G_2 \in \tilde{\mathcal{G}}(2)$, $G_1 \in \tilde{\mathcal{G}}(1)$ and possibly a new vertex. The vertices of G_2 are colored w.l.o.g. with 1 and 2, and there are three possibilities to color G_1 (see Figure 2).

In the case (c) of Figure 2, we are already done, since we reached 3 colors and $S_1(G_A(3)) = \{v_1\}$ is colored with one color and $S_1(G_A(3)) = \{v_2, v_3\}$ with two colors. In cases (a) and (b), we have to add one new vertex v_4 which is connected to two vertices with different colors to force every online coloring algorithm A to use a third color. In case (a) (equivalently, case (b)), we have $S_1(G_A(3)) = \{v_1, v_4\}$ and $S_2(G_A(3)) = \{v_2, v_3\}$. Both shores are colored with two colors, hence conditions 1, 2 and 3 are satisfied, and also condition 4, since $B(3) \leq B(2) + B(1) + 1$.

4.2 Inductive Step ($k \geq 4$)

For every online algorithm A and every $k \in \mathbb{N}^+$, we will construct $G_A(k)$ in four steps using three graphs $G_{k-1} \in \tilde{\mathcal{G}}(k-1)$, $G_{k-2} \in \tilde{\mathcal{G}}(k-2)$, $G_{k-3} \in \tilde{\mathcal{G}}(k-3)$ satisfying properties 1 to 4 from Lemma 1, and an additional vertex v.

First, we want to show that such graphs G_{k-1}, G_{k-2}, and G_{k-3} actually exist. Then, we will show that we can merge them, using an additional vertex v, to a graph $G_A(k)$ satisfying properties 1 to 4 from Lemma 1.

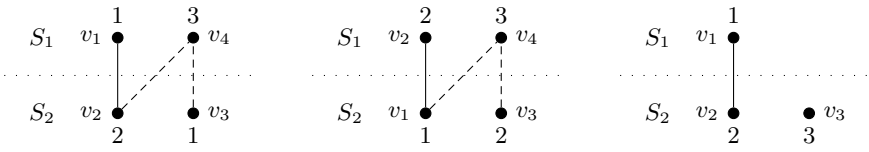


Fig. 2. The base case $k = 3$

Existence of the Graphs G_{k-1}, G_{k-2}, and G_{k-3}. We assume that $H(k-1)$ holds. Hence, for every online coloring algorithm B and $j \in \{k-1, k-2, k-3\}$ there exists a graph $G_B(j)$ with properties 1 to 4 of Lemma 1.

Step 1. Because of $H(k-1)$, we know that a graph $G_{k-1} = G_A(k-1)$ exists.
Step 2. In the next phase, algorithm A receives a second graph. We denote by $A|_{G_{k-1}}$ the work of algorithm A having already processed the graph G_{k-1}. $A|_{G_{k-1}}$ can be simulated by an algorithm B which does the same work as $A|_{G_{k-1}}$ but which did not receive any other graph before. Because of $H(k-1)$, and thus $H(k-2)$, we know that, for such algorithm B, there is a graph $G_{k-2} = G_B(k-2) = G_{A|_{G_{k-1}}}(k-2)$ satisfying properties 1 to 4 of Lemma 1.
Step 3. Now, algorithm A gets a third graph. Again, the work of $A|_{G_{k-1} \circ G_{k-2}}$ can be simulated by an algorithm C. Because of the induction hypothesis, a graph $G_{k-3} = G_C(k-3) = G_{A|_{G_{k-1} \circ G_{k-2}}}(k-3)$ exists.

Hence, we have the graphs G_{k-1}, G_{k-2}, and G_{k-3} at our disposal and can force a new color, possibly with the help of an additional vertex v.

Construction of Graph $G_A(k)$. Due to space limitations, we only sketch this part of the proof. There are two possible cases with respect to the first graph G_{k-1}:

A) A **uses exactly** $k-1$ **colors on** G_{k-1}
Here, we will distinguish five cases with respect to the colors that appear in both shores in G_{k-1} and the placement of the remaining colors. For sure, we have at least $k-4$ common colors, since otherwise properties 2 and 3 of Lemma 1 for $i = k-1$ would not be satisfied. If we have $k-1$ or $k-2$ common colors, one additional vertex v forces the algorithm to use color k and immediately all properties of Lemma 1 for $i = k$ are satisfied. With $k-3$ common colors, we have to distinguish again two cases: Either the remaining colors a and b are both in one shore or they are in different shores. In the first case, we can proceed as before. In the second case, it depends on whether one of the colors a and b appears in G_{k-2} or not. If there is neither a nor b, there will be a new color c for sure. If there are $k-4$ common colors, we can either satisfy all the conditions of Lemma 1 for $i = k$ using some new colors that appear only in G_{k-2} and G_{k-3}, or we find some of the colors that appear only in one shore of G_{k-1} in G_{k-2} and in G_{k-3}. Joining the shores in an appropriate way, we can force with a new vertex a new color k.

B) A **uses at least** k **colors on** G_{k-1}
The graph G_{k-1} already contains sufficiently many colors to satisfy condition 1 of Lemma 1 for $i = k$. In general, the graph G_{k-1} has the following colors:

$\mathrm{col}(S_1(G_{k-1}))$	$1, 2, \ldots, x, \quad b_1, b_2, \ldots, b_j$
$\mathrm{col}(S_2(G_{k-1}))$	$1, 2, \ldots, x, a_1, a_2, \ldots, a_i$

Because of $H(k-1)$, we have $x + j \geq k-3$ and $x + i \geq k-2$. Hence, we have either to add two colors to $S_1(G_{k-1})$ or one color to each shore in order to satisfy conditions 2 and 3 of Lemma 1 for $i = k$. If l is the total

number of colors in G_{k-1}, i. e., $x + i + j = l$, and m is the total number of colors in all three graphs G_{k-1}, G_{k-2}, and G_{k-3}, then we have to distinguish some cases with respect to the relation between m and l. If $m \geq l + 2$, we have sufficiently many new colors in G_{k-2} or G_{k-3} to satisfy conditions 2 and 3 of Lemma 1 for $i = k$. If $m = l + 1$, we have one new color c in one of the two smaller graphs, and either we have a second new color or we can find one of the colors $a_1, \ldots, a_i, b_1, \ldots, b_j$ in either G_{k-2} or G_{k-3}. This depends on the relation of k and x. In the case $m = l$, we have to fill the gaps in order to satisfy conditions 2 and 3 of Lemma 1 with some of the colors $a_1, \ldots, a_i, b_1, \ldots, b_j$ which again can be found in G_{k-2} and G_{k-3}, depending on the relation between x and k.

5 Advice Complexity

In this section, we investigate the advice complexity of the online coloring problem on bipartite graphs. We start with giving an upper bound on the amount of advice needed for achieving an optimal coloring.

Theorem 4. *There exists an online algorithm for* BipCol *which needs at most* $n - 2$ *advice bits to be optimal on every instance of length* n.

Proof. We present the algorithm A_2 that works as follows. The first vertex receives color 1. Then the algorithm asks for one bit of advice: if it is 1, then A_2 will assign color 1 to every isolated vertex, otherwise it will ask for a bit of advice for every other isolated vertex, to decide whether to assign color 1 or 2. Any vertex that has an edge to some previously received vertex v, receives the opposite color with respect to v. It is easy to see that, on an input of length n, whenever there are at least $n-1$ isolated vertices, the advice is a string of length one; in all other cases it is smaller than $n - 2$ bits.

Notice that, since the advice tape is infinite and it is up to the algorithm to decide how many bits to read, A_2 could not simply read one advice bit for every vertex from v_2 to v_{n-1} without knowing the length of the input in advance. □

We can complement this result by an almost matching lower bound.

Theorem 5. *Any deterministic online algorithm for* BipCol *needs at least* $n-3$ *advice bits to be optimal on every instance of length* n.

Proof (Sketch). Let $\hat{A}$ be an algorithm with advice for BipCol that uses 2 colors. Given, for any $0 \leq \alpha \leq n - 2$, the graph G_α with n vertices described in Figure 3, we consider as the set of possible instances of $\hat{A}$ any online presentation of G_α, for all $0 \leq \alpha \leq n - 2$, such that the first $n - 2$ vertices are presented as a permutation of $\{v_j\}_{1 \leq j \leq n-2}$. We say that two instances are equivalent iff the order of the first $n - 2$ vertices reflects the same shore partition, and, by a counting argument, the number of such equivalence classes is $\frac{2^{n-2}}{2} = 2^{n-3}$. It is not hard to see that, in order to avoid using a third color on $\hat{v}$ or $\hat{v}'$, the algorithm needs to assign the same color to all vertices of each shore, so instances in different equivalence classes must have different advice strings. □

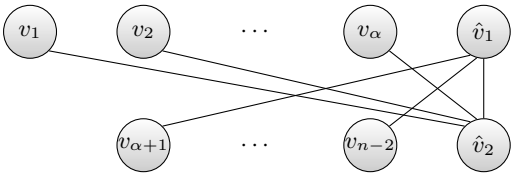


Fig. 3. Structure of the graph G_α used in the proof of Theorem 5. The set of edges is $E = \{\{v_s, \hat{v}_2\} \mid 1 \leq s \leq \alpha\} \cup \{\{v_s, \hat{v}_1\} \mid \alpha + 1 \leq s \leq n-2\} \cup \{\{\hat{v}_1, \hat{v}_2\}\}$.

We now analyze how much advice is sufficient to guarantee a given constant competitive ratio.

Theorem 6. *For any integer constant $k > 2$, there exists an online algorithm for* BipCol *that needs less than $\frac{n}{\sqrt{2^{k-1}}}$ advice bits to color every instance of length n with at most k colors.*

Proof. We will consider the same algorithm A used in the proof for Theorem 1 shown in [12]: the idea is to make A ask for an advice only when it is about to assign color $k-1$, in order to avoid assigning that color to vertices on both shores of the final graph. This implies that the algorithm will always have vertices of color $k-1$ (if any) only on one shore and vertices of color k (if any) only on the other shore, so that color $k+1$ will never be needed.

We now describe the algorithm A_k at step t, when the vertex v_t is revealed. By calling K_t (R_t, respectively) the set of colors assigned to vertices on the same (opposite, respectively) shore as v_t, A_k will choose its output as follows:

- if $\exists\ 1 \leq c \leq k-2$ such that $c \notin R_t$, then $\mathrm{col}_{A_k}(v_t) = \min\{c \geq 1 \mid c \notin R_t\}$,
- if either $k-1 \in R_t$ or $k \in K_t$, then $\mathrm{col}_{A_k}(v_t) = k$,
- if either $k \in R_t$ or $k-1 \in K_t$, then $\mathrm{col}_{A_k}(v_t) = k-1$,
- if $R_t = \{1, 2, \ldots, k-2\}$, then A_k asks for one bit of advice to decide whether to assign color $k-1$ or k to v_t.

Algorithm A_k asks for an advice only when it is about to assign color $k-1$, which may happen at most every $2^{\frac{k-1}{2}}$ vertices, as shown in [12] for algorithm A, so the maximum number of advice bits required is $\frac{n}{\sqrt{2^{k-1}}}$. □

The proof of Theorem 6 can be easily extended to the case of using a non-constant number of colors, only the size n of the input has to be encoded into the advice, using $\lceil \log(n) \rceil + 2\lceil \log\log(n) \rceil$ additional bits. This leads to the following corollary.

Corollary 2. *There is an online algorithm for* BipCol *that needs at most $O(\sqrt{n})$ advice bits to color every instance of length n with at most $\lceil \log(n) \rceil$ colors.*

In the remainder of this section, we want to analyze the case of near-optimal coloring using 3 colors. For this case, Theorem 6 gives the following upper bound on the advice complexity.

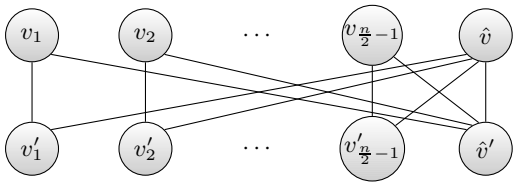


Fig. 4. Graph G' used in the proof of Theorem 7. The edges are $E' = \{\{v_s, v'_s\}, \{v_s, \hat{v}'\}, \{v'_s, \hat{v}\}, \{\hat{v}, \hat{v}'\} \mid 1 \leq s \leq \frac{n}{2} - 1\}$.

Corollary 3. *There exists an online algorithm for* BipCol *that needs at most* $\frac{n}{2}$ *advice bits to color every instance of length* n *with at most* 3 *colors.*

We conclude with an almost matching lower bound for coloring with 3 colors.

Theorem 7. *Any deterministic online algorithm for* BipCol *needs at least* $\frac{n}{2}-4$ *advice bits to color every instance of length* n *with at most* 3 *colors.*

Proof. Consider the graph $G' = (V', E')$ described in Figure 4. By calling c_t the subgraph of G' induced by the vertices $\{v_t, v'_t\}$, we will consider a set of instances where the first $\frac{n}{2} - 1$ vertices are revealed isolated: each of those can be arbitrarily chosen between the two shores of c_t, for all the c_t's from left to right. After time $\frac{n}{2} - 1$, the algorithm will receive the corresponding neighbors in each c_t of all the previously received vertices. Finally, vertices $\hat{v}$ and $\hat{v}'$ are revealed. By calling $\pi_r(i)$ the vertex revealed at step i, we identify any input instance I_r with the binary string $\sigma_r = (\sigma_r(1), \ldots, \sigma_r(\frac{n}{2} - 1))$ such that, for all $1 \leq t \leq \frac{n}{2} - 1$,

$$\sigma_r(t) = \begin{cases} 0 & \text{if } \pi_r(t) = v_t \\ 1 & \text{if } \pi_r(t) = v'_t. \end{cases}$$

In other words, σ_r tells us in which order the two vertices in each c_t are revealed.

With a slight abuse of notation, we say that $\mathrm{col}_{\sigma_r}(t) = (\alpha, \beta)$ iff, in I_r, the color assigned to v_t is α and the color assigned to v'_t is β. We also write $\mathrm{col}^R_{\sigma_r}(t) = (\beta, \alpha)$ iff $\mathrm{col}_{\sigma_r}(t) = (\alpha, \beta)$. We say that a color α is *mixed* if it appears on both shores of the bipartition, i. e., if there exist $t_1, t_2 \in \{1, \ldots, \frac{n}{2} - 1\}$ such that v_{t_1} and v'_{t_2} receive both color α. It is easy to see that an instance of the form considered above can be colored with at most 3 colors only if at most 1 color is mixed in the first $n-2$ vertices. As a consequence, we can never have an advice such that

$$\mathrm{col}_{\sigma_r}(t_1) = \mathrm{col}^R_{\sigma_r}(t_2), \tag{1}$$

for some $t_1, t_2 \in \{1, \ldots, \frac{n}{2} - 1\}$, otherwise two colors would be mixed.

In general, if the advice on an instance σ_i is such that either $\sigma_i(t_1) = \sigma_i(t_2) \wedge \mathrm{col}_{\sigma_i}(t_1) = \mathrm{col}_{\sigma_i}(t_2)$ or $\sigma_i(t_1) \neq \sigma_i(t_2) \wedge \mathrm{col}_{\sigma_i}(t_1) = \mathrm{col}^R_{\sigma_i}(t_2)$, then, for any other instance σ_j such that $\sigma_i(t_1) = \sigma_j(t_1) \wedge \sigma_i(t_2) \neq \sigma_j(t_2)$, σ_j must have a different advice, otherwise we would have $\mathrm{col}_{\sigma_j}(t_1) = \mathrm{col}^R_{\sigma_j}(t_2)$.

Our aim now is to find out how many instances can have the same advice as σ_i. We can distinguish three situations:

Table 1. The possible instances using the same advice as σ_i in the second situation in the proof of Theorem 7

	σ_i	$\bar{\sigma}_i$	σ_j	$\bar{\sigma}_j$
$t \in A$	σ_i	$1-\sigma_i$	σ_i	$1-\sigma_i$
$t \in B$	σ_i	$1-\sigma_i$	$1-\sigma_i$	σ_i

Table 2. The possible instances using the same advice as σ_i in the third situation in the proof of Theorem 7

	σ_i	$\bar{\sigma}_i$	σ_j	$\bar{\sigma}_j$	σ_k	$\bar{\sigma}_k$	σ_h	$\bar{\sigma}_h$
$t \in A$	σ_i	$1-\sigma_i$	σ_i	$1-\sigma_i$	σ_i	$1-\sigma_i$	σ_i	$1-\sigma_i$
$t \in B$	σ_i	$1-\sigma_i$	σ_i	$1-\sigma_i$	$1-\sigma_i$	σ_i	$1-\sigma_i$	σ_i
$t \in C$	σ_i	$1-\sigma_i$	$1-\sigma_i$	σ_i	$1-\sigma_i$	σ_i	σ_i	$1-\sigma_i$

1. The advice on σ_i is such that the algorithm uses only one couple of colors, i. e., for every t, $\mathrm{col}_{\sigma_i}(t) = (\alpha, \beta)$. In this case, the only other instance which can have the same advice and still avoids mixing two colors in the first $n-2$ vertices is $\bar{\sigma}_i$, such that $\bar{\sigma}_i(t) \neq \sigma_i(t)$, for all $t \in \{1, \ldots, \frac{n}{2} - 1\}$, because on any other instance we would have the situation described in Equation (1).
2. The advice is such that the algorithm uses only two couples of colors, more formally, there exists a partition $A, B \subset \{1, \ldots, \frac{n}{2} - 1\}$, for two nonempty sets A, B such that

$$\forall t \in A\colon\ \mathrm{col}_{\sigma_i}(t) = (\alpha, \beta), \qquad \forall t \in B\colon\ \mathrm{col}_{\sigma_i}(t) = (\alpha, \gamma).$$

   In order to avoid Equation (1), the only instances σ that can have the same advice as σ_i are the ones such that $\sigma(t_1) = \sigma(t_2)$ for any t_1, t_2 in the same set of the partition $\{A, B\}$, which are the four described in Table 1.
3. The advice is such that the algorithm uses all three couples of colors, i. e., there exists a partition $A, B, C \subset \{1, \ldots, \frac{n}{2} - 1\}$, with $A, B, C \neq \emptyset$, such that

$$\forall t \in A\colon\ \mathrm{col}_{\sigma_i}(t) = (\alpha, \beta), \quad \forall t \in B\colon\ \mathrm{col}_{\sigma_i}(t) = (\alpha, \gamma), \quad \forall t \in C\colon\ \mathrm{col}_{\sigma_i}(t) = (\beta, \gamma).$$

   Again, in order to avoid Equation (1), an instance σ with the same advice as σ_i must be such that $\sigma(t_1) = \sigma(t_2)$ for any t_1, t_2 in the same set of the partition $\{A, B, C\}$. This property is satisfied only by the eight instances described in Table 2.

However, the two instances σ_h and $\bar{\sigma}_h$ would have all colors mixed, if they were given the same advice as σ_i. This implies that at most six instances of the form σ_r can have the same advice, and since the number of instances of the form σ_r is $2^{\frac{n}{2}-1}$, there must be at least $\frac{2^{\frac{n}{2}-1}}{6} > 2^{\frac{n}{2}-4}$ different advice strings. □

References

1. Bean, D.R.: Effective Coloration. J. Symbolic Logic 41(2), 469–480 (1976)
2. Böckenhauer, H.-J., Komm, D., Královič, R., Rossmanith, P.: On the Advice Complexity of the Knapsack Problem. In: Fernández-Baca, D. (ed.) LATIN 2012. LNCS, vol. 7256, pp. 61–72. Springer, Heidelberg (2012)
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. 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)
6. 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. LNCS, vol. 5555, pp. 427–438. Springer, Heidelberg (2009)
7. Finch, S.R.: Mathematical Constants (Encyclopedia of Mathematics and its Applications). Cambridge University Press, New York (2003)
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. Gyárfás, A., Lehel, J.: On-line and first fit colorings of graphs. Journal of Graph Theory 12(2), 217–227 (1988)
10. 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)
11. Kierstead, H.A.: Recursive and on-line graph coloring. In: Ershov, Y.L., Goncharov, S.S., Nerode, A., Remmel, J.B., Marek, V.W., (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)
12. Kierstead, H.A., Trotter, W.T.: 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)
13. 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)
14. Sloane, N.J.A.: Sequence A000073 in The On-Line Encyclopedia of Integer Sequences (2012), Published electronically, http://oeis.org/A000073
15. Vishwanathan, S.: Randomized online graph coloring. Journal of Algorithms 13(4), 657–669 (1992)

Deep Coalescence Reconciliation with Unrooted Gene Trees: Linear Time Algorithms*

Paweł Górecki[1] and Oliver Eulenstein[2]

[1] Department of Mathematics, Informatics and Mechanics, University of Warsaw, Poland
gorecki@mimuw.edu.pl
[2] Department of Computer Science, Iowa State University, USA
oeulenst@cs.iastate.edu

Abstract. Gene tree reconciliation problems invoke the minimum number of evolutionary events that reconcile gene evolution within the context of a species tree. Here we focus on the deep coalescence (DC) problem, that is, given an unrooted gene tree and a rooted species tree, find a rooting of the gene tree that minimizes the number of DC events, or DC cost, when reconciling the gene tree with the species tree. We describe an $O(n)$ time and space algorithm for the DC problem, where n is the size of the input trees, which improves on the time complexity of the best-known solution by a factor of n. Moreover, we provide an $O(n)$ time and space algorithm that computes the DC scores for each rooting of the given gene tree. We also describe intriguing properties of the DC cost, which can be used to identify credible rootings in gene trees. Finally, we demonstrate the performance of our new algorithms in an empirical study using data from public databases.

Species trees represent the evolutionary history of species and play a key-role in a broad spectrum of applications, including comparative genomics, population divergence, and understanding patterns of diversification [3]. There has also been an increased interest in species trees to maintain biodiversity [7,10], and to study the effects of global change [9,19,21]. Species trees are traditionally inferred from the evolutionary history of gene families. Therefore, it is assumed that the history of a gene family, which can be represented as a gene tree, and the corresponding species tree are identical. While accurate species trees are crucial for their proper interpretation, for many gene families their gene trees disagree with the topology of the actual species tree [15,14,16]. Complex evolutionary events like deep coalescence, gene duplication and subsequent loss, and horizontal gene transfer can cause tremendous heterogeneity in gene trees that obscures species relationships. Reconciling such gene trees with a species tree by invoking the minimum number of evolutionary events, or *reconciliation cost*, is a common and well-studied approach to address these complications, and extensions of this approach are used to infer species trees [4,11,15,14,16]. While classical reconciliation problems are only applicable to rooted trees, there is a need to reconcile unrooted gene trees with rooted species trees. Most standard inference methods, like maximum parsimony or

* This work was supported by MNiSW (#N N301 065236) to PG, by the National Science Foundation (#0830012 and #106029) to OE, and by NCN (#2011/01/B/ST6/02777) and the NIMBioS Working Group "Gene Tree Reconciliation" to PG and OE.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 531–542, 2012.

maximum likelihood, infer only unrooted gene trees from molecular sequences. When evolutionary events cause heterogeneity in gene trees it is difficult, if not impossible, to correctly root them [18,20]. In contrast, species trees often encompass a trusted root, which, for example, can be based on the NCBI taxonomy [17]. However, reconciliation problems can be naturally extended to reconcile an unrooted gene tree with a rooted species tree, by seeking a rooting of the gene tree that invokes the minimum reconciliation cost when reconciled with the species tree [12,22]. In addition, such reconciliation problems provide a direct mechanism to root gene trees based on their reconciliation score.

Here we focus on the *DC problem*, that is defined as follows: find the rooting of a given unrooted gene tree that minimizes the number of deep coalescence events, called *DC cost*, when reconciled with a given rooted species tree. This problem can be solved in quadratic time [22]. When faced with the need to reconcile large collections of trees with thousands of genes, however, quadratic runtime becomes prohibitive. For example, the gene tree parsimony (GTP) problem for deep coalescence [23] seeks for a given collection of gene trees a species tree with the overall minimum DC cost. Local search heuristics aim to solve this intrinsically difficult problem by solving the DC problem for thousands to hundreds of thousands instances that reconcile gene trees with typically large species trees [1,5].

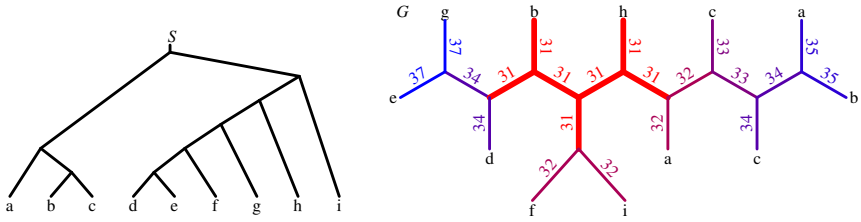


Fig. 1. Right side: An unrooted gene tree G (right). Left side: A rooted species tree S. Each edge in G is labeled by the DC score when G is rooted at this edge, and then reconciled with S. There are 7 optimal rootings forming the valley, which is the red-colored subtree in G, with the DC score 31. The remaining colors identify rootings with DC scores of 32, 33, 34, 35, and 37.

Our Contributions. Here, we describe an $O(n)$ algorithm for the DC problem, where n is the size of the given input trees, that improves on the best-known solution for this problem [22] by a factor of n. This substantial improvement can significantly speed-up applications relying on solutions to the DC problem (e.g., GTP heuristics based on the DC cost [1]). In fact, a variant of our algorithm provides the DC cost of every rooting of the given unrooted gene tree in $O(n)$ time and space. Computing these costs can be beneficial in identifying and supporting rootings of gene trees. Our original algorithms are designed for reconciling full binary trees as well as multifurcated trees. Furthermore, we show that the DC cost relates to the intriguing convexity property that states that the DC scores of all rootings in the unrooted gene tree form "terraces" surrounding a unique "valley". More precisely, the places of all optimal rootings in the gene tree induce a unique subtree, the "valley". The DC scores of rootings

along every path from this subtree to a leaf are monotonically increasing, which forms the "terraces". An example is depicted in Fig. 1. The convexity property can be crucial for the credible identification of rootings in gene trees. Optimal rootings of the gene tree can also be found using reconciliation costs other than the DC cost. For example the *duplication and loss (DL) cost* is the minimum number of duplications and losses that need to be invoked to reconcile a rooted gene tree with a rooted species tree. We show that every optimal rooting of an unrooted tree under the DL score is also an optimal rooting under the DC score. Finally, we demonstrate the performance of our algorithms and their quality in determining rootings of gene trees using an empirical data set of over 3000 gene family trees inferred from OrthoMCL database [6].

1 Basic Definitions and Preliminaries

1.1 Basic Definitions and Notation

We begin by recalling some basic definitions from phylogenetic theory. Let $\mathcal{I}$ be the set of taxa. An *unrooted gene tree* is an acyclic, connected, and undirected graph in which each node has degree 3 (internal) or 1 (leaves), and every degree-one vertex is labeled with an element from $\mathcal{I}$.

A *rooted* binary tree is defined similarly to an unrooted gene tree, with the difference that it has a distinguished vertex, called *root*, which have a degree of two. Tree T can be viewed as an upper semilattice with the least upper bound operation $+$, and the top element denoted by $\top$; that is, the root of T. In other words, for vertices $a, b \in V_T$, $a + b$ is the least common ancestor of a and b in T.

A *species tree* is a rooted binary tree with leaves uniquely labeled by the elements from $\mathcal{I}$. *A rooted gene tree* is a rooted binary tree with leaves labeled by the elements from $\mathcal{I}$. For a rooted tree T, by $T(v)$ we denote the subtree of T rooted at v. By $L(T)$, we denote the set of all leaf labels in T. For a node v in a rooted gene tree or a species tree T, by $c(v)$ we denote the cluster of v, that is, $L(T(v))$. For instance, a species tree $(a, (b, c))$ has 5 nodes with the following clusters: a, b, c, bc and abc. Let $M_G \colon V_G \to V_S$ be the *least common ancestor (lca) mapping*, from a rooted tree G into S that preserves the labeling of the leaves. Note, that if $a, b \in V_G$ are the children of v, then $M_G(v) = M_G(a) + M_G(b)$. An example is depicted in Fig. 2.

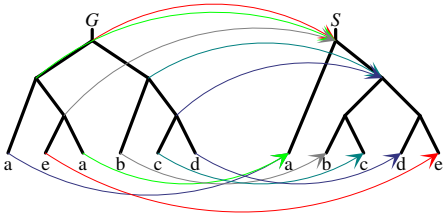


Fig. 2. The lca-mapping M from the gene tree G into the species tree S

1.2 DC Measure for Comparing Unrooted Gene Trees with Species Trees

The deep-coalescence (DC) score between a rooted gene tree T and a species tree S, such that $L(T) \subseteq L(S)$ is defined as follows [23]:

$$DC(T,S) := \sum_{\substack{g \in \text{int}(T) \\ a,b \text{ children of } g}} ||M_T(a), M_T(b)||, \tag{1}$$

where $\text{int}(T)$ is the set of internal nodes in T, $M_T \colon V_T \to V_S$ is the lca-mapping, and $||x,y||$ denotes the number of edges on the path connecting x and y in S.

For the remainder of this work we denote by $G = \langle V_G, E_G \rangle$ an unrooted gene tree and by S a species tree, such that $L(G) \subseteq L(S)$. A rooting in G is defined by selecting an edge $e \in E_G$ on which the root is to be placed. Such a rooted tree will be denoted by G_e. To distinguish between rootings of G, all defined symbols for a rooted trees will be extended by inserting index e. For example, M_e is the lca-mapping from G_e to S, etc.

Definition 1. *The unrooted* Deep coalescence (urDC) *measure between an unrooted tree G and a rooted tree S, is defined as follows:*

$$urDC(G,S) := \min\{DC(G_e,S) \colon e \in E_G\}. \tag{2}$$

The edges $e \in E_G$, for which the rooting G_e has the minimal DC score will be called *optimal*.

2 Methods

2.1 Orientation and Labeling of G

First, we transform G into a directed graph $\widehat{G}$, by replacing each undirected edge $\{v,w\}$ from G by a pair of directed edges $\langle v,w \rangle$ and $\langle w,v \rangle$. Formally, we have $\widehat{G} = \langle V_G, \widehat{E_G} \rangle$, where $\widehat{E_G} = \{\langle v,w \rangle, \langle w,v \rangle \colon \{v,w\} \in E_G\}$.

The edges in $\widehat{G}$ are labeled by nodes of S as follows. If $v \in V_G$ is a leaf labeled by a, then the edge $\langle v,w \rangle \in \widehat{E_G}$ is labeled a. When v is an internal node in $\widehat{G}$ we assume that $\langle w_1, v \rangle$ and $\langle w_2, v \rangle$ are labeled by b_1 and b_2, respectively. Then the edge $\langle v, w_3 \rangle \in \widehat{E_G}$, such that $w_3 \neq w_1$ and $w_3 \neq w_2$ is labeled $b_1 + b_2$. Such labeling will be used to explore mappings of rootings of G.

Every internal node v and its neighbors in $\widehat{G}$ define a subtree of $\widehat{E_G}$, called a *star* with a center v, as depicted in Fig. 3a. The edges $\langle v, w_i \rangle$ are called *outgoing*, while the edges $\langle w_i, v \rangle$ are called *incoming*. We will refer to the undirected edge $\{v, w_i\}$ as e_i, for $i = 1,2,3$.

Under the assumption that $M_{G_e} = \top$ (it is independent of e), there are several types of possible star topologies based on the labeling (for proofs and details see [12]): (S1) a star has one incoming edge labeled $\top$ and two outgoing edges labeled $\top$, and these edges are connected to the three siblings of the center, (S2) a star has exactly two outgoing edges labeled by $\top$, (S3) a star has all outgoing edges and exactly one incoming edge labeled by $\top$, (S4) a star has all edges labelled by $\top$, and (S5) a star has all outgoing edges and exactly two incoming edges labeled by $\top$. The star topologies are depicted in Fig. 3c, where we use the simplified notation introduced in Fig 3b.

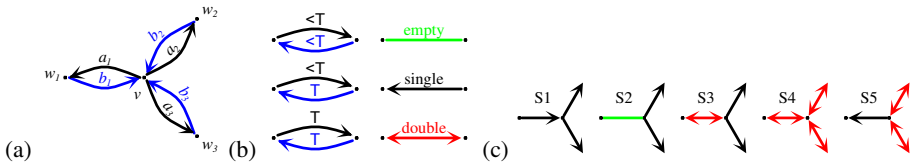


Fig. 3. (a) A star created after the transformation of gene tree edges adjacent to a node v. (b) Simplified notation of edges used in stars. (c) S1-S5 - all possible types of stars that can be present in the transformed gene trees.

2.2 DC Cost and the Star Topologies

We analyse how the cost is changing when we move the root in G by one position. Throughout this section we use the notation of a star that is introduced in Fig. 3.

For an edge e in G, let $\lambda_e = DC(G_e, S)$.

Proposition 1. *Let* $b_1 = \top$. *Then,*

$$\lambda_{e_1} \leq \lambda_{e_2} = \lambda_{e_3}.$$

Moreover, the equality holds only if $a_1 = \top$.

Proof: All rootings of G share the same subtrees attached to w_1, w_2 and w_3, therefore, all costs share the same component c coming from the partial DC score for these subtrees. The rest follows in a straightforward way from the definition of the rooted DC score and Fig. 3a: $\lambda_{e_1} = c + x + y + z$ and $\lambda_{e_2} = \lambda_{e_3} = c + x + y + 2 * z$, where $x = ||b_3, a_1||$, $y = ||b_2, a_1||$ and $z = ||a_1, \top||$ (note that $a_1 = b_3 + b_2$). Clearly, $z = 0$ if and only if $a_1 = \top$. []

From Proposition 1 it follows that all rootings of the edges in stars S3-S5 have the same cost. Moreover, in the case of S1, placing the root on the incoming edge labeled by $\top$ (the left edge of S1 in Fig.3c) yields a smaller DC score than in the other rootings.

Proposition 2. *Let* $b_i \neq \top$ *for all* $i = 1, 2, 3$. *Then, from exactly one* i, *say* $i = 1$, $a_i \neq \top$, *and*

$$\lambda_{e_1} < \lambda_{e_2} = \lambda_{e_3}.$$

Proof: Observe that this case is related to the star S2, where $a_2 = a_3 = \top$ and $a_1, b_1 \neq \top$. Similarly, to the previous proof, we have the following equalities: $\lambda_{e_1} = c + v + x + y + v$ and $\lambda_{e_2} = \lambda_{e_3} = c + v + x + y + 2 * z$, where $x = ||b_3, a_1||$, $y = ||b_2, a_1||$, $z = ||a_1, \top||$ and $v = ||b_1, \top||$. By assumption that $a_1 \neq \top$, $z > 0$. This completes the proof. []

2.3 Algorithm

Theorem 1. *Let* G *be an unrooted tree and* S *be a species tree such that* $L(G) \subseteq L(S)$. *Let* $\mathbf{Min}_G$ *be the set of optimal edges in* G *for the urDC score computed for* G *and* S.

(i) If $\widehat{G}$ has a double edge then $\mathbf{Min}_G$ contains edges from all stars S3-S5 present in $\widehat{G}$.
(ii) If $\widehat{G}$ has an empty edge then this edge is the only optimal edge (star S2).
(iii) $\mathbf{Min}_G$ is a full subtree[1] of G.
(iv) An optimal edge in $\mathbf{Min}_G$ can be found by using a greedy method of gradient descent: leave a star in the reverse direction of the outgoing edge labeled $\top$. Terminate when a symmetric edge is found.

Proof: (i) Assume that $\widehat{G}$ has a double edge. It follows from the star topologies that $\widehat{G}$ has a star S3, S4 or S5. Then, it is easy to show that the set of stars from G contains only stars: S1, S3, S4 or S5. Moreover, the set of double edges induces a connected subgraph of G. Now, with Proposition 1 it follows directly that all rootings of edges from stars S3-S5 share the same DC score. Additionally, by Proposition 2 this cost is smaller than scores of the other edges; that is, the edges that are not present in S3-S5 stars. The proof (inductive) is straightforward and omitted for brevity. This completes the proof of case (ii).

(ii) Observe that $\widehat{G}$ has at most 2 stars of type S2 that share the empty edge e. All remaining stars are of type S1. From Proposition 2 it follows that the score of e is smaller than scores of the other edges in stars of type S2. By Proposition 1 we have that the score of e is smaller than scores of all other edges (not present in S2 stars). Again, we omit the straightforward inductive proof for brevity. This completes the proof of case (ii).

(iii) Suppose that $\mathbf{Min}_G$ has more than one edge. Then, $\mathbf{Min}_G$ contains at least one double edge (see (i) and (ii)). Then, by (i), $\mathbf{Min}_G$ consists of all edges present in stars S3-S5. Observe that these edges induce a connected subgraph of G, which is a full subtree of G.

(iv) The claim follows directly from statements (i), (ii) and (iii). □

Algorithm 1 details the greedy descent procedure (Thm. 1 (iv)) and the urDC score computation. An example of star topologies is depicted in Fig. 4.

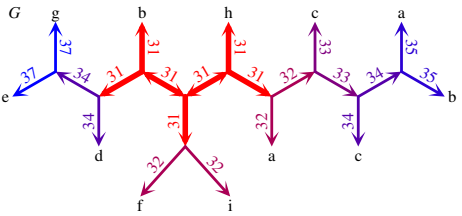


Fig. 4. Gene tree G from Fig. 1 converted into the star representation. This tree have 3 optimal stars: two of type S3 and one of type S4.

The proof of correctness of the optimal edge search follows immediately from Thm. 1 and the following property (easy details omitted):

Lemma 1. *For any edge $e = \{x, y\}$ in G, $m_{x,y} = M_e(x)$. Moreover, if e' is an edge of G not present in $G_e(x)$, then $m_{x,y} = M_{e'}(x)$.*

[1] A subtree T' of T is a full subtree of T if for any node $v \in T$ and the set of edges E adjacent with v in T, either $E \subseteq E_{T'}$ or $E \cap E_{T'}$ has at most one element.

Algorithm 1. Optimal edge and urDC score computation

1: **Input:** A species tree S, an unrooted gene tree G with at least three leaves, $L(G) \subseteq L(S)$.
2: **Output:** $urDC(G, S)$.
3: Compute $\widehat{G}$ with the labelling of edges by as follows. Let $m_{v,w}$ be the label, that is a species node, of $\langle v, w\rangle$ in $\widehat{G}$. For each directed edge $\langle v, w\rangle$ in $\widehat{G}$, let
$$m_{v,w} := \begin{cases} s & v \in G \text{ and } s \in S \text{ are leaves with the same label,} \\ m_{x,v} + m_{y,v} & \text{if } \{x, y, w\} \text{ is the set of all neighbours of } v \text{ in } G. \end{cases}$$
4: For some edge $e = \{x, y\}$ of G, let $\top := m_{x,y} + m_{y,x}$ (here $\top$ is the mapping of the root of G_e; it is independent of e).
5: **Let** v be a node from $\widehat{G}$.
6: **While** there exits a node w adjacent with v such that $m_{w,v} = \top \neq m_{v,w}$:
let $v := w$ (star S1, continue search).
7: **Let** $e = \{v, w\}$, where w is a neighbor of v such that $m_{v,w} = \top = m_{w,v}$ or $m_{v,w} \neq \top \neq m_{w,v}$ (e is an optimal and symmetric edge).
8: **Return:** $\delta_{v,w} + \delta_{w,v} + ||m_{v,w}, m_{w,v}||$, where
$$\delta_{a,b} = \begin{cases} 0 & \text{if } a \text{ is a leaf} \\ \delta_{c,a} + \delta_{d,a} + ||m_{c,a}, m_{d,a}|| & \text{if } \{c, d\} \text{ are the children of } a \text{ in } G_e. \end{cases}$$
.

In other words, the mappings m computed in the third line of Alg. 1, are shared among rootings of G. These mappings can be used to compute the optimal cost or even costs of all rootings, by using the values of $\delta_{a,b}$. The following lemma justifies this observation.

Lemma 2. *Under notation introduced in Alg.1. If $\langle a, b\rangle \in \widehat{E_G}$ such that b is not a child of a in G_e, we have $\delta_{a,b} = DC(G_e(a), S)$, where e is the optimal edge found by Alg. 1.*

Theorem 2. *The time complexity of Alg. 1 is $O(|G| + |S|)$.*

Proof: Let $|E_G| = k$ and $|S| = n$. It follows from [2] that a single least common ancestor query can be computed in $O(1)$ number of steps after an initial preprocessing requiring $O(n)$ steps. The labeling of $\widehat{G}$ requires $2k$ lca-queries. Therefore, the procedure of finding an optimal edge in G can be completed in $O(k + n)$ time. Now, we analyze the time complexity of the score computation. Assume that the distances from $\top$, called here $\top$-queries, are calculated for each node of S. Observe that such preprocessing can be done in $O(n)$ time. Then, each distance of the form $||x, y||$ we compute by using three $\top$-queries and one lca-query: $||x, y|| = ||x, \top|| + ||y, \top|| - 2 * ||x + y, \top||$. In total, we have $3k$ $\top$-queries and k lca-queries in the 8th line of Alg.1. Finally, we have the preprocessing phase that can be done in $O(n)$ time, $3k$ lca-queries and $3k$ $\top$-queries to compute the optimal DC score. This completes the proof. []

It should be clear, that the space complexity of Alg. 1 and is $O(|G| + |S|)$. The next theorem presents the most general result for the DC score computation.

Theorem 3. *For an unrooted gene tree G and a species tree S, computing DC scores for all rootings of G can be done in $O(|S| + |G|)$ time.*

Proof: Let $|E_G| = k$ and $|S| = n$. Lemma 1 and Lemma 2 state that the values of m and δ are shared among rootings of G. Therefore, if $e = \{x, y\}$ is an edge in G, then $DC(G_e, S) = \delta_{x,y} + \delta_{y,x} + ||m_{x,y}, m_{y,x}||$. Thus, first we need to compute m and δ for each directed edge from $\widehat{E_G}$ ($2k$ edges). Computing δ requires $6k$ $\top$-queries (see the proof of Thm. 2) and $2k$ lca-queries. Additionally, for the computation of DC scores, we need $3k$ $\top$-queries and k lca-queries. Finally,

after the preprocessing phase that can be completed in $O(n)$ time, we have $5k$ lca-queries and $9k$ $\top$-queries to compute m (see Thm. 2), δ and all DC scores. []

2.4 Multifurcated Trees

Most of gene trees inferred from sequences and species taxonomies have multifurcations that represent uncertainties. The analysis of multifurcations in gene and species trees leads to essentially the same results. Formally, a multifurcated tree has potentially higher degrees of internal nodes than the binary gene and species trees (see definitions in Section 1.1). For this reason, stars in the multifurcated case can have more than 3 edges. Their construction, which is omitted here for brevity, is based on the approach taken from binary trees (see Fig. 3). All multifurcated stars are depicted in Fig. 5. We conclude that all binary stars can be naturally transformed into the multifurcated ones. However, one new topology has to be defined. For instance, when $G = S = (a, b, c)$, the unrooted gene tree G has exactly one star of a new type M6. Observe, that a star of type M6 cannot be inferred when both G and S are binary.

Similarly to Thm. 1, we have the following result:

Theorem 4. *Let G be an unrooted tree and S be a rooted species tree, both with multifurcations, such that $L(G) \subseteq L(S)$. Let $\mathbf{Min}_G$ be the set of optimal edges in G for the urDC score computed for G and S.*

(i) If $\widehat{G}$ has a double edge then $\mathbf{Min}_G$ contains edges from all stars M3-M5 present in $\widehat{G}$.
(ii) If $\widehat{G}$ has an empty edge then this edge is the only optimal edge (star M2).
(iii) If $\widehat{G}$ has no symmetric edges then $\widehat{G}$ has exactly one star of type M6. All edges if this star are optimal.
(iv) $\mathbf{Min}_G$ is a full subtree of G.
(v) An optimal edge in $\mathbf{Min}_G$ can be found a greedy method of gradient descent similarly to the binary case (based on Alg. 1).

Note that Alg. 1 and its properties (Thm. 2 and Thm. 3) can be easily adopted to cover multifurcated trees and stars of type M6. Observe, that such modified algorithm will have the same time and space complexity as Alg. 1. Here, we omit proofs and formal details for brevity.

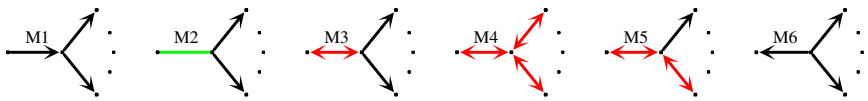


Fig. 5. Star topologies for the multifurcated trees, M1 is a multifurcated variant of S1, M2 of S2, etc. The star M5 has at least two double edges and at least one single edge. Note, that M6 is not present if both the gene tree and the species tree are binary.

2.5 Rooting at Nodes

Gene trees can be either rooted at an edge or an internal node. Our approach can naturally cover this case, and it can be shown, that the set of optimal edge-rootings can be extended with the rootings placed on internal nodes of the optimal valley. We skip formal details for brevity.

2.6 Relations between urDC and the Duplication-Loss Cost

We present relationships between the duplication-loss and the urDC cost. For more details, please refer to [12].

Theorem 5. *Let S be a binary species tree and G a binary unrooted gene tree. Then each optimal rooting of the weighted duplication-loss cost is optimal for the urDC measure.*

Proof: The optimal edges for the urDC cost (see Fig. 3) are: the symmetric edge in S2-S3 stars and all edges present in S4-S5 stars. From Thm. 7 [12], the optimal rootings for the duplication-loss cost are determined by all edges of stars S4-S5 and by the symmetric edge of stars S2-S3. By Thm. 1, these edges are also optimal for the urDC cost. □

Fig. 6 illustrates an example of the duplication-loss cost analysis for the trees from the previous figures. It follows from the theory of unrooted reconciliation for the duplication-loss cost that the optimal rootings determine evolutionary scenarios that differ only in the duplications mapped to the rooting edge. This observation supports the biological importance of the urDC cost evaluation.

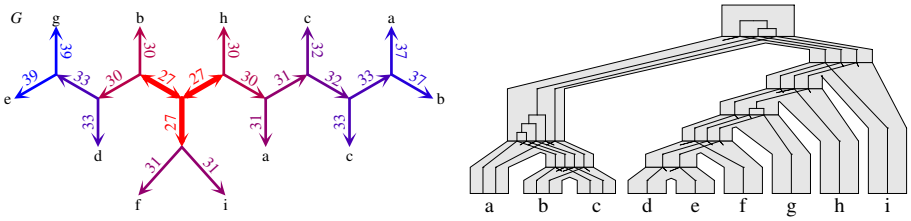


Fig. 6. Left: the duplication-loss cost example with the star representation of the trees from Fig. 1. There are 3 optimal edges (star S4) with cost 27. Observe that these edges are also optimal for the urDC cost (Fig. 4). Right: visualization of an optimal evolutionary scenario, that is, the embedding of the rooting of G placed on the bottom optimal edge into S.

3 Experiment

In our experimental study we computed 3150 unrooted, and mostly multifurcated, gene family trees from the OrthoMCL-DB database [6]. Our new urDC measure allowed us

Fig. 7. Top right: an unrooted gene tree for ribosome-binding factors A and similar proteins from the OrthoMCL family OG5_131740[6]. Top left: a rooted species tree based on the NCBI taxonomy [17] inferred for the set of species present in the OG5_131740 family. Bottom: the result of the urDC cost analysis. The minimum DC score (81) has 14 edges.

to compare each of these unrooted gene trees with the rooted species taxonomy (which includes multifurcations) from the NCBI taxonomy [17].

To infer the gene trees, we first extracted orthologous groups of protein sequences from the OrthoMCL DB v5.0 [6] with at least 11 members. Then, we aligned the protein sequences of each group using the program Muscle [8] by applying the default parameter setting. Maximum likelihood gene trees were inferred from each of these alignments using the program PhyML [13]. The resulting unrooted and binary ML-trees usually contained short edges. To avoid uncertain edges we collapsed edges with an edge weight that was smaller than 0.06. As a result we inferred 3150 unrooted gene trees with multifurcations.

The species tree is based on the NCBI taxonomy [17]. For each inferred gene tree G, we transformed the NCBI taxonomy into a rooted species tree that contains only the species from G by removing unused species and contracting edges.

Using an implementation of our algorithm for the urDC measure, we reconciled each of the 3150 gene trees with the corresponding rooted species taxonomy. The overall computation of the scores was completed within a minute on a standard workstation. An example for one of these reconciled gene trees is depicted in Figure 7. The minimum urDC score for this gene tree is 81 and can be obtained by rooting the tree at one of the edges that are forming the valley consisting of 14 optimal edges.

Software and exemplary datasets are freely available through the following website: `http://bioputer.mimuw.edu.pl/gorecki/urdc`

4 Conclusion and Outlook

In this article we addressed the problem of reconciling an unrooted gene tree with a rooted species tree under the deep coalescence cost. We presented novel linear time and space algorithms for computing DC costs, which can be used to support a wide variety of applications, including (i) the rooting of unrooted gene trees, (ii) the reconciliation of unrooted gene trees with rooted species trees based on the DC cost, and (iii) the comparative phylogenetic analyses of unrooted gene trees with rooted species trees. Further, we showed that the optimal rootings under the DC cost and the duplication-loss cost are closely related, and that these rootings form a valley in the gene tree. These properties can provide a valuable approach for establishing credible rootings.

Our results suggest to investigate on extending our algorithms to solve more general problems under the DC cost, which include (i) the reconciliation of an unrooted gene tree with an unrooted species tree, (ii) the error correction in unrooted gene trees, and (iii) local search algorithms for unrooted GTP heuristics.

References

1. Bansal, M.S., Burleigh, J.G., Eulenstein, O.: Efficient genome-scale phylogenetic analysis under the duplication-loss and deep coalescence cost models. BMC Bioinformatics 11(suppl. 1), S42 (2010)
2. Bender, M.A., Farach-Colton, M.: The LCA Problem Revisited. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000)

3. Bininda-Emonds, O.R.P., Gittleman, J.L., Steel, M.A.: The (super) tree of life: procedures, problems, and prospects. Annual Review of Ecology and Systematics 33, 265–289 (2002)
4. Burleigh, J.G., Bansal, M.S., Eulenstein, O., Hartmann, S., Wehe, A., Vision, T.J.: Genome-scale phylogenetics: inferring the plant tree of life from 18,896 discordant gene trees. Systematic Biology 60, 117–125
5. Chaudhary, R., Bansal, M., Wehe, A., Fernández-Baca, D., Eulenstein, O.: iGTP: A software package for large-scale gene tree parsimony analysis. BMC Bioinformatics 11(1), 574 (2010)
6. Chen, F., Mackey, A.J., Stoeckert, C.J., Roos, D.S.: Orthomcl-db: querying a comprehensive multi-species collection of ortholog groups. Nucleic Acids Research 34(suppl. 1), D363–D368
7. Davies, J.T., Fritz, S.A., Grenyer, R., Orme, C.D.L., Bielby, J., Bininda-Emonds, O.R.P., Cardillo, M., Jones, K.E., Gittleman, J.L., Mace, G.M., Purvis, A.: Phylogenetic trees and the future of mammalian biodiversity. PNAS 105, 11556–11563 (2008)
8. Edgar, R.C.: MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Research 32, 1792–1797 (2004)
9. Edwards, E.J., Still, C.J., Donoghue, M.J.: The relevance of phylogeny to studies of global change. Trends In Ecology & Evolution 22(5), 243–249 (2007)
10. Forest, F., et al.: Preserving the evolutionary potential of floras in biodiversity hotspots. Nature 445(7129), 757–760 (2007)
11. Górecki, P., Tiuryn, J.: DLS-trees: A model of evolutionary scenarios. Theor. Comput. Sci. 359(1-3), 378–399 (2006)
12. Górecki, P., Tiuryn, J.: Inferring phylogeny from whole genomes. Bioinformatics 23(2), e116–e122 (2007)
13. Guindon, S., Delsuc, F., Dufayard, J., Gascuel, O.: Estimating maximum likelihood phylogenies with PhyML. Methods Mol. Biol. 537, 113–137 (2009)
14. Koonin, E.V., Galperin, M.Y.: Sequence - evolution - function: computational approaches in comparative genomics. Kluwer Academic (2003)
15. Maddison, W.P.: Gene trees in species trees. Systematic Biology 46, 523–536 (1997)
16. Page, R.D.M., Holmes, E.C.: Molecular evolution: a phylogenetic approach. Blackwell Science (1998)
17. Sayers, E.W., et al. Database resources of the national center for biotechnology information. Nucleic Acids Research 37(suppl. 1), D5–D15 (2009)
18. Smith, A.: Rooting molecular trees: problems and strategies. Biol. J. Linn. Soc. 51, 279–292
19. Thuiller, W., Lavergne, S., Roquet, C., Boulangeat, I., Lafourcade, B., Araujo, M.: Consequences of climate change on the tree of life in Europe. Nature 470(7335), 531–534 (2011)
20. Wheeler, W.: Nucleic acid sequence phylogeny and random outgroups. Cladistics – The International Journal of the Willi Hennig Society 51, 363–368 (1990)
21. Willis, C.G., Ruhfel, B., Primack, R.B., Miller-Rushing, A.J., Davis, C.C.: Phylogenetic patterns of species loss in thoreau's woods are driven by climate change. PNAS 105, 17029–17033 (2008)
22. Yu, Y., Warnow, T., Nakhleh, L.: Algorithms for MDC-Based Multi-locus Phylogeny Inference. In: Bafna, V., Sahinalp, S.C. (eds.) RECOMB 2011. LNCS, vol. 6577, pp. 531–545. Springer, Heidelberg (2011)
23. Zhang, L.: From Gene Trees to Species Trees II: Species Tree Inference by Minimizing Deep Coalescence Events. IEEE/ACM Transactions on Computational Biology and Bioinformatics 8, 1685–1691 (2011)

On the 2-Central Path Problem*

Yongding Zhu and Jinhui Xu

Department of Computer Science and Engineering,
State University of New York at Buffalo,
Buffalo, NY 14260, USA
{yzhu3,jinhui}@buffalo.edu

Abstract. In this paper we consider the following 2-Central Path Problem (2CPP): Given a set of m polygonal curves $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ in the plane, find two curves P_u and P_l, called 2-*central paths*, that best represent all curves in $\mathcal{P}$. Despite its theoretical interest and wide range of practical applications, 2CPP has not been well studied. In this paper, we first establish criteria that P_u and P_l ought to meet in order for them to best represent $\mathcal{P}$. In particular, we require that there exists parametrizations $f_u(t)$ and $f_l(t)$ ($t \in [a, b]$) of P_u and P_l respectively such that the maximum distance from $\{f_u(t), f_l(t)\}$ to curves in $\mathcal{P}$ is minimized. Then an efficient algorithm is presented to solve 2CPP under certain realistic assumptions. Our algorithm constructs P_u and P_l in $O(nm \log^4 n 2^{\alpha(n)}\alpha(n))$ time, where n is the total complexity of $\mathcal{P}$ (i.e., the total number of vertices and edges), m is the number of curves in $\mathcal{P}$, and $\alpha(n)$ is the inverse Ackermann function.Our algorithm uses the parametric search technique and is faster than arrangement-related algorithms (i.e. $\Omega(n^2)$) when $m \ll n$ as in most real applications.

1 Introduction

In this paper, we introduce a relatively new concept of *k-central paths* among a set of paths. Given a set of polygonal curves $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ in the plane, the k-central path problem is to find k curves, called *k-central paths*, that "best" represent all curves in $\mathcal{P}$, where k is an integer and $1 \leq k \leq m$. The problem is a generalization of the well-known k-center problem of points, where the centers are naturally the best representatives of the set of input points. For polygonal curves, however, it is less clear what are the "best representatives". To define better representatives, we let $f_1(t), f_2(t), \ldots, f_k(t)$ ($t \in [a, b]$) be a parametrization of the k-central paths and propose two criteria for the selection of the k-central paths: (1) Each k-central path preserves the shapes of curves in $\mathcal{P}$; (2) For any given $t \in [a, b]$, the distance from $\{f_1(t), f_2(t), \ldots, f_k(t)\}$ to any input path in $\mathcal{P}$ is minimized, that is, we need to find the smallest radius r such that every path in $\mathcal{P}$ intersects some disk $D(f_i(t), r)$ centered at $f_i(t)$ with radius r. The first criterion is relatively easy to fulfill by requiring all k-central paths and the paths in $\mathcal{P}$ (except some outliers) start at the same region and end at some common region in the plane. For the second criterion, we need to solve an optimization problem of minimizing $\max_{a \leq t \leq b, i=1,2,\ldots,m} d(\{f_1(t), f_2(t), \ldots, f_k(t)\}, P_i)$, where $d(\cdot, \cdot)$ is the Euclidean distance (i.e., the closest distance).

* This research was partially supported by NSF through a CAREER award CCF-0546509 and grants IIS-0713489 and IIS-1115220.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 543–555, 2012.

The problem arises in a variety of application fields. With the rapid advancement of new technologies like sensor networks, global positioning systems, and mobile phone networks, it is now possible to record the traces of certain objects such as patients, animals, vehicles, hurricanes for later analysis. In such applications, traces are usually represented as *trajectories*, which is a time-stamped path that a moving object traverses [4], i.e., a sequence of 2D or 3D points with time stamps. In some applications though, the temporal information (time stamps) may be discarded or ignored when studying their spatial patterns, thus resulting in a set of polygonal trajectories (or paths) in the plane. Effective analysis of the recorded data usually requires finding the common or representative patterns of the trajectories, and thus can be formulated as a k-central path problem. The k-central path problem is also motivated by problems in segmentation, medical imaging, data mining and pattern recognition. In multi-surfaces segmentation [15], the accurate boundary of an object in noisy images often needs to first generate a set of candidate segmentations using algorithms following different optimization criteria and then derive the most likely solution from them (e.g., segmenting by ensemble clustering method [14]), which can be formulated as a problem of finding the "k-central" paths. Despite its importance, studies on "k-center" or "k-median" of curves have only been initiated recently.

In article [4], Buchin *et al.* investigated the so-called *median trajectory* (actually median path as no temporal information is considered). In their definition, the median trajectory has to be part of the input trajectories. In addition, the median trajectory must be in the "middle" of the input trajectories. More precisely, for any point p of the median trajectory, the minimum number of distinct trajectories that p must cross to reach the unbounded face is $(m+1)/2$, where m is the number of input trajectories. They showed that by applying an arrangement algorithm, the median trajectory can be computed in $O(n^2)$ time, where n is the total complexity of the input [4].

In this paper, we extend the concept of "1-center (e.g., median)" to "2-center" and introduce the following 2-central path problem (2CPP): Given a set of polygonal curves $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ in the plane, find two curves P_u and P_l, called 2-*central paths*, so that the maximum distance from $\{f_u(t), f_l(t)\}$ to curves in $\mathcal{P}$ is minimized for all t. To meet the first criterion proposed above, we assume that there are two vertical lines L_s and L_t such that every path in $\mathcal{P}$ starts on L_s and ends on L_t (i.e., the common starting and ending regions are lines). Note that this assumption is reasonable in many applications. For example, shopping carts with RFID tags are usually starting and ending at common locations.

To our best knowledge, the concept of "2-center" of curves has not been explored in the past. Existing algorithms are mainly for the 2-center problem of points [6,3,9,10,13,7,5], which seeks two congruent disks with minimum radius to cover a set of points. The point case of the 2-center problem has been studied extensively. It was believed to be at least quadratic-time solvable for decades [6,9,10]. The major breakthrough was achieved by Sharir. In [13], he showed that the planar 2-center problem can be solved in subquadratic time (i.e., $O(n \log^9 n)$). This result was improved to $O(n \log^2 n)$ by Eppstein by using randomized algorithm [7]. Later, Chan improved the deterministic running time to $O(n \log^2 n \log^2 \log n)$ [5].

In this paper, we investigate the geometric properties of the 2-central paths P_u and P_l and show that they can be constructed in $O(nm \log^4 n 2^{\alpha(n)} \alpha(n))$ time under certain realistic assumptions by using parametric search. Our result is near linear time when m is small (actually m is a constant in many applications).

2 Preliminary

Assumptions. For simplicity, we make the following assumptions on $\mathcal{P}$:

(1) General position assumption: No three edges intersect at one common point.
(2) Non-overlapping assumption: No two edges partially or fully overlap, and no vertex of one input curve lies on another curve.
(3) Monotonicity assumption: All curves in $\mathcal{P}$ are x-monotone.
(4) Common x-span assumption: All curves start at the same vertical starting line L_s, end at the same vertical ending line L_t, and lie completely between them (see Figure 1).

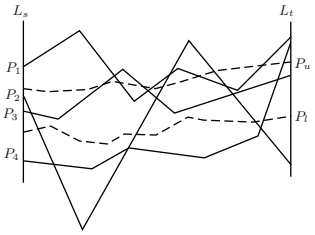


Fig. 1. An example of 2CPP. P_1, P_2, P_3 and P_4 are 4 input curves; P_u and P_l (dashed curves) are the 2-central paths.

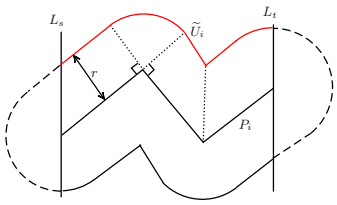


Fig. 2. The Minkowski sum $\mathcal{M}(P_i, r)$. The red curve is the upper boundary $\widetilde{U}_i$.

The above assumptions are reasonable in many applications. For instance, the first two assumptions can be easily satisfied by slightly perturbing the vertices of the curves. For the third assumption, since many moving objects (except for possible outliers) share similar spatial patterns, it is often possible to discard outliers in advance and uses certain transformations (e.g., using radial scanning and polar coordinates locally at those non-monotone places) to make them x-monotone. For the fourth assumption, one can find two vertical lines L_s and L_t and extends (or truncates) the two ends of each curve to them, without compromising the quality of the 2-central paths. With these assumptions, a natural way to parameterize curves in $\mathcal{P}$ and the 2-central paths P_u and P_l is to use x as the parameter. Thus throughout the paper, we use $f_u(x)$ and $f_l(x)$ to represent P_u and P_l, where $x \in [a, b]$ and a and b are the x-coordinates of L_s and L_t respectively.

Parametric Search. *Parametric search* is an important optimization technique developed by Meggido [11,12], and applied to numerous geometric optimization problems like the ones in [1,2]. Let n be the input data size and λ be a parameter. The goal of parametric search is to find a value λ^* of λ so that the output satisfies certain properties.

Assume that an oracle A_s exists to solve the following decision problem in T_s time: Given a constant λ, decide if it is larger than, smaller than, or equal to the desired value λ^*. Further, assume that A_s can be parallelized (denoted as A_p) using p processors in T_p parallel steps. Then, we can simulate A_p sequentially by generating p values of λ for each parallel step of A_p, sorting the p values of λ, and applying binary search using A_s as comparator. In this way, we can solve the original problem in $O(pT_p + T_sT_p \log p)$.

3 Decision Problem

As mentioned in last section, to solve 2CPP by parametric search, it is essential to solve the following fixed-size decision problem.

Definition 1 (Decision 2CPP or D2CPP). *Given a set of 2D polygonal curves $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ with total complexity n and a parameter r, decide if there exist two curves P_u and P_l such that $\max_{a \le x \le b, i=1,2,\ldots,m} d(\{f_u(x), f_l(x)\}, P_i)$ is less than or equal to r, where d is the Euclidean distance of two sets of points.*

To solve D2CPP, we first consider the Minkowski sums of input curves.

3.1 Minkowski Sum

Definition 2 (Minkowski Sum). *Let P_i be a path in $\mathcal{P}$, $i = 1, 2, \ldots, m$, and D_r be a closed disk of radius r and centered at the origin. The* Minkowski Sum *of P_i and D_r is $\mathcal{M}(P_i, r) = \{p_1 + p_2 | p_1 \in P_i, p_2 \in D_r, p_1 + p_2$ is located between L_s and $L_t\}$.*

The following few lemmas show some good properties of the Minkowski sum $\mathcal{M}(P_i, r)$.

Lemma 1. *Both $\mathcal{M}(P_i, r)$ and $\mathbf{R}^2 \backslash \mathcal{M}(P_i, r)$ are connected.*

Lemma 1 implies that $\mathcal{M}(P_i, r)$ is a closed region bounded by the upper and lower boundary curves, L_s and L_t. Next, we present an efficient algorithm to compute the upper and lower boundary curves of $\mathcal{M}(P_i, r)$, which can be easily parallelized. Without loss of generality, only the upper boundary of $\mathcal{M}(P_i, r)$ is considered; the lower boundary is symmetric and can be handled similarly. Let $\widetilde{U}_i$ be the upper boundary curve of $\mathcal{M}(P_i, r)$. As showed in Figure 2, $\widetilde{U}_i$ consists of line segments and circular arcs (i.e., an edge of $\widetilde{U}_i$ is either a line segment or a circular arc).

Definition 3. *A* candidate edge *of a site (vertex or edge) in P_i for $\widetilde{U}_i$ is defined as follows.*

1. *If the site is an (closed) edge e, the candidate edge C(e) is a segment parallel to e and above it by a distance of r with the endpoints of e and C(e) forming a rectangle (see Figure 3 (left));*
2. *If the site is a reflex vertex (looking from $+\infty$ of the y axis) v, the candidate edge C(v) is a circular arc centered at v, of radius r, and lying above P_i with its two endpoints on the rays emanating from v and orthogonal to the two adjacent edges of v respectively (see Figure 3 (right));*

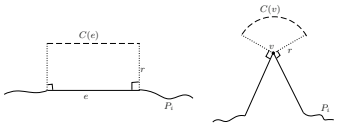


Fig. 3. Two cases of candidate edges. The input curve P_i is in solid lines. The dashed curves are candidate edges.

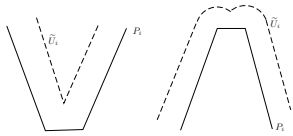


Fig. 4. Partial structures of the upper boundary of $\mathcal{M}(P_i, r)$. The upper boundary curve is in dashed line.

3. *If the site is a non-reflex vertex, no candidate edge exists.*

If v is one of the two endpoints of P_i, we treat v as a reflex vertex, in which case the reflex angle is equal to 2π.

Lemma 2. *Each edge of $\widetilde{U}_i$ is part of some candidate edge. Each candidate edge can only contain at most one edge of $\widetilde{U}_i$.*

It is possible that a candidate edge contains none of the upper boundary edges of $\mathcal{M}(P_i, r)$. For example, in Figure 4, the horizontal edges of P_i do not have edges of $\widetilde{U}_i$ associated with it.

Lemma 3. *The upper boundary of $\mathcal{M}(P_i, r)$ $\widetilde{U}_i$ is x-monotone.*

The following lemma shows that the ordering of the upper boundary curve is consistent with that of the input curve P_i.

Lemma 4. *Let α and β be two sites (either an edge or a vertex) of P_i. Let e_α and e_β be two edges of $\widetilde{U}_i$ contained by $C(\alpha)$ and $C(\beta)$ respectively. Then α is on the left of β if and only if e_α is on the left of e_β.*

Proof. Suppose α is on the left of β but e_α is on the right of β. Let $p_\alpha \in e_\alpha$ and $p_\beta \in e_\beta$ be two arbitrary points. Let $q_\alpha \in \alpha$ and $q_\beta \in \beta$ are the two closest points on P_i of p_α and p_β respectively. Then $d(p_\alpha, q_\alpha) = r$ and $d(p_\beta, q_\beta) = r$. By Lemma 3, we know that both P_i and $\widetilde{U}_i$ are x-monotone. Thus, the two line segments $p_\alpha q_\alpha$ and $p_\beta q_\beta$ must intersect at some point. By triangle inequality, we have $d(p_\alpha, q_\alpha)+d(p_\beta, q_\beta) > d(p_\alpha, q_\beta)+d(p_\beta+q_\alpha)$, i.e., $d(p_\alpha, q_\beta)+d(p_\beta+q_\alpha) < 2r$. Thus one of the two points p_α and p_β must have distance to P_i less than r. This contradicts the fact that both p_α and p_β are on $\widetilde{U}_i$.

The above lemma suggests that we can construct the upper boundary of $\mathcal{M}(P_i, r)$ by a simple plane sweep algorithm as follows.

Minkowski-Sum Algorithm

1. Compute the candidate edge for each edge or reflex vertex of P_i;
2. Let $\widetilde{U}_i = \{\}$ be the initial upper boundary of $\mathcal{M}(P_i, r)$;
3. Starting from the leftmost endpoint of P_i, for each candidate edge, compute the intersection between the candidate edge and the last edge of $\widetilde{U}_i$;
4. If there is no intersection point, discard the last edge from $\widetilde{U}_i$ and go to step 3;

5. If there is an intersection point p, truncate both edges at p and add the truncated candidate edge at the end of $\widetilde{U}_i$;
6. Proceed to the next candidate edge in left-to-right order, repeat step 3 – 6 until the rightmost endpoint of P_i is reached;
7. Discard the portions of $\widetilde{U}_i$ not lying between L_s and L_t.

Theorem 1. *Given a path $P_i \in \mathcal{P}$ with n_i vertices and edges and a positive constant r, the complexity of $\mathcal{M}(P_i, r)$ is $O(n_i)$ and the Minkowski-Sum Algorithm correctly computes $\mathcal{M}(P_i, r)$ in $O(n_i)$ time.*

Note that the Minkowski sum $\mathcal{M}(P_i, r)$ can be alternatively computed using existing approaches (such as those based on medial axis or Voronoi diagram). However, those algorithms can not be easily parallelized for parametric search. In contrast, our algorithm is much simpler and more friendly to parallelization.

3.2 Double Stabbing Problem

The rough idea of our D2CPP algorithm is to first compute $\mathcal{M}(P_i, r)$ for all $P_i \in \mathcal{P}$ and sort the vertices of the upper and lower boundary curves. Then compute the two leftmost endpoints of P_u and P_l on L_s, denoted by s_u and s_l respectively. At last, starting from s_u and s_l, we sweep a vertical line from left to right and build P_u and P_l if they exist.

Let L be an arbitrary vertical line between L_s and L_t. $L \cap \mathcal{M}(P_i, r)$ forms a connected line segment by Lemma 1. We denote the line segment by an interval $I_i = [y_1^i, y_2^i]$, where y_1^i and y_2^i are the y-coordinate of the two endpoints. Let $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$. To determine if there exists a feasible pair of points on L like s_u and s_l, we need to solve the following *double stabbing problem*.

Definition 4 (Double Stabbing Problem (DSP)). *Given a set of intervals $\mathcal{I}$, find two points p_u and p_l such that there exists a partition $\mathcal{I}_u$ and $\mathcal{I}_l$ (i.e., $\mathcal{I} = \mathcal{I}_u \cup \mathcal{I}_l$) such that $p_u \in \cap_{I \in \mathcal{I}_u} I$ and $p_l \in \cap_{I \in \mathcal{I}_l} I$. In other words, $\cap_{I \in \mathcal{I}_u} I$ and $\cap_{I \in \mathcal{I}_l} I$ are not empty.*

Lemma 5. *Given $\mathcal{P}$ and $r > 0$, if there exist P_u and P_l for D2CPP, then there exists a feasible solution for the double stabbing problem on $\mathcal{I}$ for any L between L_s and L_t.*

Proof. Let p_u and p_l be two intersection points between L and P_u, P_l respectively. Since P_u and P_l are the 2-central paths for $r > 0$, by definition we have $\min(d(p_u, P_i), d(p_l, P_i)) \leq r$ for all $P_i \in \mathcal{P}$. Thus either p_u or p_l is contained by I_i.

We can solve the double stabbing problem in linear time by using the plane sweep paradigm if the endpoints are sorted. The main idea is to scan a vertical line from left to right with the right endpoints of the intervals in $\mathcal{I}$ as event points. Meanwhile, maintain $\cap_{I \in \mathcal{I}_u} I$ and $\cap_{I \in \mathcal{I}_l} I$, add an interval to either $\mathcal{I}_u$ or $\mathcal{I}_l$ with adding to $\mathcal{I}_l$ having a higher priority. Let $I_u = \cap_{I \in \mathcal{I}_u} I$ and $I_l = \cap_{I \in \mathcal{I}_l} I$.

Lemma 6. *The above algorithm correctly solves the double stabbing problem in $O(m)$ time if the endpoints are sorted, where m is the total number of intervals.*

3.3 Solving the D2CPP

After solving the double stabbing problem on L_s, if there is no solution, we immediately know that there exist no pair of paths P_u and P_l for the D2CPP. Otherwise, we sweep a vertical line L from L_s to L_t and maintain I_u and I_l. Let $\mathcal{P}_u \subset \mathcal{P}$ and $\mathcal{P}_l \subset \mathcal{P}$ be the two sets of input curves corresponding to $\mathcal{I}_u$ and $\mathcal{I}_l$ respectively. Whenever I_u or I_l is shrunk to a single point (called *singularity event point*, see Figure 5), we have to update $\mathcal{I}_u$ and $\mathcal{I}_l$. The reason is that if we continue sweeping L without updating $\mathcal{I}_u$ and $\mathcal{I}_l$, one of the two intervals I_u and I_l that contains the singularity event point will disappear. But that does not necessarily mean there is no solution for the double stabbing problem on L. One way to resolve this problem is to solve the double stabbing problem again. However, there are three major drawbacks for this approach: 1) We can not afford it because we potentially have $\Omega(n)$ event points; Thus an $o(\log m)$ update scheme is desired; 2) The Double Stabbing Algorithm possibly generates the same I_u and I_l as they are still feasible solution; 3) The new I_u (or I_l) is likely to be disjoint with the old I_u (or I_l), which leads to discontinuous or non-x-monotone 2-central paths (see Figure 5). The following lemma suggests that we only need to find and remove one curve from $\mathcal{I}_u$ (or $\mathcal{I}_l$) and then add it to $\mathcal{I}_l$ (or $\mathcal{I}_u$) to avoid all the drawbacks. Furthermore, the update only takes $O(1)$ time if the two curves intersecting at the singularity event point are known.

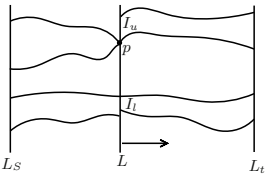


Fig. 5. The singularity event point p in I_u. After an update, the old I_u is a subset of the new I_u and the new I_l is a subset of the old I_l. But I_l can become empty.

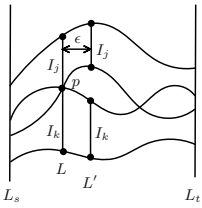


Fig. 6. Illustration of the proof of Lemma 7

Lemma 7. *While sweeping L, if one of the two intervals I_u and I_l becomes singular (i.e., the interval only contains one single point) and $L \neq L_t$, there are two cases:*

1. *I_u is composed of a single point p and I_l is not empty. Let $I_j \in \mathcal{I}_u$ be the interval whose left endpoint is p and $I_k \in \mathcal{I}_u$ be the interval whose right endpoint is p. If $I_k \cap I_l = \emptyset$, then no solution for the D2CPP (see Figure 5);*
2. *I_l is composed of a single point p and I_u is not empty. Let $I_j \in \mathcal{I}_l$ be the interval whose left endpoint is p and $I_k \in \mathcal{I}_l$ be the interval whose right endpoint is p. If $I_j \cap I_u = \emptyset$, then no solution for the D2CPP;*

Proof. We only prove the first case (the second case is similar). If $I_k \cap I_l = \emptyset$, there must exist an interval $I'_k \in \mathcal{I}_l$ such that $I_k \cap I'_k = \emptyset$. Furthermore, the upper boundary curve of $\mathcal{M}(P_k, r)$ intersects the lower boundary curve of $\mathcal{M}(P_j, r)$ at a point on L. Consider a vertical line L' which is infinitesimally close to L on the right. Then on L' we must

have $I_j \cap I_k = \emptyset$ (see Figure 6). Thus there exist three pairwise disjoint intervals I_j, I_k, I'_k on L', i.e., there do not exist two paths P and P' for the D2CPP. □

Remarks: Lemma 7 tells us exactly when we should stop our plane sweep algorithm and assert no solution exists. However, there is one situation that the lemma has difficulty to handle. Consider the interval I_k in case 1. There are basically three subcases: 1) if $I_k \cap I_l = \emptyset$, no solution (quit and return failure); 2) if $I_k \cap I_l$ is a non-singular interval, we continue our plane sweeping algorithm with the updated partition; 3) $I_k \cap I_l$ is singular. The problematic subcase 3 brings us back to the singular event point problem that we are trying to solve. If we use case 2 of Lemma 7 to solve subcase 3, the only thing it would do is to add I_k back to $\mathcal{I}_u$. So one of I_u and I_l must be singular. One way to resolve the issue is to find the interval $I_i \in \mathcal{I}_l$ that shares an endpoint with I_k at the new singular point. Then find the two arcs η_k and η_i of the lower boundary curves of $\mathcal{M}(P_k, r)$ and the upper boundary curve of $\mathcal{M}(P_i, r)$ that passes the singular point. Check if η_k lies above η_i and on the right side of L. If yes, no solution for D2CPP. Otherwise, we continue the plane sweep algorithm. Since this procedure does not increase the asymptotic running time, for simplicity of description, we assume that this subcase will never happen, i.e. $I_k \cap I_l$ is either empty or a non-singular interval.

In order to determine if $I_k \cap I_l$ or $I_j \cap I_u$ is empty in Lemma 7, we have to maintain I_u and I_l (and the partition $\mathcal{I}_u$ and $\mathcal{I}_l$). However, the data structure changes at each intersection point between curves within $\mathcal{P}_u$ or $\mathcal{P}_l$. There are $\omega(n^2)$ intersection points in the worst case (i.e., the complexity of the arrangement of $O(n)$ circular arcs). To resolve this issue, we need to solve the following four problems: 1) Find all singularity event points (this type of event points have to be considered because the existence of P_u and P_l is determined by them); 2) Reduce the total number of event points to about that of the singularity event points; 3) Determine if $I_k \cap I_l$ or $I_j \cap I_u$ is empty at each singularity event point in constant or logarithmic time; 4) Construct P_u and P_l on the fly or efficiently. The last issue is relatively easy to solve by exploring the features of P_u and P_l (Lemma 8).

Let $\mathcal{U} = \{\widetilde{U}_1, \widetilde{U}_2, \ldots, \widetilde{U}_m\}$ and $\mathcal{L} = \{\widetilde{L}_1, \widetilde{L}_2, \ldots, \widetilde{L}_m\}$ be the set of upper and lower boundary curves of all $\mathcal{M}(P_i, r)$ respectively. Let $UL(\mathcal{L})$ be the upper envelope of $\mathcal{L}$ and $LU(\mathcal{U})$ be the lower envelope of $\mathcal{U}$.

Lemma 8. *Given $\mathcal{P}$, $r > 0$ and L, if there exists a solution to D2CPP, $P_u = UL(\mathcal{L})$ and $P_l = LU(\mathcal{U})$ are the 2-central paths for D2CPP.*

Proof. By Lemma 5, for any vertical line L between L_s and L_t, there exists a solution to the stabbing problem. By Lemma 6 (and an observation in its proof), for the intervals generated by the intersection between L and all $\mathcal{M}(P_i, r)$'s, the rightmost left endpoint and the leftmost right endpoint are eligible for being the pair of stabbing points p_u and p_l. p_u and p_l are actually the intersection points between L and $UL(\mathcal{L})$, $LU(\mathcal{U})$ respectively. Since L is arbitrary, the loci of p_u and p_l, when L sweeps from L_s to L_t, respectively form the upper envelope of $\mathcal{L}$ and the lower envelope of $\mathcal{U}$. In other words, the two envelopes are the solution to D2CPP if exist. □

The following lemma reveals the complexities of $UL(\mathcal{L})$ and $LU(\mathcal{U})$.

Lemma 9. *Given n circular arcs in the plane, the upper (or lower) envelope has a complexity of $O(n2^{\alpha(n)})$ and can be computed in $O(n\alpha(n)\log n)$ time, where $\alpha(n)$ is the inverse Ackermann function.*

Since $UL(\mathcal{L})$ and $LU(\mathcal{U})$ have near-linear complexity, we can take advantage of the data structures to identify the singularity event points and reduce the total number of event points. Let $\mathcal{E}$ be the intersection points between $\widetilde{U}_i$ and $UL(\mathcal{L})$, $\widetilde{L}_i$ and $LU(\mathcal{U})$ for all $i = 1, 2, \ldots, m$. The following lemma shows that $\mathcal{E}$ contains all critical singularity event points.

Lemma 10. *Let p be a singularity event point of case 1 or case 2 in Lemma 7. p is contained in $\mathcal{E}$.*

Lemma 11. *$\mathcal{E}$ has a complexity of $O(nm4^{\alpha(n)})$ and can be computed in $O(nm2^{\alpha(n)}\alpha(n)\log n)$ time.*

With Lemma 11, the total number of event points is now reduced to $O(nm4^{\alpha(n)})$ from $\omega(n^2)$. The remaining problem is to examine the emptiness of $I_k \cap I_l$ or $I_j \cap I_u$ at each singularity event point in constant or logarithmic time. The interval I_j or I_k can be stored at each event point while computing $\mathcal{E}$. Thus checking the emptiness of $I_k \cap I_l$ or $I_j \cap I_u$ can be done in $O(1)$ time.

We are now ready to present the D2CPP algorithm. First, compute the Minkowski sum $\mathcal{M}(P_i, r)$ for each $P_i \in \mathcal{P}$, construct the two envelopes $UL(\mathcal{L})$ and $LU(\mathcal{U})$, and compute all event points in $\mathcal{E}$. When sweeping L from L_s to L_t, at each event point $v \in \mathcal{E}$, check if there is a solution to D2CPP by Lemma 7. Below are the main steps of the algorithm.

D2CPP Algorithm

1. For each input curve $P_i \in \mathcal{P}$, compute $\mathcal{M}(P_i, r)$;
2. Compute $UL(\mathcal{L})$ and $LU(\mathcal{U})$;
3. Compute $\mathcal{E}$ and store the arc intersecting $UL(\mathcal{L})$ or $LU(\mathcal{U})$ at each event point;
4. Sweep L from L_s to L_t and for each encountered event point v_i (initially, $i = 1$) do;
   (a) Find the three input curves $P_i, P_j, P_k \in \mathcal{P}$ such that $\widetilde{L}_i \cap L = UL(\mathcal{L}) \cap L$, $\widetilde{U}_j \cap L = LU(\mathcal{U}) \cap L$, and $\widetilde{L}_k \cap L = v_i$ or $\widetilde{U}_k \cap L = v_i$;
   (b) If $\widetilde{L}_k \cap L = v_i$, check if $\widetilde{U}_k \cap L > \widetilde{L}_i \cap L$; If yes, $i = i + 1$; otherwise, stop and return failure;
   (c) If $\widetilde{U}_k \cap L = v_i$, check if $\widetilde{U}_j \cap L > \widetilde{L}_k \cap L$; If yes, $i = i + 1$; otherwise, stop and return failure;

Theorem 2. *The D2CPP Algorithm correctly solves D2CPP in $O(nm2^{\alpha(n)}\alpha(n)\log n)$ time, where n is the total complexity of $\mathcal{P}$ and m is the number of curves in $\mathcal{P}$.*

Proof. Step 1 takes $O(n)$ time by Theorem 1. Step 2 takes $O(n\alpha(n)\log n)$ time by Lemma 9. Step 3 takes $O(nm2^{\alpha(n)}\alpha(n)\log n)$ time by Lemma 11. For the loop (step 4), there are $O(nm4^{\alpha(n)})$ event points. At each event point, steps $4(a) - 4(c)$ take $O(1)$ time each. Thus the total running time is $O(nm2^{\alpha(n)}\alpha(n)\log n)$. Note that the two intersection points $UL(\mathcal{L}) \cap L$ and $LU(\mathcal{U}) \cap L$ can be computed during the sweep process by adding the vertices of $UL(\mathcal{L})$ and $LU(\mathcal{U})$ into $\mathcal{E}$ without increasing the asymptotic running time. Hence we know that the running time of steps $4(b)$ and $4(c)$ is $O(1)$. □

4 Solving the 2CPP

To apply the parametric search technique, an efficient parallel algorithm (in Valiant's comparison model) for D2CPP has to be designed. Most of the steps in the D2CPP Algorithm are routine to parallelize. For example, sorting can be performed in $O(\log n)$ parallel steps by using n processors if the input size is n [1,2]. The lower and upper envelopes can be computed in $O(\log^2 n)$ parallel steps by using n processors [8]. The main steps of our parallel algorithm are sketched below.

Parallel D2CPP Algorithm

1. For each input curve $P_i \in \mathcal{P}$, compute $\mathcal{M}(P_i, r)$ by a parallel algorithm;
2. Compute $UL(\mathcal{L})$ and $LU(\mathcal{U})$ by parallel envelope algorithms;
3. Compute $\mathcal{E}$ by parallel envelope algorithms;
4. Let L_i be the vertical line passing v_i ($v_i \in \mathcal{E}$);
5. In parallel, execute steps 4(a)-4(c) of the D2CPP Algorithm for each event point v_i;

Next, we analyze the parallel running time for each step of the algorithm.

We first consider how to compute the upper boundary curve $\widetilde{U}_i$ of $\mathcal{M}(P_i, r)$ by using $\Theta(n_i)$ processors (the lower boundary curve $\widetilde{L}_i$ can be similarly computed), where n_i is the total number of edges and vertices in P_i. As mentioned in the last section, the Minkowski-Sum Algorithm is friendly to parallelization. The reason is that we can simulate Minkowski-Sum Algorithm by using a balanced binary tree structure $\mathcal{T}_i$ (see Figure 7). For every edge or reflex vertex of $P_i \in \mathcal{P}$, there is a leaf node associated with it. All leaf nodes of $\mathcal{T}_i$ are ordered from left to right consistent with P_i. Clearly, $\mathcal{T}_i$ can be constructed in a bottom-up fashion in $O(n_i)$ time, since there are at most n_i leaf nodes (these tree structures are constructed at the beginning of the whole algorithm). At each node p of $\mathcal{T}$, we store the partial Minkowski sum curve formed by the candidate edges in the subtree rooted at p. If p is a leaf node, the candidate edge is stored. Thus, the main idea of our parallel algorithm is to compute the partial Minkowski sum structures layer-by-layer in parallel, starting from the lowest layer (i.e., the leaf nodes). Then the Minkowski sum curve stored at the root becomes $\widetilde{U}_i$.

Now we consider the problem of constructing the partial Minkowski sum curve of a non-leaf node from the partial Minkowski sums of its children. Let p be a node of height j ($1 \le j \le \lceil n \rceil$), and p_l and p_r be its two children. Then the complexity of the partial Minkowski sum curve at p is at most 2^j. And the Minkowski sum structures at p_l and p_r have a complexity of at most 2^{j-1}. Let $\mathcal{A}_l = \{\beta_{l1}, \beta_{l2}, \ldots, \beta_{l2^{j-1}}\}$ and $\mathcal{A}_r = \{\beta_{r1}, \beta_{r2}, \ldots, \beta_{r2^{j-1}}\}$ be the sets of arcs of the partial Minkowski sum curve at p_l and p_r respectively. Then we have the following lemma.

Lemma 12. *There is exactly one intersection point between $\mathcal{A}_l$ and $\mathcal{A}_r$.*

Proof. First of all, there is at least one intersection point by Lemma 1. Secondly, suppose that there are two intersection points q_1 and q_2. Let $P_{il} \subset P_i$ and $P_{ir} \subset P_i$ be the two continuous portions of P_i whose candidate edges contain $\mathcal{A}_l$ and $\mathcal{A}_r$ respectively. Then $d(q_1, P_{il}) = d(q_2, P_{ir}) = r$. Thus, $d(q_1, P_{il} \cup P_{ir}) = d(q_2, P_{il} \cup P_{ir}) = r$, which means that the merged partial Minkowski sum structure at p must have both q_1 and q_2 on its upper boundary. This is impossible by Lemma 4. □

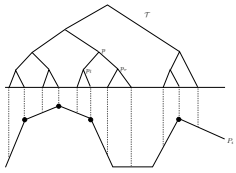


Fig. 7. The binary tree $\mathcal{T}$ constructed on input path P_i

As mentioned in the proof of Theorem 1, if the Minkowski sum curve at p consists of only one arc, but both Minkowski sum structures at its children have a complexity of 2^{j-1}, we have to compute the intersection points for 2^j times. This will be a problem for our parallel Minkowski-Sum algorithm. To fix this problem, instead of computing the partial Minkowski sum curve sequentially, we use 2^{j-1} processors to compute it concurrently. Since the total complexity of all partial Minkowski sum curves in any layer is at most n_i, we only need $n_i/2$ processors for each layer. Furthermore, we use binary search to reduce the dependency. The following algorithm finds the partial Minkowski sum curve at p.

Partial-Minkowski-Sum Algorithm

1. In parallel, for each arc β_{lk_l} in $\mathcal{A}_l$ (where $1 \leq k_l \leq 2^{j-1}$), perform a binary search on $\mathcal{A}_r$, i.e., if there is no intersection between β_{lk_l} and $\beta_{lk_r} \in \mathcal{A}_r$, compute the intersection point between β_{lk_l} and $\beta_{l\frac{k_r+2^{j-1}}{2}}$; otherwise, the intersection point between $\mathcal{A}_l$ and $\mathcal{A}_r$ has already been found;
2. Merge $\mathcal{A}_l$ and $\mathcal{A}_r$ at the intersection point found at step 1;

Lemma 13. *The Partial-Minkowski-Sum Algorithm runs in $O(j)$ parallel steps by using 2^{j-1} processors.*

Lemma 14. *The Minkowski sum $\mathcal{M}(P_i, r)$ can be computed in $O(\log^2 n_i)$ parallel steps by using n_i processors, where n_i is the complexity of P_i.*

Proof. At the layer of height j in $\mathcal{T}$, there are $\frac{n_i}{2^j}$ nodes. By Lemma 13, the partial Minkowski sum curve can be computed in $O(j)$ parallel steps by using 2^{j-1} processors. Thus all partial Minkowski sum curves in that layer can be computed in $O(j)$ steps, using $\frac{n_i}{2^j} * 2^{j-1} = \frac{n_i}{2}$ processors, since each node can be computed independently. Hence, by using n_i processors, the Minkowski sum $\mathcal{M}(P_i, r)$ (i.e., both upper and lower boundary curves) can be computed in $O(1 + 2 + \ldots + \lceil \log n_i \rceil) = O(\log^2 n_i)$ parallel steps. □

From Lemma 14, we immediately know that Minkowski sums for all $P_i \in \mathcal{P}$ can be computed in $O(\log^2 \max_{i=1,2,\ldots,m} n_i) = O(\log^2 n)$ parallel steps using n processors.

Lemma 15. *The envelopes $UL(\mathcal{L})$ and $LU(\mathcal{U})$ in step 2 of the Parallel D2CPP Algorithm can be computed in $O(\log^2 n)$ time by using n processors; Step 3 takes $O(\log^2 n)$ parallel steps by using $nm2^{\alpha(n)}$ processors.*

Proof. The first claim is a direct application of [8]. By Lemma 11, we know that computing the intersection points between $\widetilde{U}_i$ and $UL(\mathcal{L})$ is equivalent to computing

their upper or lower envelope. There are $O(n_i + n2^{\alpha(n)})$ curves in total. Thus the intersection between $\widetilde{U}_i$ and $UL(\mathcal{L})$ can be computed in $O(\log^2 n)$ parallel steps by using $O(n_i + n2^{\alpha(n)})$ processors. Then by using $\sum_{i=1}^{m} O(n_i + n2^{\alpha(n)}) = O(nm2^{\alpha(n)})$ processors, $\mathcal{E}$ can be computed in $O(\log^2 n)$ parallel steps. □

Lemma 16. *Step 5 of the Parallel D2CPP Algorithm takes $O(1)$ time by using $O(nm4^{\alpha(n)})$ processors.*

Theorem 3. *The Parallel D2CPP Algorithm can be computed in $O(\log^2 n)$ parallel steps by using $O(nm4^{\alpha(n)})$ processors.*

Theorem 4. *The 2CPP can be solved in $O(nm\log^4 n2^{\alpha(n)}\alpha(n))$ time, where n is the total complexity of $\mathcal{P}$, m is the number of curves in $\mathcal{P}$, and $\alpha(n)$ is the inverse Ackermann function.*

Acknowledgment. The authors would like to thank Professor Joseph S.B. Mitchell, State University of New York at Stony Brook, for helpful suggestions and discussions.

References

1. Agarwal, P.K., Sharir, M.: Applications of parametric searching in geometric optimization. Journal of Algorithms 17, 292–318 (1994)
2. Agarwal, P.K., Sharir, M.: Efficient algorithms for geometric optimization. ACM Computing Surveys 30, 412–458 (1998)
3. Agarwal, P.K., Sharir, M., Welzl, E.: The discrete 2-center problem. In: Proceedings of the 13th Annual ACM Symposium Computation Geometry, pp. 147–155 (1997)
4. Buchin, K., Buchin, M., van Kreveld, M., Löffler, M., Silveira, R.I., Wenk, C., Wiratma, L.: Median Trajectories. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part I. LNCS, vol. 6346, pp. 463–474. Springer, Heidelberg (2010)
5. Chan, T.M.: More planar two-center algorithms. Computational Geometry Theory Application 13, 189–198 (1997)
6. Drezner, Z.: The planar two-center and two-median problem. Transportation Science 18, 351–361 (1984)
7. Eppstein, D.: Faster construction of planar two-centers. In: Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 131–138 (1997)
8. Goodrich, M.T.: 42 parallel algorithms in geometry
9. Hershberger, J.: A faster algorithm for the two-center decision problem. Information Processing Letters 47, 23–29 (1993)
10. Jaromczyk, J.W., Kowaluk, M.: A geometric proof of the combinatorial bounds for the number of optimal solutions to the 2-center euclidean problem. In: Proceedings of the 7th Canadadian Conference Computational Geometry, pp. 19–24 (1995)
11. Megiddo, N.: Combinatorial optimization with rational objective functions. In: Proceedings of the 10th Annual ACM Symposium on Theory of Computing, pp. 1–12 (1978)
12. Megiddo, N.: Applying parallel computation algorithms in the design of serial algorithms. Journal ACM 30, 852–865 (1983)

13. Sharir, M.: A near-linear algorithm for the planar 2-center problem. In: Discrete Computational Geometry, pp. 147–155 (1996)
14. Singh, V., Mukherjee, L., Peng, J., Xu, J.: Ensemble clustering using semidefinite programming. In: Proceedings of the 21st Advances in Neural Information Processing Systems (2007)
15. Xu, L., Stojkovic, B., Zhu, Y., Song, Q., Wu, X., Sonka, M., Xu, J.: Efficient algorithms for segmenting globally optimal and smooth multi-surfaces. In: Proceedings of the 22nd Biennial International Conference on Information Processing in Medical Imaging, pp. 208–220 (2011)

Making Profit in a Prediction Market

Jen-Hou Chou, Chi-Jen Lu*, and Mu-En Wu

Institute of Information Science, Academia Sinica, Taipei, Taiwan
{jhchou,cjlu,mn}@iis.sinica.edu.tw

Abstract. Suppose one would like to estimate the outcome distribution of an uncertain event in the future. One way to do this is to ask for a collective estimate from many people, and prediction markets can be used to achieve such a task. By selling securities corresponding to the possible outcomes, one can infer traders' collective estimate from the market price if it is updated properly. In this paper, we study prediction markets from the perspectives of both traders and market makers. First, we show that in any prediction market, a trader has a betting strategy which can guarantee a positive expected profit for him when his estimate about the outcome distribution is more accurate than that from the market price. Next, assuming traders playing such a strategy, we propose a market which can update its price to converge quickly to the average estimate of all traders if the average estimate evolves smoothly. Finally, we show that a trader in our market can guarantee a positive expected profit when his estimate is more accurate than the average estimate of all traders if the average estimate again evolves in a smooth way.

1 Introduction

Prediction markets are financial markets which are designed to aggregate knowledge or opinions about uncertain events. For example, to predict if a political candidate will win an election or if a horse will win a race, one can set up a market which offers a security that will pay off one dollar if and only if that outcome actually happens. In general, for a future event of n possible outcomes, one can have a market with n securities, one for each possible outcome. Then a risk neutral trader who believes that the probability of an outcome is p would be willing to buy (or sell) a share of the corresponding security for any price below (or above) p. Thus, the current price of a security may be interpreted as the collective estimate of traders on the likelihood of that outcome. By harnessing the wisdom of the crowd, the estimate can be quite accurate, even more accurate than estimates made by traditional methods. As a result, prediction markets have been widely adopted in diverse areas such as politics, business, and sports [8,2,7,10,3].

* Also with Department of Computer Science, National Chiao-Tung University, Taiwan. Supported in part by the National Science Council under Grant NSC 100-2221-E-001-008-MY3.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 556–567, 2012.

To collect more information from the crowd, a market would like to attract more traders to participate. It helps when there is an automated market maker which can take orders from traders arriving at any time. Furthermore, it also helps if the market maker can be expected to lose money, in which case the market turns into a positive-sum game for traders. The loss of the market maker can be seen as the cost of obtaining valuable information from traders, but still a market designer would like to minimize it. Several mechanisms have been known to guarantee a bounded loss for the market maker, and the most well-known one is perhaps Hanson's market [11,12] based on the logarithmic market scoring rule (LMSR). However, the pricing mechanisms used by most prediction markets may not look intuitive enough for traders to plan their orders easily. For example, consider a cost function based market [4] which calculates the payment of an order according to a cost function C in the following way. Suppose a trader wants to buy q_i shares of security i whose current number of outstanding shares is s_i, for every $i \in \{1, \ldots, n\}$. Then the trader pays $C(s+q) - C(s)$ dollars to the market maker, where $s = (s_1, \ldots, s_n)$, $q = (q_1, \ldots, q_n)$, and $s+q = (s_1+q_1, \ldots, s_n+q_n)$. A simpler pricing mechanism, just as in most product markets, is to have a price vector $(P_1, \ldots, P_n)$, where P_i is the price for buying one share of security i, so that the trader simply pays $\sum_{i=1}^{n} P_i q_i$ dollars to the market maker.

We feel that trading according to such a price vector may be more intuitive and more appealing to traders, which may attract more traders to participate, resulting in more information being collected. Thus, we believe that such markets deserve more study, and we will focus on such markets in this paper. We must stress that we do not claim that such markets are superior; our intention is to explore other alternatives and study their strength and weakness. For such markets, the first question one may ask is whether or not such markets can still have the desirable properties such as the guarantee of bounded loss for the market maker. Chen et al. [5] provided one such example, which is based on the well-known multiplicative update algorithm [13,9] in machine learning and can be seen as a discrete implementation of LMSR. However, in order to achieve a bounded loss, it is required that only a very small fraction of a share can be purchased for each security at each step, which means that a larger order must be divided into several smaller ones. As a result, even an order from one trader may need to be processed in several steps and at different prices. Then it is not clear how fast such a market can aggregate beliefs from traders. Furthermore, it is not clear if the price, which is supposed to reflect the beliefs of traders, will converge even when the beliefs of traders remain fixed over time, if they disagree with each other considerably.

We would like to have a market for which we can prove that the market price does converge quickly to the average belief of traders. In order to achieve that, it may help to understand the behavior of traders. It seems that most previous works on prediction markets take the perspective of a market maker, but we feel that it is also important to study prediction markets from the perspective of a trader. We consider the situation in which a trader has confidence in his own belief, which may evolve over time. That is, although he can infer from

the current market price some information about the current beliefs of other traders, he is not affected too much by that and he has more trust in his own judgement and his own source of information. We do not consider the situation in which a trader can obtain some outside payoff and thus has outside incentive to manipulate the market price [6]. Instead, as in most previous works on prediction markets, we consider the model in which a trader can only obtain the payoff from inside the market. Then for such a trader, what is a good strategy to bet? Under which condition can he make a profit? How much money can he make?

Our Results. Our first result shows that in a prediction market with any pricing mechanism, a trader can make a positive profit if he has a more accurate estimation of the outcome distribution than that from the market price. More precisely, suppose the market runs for T steps, and let $P^{(t)}$ denote the price vector at step t. Then a simple strategy, which we call the uniform strategy, for a trader is to bet the same amount of money at each step t, with a fraction $b_i^{(t)}$ of that amount on security $i \in [n]$, where $b_i^{(t)}$ is the probability he believes at step t that outcome i will happen. Note that this is similar to the famous Kelly strategy, except that we divide the money into T parts and spend only one part at each step. We show that a trader spending m dollars using this strategy can make an expected profit of at least

$$\frac{m}{T} \sum_{t \in [T]} \left(\mathrm{KL}(R \| P^{(t)}) - \mathrm{KL}(R \| b^{(t)}) \right)$$

dollars, where $R = (R_1, \ldots, R_n)$ is the outcome distribution (with R_i being the probability of outcome i), $b^{(t)} = (b_1^{(t)}, \ldots, b_n^{(t)})$, and $\mathrm{KL}(\cdot \| \cdot)$ is the KL-divergence which measures the distance between its two input distributions. From this we see that a trader with a more accurate belief can have a larger expected profit. A somewhat related result can be found in [1], which shows that the Kelly strategy in fact maximizes a trader's expected log utility in a prediction market with a binary event. This may provide a further justification for our uniform strategy.

With this in mind, we next propose a prediction market, which is based on the market of Chen et al. [5] but differs in the following ways. At each step t, the market maker waits for a period of time to collect orders from all possible traders, and sells the requested shares to the traders all at the price $P^{(t)}$. Then the price at step $t+1$ is updated as

$$P_i^{(t+1)} = P_i^{(t)} \exp\left(\eta B_i^{(t)} / P_i^{(t)} \right) / Z^{(t)},$$

for some parameter η, where $B_i^{(t)}$ is the fraction of total money spent at step t on security i and $Z^{(t)}$ is a normalization factor. The normalization factor is used to make $\sum_{i \in [n]} P_i^{(t+1)} = 1$ in order for our market to be arbitrage free. For simplicity, here we only allow traders to buy shares, but we show that even with this simplification, our market can still aggregate beliefs from traders effectively. More precisely, our second result shows that in our market, the market price

$P^{(t)}$ converges quickly to the distribution $B^{(t)} = (B_1^{(t)}, \ldots, B_n^{(t)})$, if $B^{(t)}$ evolves in a smooth way. Note that $B^{(t)}$ reflects what the traders believe on average at step t about how likely each outcome will happen. In fact, it becomes exactly (approximately) the average belief of the traders when all (most) traders play the uniform strategy, which is the case when all (most) traders have confidence in their beliefs. Furthermore, we can also guarantee a bounded loss for the market maker by limiting the number of shares that can be purchased at each step or by charging a small fee for each transaction.

Finally, recall from our first result that a trader can have a positive expected profit if his belief is more accurate compared to the market price. However, a more natural comparison may be to compare a trader's belief to others' beliefs instead of to the market price. With the help of our second result, we show in our third result that in our market, a positive expected profit can still be guaranteed for a trader if his belief is slightly more accurate than the average belief of all traders and the average belief again evolves in a smooth way.

Comparison to Related Works. Although it is commonly believed that one can treat the market price as traders' average belief, the justification seems to be mostly based on empirical analysis, and relatively few theoretical results seem to be known. One such theoretical result is by Wolfers and Zitzewitz [15], who considered a model in which each trader has his own belief and tries to maximize some logarithmic (or some other) utility function. They showed that if the market ever reaches an equilibrium with supply equal to demand, the price coincides exactly (or approximately) with the average belief of traders. However, they did not address the issue of how the market reaches an equilibrium, so they did not have any convergence analysis. Another result is by Ostrovsky [14], who considered markets based on market scoring rules with the so-called strictly proper property. Assuming that there is only one security in the market and each trader initially holds some partial information about the value of the security, he showed that if traders iteratively revise their beliefs and trade according their strategies in a perfect Bayesian equilibrium, then the beliefs of traders will eventually all converge to a consensus which equals the value of the security. Note that the setting considered by Ostrovsky is different from ours. He considered traders who have reliable but partial information and have trust on the information revealed by other traders, and as a result, they can converge to the same belief. On the other hand, we consider the situation in which traders may have incorrect information and thus some traders may have confidence in their own beliefs and may not trust the information revealed by others. Consequently, traders may never reach the same belief, but our market price still converges, to the average belief of traders. Furthermore, we derive bounds on the convergence rate of our market, while Ostrovsky did not provide such a bound. Finally, although it is generally considered true that a trader with a more accurate belief should be able to make a profit, not much work has been done to study the strategies such a trader can use to increase his profit, and according to our limited knowledge, we are not aware of previous work which quantifies the amount of profit such a trader can make. In general, it seems that most previous results on prediction markets focus on qualitative aspects, while we strive to obtain quantitative results.

2 Preliminaries

For $n \in \mathbb{N}$, let $[n]$ denote the set $\{1, \ldots, n\}$. For an n-dimensional vector x and $i \in [n]$, let x_i denote the component in the i'th dimension of x. For two n-dimensional vectors x and y, let $\|x - y\|_1$ denote their L_1-distance, defined by $\|x-y\|_1 = \sum_{i=1}^{n} |x_i - y_i|$. We will see a probability distribution over n elements as a vector in $[0,1]^n$. We will use the KL-divergence to measure the distance between two probability distributions $P, Q \in [0,1]^n$, defined by $\mathrm{KL}(P\|Q) = \sum_{i=1}^{n} P_i \ln \frac{P_i}{Q_i}$.

In this paper, we consider prediction markets with $n \geq 2$ securities corresponding to n possible and mutually exclusive outcomes of some event, which operate for a total of T steps in the following way, for a large enough T. At each step $t \in [T]$, the market maker first announces a price vector $P^{(t)} = (P_1^{(t)}, \ldots, P_n^{(t)})$, and any purchase of one share of security $i \in [n]$ made by a trader at this step costs $P_i^{(t)}$ dollars (or any unit of money). Here we assume that the price vector at each step t satisfies the condition that $\sum_{i \in [n]} P_i^{(t)} = 1$, which can be easily achieved by scaling the unit of money. After the T steps, some outcome $i \in [n]$ happens, and the market maker pays only to shares of security i, one dollar for every share.

We will consider the following variant of the prediction market introduced by Chen et al. [5], which is based on the well-known multiplicative update algorithm [13,9]. We call our market the MU-Market. For simplicity, we will only allow traders to buy but not sell shares in our market.

Definition 1. *(MU-Market) The market is associated with a parameter η in the range $1/\sqrt{T} \leq \eta \leq 1/n^4$, and operates for T steps in the following way, starting with the price $P_i^{(1)} = 1/n$ for every $i \in [n]$. At each step t, any trader is allowed to make a request, and the market maker sells the requested shares to the traders all at the price $P^{(t)}$. Then the market maker updates the next price to*

$$P_i^{(t+1)} = P_i^{(t)} \exp\left(\eta B_i^{(t)} / P_i^{(t)}\right) / Z^{(t)}, \quad \textit{with } Z^{(t)} = \sum_{j=1}^{n} P_j^{(t)} \exp\left(\eta B_j^{(t)} / P_j^{(t)}\right),$$

where $B_i^{(t)}$ is the fraction of total money spent at step t on security i.

As a market maker, he would like to have the guarantee that the money he loses in the worst case is small. One way to achieve this is to have a large enough T, so that the money involved at each step is small. Following the analysis in [5], which is based on a standard regret analysis of the multiplicative update algorithm, we can have the following.

Lemma 1. *Suppose that the total amount of money spent by all traders at each step is at most β and $P_i^{(t)} \geq \gamma$ for any t and i. Then the amount of money the market maker loses in the worst case is at most $(\beta/\eta) \ln n + (\eta/\beta)(\beta/\gamma)^2 T$.*

3 Betting Strategies for Traders

Consider any prediction market, and let $P^{(t)} = (P_1^{(t)}, \ldots, P_n^{(t)})$ be its price vector at step t. Let $R = (R_1, \ldots, R_n)$ denote the outcome distribution, where R_i is the probability that the outcome i will happen. Now, consider a trader who has the belief $b^{(t)} = (b_1^{(t)}, \ldots, b_n^{(t)})$ at step t, where $b_i^{(t)}$ is the probability he believes at step t that the outcome i will happen and $\sum_{i\in[n]} b_i^{(t)} = 1$. We will show that if he has confidence in his belief, then a good betting strategy for him is the following.

– Uniform Strategy: Bet the same amount of money at each step t, with a fraction $b_i^{(t)}$ of that amount on security $i \in [n]$.

Our main result of this section is the following, which provides a bound on the expected profit a trader can make with one dollar using the uniform strategy; for a trader with m dollars, his expected profit is multiplied by m by linearity of expectation.

Theorem 1. *For a trader using the uniform strategy, his expected profit per dollar is at least*

$$\frac{1}{T}\sum_{t\in[T]}\left(\mathrm{KL}(R\|P^{(t)}) - \mathrm{KL}(R\|b^{(t)})\right).$$

Note that $\mathrm{KL}(R\|P^{(t)})$ measures the distance between the outcome distribution R and the market price $P^{(t)}$ at step t, while $\mathrm{KL}(R\|b^{(t)})$ measures the distance between R and the trader's belief $b^{(t)}$ at step t. Thus, $\mathrm{KL}(R\|P^{(t)}) - \mathrm{KL}(R\|b^{(t)}) > 0$ reflects in a sense that the trader's belief is more accurate than the market price at step t about the outcome distribution. Then, according to Theorem 1, if the trader's belief is more accurate than the market price for most steps, he is guaranteed to have a positive expected profit. Now we proceed to prove the theorem.

Proof. (of Theorem 1) Assume that the trader has 1 dollar, and using the uniform strategy he spends exactly $\frac{1}{T}$ dollar at each step. Then at step $t \in [T]$, his expected profit is

$$\sum_{i\in[n]} R_i \frac{b_i^{(t)}/T}{P_i^{(t)}} - \frac{1}{T} = \frac{1}{T}\left(\sum_{i\in[n]} R_i \frac{b_i^{(t)}}{P_i^{(t)}} - 1\right) \geq \frac{1}{T}\left(\ln \sum_{i\in[n]} R_i \frac{b_i^{(t)}}{P_i^{(t)}}\right),$$

using the fact that $1 + \ln x \leq x$ and hence $x - 1 \geq \ln x$, for any x. By Jensen's inequality, the righthand side of the above inequality is at least

$$\frac{1}{T}\left(\sum_{i\in[n]} R_i \ln \frac{b_i^{(t)}}{P_i^{(t)}}\right) = \frac{1}{T}\left(\mathrm{KL}(R\|P^{(t)}) - \mathrm{KL}(R\|b^{(t)})\right).$$

As a result, by summing the bound over t, we have the theorem. □

4 Convergence of Price to Belief in the MU-Market

Recall that the main purpose of a prediction market is to aggregate the beliefs of the traders. In this section, we show that the MU-market defined in Definition 1 can indeed achieve this effectively. Recall that $B_i^{(t)}$ denotes the fraction of total money spent by all traders at step t on security i, so the distribution $B^{(t)} = (B_1^{(t)}, \ldots, B_n^{(t)})$ reflects what the traders believe on average at step t about how likely each outcome will happen. In fact, $B^{(t)}$ becomes exactly (approximately) the weighted average of the beliefs of all the traders at step t, with each trader's belief weighted by the money he spends, when all (most) traders play the uniform strategy, which is the case when all (most) traders have confidence in their beliefs. Our result in this section is the following, which shows that the market price converges to the average belief when it evolves in a smooth way. In fact, the convergence is quick as it takes $O(1/\eta)\ln(1/\eta)$ steps to have the price come within a distance of η from the average belief.

Theorem 2. *Suppose $P_i^{(t)} \geq \eta$ for any t and i, and $\|B^{(t)} - B^{(t-1)}\|_1 \leq \eta^2/\ln(1/\eta)$ for any $t \geq 2$. Then for any $t \geq 1 + (4/\eta)\ln(1/\eta)$,*

$$\mathrm{KL}(B^{(t)} \| P^{(t)}) \leq 5\eta.$$

We will prove the theorem in Subsection 4.1. The key is to show that after one step of update, the market price moves closer to the average belief. For this, we need the following lemma, which we will prove in Subsection 4.2.

Lemma 2. *Suppose $P_i^{(t)} \geq \eta$ for any t and i. Then for any $t \geq 2$,*

$$\mathrm{KL}(B^{(t-1)} \| P^{(t)}) \leq (1 - \eta/2) \cdot \mathrm{KL}(B^{(t-1)} \| P^{(t-1)}) + \eta^2.$$

Note that the lemma shows that the new price $P^{(t)}$ moves towards to the old average belief $B^{(t-1)}$ instead of the new average belief $B^{(t)}$. If the average belief stays the same for all the time steps, then the lemma immediately implies that the market price converges quickly towards the average belief. Interestingly, our Theorem 2 shows that even when the average belief can change over time, the market price can still move close to it, as long as the average belief changes smoothly. The idea is to show that $\mathrm{KL}(B^{(t)} \| P^{(t)})$ is in fact close to $\mathrm{KL}(B^{(t-1)} \| P^{(t)})$.

4.1 Proof of Theorem 2

By definition, we have

$$\begin{aligned}
&\mathrm{KL}(B^{(t)} \| P^{(t)}) - \mathrm{KL}(B^{(t-1)} \| P^{(t)}) \\
&= \sum_{i \in [n]} B_i^{(t)} \ln \frac{B_i^{(t)}}{P_i^{(t)}} - \sum_{i \in [n]} B_i^{(t-1)} \ln \frac{B_i^{(t-1)}}{P_i^{(t)}} \\
&= \sum_{i \in [n]} \left(B_i^{(t)} - B_i^{(t-1)} \right) \ln \frac{B_i^{(t)}}{P_i^{(t)}} + \sum_{i \in [n]} B_i^{(t-1)} \ln \frac{B_i^{(t)}}{B_i^{(t-1)}},
\end{aligned}$$

where in the last line, the second term is $-\mathrm{KL}(B^{(t-1)}\|B^{(t)}) \le 0$ and the first term is at most

$$\sum_{i\in[n]} \left|B_i^{(t)} - B_i^{(t-1)}\right| \cdot \ln(1/\eta) = \|B^{(t)} - B^{(t-1)}\|_1 \cdot \ln(1/\eta) \le \eta^2,$$

since we assume that $P_i^{(t)} \ge \eta$ and $\|B^{(t)} - B^{(t-1)}\|_1 \le \eta^2/\ln(1/\eta)$. From this together with Lemma 2, we have

$$\begin{aligned}\mathrm{KL}(B^{(t)}\|P^{(t)}) &= \mathrm{KL}(B^{(t-1)}\|P^{(t)}) + \left(\mathrm{KL}(B^{(t)}\|P^{(t)}) - \mathrm{KL}(B^{(t-1)}\|P^{(t)})\right)\\ &\le (1-\eta/2)\cdot \mathrm{KL}(B^{(t-1)}\|P^{(t-1)}) + \eta^2 + \eta^2.\end{aligned}$$

Then a simple induction shows that

$$\begin{aligned}\mathrm{KL}(B^{(t)}\|P^{(t)}) &\le (1-\eta/2)^{t-1}\cdot \mathrm{KL}(B^{(1)}\|P^{(1)}) + 2\eta^2 \sum_{i=1}^{t-1}(1-\eta/2)^{i-1}\\ &\le (1-\eta/2)^{t-1}\cdot \mathrm{KL}(B^{(1)}\|P^{(1)}) + 4\eta,\end{aligned}$$

where we have

$$\mathrm{KL}(B^{(1)}\|P^{(1)}) = \sum_{i\in[n]} B_i^{(1)} \ln \frac{B_i^{(1)}}{P_i^{(1)}} \le \sum_{i\in[n]} B_i^{(1)} \ln(1/\eta) = \ln(1/\eta),$$

since $P_i^{(1)} \ge \eta$. As a result, for any $t \ge 1 + (4/\eta)\ln(1/\eta)$, we have

$$\mathrm{KL}(B^{(t)}\|P^{(t)}) \le (1-\eta/2)^{t-1}\ln(1/\eta) + 4\eta \le \eta^2 \ln(1/\eta) + 4\eta \le 5\eta.$$

This proves Theorem 2.

4.2 Proof of Lemma 2

Let $\delta \equiv \mathrm{KL}(B^{(t-1)}\|P^{(t)}) - \mathrm{KL}(B^{(t-1)}\|P^{(t-1)})$, which by definition equals

$$\begin{aligned}\sum_{i\in[n]} B_i^{(t-1)} \ln \frac{P_i^{(t-1)}}{P_i^{(t)}} &= \sum_{i\in[n]} B_i^{(t-1)} \ln \frac{Z^{(t-1)}}{\exp\left(\eta B_i^{(t-1)}/P_i^{(t-1)}\right)}\\ &= \ln Z^{(t-1)} - \eta \sum_{i\in[n]} B_i^{(t-1)}\left(B_i^{(t-1)}/P_i^{(t-1)}\right).\end{aligned}$$

Note that

$$\begin{aligned}Z^{(t-1)} &= \sum_{j\in[n]} P_j^{(t-1)} \cdot \exp\left(\eta B_j^{(t-1)}/P_j^{(t-1)}\right)\\ &\le \sum_{j\in[n]} P_j^{(t-1)} \cdot \left(1 + \eta B_j^{(t-1)}/P_j^{(t-1)} + \left(\eta B_j^{(t-1)}/P_j^{(t-1)}\right)^2\right)\\ &= 1 + \eta + \eta^2 \sum_{j\in[n]} B_j^{(t-1)}\left(B_j^{(t-1)}/P_j^{(t-1)}\right),\end{aligned}$$

where the inequality uses the fact that $\exp(x) \le 1 + x + x^2$ for any $x \in [0,1]$ and $\eta B_j^{(t-1)}/P_j^{(t-1)} \le \eta/\eta = 1$. Then using the fact that $\ln(1+x) \le x$ for any x, we have

$$\ln Z^{(t-1)} \le \eta + \eta^2 \sum_{i\in[n]} B_i^{(t-1)} \left(B_i^{(t-1)}/P_i^{(t-1)}\right),$$

which implies that

$$\delta \le \eta + (\eta^2 - \eta) \sum_{i\in[n]} B_i^{(t-1)} \left(B_i^{(t-1)}/P_i^{(t-1)}\right).$$

Note that $\eta^2 - \eta < 0$ and $B_i^{(t-1)}/P_i^{(t-1)} \ge 1 + \ln\left(B_i^{(t-1)}/P_i^{(t-1)}\right)$ since $x \ge 1 + \ln x$ for any x. As a result, we have

$$\begin{aligned}
\delta &\le \eta + (\eta^2 - \eta) \sum_{i\in[n]} B_i^{(t-1)} \left(1 + \ln\left(B_i^{(t-1)}/P_i^{(t-1)}\right)\right) \\
&= \eta + (\eta^2 - \eta)\left(1 + \mathrm{KL}(B^{(t-1)} \| P^{(t-1)})\right) \\
&= \eta^2 + (\eta^2 - \eta) \cdot \mathrm{KL}(B^{(t-1)} \| P^{(t-1)}) \\
&\le \eta^2 - (\eta/2) \cdot \mathrm{KL}(B^{(t-1)} \| P^{(t-1)}),
\end{aligned}$$

since $\eta \le 1/2$. This proves Lemma 2.

5 Trader's Profit in the MU-Market

Recall from Theorem 1 that in any prediction market, a trader can have a positive expected profit when his belief is more accurate compared to the market price. However, it seems that a more natural comparison is to the beliefs of other traders, instead of to the price. With the help of Theorem 2, we have the following, which shows that in the MU-market, a trader can still have a positive expected profit when his belief is slightly more accurate than the average belief of all traders.

Theorem 3. *Suppose $B_i^{(t)} \ge 4\sqrt[4]{\eta}$ for any t and i, and $\|B^{(t)} - B^{(t-1)}\|_1 \le \eta^2/\ln(1/\eta)$ for any $t \ge 2$. Then for a trader using the uniform strategy with belief $b^{(t)}$ for $t \in [T]$, his expected profit per dollar is at least*

$$\frac{1}{T} \sum_{t\in[T]} \left(\mathrm{KL}(R\|B^{(t)}) - \mathrm{KL}(R\|b^{(t)})\right) - 2\sqrt[4]{\eta}.$$

We will prove Theorem 3 in Subsection 5.1. Note that in order to apply Theorem 2, we need the condition that $P_i^{(t)} \ge \eta$ for any t and i. The following lemma, which we will prove in Subsection 5.2, shows that the condition follows from the assumption of Theorem 3.

Lemma 3. *If $B_i^{(t)} \ge 4\sqrt[4]{\eta}$ for every t and i, then $P_i^{(t)} \ge \sqrt[4]{\eta}$ for every t and i.*

Let us make some remarks about Theorem 3. First, in the theorem we use the assumption that $B^{(t)}$ is close to $B^{(t-1)}$ for every $t \geq 2$, which corresponds to the case that the average belief evolves in a smooth way. The result, nevertheless, can be easily extended to the case when abrupt changes only occur sporadically, in which case we can divide the T steps into a small number of smoothly-evolving intervals and apply the theorem on each interval. Next, with the parameter setting of the theorem, let us see how much money the market maker may lose in the worst case. Consider again that every trader plays the uniform strategy. Then for a large enough T, we can assume that the money spent by traders at each step is $\beta \leq \eta$. Then Lemma 1 can only guarantee that the loss of the market maker is at most $\ln n + (\eta/\beta)(\beta/\sqrt[4]{\eta})^2 T = \ln n + \sqrt{\eta}\beta T$, which unfortunately is unbounded for our choice of $\eta \geq 1/\sqrt{T}$. One way around this is to allow the market maker to charge a transaction fee of $\sqrt{\eta}$ dollar for every dollar of purchase, and the loss of the market maker becomes at most

$$\ln n + \sqrt{\eta}\beta T - \sqrt{\eta}\beta T = \ln n,$$

which is now bounded. With such a transaction fee, the trader's expected profit per dollar is still at least

$$\frac{1}{T}\sum_{t\in[T]} \left(\mathrm{KL}(R\|B^{(t)}) - \mathrm{KL}(R\|b^{(t)})\right) - 2\sqrt[4]{\eta} - \sqrt{\eta}.$$

5.1 Proof of Theorem 3

From Theorem 1, we know that the trader's expected profit per dollar is at least

$$\frac{1}{T}\sum_{t\in[T]} \left(\mathrm{KL}(R\|P^{(t)}) - \mathrm{KL}(R\|b^{(t)})\right).$$

To show that it is at least

$$\frac{1}{T}\sum_{t\in[T]} \left(\mathrm{KL}(R\|B^{(t)}) - \mathrm{KL}(R\|b^{(t)})\right) - 2\sqrt[4]{\eta},$$

it suffices to show that $\mathrm{KL}(R\|B^{(t)})$ and $\mathrm{KL}(R\|P^{(t)})$ are close for most t, or more precisely, to show that

$$\frac{1}{T}\sum_{t\in[T]} \Delta_t \leq 2\sqrt[4]{\eta}, \qquad \text{where } \Delta_t \equiv \mathrm{KL}(R\|B^{(t)}) - \mathrm{KL}(R\|P^{(t)}).$$

With the help of Lemma 3, we can apply Theorem 2 to show that $P^{(t)}$ converges to $B^{(t)}$. More precisely, consider any $t \geq 1 + \ell$, where $\ell \equiv (4/\eta)\ln(1/\eta)$. From Theorem 2, we have

$$\mathrm{KL}(B^{(t)}\|P^{(t)}) \leq 5\eta.$$

Then by Pinsker's inequality, we have $|P_i^{(t)} - B_i^{(t)}| \le \sqrt{5\eta}$ for any i, and hence

$$\frac{P_i^{(t)}}{B_i^{(t)}} \le \frac{B_i^{(t)} + \sqrt{5\eta}}{B_i^{(t)}} = 1 + \frac{\sqrt{5\eta}}{B_i^{(t)}} \le 1 + \frac{\sqrt{5\eta}}{4\sqrt[4]{\eta}} \le 1 + \sqrt[4]{\eta} \le \exp\left(\sqrt[4]{\eta}\right),$$

where the second inequality uses the assumption that $B_i^{(t)} \ge 4\sqrt[4]{\eta}$ and the last inequality uses the fact that $1 + x \le \exp(x)$. Thus for any $t \ge 1 + \ell$, we have

$$\Delta_t = \sum_{i\in[n]} R_i \ln \frac{P_i^{(t)}}{B_i^{(t)}} \le \sum_{i\in[n]} R_i \ln\left(\exp\left(\sqrt[4]{\eta}\right)\right) = \sqrt[4]{\eta}.$$

On the other hand, for any $t \le \ell$, with $B_i^{(t)} \ge 4\sqrt[4]{\eta}$, we have

$$\Delta_t = \sum_{i\in[n]} R_i \ln \frac{P_i^{(t)}}{B_i^{(t)}} \le \sum_{i\in[n]} R_i \ln \frac{1}{4\sqrt[4]{\eta}} \le \frac{1}{4}\ln(1/\eta).$$

By combining the two bounds of Δ_t for different ranges of t, we have

$$\frac{1}{T}\sum_{t\in[T]} \Delta_t \le \frac{T-\ell}{T} \cdot \sqrt[4]{\eta} + \frac{\ell}{T} \cdot \ln(1/\eta) \le \sqrt[4]{\eta} + \frac{1}{\eta T}\ln^2(1/\eta) \le 2\sqrt[4]{\eta},$$

since $\eta \ge 1/\sqrt{T}$ and T is large enough. This proves Theorem 3.

5.2 Proof of Lemma 3

We show this by induction on t. It holds for $t = 1$ since $P_i^{(1)} = 1/n \ge \sqrt[4]{\eta}$ for any $i \in [n]$. Now suppose it holds for $t-1$ and we next show that it holds for t. Consider any $i \in [n]$. Recall that

$$P_i^{(t)} = P_i^{(t-1)} \cdot \exp\left(\eta B_i^{(t-1)}/P_i^{(t-1)}\right)/Z^{(t-1)},$$

where

$$Z^{(t-1)} = \sum_{j\in[n]} P_j^{(t-1)} \cdot \exp\left(\eta B_j^{(t-1)}/P_j^{(t-1)}\right).$$

Since $\exp(x) \le 1 + 2x$ for $x \in [0,1]$ and $\eta B_j^{(t-1)}/P_j^{(t-1)} \le \eta/\sqrt[4]{\eta} \le 1$ for any j by inductive hypothesis, we have

$$Z^{(t-1)} \le \sum_{j\in[n]} P_j^{(t-1)}\left(1 + 2\left(\eta B_j^{(t-1)}/P_j^{(t-1)}\right)\right) = 1 + 2\eta.$$

Moreover, since $\exp(x) \ge 1 + x$ for any x, we can also have

$$\exp\left(\eta B_i^{(t-1)}/P_i^{(t-1)}\right) \ge 1 + \left(\eta B_i^{(t-1)}/P_i^{(t-1)}\right).$$

As a result, we have

$$P_i^{(t)} \ge P_i^{(t-1)} \cdot \left(1 + \left(\eta B_i^{(t-1)}/P_i^{(t-1)}\right)\right)/(1+2\eta).$$

Therefore, if $P_i^{(t-1)} \le B_i^{(t-1)}/2$, we have $P_i^{(t)} \ge P_i^{(t-1)} \cdot 1 \ge \sqrt[4]{\eta}$; otherwise, we still have $P_i^{(t)} \ge P_i^{(t-1)} \cdot (1/2) \ge B_i^{(t-1)}/4 \ge \sqrt[4]{\eta}$. This proves Lemma 3.

References

1. Beygelzimer, A., Langford, J., Pennock, D.: Learning performance of prediction markets with Kelly bettors, CoRR abs/1201.6655 (2012)
2. Berg, J., Forsythe, R., Nelson, F., Rietz, T.: Results from a dozen years of election futures markets research. In: Handbook of Experimental Economic Results
3. Chen, K., Plott, C.: Information aggregation mechanisms: concept, design and implementation for a sales forecasting problem. Working paper No. 1131, California Institute of Technology, Division of the Humanities and Social Sciences (2002)
4. Chen, Y., Pennock, D.M.: A utility framework for bounded-loss market makers. In: Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI), pp. 49–56 (2007)
5. Chen, Y., Fortnow, L., Lambert, N., Pennock, D.M., Wortman, J.: Complexity of combinatorial market makers. In: Proceedings of the 9th ACM Conference on Electronic Commerce (EC), pp. 190–199 (2008)
6. Chen, Y., Gao, X.A., Goldstein, R., Kash, I.A.: Market manipulation with outside incentives. In: AAAI 2011: Proceedings of the 25th Conference on Artificial Intelligence (2011)
7. Debnath, S., Pennock, D., Giles, C., Lawrence, S.: Information incorporation in online in-game sports betting markets. In: Proceedings of the Fourth Annual ACM Conference on Electronic Commerce (EC), pp. 258–259 (2003)
8. Forsythe, R., Rietz, T., Ross, T.: Wishes, expectations, and actions: a survey on price formation in election stock markets. Journal of Economic Behavior and Organization 39, 83–110 (1999)
9. Freund, Y., Schapire, R.: A decision theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences 55(1), 119–139 (1997)
10. Goel, S., Pennock, D., Reeves, D.M., Yu, C.: Yoopick: A combinatorial sports prediction market. In: Proceedings of the 23rd Conference on Artificial Intelligence (AAAI), pp. 1880–1881 (2008)
11. Hanson, R.D.: Combinatorial information market design. Information Systems Frontiers 5(1), 107–119 (2003)
12. Hanson, R.D.: Logarithmic market scoring rules for modular combinatorial information aggregation. Journal of Prediction Markets 1(1), 1–15 (2007)
13. Littlestone, N., Warmuth, M.: The weighted majority algorithm. Information and Computation 108(2), 212–261 (1994)
14. Ostrovsky, M.: Information aggregation in dynamic markets with strategic traders. In: Proceedings of the 10th ACM Conference on Electronic Commerce (EC), pp. 253–254 (2009)
15. Wolfers, J., Zitzewitz, E.: Interpreting Prediction Market Prices as Probabilities. NBER Working Paper No. 12200

Computing Shapley Value in Supermodular Coalitional Games*

David Liben-Nowell[1], Alexa Sharp[2], Tom Wexler[2], and Kevin Woods[3]

[1] Department of Computer Science, Carleton College
[2] Department of Computer Science, Oberlin College
[3] Department of Mathematics, Oberlin College
dlibenno@carleton.edu, {alexa.sharp,tom.wexler,kevin.woods}@oberlin.edu

Abstract. Coalitional games allow subsets (coalitions) of players to cooperate to receive a collective payoff. This payoff is then distributed "fairly" among the members of that coalition according to some division scheme. Various solution concepts have been proposed as reasonable schemes for generating fair allocations. The *Shapley value* is one classic solution concept: player i's share is precisely equal to i's expected marginal contribution if the players join the coalition one at a time, in a uniformly random order. In this paper, we consider the class of supermodular games (sometimes called convex games), and give a fully polynomial-time randomized approximation scheme (FPRAS) to compute the Shapley value to within a $(1 \pm \varepsilon)$ factor in monotone supermodular games. We show that this result is tight in several senses: no deterministic algorithm can approximate Shapley value as well, no randomized algorithm can do better, and both monotonicity and supermodularity are required for the existence of an efficient $(1 \pm \varepsilon)$-approximation algorithm. We also argue that, relative to supermodularity, monotonicity is a mild assumption, and we discuss how to transform supermodular games to be monotonic.

1 Introduction

Game theory is broadly defined as the study of self-interested players. These players may be restricted to make independent decisions, leading to *competitive* games; alternatively, players may be allowed to cooperate in order to achieve their goals, leading to *coalitional*, or *cooperative*, games. Although the recent increased attention on algorithmic game theory from theoretical computer scientists largely focuses on competitive games, both models provide rich computational and theoretical challenges, as demonstrated by a long history of research across the fields of economics, politics, and biology.

In any game, we are interested in the outcomes that might be achieved as a result of players' self-interested behavior. For competitive games, the standard solution concept is the Nash equilibrium; for coalitional games, a number of

* This work was supported in part by NSF grant CCF-0728779 and by grants from Oberlin College and Carleton College. Thanks to Josh Davis for helpful discussions.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 568–579, 2012.

reasonable solution concepts exist. One class of solution concepts focuses on "fair" divisions of wealth. Assume that all players work together to form the *grand coalition* of all players. How can the total utility generated be divided so that each player's portion is proportional to his or her influence or power? The *Shapley value* [35] is one such fair allocation scheme. Other well-known solution concepts include the Banzhaf power index [4], the core [17,37], and the kernel [9]. The Banzhaf index is similar in spirit to the Shapley value, while the other solution concepts aim to describe outcomes that are "stable" (robust to deviating subcoalitions) rather than fair.

Generally, under these solution concepts for coalitional games, solutions are hard to compute. In some restricted domains, however, it is possible to find exact or approximate solutions efficiently. In this paper, we consider *supermodular* games (also known as *convex* games in the economics literature [36]), a class of coalitional games in which incentives for joining a coalition increase as the coalition grows. This paper is devoted to the efficient computation of Shapley value in these games.

1.1 Related Work: Computing the Shapley Value

Shapley first introduced his eventually eponymic solution concept in 1953 [35]. The Shapley value, Banzhaf index, core, kernel, and related measures have been studied extensively; they are described and surveyed in, e.g., [2,5,10,11,16,24].

Finding the allocations described by these solution concepts is computationally intractable in general. Hence, much of the research in this area focuses on a variety of restricted domains in which one can hope to find either an exact or approximate solution efficiently.

One such domain is *weighted majority games*, in which a coalition receives a payoff of 1 if its members constitute a majority of all players' weights, and 0 otherwise. Mann–Shapley [25] motivate this class of coalitional games and propose a Monte Carlo sampling algorithm to approximate the Shapley value. They apply their algorithm to U.S. electoral college data, but do not provide formal analysis. Deng–Papadimitriou [12] and Matsui–Matsui [29] show that it is NP-hard to determine whether a given player has nonzero Shapley value, and #P-hard to calculate it exactly. Matsui–Matsui [28] make the exact computation with a pseudo-polynomial dynamic programming algorithm, and find that Mann–Shapley's algorithm has error that goes to zero like $1/\sqrt{\#\text{samples}}$. Fatima et al. [13,14] give exact polynomial-time algorithms to compute the Shapley value for specific subclasses of weighted majority games, characterize when this approach becomes intractable, and consider linear-time approximations.

Another domain to receive attention is that of *simple* coalitional games, a generalization of weighted majority games in which every coalition has a payoff of 0 or 1. Bachrach et al. [3] apply the Mann–Shapley algorithm in this more general setting, and give an oracle-based sampling algorithm to approximate Shapley value in polynomial time, also with $1/\sqrt{\#\text{samples}}$ additive error. They also show that no approximation algorithm can do much better by giving lower bounds for both deterministic and randomized algorithms for these calculations.

A third domain with interesting results is that of *submodular* games [33,34], in which incentives for joining a coalition *decrease* as the coalition grows. Approximability results are known for the *least-core value* but not for the Shapley value.

1.2 Related Work: Supermodular Games

Supermodular coalitional games are another restricted class of coalitional games, and the class upon which we focus in this paper. These games, first introduced by Shapley [36] as *convex* games, capture the intuitive notion that incentives for joining a coalition increase as the coalition grows. In addition to many natural applications that result in supermodular coalitional games, these games also have pleasing theoretical properties: the core is nonempty [36], the Shapley value is in the core and is the center of mass of the core's vertices [36], the kernel is a single point corresponding to the *nucleolus* [27], and the *stable set* and *bargaining set* for the grand coalition coincide with the core [26]. Many specific and natural examples of supermodular games are studied in the literature; a sampling of these games are described in the remainder of this section.

The *multicast tree game* [1,15,18,21] is used to model distribution networks such as waterways and telecommunication networks. Players receive a payoff for being connected to the source but must cover the cost of building the underlying (and fixed) tree. As more players join the network, the cost to a previously connected player cannot increase, and thus the game is supermodular.

The *edge synergy game* [12] takes place on an undirected graph, with the nodes as players. Each edge of a graph has an associated nonnegative benefit, and the value of a coalition is the sum of the benefits in its induced subgraph. This game possesses increasing returns of scale, as a player who joins a coalition adds value for each neighbor already in that coalition.

The *bankruptcy game*, first studied by O'Neill [31], is another natural supermodular game. In this setting, there is an estate to which each player has some claim, but not all claims can be satisfied. The value of a coalition S is what remains of the estate after satisfying the claims of the players *not* in S. Curiel et al. [8] show that bankruptcy games are supermodular.

These examples are meant to sample just some of the supermodular games in the literature; other specific examples have been recently studied algorithmically by, e.g., Jain–Vazirani [21] and Ieong–Shoham [19].

1.3 Our Results

In many of the games described in Section 1.2, computing the exact Shapley value is fairly easy; however, to the best of our knowledge, efficient computation of Shapley value for general supermodular games has not been addressed.

Our main result (Theorem 2) resolves the open question regarding the computation of Shapley value in supermodular games: we give an efficient randomized algorithm to approximate arbitrarily well the Shapley value of any monotone supermodular coalitional game. Specifically, we show that the Mann–Shapley

Monte Carlo sampling algorithm [25] is a fully polynomial-time randomized approximation scheme (FPRAS) for the Shapley value of a monotone supermodular game, assuming access to an oracle that returns the value of any given coalition. We also show that this result is the best possible in the following senses: no fully polynomial deterministic algorithm can approximate Shapley value as well; no randomized algorithm can do better; and *both* monotonicity and supermodularity are required to approximate Shapley value within *any* multiplicative factor. Note that our definition of polynomial running time is slightly atypical, as we do not have the entire game as input, but rather have only oracle access to it. Also note that our negative results hold regardless of whether $\mathsf{P} \neq \mathsf{NP}$.

The remainder of this paper is structured as follows. Section 2 formally defines coalitional games and the Shapley value. Section 3 presents and analyzes the FPRAS for Shapley value in monotone supermodular games, and Section 4 shows that this FPRAS is essentially the best result that one can hope to achieve.

Finally, Section 5 gives reason to believe that the additional assumption of monotonicity in a supermodular game is reasonably natural. Specifically, we examine two ways to convert any supermodular game v into a monotone supermodular game. The *zero-normalization transform*, v_Z, translates the utility function so that $v_Z(\{i\}) = 0$ for each player i. This shift does not change the strategies of the players, and, hence for any supermodular v we can approximate player i's gain over $v(\{i\})$ under the Shapley allocation. The *opt-out transform*, v_O, allows any players who contribute negative value to a coalition not to participate, thereby receiving zero utility. While v_O is a more substantive transform of v, in many settings it is a natural operation. For both transforms, we prove that if the original game is supermodular, then the transformed game is both supermodular and monotone, and furthermore we can compute the value of any coalition efficiently. Thus our results from Sections 3 and 4 apply to both v_O and v_Z.

2 Model and Definitions

A coalitional game v is defined by a set N of n players, and a function $v : \mathscr{P}(N) \to \mathbb{R}$, where $v(S)$ denotes the value generated by a coalition $S \subseteq N$. Without loss of generality, we assume throughout that $v(\emptyset) = 0$. We further assume that a game is represented by an oracle that, given $S \subseteq N$, returns $v(S)$.

A game is *monotone* if, for all $S \subseteq T \subseteq N$, we have $v(S) \leq v(T)$.

A game is *supermodular* if $v(S \cup T) + v(S \cap T) \geq v(S) + v(T)$ for any two sets $S, T \subseteq N$, or equivalently, if $v(S \cup \{i\}) - v(S) \leq v(T \cup \{i\}) - v(T)$ whenever $S \subseteq T$ and $i \notin T$. That is, the marginal value that a player i adds to a coalition S is no greater than the marginal value i adds to a coalition $T \supseteq S$.

A weaker notion than supermodularity is *superadditivity*: a game is superadditive if, for all *disjoint* sets $S, T \subseteq N$, we have $v(S \cup T) \geq v(S) + v(T)$. In a superadditive game, cooperation is always beneficial, and the *grand coalition* of all players will form. We will assume, at minimum, that a game is superadditive. The question, then, is how the players will divvy up $v(N)$, the value of the grand coalition. An allocation $x = \langle x_1, \ldots, x_n \rangle$ should certainly be *(economically) efficient* (no money is left on the table) and *individually rational* (no player makes

less than he could make by acting alone): formally, we require $\sum_i x_i = v(N)$ and $x_i \geq v(\{i\})$ for all i. Solution concepts for coalitional games further refine these requirements.

The *Shapley value* [35] is the most well-known solution concept based on the notion that the allocation for player i should be proportional to i's "power" in the game—that is, how much value i creates. Formally, it is defined as follows:

Definition 1. *The* Shapley value *is the allocation where*

$$x_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!\,(n-|S|-1)!}{n!} \left[v(S \cup \{i\}) - v(S)\right].$$

Given a permutation π ordering the arrival of the players, the marginal contribution of π_j is $v(\{\pi_1, \ldots, \pi_j\}) - v(\{\pi_1, \ldots, \pi_{j-1}\})$; the Shapley value for player i is the average, over all permutations, of the marginal contribution of i to the set of players who arrive before i.

3 Algorithms to Approximate the Shapley Value

We say that a vector $\overline{s}$ is a (multiplicative) *ε-approximation to the vector s* if $|\overline{s}_i - s_i| \leq \varepsilon s_i$ for all indices i. In this section, we show that there is an oracle-based fully polynomial-time randomized approximation scheme (FPRAS) for the Shapley value, as long as the game is *both* supermodular and monotone. That is, we give a $\text{poly}(n, 1/\varepsilon)$-time randomized algorithm to ε-approximate the Shapley value in monotone supermodular games with high probability. In Section 4, we show that this result is essentially the best possible. Our approach is based on sampling: we compute the marginal value of each player in a random permutation, and average over many permutations. (This type of sampling is also used by Mann–Shapley [25] and Bachrach et al. [3]; however, Mann–Shapley provides no theoretical analysis, and Bachrach et al.'s analysis does not apply to non-simple coalitional games. The comparative difficulty here is that our game may have payoffs besides 0 and 1, and so there is not an immediate bound on the variance of the sampling.)

Algorithm SV-Sample [25]. Given an n-player game v and $\varepsilon > 0$:
Generate $m = 4n(n-1)/\varepsilon^2$ random permutations of the players $\{1, \ldots, n\}$. For each player, define $\overline{s}_i$ to be the average marginal contribution of player i over these m permutations. Return the vector $\overline{s}$.

Theorem 2. *There is an FPRAS for the Shapley value of any game v that is both supermodular and monotone. In particular, Algorithm* SV-Sample(v, ε) *produces an ε-approximation to the Shapley value with probability at least* 3/4.

Proof. Let X_i denote the marginal contribution of player i in a random permutation. Because v is both supermodular and monotone, we have $X_i \geq 0$. By definition, the Shapley value for player i is $s_i := E[X_i]$. By supermodularity, the maximum possible value achieved by X_i occurs when i is the last player in the permutation, which happens in a $1/n$ fraction of permutations. Thus X_i achieves its maximum value with probability at least $1/n$, and so X_i is at most $n \cdot s_i$.

To upper bound the variance of X_i, we first define a new random variable Y_i that is $n \cdot s_i$ with probability $1/n$ and 0 otherwise. Note that the variances of X_i and Y_i satisfy $\sigma^2_{X_i} \leq \sigma^2_{Y_i}$, because X_i and Y_i have the same expectation, and we have simply pushed individual values to the extremes as much as possible in Y_i. Therefore we have

$$\sigma^2_{X_i} \leq \sigma^2_{Y_i} = E[Y_i^2] - E[Y_i]^2 = \frac{1}{n}(n \cdot s_i)^2 - s_i^2 = (n-1) \cdot s_i^2.$$

Compute the sample mean $\overline{s}_i = \frac{1}{m} \sum_{j=1}^{m} X_i^{(j)}$, where each $X_i^{(j)}$ is an independent trial as above. Now

$$\sigma^2_{\overline{s}_i} = \sigma^2_{X_i}/m \leq (n-1)s_i^2/m \qquad \text{and} \qquad E[\overline{s}_i] = s_i.$$

Using Chebyshev's inequality [30], we have

$$\Pr[|\overline{s}_i - s_i| \geq \varepsilon \cdot s_i] \leq \frac{\sigma^2_{\overline{s}_i}}{s_i^2 \varepsilon^2} \leq \frac{(n-1)s_i^2/m}{s_i^2 \varepsilon^2} = \frac{n-1}{m \cdot \varepsilon^2}.$$

Taking a union bound, we have that

$$\Pr[\exists i : |\overline{s}_i - s_i| \geq \varepsilon \cdot s_i] \leq \frac{n(n-1)}{m\varepsilon^2}.$$

Because we defined $m = 4n(n-1)/\varepsilon^2$, this upper bound on the failure probability is $1/4$. Thus $\overline{s}$ is an ε-approximation to s with probability at least $3/4$. □

The choice of $3/4$ as the success probability in Theorem 2 is arbitrary. By re-running Algorithm SV-Sample $\Theta(\log(1/\delta))$ times, taking the coordinate-wise median value for each player, and rescaling the resulting vector to preserve economic efficiency (i.e., ensuring that $\sum_{i=1}^{n} \overline{s}_i$ and $v(N)$ are equal), we get an ε-approximation to s with failure probability at most δ.

4 Lower Bounds for Approximating Shapley Value

In this section, we prove that the randomized approximation scheme from Section 3 is the best possible, in several senses: no deterministic algorithm can do as well, a randomized algorithm can do no better, and *both* the monotonicity and supermodularity conditions are required to achieve this approximation.

We will use the following class of n-player supermodular games for several of the lower bounds, for an even number n. Let $\mathcal{C}$ be a collection of subsets of $\{1, \ldots, n\}$, each of cardinality $n/2$. Define the game $v_{\mathcal{C}}$ as follows:

$$v_{\mathcal{C}}(A) = \begin{cases} 2|A| - n & \text{if } |A| > n/2 \\ 1 & \text{if } |A| = n/2 \text{ and } A \in \mathcal{C} \\ 0 & \text{otherwise.} \end{cases}$$

In other words, no coalition of fewer than half the players can receive any value, only some coalitions of size exactly $n/2$ (those in $\mathcal{C}$) receive some value, and

larger coalitions receive linearly increasing value as the size grows (regardless of membership).

By examining each player's marginal contributions, we can see that $v_{\mathcal{C}}$ is supermodular. The Shapley value of each player in the game $v_\emptyset$ is 1, and the games $v_\emptyset$ and $v_{\mathcal{C}}$ differ only on the sets in $\mathcal{C}$. Thus, writing $\mathcal{A}_i = \{A : |A| = n/2 \text{ and } i \in A\}$ and $\overline{\mathcal{A}}_i = \{A : |A| = n/2 \text{ and } i \notin A\}$, we have

$$\text{the Shapley value for player } i \text{ in } v_{\mathcal{C}} \quad = \quad 1 + \frac{|\mathcal{C} \cap \mathcal{A}_i|}{\binom{n-1}{n/2-1} \cdot n} - \frac{|\mathcal{C} \cap \overline{\mathcal{A}}_i|}{\binom{n-1}{n/2} \cdot n}.$$

A fully polynomial-time deterministic approximation scheme (FPTAS) is the deterministic analog to an FPRAS. Our next result says that the randomization used in Theorem 2 is in fact necessary: there is no FPTAS for Shapley value in monotone supermodular games. (That is, there is no poly$(n, 1/\varepsilon)$-time deterministic algorithm to ε-approximate Shapley value in n-player monotone supermodular games.)

Theorem 3. *There is no FPTAS for the Shapley value, even for games that are both supermodular and monotone.*

Proof. Assume that such an algorithm exists. For any n, we take $\varepsilon = 1/2n$. The algorithm must ε-approximate a player i's Shapley value with only poly(n) oracle calls. Define $\mathcal{A}_i = \{A : |A| = n/2 \text{ and } i \in A\}$. Assume that the oracle responds to all queries as if the game is $v_\emptyset$, and let $\mathcal{Q}_i \subseteq \mathcal{A}_i$ be the collection of sets among $\mathcal{A}_i$ queried by the algorithm. Then these queries cannot distinguish between the games $v_\emptyset$ and $v_{\mathcal{A}_i \setminus \mathcal{Q}_i}$. The Shapley values for player i in these two games are

$$1 \qquad \text{and} \qquad 1 + \frac{1}{n} - \frac{|\mathcal{Q}_i|}{\binom{n-1}{n/2-1} n},$$

respectively. Because $|\mathcal{Q}_i|$ is polynomial in n, and $\binom{n-1}{n/2-1}$ grows faster than any polynomial, we may take n large enough so that $|\mathcal{Q}_i| / \binom{n-1}{n/2-1} < 1/2$. In this case, the purported algorithm cannot distinguish between two games whose Shapley values differ by a multiplicative factor of $\varepsilon = 1/2n$, as was required. Therefore, such an algorithm cannot exist. □

The randomized sampling algorithm from Section 3 requires poly$(n, 1/\varepsilon)$ time to ε-approximate Shapley values. In other words, for any polynomial $q(m)$, we can get a $1/q(m)$ approximation in poly(n, m) time. One might hope for a better algorithm—for example, a $1/2^m$ approximation in poly(n, m) steps. We now apply Yao's Minimax Principle to show that no such algorithm exists. (A similar argument is used by Bachrach et al. [3].)

Theorem 4. *Suppose $\varepsilon(m)$ is a function that converges to zero faster than $1/q(m)$ for any polynomial $q(m)$. Then no randomized algorithm can $\varepsilon(m)$-approximate the Shapley value of an n-player monotone supermodular game in* poly(n, m) *time.*

Proof. Yao's Minimax Principle [39] states that it suffices to prove that no deterministic polynomial-time algorithm can give an $\varepsilon(m)$ approximation on any particular probability distribution of games. We define the distribution as follows. Let i be a particular player, let $\mathcal{A}_i = \{A : |A| = n/2 \text{ and } i \in A\}$, and let $k = \varepsilon(n)\binom{n-1}{n/2-1}n$. (Assume that n is large enough that $\varepsilon(n) < 1/n$.) With probability $1/2$, we choose the game $v_\emptyset$, and with probability $1/2$ we choose uniformly at random a subcollection $\mathcal{Q}_i$ of $\mathcal{A}_i$ of exactly k sets. The respective Shapley values for player i in $v_\emptyset$ and $v_{\mathcal{Q}_i}$ are

$$1 \qquad \text{and} \qquad 1 + \frac{k}{\binom{n-1}{n/2-1}n} = 1 + \varepsilon(n).$$

Thus the algorithm must be able to distinguish $v_\emptyset$ and $v_{\mathcal{Q}_i}$ with probability $3/4$. But the only way to differentiate is to query a set that is in $\mathcal{Q}_i$. The probability of querying a set in $\mathcal{Q}_i$ in one query is $k/\binom{n-1}{n/2-1} = n\varepsilon(n)$, and the probability of querying a set in $\mathcal{Q}_i$ in $p(n)$ queries is at most $p(n) \cdot n\varepsilon(n)$. As $1/\varepsilon(n)$ eventually exceeds any polynomial and the number of queries $p(n)$ is polynomial in n, this probability approaches zero, contradicting the requirement that the algorithm distinguish $v_\emptyset$ and $v_{\mathcal{Q}_i}$ with probability $3/4$. □

Finally, we prove that both the supermodularity and monotonicity conditions are required, in a very strong sense: with only one of the two properties, no polynomial-time algorithm can distinguish a zero from a nonzero Shapley value (either deterministically or probabilistically). Therefore, there is no ε-approximation algorithm that runs in polynomial time, for any $\varepsilon > 0$.

Theorem 5. *No polynomial-time (deterministic or randomized) algorithm can determine whether the Shapley value of a supermodular game is nonzero.*

Proof. First we prove the deterministic version. For a given collection $\mathcal{C}$ of subsets of size $n/2$, define a new game $v'_\mathcal{C}$ by $v'_\mathcal{C}(A) = v_\mathcal{C}(A) - |A|$ for all subsets A. Each player's Shapley value is decreased by 1 under this transformation. Such a game is still supermodular, but $v'_\mathcal{C}$ is not monotone, because $v'_\mathcal{C}(\emptyset) = 0 > -1 = v'_\mathcal{C}(\{i\})$ for any player i. Suppose that the oracle answers queries as if the game is $v'_\emptyset$, in which every player has Shapley value 0. Because the algorithm can make only poly(n) oracle calls and there are $\binom{n}{n/2}$ sets of size $n/2$, one of these sets, A, has not been queried (for sufficiently large n). Then in the game $v'_{\{A\}}$ each player has nonzero Shapley value—the players in A have positive Shapley value, those not in A have negative Shapley value—but the algorithm cannot distinguish $v'_{\{A\}}$ from $v'_\emptyset$ based on its oracle calls.

For the randomized version, we use Yao's Minimax Principle, as in Theorem 4. For the random distribution, with probability $1/2$ we take $v'_\emptyset$, and otherwise we take $v'_{\{A\}}$ for a set A of size $n/2$ chosen uniformly at random. □

Theorem 6. *No polynomial-time (deterministic or randomized) algorithm can determine whether the Shapley value of a monotone game is nonzero, even assuming superadditivity.*

Proof. Fix a player i. Suppose the oracle answers a query about $v(A)$ as if we have the following monotone, superadditive game: if $|A| > n/2$, or if $|A| = n/2$ and $i \notin A$, then $v(A) = 1$; otherwise $v(A) = 0$. Player i has Shapley value 0 in this game. A polynomial number of oracle calls cannot differentiate this game from a monotone, superadditive game v' where one set B, of size $n/2$ and with $i \notin B$, is changed from value 1 to value 0 (which gives player i nonzero Shapley value in v'). The randomized version follows as before. □

Note that, while Theorem 3 shows that there is no FPTAS for computing the Shapley value of a supermodular game, it is an open question whether there is a PTAS—i.e., a deterministic ε-approximation algorithm that runs in time poly(n) for any *fixed* $\varepsilon > 0$.

5 Ensuring Monotonicity in Supermodular Games

Section 4 shows that computing the Shapley value of a supermodular game, even approximately, is difficult when the game is not monotone. There are, however, two natural transforms that add monotonicity to any supermodular game, while maintaining supermodularity.

Zero-Normalization Transform. Given a coalitional game v, define a new game v_Z where

$$v_Z(A) = v(A) - \sum_{i \in A} v(\{i\}).$$

The zero-normalization transform offsets the value of any coalition A by the value that each member of A would gather alone, be that amount positive or negative. Thus the value of any singleton coalition is normalized to 0. This change does not affect the strategic character of the game, as only the relative utility of a player's options are important. Note that player i's Shapley values in v and in v_Z differ by exactly $v(\{i\})$; that is, the value in v_Z is the share of the *gains* due to cooperation that are allocated to a player. If v is supermodular, then we will show shortly that v_Z is both supermodular and monotone, meaning these Shapley values can be approximated efficiently, using Theorem 2.

Opt-Out Transform. Given a coalitional game v, define a new game v_O where

$$v_O(A) = \max_{S \subseteq A} v(S).$$

The opt-out transform essentially allows players to "opt out" of any coalition and receive zero utility. Thus, whenever $v(\{i\})$ is negative, we can think of $v(\{i\})$ as the cost for player i to participate in the game, and he will do so only if this cost is offset by the benefits of cooperating.

If v is supermodular, then the following lemma shows that the game v_O (like the game v_Z) is both supermodular and monotone, and $v_O(A)$ can be efficiently computed. Thus Shapley values can be efficiently approximated here as well.

Lemma 7. *If v is supermodular, then both v_Z and v_O are supermodular and monotone. Furthermore, we can compute $v_Z(A)$ and $v_O(A)$ in polynomial time.*

Proof. Supermodularity of v_Z follows from the definition. Because the marginal contribution of a player to the empty set is now zero, his marginal contribution to any set is nonnegative (by supermodularity), so v_Z must be monotone. Computation of $v_Z(A)$ is straightforward.

Monotonicity of v_O is immediate by definition. For supermodularity, we use a simpler version of a result of Topkis [38]. For all sets B_1 and B_2 and all subsets $A_1 \subseteq B_1$ and $A_2 \subseteq B_2$, we have

$$\begin{aligned} v_O(B_1 \cup B_2) + v_O(B_1 \cap B_2) &\geq v(A_1 \cup A_2) + v(A_1 \cap A_2) \quad \text{(by definition of } v_O\text{)} \\ &\geq v(A_1) + v(A_2) \qquad\quad \text{(by supermodularity of } v\text{)} \end{aligned}$$

Maximizing the right-hand side over all $A_1 \subseteq B_1$ and $A_2 \subseteq B_2$ gives us

$$v_O(B_1 \cup B_2) + v_O(B_1 \cap B_2) \geq v_O(B_1) + v_O(B_2)$$

as desired. The ability to compute $v_O(A)$ in polynomial time follows from our ability to maximize the supermodular function v [20,32]. □

To illustrate these two transforms, consider the following coalitional game v. Let G be an undirected graph. The players of v are the vertices of G, and the value of a coalition A is as follows. Every vertex in A pays an activation *cost* c, and gains a *benefit* $b \geq 0$ for each neighbor in A. That is,

$$v(A) = 2b \cdot |E_A| - c \cdot |A|,$$

where E_A is the set of edges induced by the vertex set A. One can verify that the game v is supermodular, even when generalized to weighted costs and benefits.

Notice that v_Z is equivalent to v but with $c = 0$; thus, v_Z is precisely the edge synergy game studied by Deng and Papadimitriou [12], who show that the Shapley value of player i is exactly $\deg(i) \cdot b$. Thus, in v, the Shapley value for player i is $\deg(i) \cdot b - c$.

The game v_O provides another interesting variant of v. In particular, we can think of v_O as a version of v in which we allow players to opt out of a given coalition A if their participation would incur a net loss. This game appears to be markedly different from v_Z. We can efficiently compute the exact Shapley value of each player when $b \geq 3c/4$. In this case, player i's value is a function of both $\deg(i)$ and the degrees of the nodes within a small radius of i in G. For smaller b, we can approximate Shapley values using the algorithm SV-Sample. Broadly speaking, a player's Shapley value in v_O increases as her degree increases, and it also increases when nodes near her in the network become more valuable. One potentially intriguing way of interpreting Shapley value in v_O is as a new measure of influence of nodes in a network—as in [6,7,22,23], among others.

References

1. Archer, A., Feigenbaum, J., Krishnamurthy, A., Sami, R., Shenker, S.: Approximation and collusion in multicast cost sharing. Games and Economic Behavior 47, 36–71 (2004)
2. Aziz, H.: Algorithmic and complexity aspects of simple coalitional games. PhD thesis, University of Warwick (2009)
3. Bachrach, Y., Markakis, E., Resnick, E., Procaccia, A.D., Rosenschein, J.S., Saberi, A.: Approximating power indices: theoretical and empirical analysis. In: Autonomous Agents and Multi-Agent Systems, vol. 20, pp. 105–122 (2010)
4. Banzhaf, J.F.: Weighted voting doesn't work: A mathematical analysis. Rutgers Law Review 19, 317–343 (1965)
5. Barua, R., Chakravarty, S.R., Roy, S.: Measuring power in weighted majority games. Technical report. Department of Economics, Iowa State University (2007)
6. Bonacich, P.: Power and centrality: A family of measures. American Journal of Sociology 92(5), 1170–1182 (1987)
7. Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems 30(1–7), 107–117 (1998)
8. Curiel, I.J., Maschler, M., Tijs, S.: Bankruptcy games. Zeitschrift für Operations Research 31, 143–159 (1987)
9. Davis, M., Maschler, M.: The kernel of a cooperative game. Naval Research Logistics Quarterly 12, 223–259 (1965)
10. Deegan, J., Packel, E.W.: A new index of power for simple n-person games. International Journal of Game Theory 7(2), 113–123 (1978)
11. Deng, X., Fang, Q.: Algorithmic cooperative game theory. Pareto Optimality, Game Theory and Equilibria 17(1), 159–185 (2008)
12. Deng, X., Papadimitriou, C.: On the complexity of cooperative solution concepts. Mathematics of Operations Research 19(2), 257–266 (1994)
13. Fatima, S.S., Wooldridge, M., Jennings, N.R.: A randomized method for the Shapley value for the voting game. In: Proc. 6th Conference on Autonomous Agents and Multiagent Systems, pp. 955–962 (May 2007)
14. Fatima, S.S., Wooldridge, M., Jennings, N.R.: A linear approximation method for the Shapley value. Artificial Intelligence 172(14), 1673–1699 (2008)
15. Feigenbaum, J., Papadimitriou, C., Shenker, S.: Sharing the cost of multicast transmissions. J. Computing Systems Sciences 63, 21–41 (2001)
16. Felsenthal, D., Machover, M.: The measurement of voting power: Theory and practice, problems and paradoxes. Public Choice 102(3-4), 373–376 (2000)
17. Gillies, D.B.: Solutions to general non-zero-sum games. Contributions to the Theory of Games 4, 47–85 (1959)
18. Herzog, S., Shenker, S., Estrin, D.: Sharing the "cost" of multicast trees: an axiomatic analysis. IEEE/ACM Transactions on Networking 5, 847–860 (1997)
19. Ieong, S., Shoham, Y.: Marginal contribution nets: a compact representation scheme for coalitional games. In: Proc. 6th ACM Conference on Electronic Commerce (2005)
20. Iwata, S., Fleischer, L., Fujishige, S.: A combinatorial, strongly polynomial-time algorithm for minimizing submodular functions. In: Proc. 32nd Symposium on Theory of Computing, pp. 97–106 (2000)
21. Jain, K., Vazirani, V.: Applications of approximation algorithms to cooperative games. In: Proc. 33rd ACM Symposium on Theory of Computing (2001)

22. Katz, L.: A new status index derived from sociometric analysis. Psychometrika 18(1), 39–43 (1953)
23. Kleinberg, J., Tardos, E.: Balanced outcomes in social exchange networks. In: Proc. 40th Symposium on Theory of Computing (2008)
24. Leech, D.: An empirical comparison of the performance of classical power indices. Political Studies 50(1), 1–22 (2002)
25. Mann, I., Shapley, L.S.: Values of large games, IV: Evaluating the electoral college by Monte-Carlo techniques. Technical report. The Rand Corporation, Santa Monica, CA (1960)
26. Maschler, M., Peleg, B., Shapley, L.S.: The kernel and bargaining set for convex games. International Journal of Game Theory 1, 73–93 (1971)
27. Maschler, M., Peleg, B., Shapley, L.S.: Geometric properties of the kernel, nucleolus, and related solution concepts. Mathematics of Operations Research 4(4), 303–338 (1979)
28. Matsui, Y., Matsui, T.: A survey of algorithms for calculating power indices of weighted majority games. Journal of the Operations Research Society of Japan 43(1), 71–86 (2000)
29. Matsui, Y., Matsui, T.: NP-completeness for calculating power indices of weighted majority games. Theoretical Computer Science 263(1-2), 305–310 (2001)
30. Motwani, R., Raghavan, P.: Randomized Algorithms, Cambridge (1995)
31. O'Neill, B.: A problem of rights arbitration from the Talmud. Mathematical Social Sciences 2(4), 345–371 (1982)
32. Schrijver, A.: A combinatorial algorithm minimizing submodular functions in strongly polynomial time. Journal of Combinatorial Theory B 80, 346–355 (2000)
33. Schulz, A.S., Uhan, N.A.: Approximating the least core value and least core of cooperative games with supermodular costs. Working paper (2010)
34. Schulz, A.S., Uhan, N.A.: Sharing supermodular costs. Operations Research 58(4), 1051–1056 (2010)
35. Shapley, L.S.: A value for n-person games. In: Kuhn, H., Tucker, A.W. (eds.) Contributions to the Theory of Games II, pp. 307–317. Princeton University Press (1953)
36. Shapley, L.S.: Cores of convex games. International Journal of Game Theory 1, 11–26 (1971)
37. Shapley, L.S., Shubik, M.: On the core of an economic system with externalities. American Economic Review 59, 678–684 (1969)
38. Topkis, D.M.: Minimizing a submodular function on a lattice. Operations Research 26(2), 305–321 (1978)
39. Yao, A.C.: Probabilistic computations: Toward a unified measure of complexity. In: Proc. 18th Symposium on Foundations of Computer Science, pp. 222–227 (1977)

Equilibria of GSP for Range Auction

H.F. Ting and Xiangzhong Xiang

Department of Computer Science, The University of Hong Kong, Hong Kong
{hfting,xzxiang}@cs.hku.hk

Abstract. Position auction is a well-studied model for analyzing online auctions for internet advertisement, in which a set of advertisers bid for a set of slots in a search result page to display their advertisement links. In particular, it was proved in [10,11] that the Generalized Second Price (GSP) mechanism for position auction has many interesting properties. In this paper, we extend these results to range auction, in which a bidder may specify a range of slots he is interested in. We prove GSP for range auction has an envy free equilibrium, which is bidder optimal and has the minimum pay property. Further, this equilibrium is equal to the outcome of the Vickrey-Clarke-Groves mechanism. We also show that the social welfare of any equilibrium of GSP for range auctions is not far from the optimal; it is at least 1/2 of the optimal.

Classification: Algorithmic Game Theory and Online Algorithms.

Keywords: Online Auctions, Nash Equilibria, Envy-free, Valuations.

1 Introduction

Internet advertising, which sells online advertisements via search engines, is one of the biggest and fastest growing businesses on internet. The idea of this new type of advertisement is simple: in response to a user query, a search engine displays not only the result of the search, but also a set of *sponsored advertisement links*. When a user clicks a sponsored link and is brought to the advertisement page, the owner of the link will be charged a price, which is called *price-per-click* in the industry. To determine in nearly real-time whose advertisement links would be displayed, most search engines run online auctions. To enter the auctions, an advertiser specifies a set of keywords related to his product, together with a bid indicating the maximum price-per-click he is willing to pay. Given a query, which consists of keywords, those advertisers whose sets of keywords include some keywords in the query would participate in an auction to compete for positions for their advertisements.

The Generalized Second Price (GSP) mechanism has been the most widely used auction mechanism in internet advertising since its introduction in 2002 (for Google's AdWords Select system). It is tailor-made for internet advertising, which has the unique feature that there is an ordering on the slots of sponsored links. For example, in the case of Google, Yahoo and MSN, these slots are located near the right border of a page, ordered linearly from top to bottom. Experiences

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 580–591, 2012.

confirm that a higher slot often receives more clicks and thus is more desirable. GSP allocates the slots (from top to bottom) to the advertisers in descending order of the bids and charges the advertiser the bid of the next one in this order.

GSP is simple, transparent, and easy to be explained to ordinary advertisers. Search engine companies have made a successful business out of GSP. For example, the combined revenue of Google and Yahoo exceeded $34 billion in 2010. Given this huge commercial success, there were many studies on GSP in recent years. For instance, Aggarwal, Goel and Motwani [4] showed that although GSP equals the optimal Vickrey-Clarke-Groves (VCG) mechanism if there is only one slot, they are very different otherwise. In particular, they proved that while VCG is always truthful [7,12], GSP may not be truthful for auctions with more than one slot. Later, Edelman, Ostrovsky and Schwarz [10], and independently Varian [11] studied the equilibria of GSP. Note that popular search engines such as Google and Yahoo provide tools for advertisers to analyze the performance of their advertisements and allow them to change their bids frequently. Hence, a key concern about the auction is whether this auction has equilibrium. Edelman, Ostrovsky and Schwarz [10] and Varian [11] proposed the following *position auction* model to study GSP's equilibria formally: There are n advertisers and k slots where $n \geq k$. For $1 \leq i \leq n$, advertiser i has a value of v_{ij} for his advertisement provided that it is allocated at slot j. For $1 \leq j \leq k$, slot j has a *click-through-rate* $c_j \geq 0$ where $c_1 \geq c_2 \geq \ldots \geq c_k$. If i has his advertisement assigned to j and is charged a total price of p_j, he has a *net profit* of $v_{ij}c_j - p_j$.

It is proved in [10,11] that GSP for position auction has Nash equilibrium if the value of an advertisement is independent of its location, i.e., $v_{i1} = v_{i2} = \ldots = v_{ik}$ for every i. Furthermore, this equilibrium is an *envy-free Nash equilibrium*, and its assignment and pricing are identical to those of the VCG mechanism. Following [10,11], there are many studies on position auction [2,3,5,8]. In particular, Aggarwal, Feldman and Muthukrishnan [2] observed that the value of an advertisement may not be independent of its location; a higher slot not only has a higher click-through-rate but also has other advantages. They cited a study by the Interactive Advertising Bureau confirming that higher slots have better brand awareness effect. Furthermore, users have a tendency of clicking the advertisement links from top to bottom, and one at a higher slot may capture the attention of a user earlier. To provide better control for the advertisers, Aggarwal *et al.* generalized position auction to the *prefix position auction* model, or simply prefix auction, in which an advertiser i specifies a bid b_i and a *bottom cutoff* $\kappa_i \in [1, k]$, indicating that he is only interested in the first κ_i slots, and is willing to pay a maximum price-per-click of b_i if his advertisement is placed at one of them. They proved that under the assumption that $v_{i1} = v_{i2} = \ldots = v_{i\kappa_i} \geq 0$, and $v_{i(\kappa_i+1)} = \ldots = v_{ik} = 0$ for every $1 \leq i \leq n$, GSP for prefix auction has envy-free Nash equilibrium, and its assignment and pricing are identical to those of VCG. Furthermore, this equilibrium is *bidder optimal*, and has the *ordering* and the *minimum pay* properties.

Our Contributions. In this paper, we study a natural extension of the prefix auction. This extension is proposed by Aggarwal *et al.* [2], who observed that

sometimes an advertiser may also want a *top cutoff* (in addition to a bottom cutoff). For example, he may want his advertisement to appear consistently in a small range of positions (or even at some fixed position), or he may want to avoid the topmost slots in order to "weed out clickers who do not persist through the topmost advertisements to choose the most appropriate one" [2]. Thus, they proposed the *range auction*, in which every advertiser i specifies a top cutoff τ_i and a bottom cutoff κ_i, indicating that he is only interested in the slots $\tau_i, \tau_i + 1, \ldots, \kappa_i$. Despite this seemingly insignificant extension, it is surprising that range auction is fundamentally different from the position and the prefix auctions. Recall that for both the position and the prefix auctions, GSP has a Nash equilibrium whose allocation and pricing equal those of VCG, and it has the ordering and the minimum pay properties [2, 10, 11]. However, Aggarwal *et al.* [2] showed that for range auction, GSP can no longer guarantee that it has a Nash equilibrium with allocation and pricing equal those of VCG.

After a careful study, we are convinced that submitting a single bid for all preferred slots is not sufficiently expressive for an advertiser to capture his preferences. This paper investigates the fundamental question on what properties of GSP for range auction can be restored if each advertiser i is allowed to submit more than one bid (i.e., allow i to submit some different bids b_{ij} for those slots j that he has different preferences). With this extension, we prove that GSP for range auction has once again an envy-free Nash equilibrium, and this equilibrium is *bidder optimal* and its assignment and pricing are identical to those of VCG. Furthermore, we show that this extended GSP for range auction has the minimum pay property, but it does not have the ordering property. In fact, we show that no reasonable mechanism has the ordering property for range auction.

We note that previous proofs of the same properties of the equilibria of GSP for prefix auctions are based on a fundamental lemma (Lemma 1 in [2]) on the allocations of VCG. In *Example 2*, we show that this lemma no longer holds for range auction. Hence, we have to prove these properties of equilibria in a different, but more direct way. Interestingly, our proofs do not need the assumption used in all previous works, namely the value of an advertisement is independent of the slot it locates, i.e., $v_{i\tau_i} = v_{i(\tau_i+1)} = \ldots = v_{i\kappa_i} > 0$, and $v_{ij} = 0$ for $j \notin [\tau_i, \kappa_i]$. Instead, we need the less restrictive and more reasonable assumption that the value of an advertisement is not diminished if it is relocated to some higher slot: $v_{i\tau_i} \geq v_{i(\tau_i+1)} \geq \ldots \geq v_{i\kappa_i} > 0$, and $v_{ij} = 0$ for $j \notin [\tau_i, \kappa_i]$.

We also study how the extension from position auction to range auction affects the social welfare of GSP. Given an allocation which assigns advertiser i to slot j, its *social welfare* is defined to be $\sum_{1 \leq j \leq k} v_{ij} c_j$. In [9], Paes Leme and Tardos proved that for position auction, any Nash equilibrium of GSP has social welfare at least $\frac{1}{1.618}$ times that of the optimal allocation. This bound is recently improved to $\frac{1}{1.282}$ in [6]. We generalize these results to the range auction. We prove that any Nash equilibrium of GSP for range auction has social welfare at least $\frac{1}{2}$ times that of the optimal one. Furthermore, we show that this bound is nearly tight, or more precisely, we prove that for any small enough positive ξ, we can construct a Nash equilibrium whose social welfare is at most $\frac{1}{2-\xi}$

times that of the optimal. The main difficulty for extending existing results for position auction to range auction is as follows. For position auction the optimal allocation is the greedy allocation, which assigns the advertiser with the largest value to the highest slot, the one with the second highest value to the second slot and so on. Hence, we can use mathematical induction to bound the social welfare of the greedy allocation with that of the GSP allocation. However, this greedy allocation property does not hold for range auction, and we can no longer inductively get the bound.

2 Preliminaries and Notations

2.1 Range Auction

In our discussion, we assume that there are n advertisers $1, 2, \ldots, n$ and k slots $1, 2, \ldots k$ where $n \geq k$. Each slot j is associated with a click-through-rate c_j, and each advertiser i has a value v_{ij} and submits a bid b_{ij} for slot j. He also specifies a top cutoff τ_i and a bottom cutoff κ_i. These values have the following relationships:

- $c_1 \geq c_2 \geq \ldots \geq c_k \geq 0$.
- For each i, $v_{i\tau_i} \geq v_{i(\tau_i+1)} \geq \ldots \geq v_{i\kappa_i} > 0$, and $v_{ij} = 0$ for every $j \notin [\tau_i, \kappa_i]$.

If advertiser i is assigned to slot j, and is charged a price of p_j, the net-profit of i is $v_{ij}c_j - p_j$. Note that if we let $\rho_j = p_j/c_j$ denote the price-per-click, then his net-profit can be rewritten as $v_{ij}c_j - \rho_j c_j$.

We say that a given allocation and pricing is at *envy-free Nash equilibrium* if each advertiser's net-profit will not increase if he is reassigned to another slot and pay the price there. To be more precise, suppose that p_x is the price that the current occupant at slot x is paying. If advertiser i is currently assigned to slot j and pay the price p_j, then for any other slot x, we have $c_x v_{ix} - p_x \leq c_j v_{ij} - p_j$; if i is not assigned to any slot, then $c_x v_{ix} - p_x \leq 0$. We say that the allocation and pricing is *bidder optimal* if each advertiser's net profit is the maximum net profit he can make among all envy-free equilibria.

We are also interested in the following properties of an auction mechanism.

Minimum Pay Property. If for any input bids, it satisfies the following: Suppose the advertiser assigned to the slot j is charged the price-per-click ρ_j, then he pays the minimum amount he would have needed to bid in order to be assigned to slot j, i.e., if he has bidden less than ρ_j, he could not be assigned to slot j, and if he has bidden more than ρ_j, he will be assigned slot j or some higher slot.

Ordering Property. If advertiser x_j is assigned to slot j $(1 \leq j \leq k)$, then $b_{x_1 1} \geq b_{x_2 2} \geq \ldots \geq b_{x_k k}$.

2.2 The Vickrey-Clarke-Groves (VCG) Auction

In our analysis of GSP for range auction, we need to refer to the pricing method of VCG. This section gives minimum details about VCG that are necessary for

our discussion. The allocation $\Theta \subseteq \{(i,j) \mid 1 \le i \le n,\ 1 \le j \le k\}$ returned by VCG is a matching from the set of advertisers to the set of slots such that

(i) it is feasible, i.e., if $(i,j) \in \Theta$ then $\tau_i \le j \le \kappa_i$; and
(ii) its value $v(\Theta) = \sum_{(i,j)\in\Theta} v_{ij}c_j$ is the maximum over all feasible matchings.

If i is assigned to j, i.e, $(i,j) \in \Theta$, then it is charged by VCG a price of

$$v(\Theta_{-i}) - (v(\Theta) - c_j v_{ij})$$

where Θ_{-i} is the allocation returned by VCG if we take i out of the auction. Following is an example.

Example 1. There are three slots with $c_1 = 4, c_2 = 2, c_3 = 1$, and four advertisers with $(v_{11}, v_{12}, v_{13}) = (\$20, \$0, \$0), (v_{21}, v_{22}, v_{23}) = (\$15, \$15, \$0), (v_{31}, v_{32}, v_{33}) = (\$10, \$10, \$0)$, and $(v_{41}, v_{42}, v_{43}) = (\$0, \$5, \$5)$. Then, $\Theta = \{(1,1), (2,2), (4,3)\}$. Note that $(1,1) \in \Theta$ and $\Theta_{-1} = \{(2,1), (3,2), (4,3)\}$; thus the price charged to advertiser 1 by VCG is $v(\Theta_{-1}) - (v(\Theta) - c_1 v_{11}) = 85 - (115 - 80) = 50$.

To represent the difference between Θ and Θ_{-i} succinctly, observe that when converting Θ to Θ_{-i}, i is moved out and some advertiser i_0 takes its slot. Then, another advertiser i_1 takes the slot vacated by i_0, and so on until some previously unassigned advertiser j is assigned to the last vacated slot (or no advertiser bids for this slot). We call this sequence $j \to \ldots \to i_1 \to i_0 \to i$ of moving advertisers the *chain* for converting Θ to Θ_{-i}, and a transition, say $\ell \to \ell'$, a link in the chain. Note that for those slots occupied by other advertisers, we can assume their assignments are the same in both Θ and Θ_{-i} (to see this, try substituting a purported better assignment on these slots back into Θ). We say that a chain is of minimum length if there is no shorter chain that transforms Θ to some allocation not involving i and having the same valuation as Θ_{-i}. In Example 1, the chain for converting Θ to Θ_{-1} is $3 \to 2 \to 1$.

Before leaving this section, we show a fundamental difference between range auction and position/prefix auction. Consider any link $i_0 \to i_1$ in the chain. Suppose that i_0 and i_1 are located at slots j_0 and j_1, respectively. We say that the link is an upward link if $j_0 > j_1$; otherwise, it is downward. To prove the above properties of equilibria for prefix auction, Aggarwal *et al.* [2] needed a fundamental lemma on downward-upward-link (Lemma 1 in [2]), which asserts that the minimum length chain for converting Θ to Θ_{-i} does not contain a downward link followed by an upward link. This is not true for range auction. In range auction, we have an example in which the minimum length chain for converting Θ to Θ_{-i} contains a downward link followed by an upward link.

Example 2. There are three slots 1,2,3 and five advertisers $1, 2, 3, 4, 5$. The click through rates are $c_1 = 6$, $c_2 = 5$, $c_3 = 4$. Their values for these slots are: $(v_{11}, v_{12}, v_{13}) = (\$0, \$10, \$10), (v_{21}, v_{22}, v_{23}) = (\$9, \$9, \$9), (v_{31}, v_{32}, v_{33}) = (\$100, \$0, \$0)$, $(v_{41}, v_{42}, v_{43}) = (\$0, \$8, \$0)$ and $(v_{51}, v_{52}, v_{53}) = (\$1, \$1, \$1)$. Then, the VCG-allocation $\Theta = \{(3,1), (1,2), (2,3)\}$ and $\Theta_{-3} = \{(2,1), (4,2), (1,3)\}$. So the chain for converting Θ to Θ_{-3} is $4 \to 1 \to 2 \to 3$. Note that in this minimum length chain: $1 \to 2$ is a downward link (advertiser 1 is at slot 2, and 2 is at slot 3) while $2 \to 3$ (advertiser 2 is at slot 3 and 3 is at slot 1) is an upward link.

3 GSP and VCG for Range Auction

In this section, we first study some useful properties of GSP for range auction, which works as follows.

> For each slot $j = 1, 2, \ldots, k$ in order from top to bottom, iteratively run the following standard second-price auction: Let U be the set of advertisers i who are interested in the current slot j, i.e., $\tau_i \leq j \leq \kappa_i$ and have not been allocated any slot yet. The advertiser i in U with the highest bid b_{ij} is allocated slot j, and is charged a price-per-click equal to the second-highest bid in $\{b_{xj} \mid x \in U\}$.

It is trivial to prove that GSP for range auction has the *Minimum Pay Property*. Suppose x_j is allocated slot j and pays the price p_j. Then, according to the pricing mechanism, p_j is the highest bid among all those unassigned advertisers interested in j other than x_j. Obviously, if x_j changes its bid to some value lower than p_j, he will lose slot j to the one who currently has this bid p_j; otherwise, he will not lose it.

GSP for range auction cannot guarantee the *Ordering Property*. The following simple example shows that no reasonable mechanism can guarantee the ordering property for range auction. There are two advertisers $1, 2$ and two slots $1, 2$. The bid of the advertiser 1 is $b_{11} = 100, b_{12} = 0$, and for advertiser 2 is $b_{21} = 0, b_{22} = 200$. Note that advertiser 1 only wants slot 1 and advertiser 2 slot 2. Since each slot has only one effective bidder, a reasonable mechanism should assign advertiser 1 to slot 1 and 2 to slot 2. Note that it is not true that advertisers with higher bids win higher slots. The ordering property does not hold.

In the rest of this section, we prove a fundamental property about VCG for range auction: Suppose VCG assigns advertisers i and i' to slots j and j', respectively. Suppose that $j < j'$ and i' is also interested in this higher slot j. Then, VCG must charge i a higher price-per-click than that it charges i'. Note that previous analysis also needs this property, but their proofs are based on the upward-downward link lemma, which is not true for range auction.

To prove this price-per-click ordering property, we need some notations. We use $\binom{j}{i}$ to denote the fact that slot j is allocated to advertiser i, and in general, we use $\binom{j_0\ j_1\ \cdots\ j_m}{i_0\ i_1\ \ldots\ i_m}$ to denote the fact that the slots $j_0, j_1, \ldots, j_m$ are allocated to the advertisers $i_0, i_1, \ldots, i_m$, respectively. Define

$$v\left(\binom{j_0\ j_1\ \cdots\ j_m}{i_0\ i_1\ \ldots\ i_m}\right) = v_{i_0 j_0} c_{j_0} + \ldots + v_{i_m j_m} c_{j_m}.$$

Recall that Θ denotes the VCG allocation, which has the maximum value among all possible allocations, and Θ_{-i} is the one returned by VCG after taking advertiser i out of the auction. To analyze the price-per-click ordering property, we need to compare the value of Θ_{-x} and Θ_{-y} for any two advertisers x and y. First, we have the following trivial fact.

Fact 31. *Suppose that in Θ_{-y}, x is assigned to slot j, and y is also interested in j, i.e., $\tau_y \leq j \leq \kappa_y$. Then, $v(\Theta_{-x}) \geq v(\Theta_{-y}) - v(\binom{j}{x}) + v(\binom{j}{y})$.*

To see this, just replace x by y at slot j in Θ_{-y}, and we get an allocation without x with value $v(\Theta_{-y}) - v(\binom{j}{x}) + v(\binom{j}{y})$, which, by optimality of Θ_{-x}, is no greater than $v(\Theta_{-x})$. The next lemma shows similar relationship for some other slot determined by Θ, not Θ_{-y}.

Lemma 32. *Suppose that in Θ, x is assigned to slot j, and y is also interested in j, i.e., $\tau_y \le j \le \kappa_y$. Then, $v(\Theta_{-x}) \ge v(\Theta_{-y}) - v(\binom{j}{x}) + v(\binom{j}{y})$.*

Proof. Note that if x is also assigned to slot j in Θ_{-y}, then the lemma follows directly from Fact 31. Suppose that x is not assigned to slot j in Θ_{-y}. Since x is in slot j in Θ, but is moved out of j in Θ_{-y}, it must be in the chain

$$y \leftarrow i_\ell \leftarrow \ldots \leftarrow i_1 \leftarrow x \leftarrow w \leftarrow \ldots$$

that converts Θ to Θ_{-y}. Suppose that in Θ, $x, i_1, \ldots, i_\ell$ and y are assigned to $j, j_1, \ldots, j_\ell, j_{\ell+1}$, respectively, i.e.,

$$\Theta = \Theta' \cup \left(\begin{smallmatrix} j & j_1 & j_2 & \cdots & j_\ell & j_{\ell+1} \\ x & i_1 & i_2 & \ldots & i_\ell & y \end{smallmatrix}\right)$$

where Θ' denotes the allocation of the other slots in Θ. After converting Θ to Θ_{-y} by the chain, we have

$$\Theta_{-y} = \Theta'_{-y} \cup \left(\begin{smallmatrix} j & j_1 & j_2 & \cdots & j_\ell & j_{\ell+1} \\ w & x & i_1 & \ldots & i_{\ell-1} & i_\ell \end{smallmatrix}\right),$$

where Θ'_{-y} denotes the other allocation in Θ'_{-y}. Now, consider the new allocation

$$X = \Theta' \cup \left(\begin{smallmatrix} j & j_1 & j_2 & \cdots & j_\ell & j_{\ell+1} \\ y & x & i_1 & \ldots & i_{\ell-1} & i_\ell \end{smallmatrix}\right).$$

The optimality of Θ suggests that $v(\Theta) \ge v(X)$, i.e.,

$$v\left(\Theta' \cup \left(\begin{smallmatrix} j & j_1 & j_2 & \cdots & j_\ell & j_{\ell+1} \\ x & i_1 & i_2 & \ldots & i_\ell & y \end{smallmatrix}\right)\right) \ge v\left(\Theta' \cup \left(\begin{smallmatrix} j & j_1 & j_2 & \cdots & j_\ell & j_{\ell+1} \\ y & x & i_1 & \ldots & i_{\ell-1} & i_\ell \end{smallmatrix}\right)\right),$$

or equivalently,

$$v\left(\binom{j}{x}\right) + v\left(\left(\begin{smallmatrix} j_1 & j_2 & \cdots & j_\ell & j_{\ell+1} \\ i_1 & i_2 & \ldots & i_\ell & y \end{smallmatrix}\right)\right) \ge v\left(\binom{j}{y}\right) + v\left(\left(\begin{smallmatrix} j_1 & j_2 & \cdots & j_\ell & j_{\ell+1} \\ x & i_1 & \ldots & i_{\ell-1} & i_\ell \end{smallmatrix}\right)\right). \quad (1)$$

Recall that $\Theta_{-y} = \Theta'_{-y} \cup \left(\begin{smallmatrix} j & j_1 & j_2 & \cdots & j_\ell & j_{\ell+1} \\ w & x & i_1 & \ldots & i_{\ell-1} & i_\ell \end{smallmatrix}\right)$. Let $Y = \Theta'_{-y} \cup \left(\begin{smallmatrix} j & j_1 & j_2 & \cdots & j_\ell & j_{\ell+1} \\ w & i_1 & i_2 & \ldots & i_\ell & y \end{smallmatrix}\right)$. Then,

$$\begin{aligned} v(Y) + v\left(\binom{j}{x}\right) &= v(\Theta'_{-y}) + v\left(\binom{j}{w}\right) + v\left(\left(\begin{smallmatrix} j_1 & j_2 & \cdots & j_\ell & j_{\ell+1} \\ i_1 & i_2 & \ldots & i_\ell & y \end{smallmatrix}\right)\right) + v\left(\binom{j}{x}\right) \\ &\ge v(\Theta'_{-y}) + v\left(\binom{j}{w}\right) + v\left(\left(\begin{smallmatrix} j_1 & j_2 & \cdots & j_\ell & j_{\ell+1} \\ x & i_1 & \ldots & i_{\ell-1} & i_\ell \end{smallmatrix}\right)\right) + v\left(\binom{j}{y}\right) = v(\Theta_{-y}) + v\left(\binom{j}{y}\right). \end{aligned}$$

(The inequality is from (1).) It can be verified that Y does not assign x to any slot, and by the optimality of Θ_{-x}, we conclude that $v(\Theta_{-x}) \ge v(Y)$.

Lemma 33. *Suppose that in the allocation Θ of VCG, advertisers x and y are assigned to slots j and j', respectively. Suppose that $j < j'$, and y is also interested in this higher slot j, i.e., $\tau_y \le j \le \kappa_y$. Then, the price-per-click that VCG charges x is no smaller than that VCG charges y.*

Proof. The price-per-click that VCG charges x is $\rho_x = \frac{v(\Theta_{-x})-(v(\Theta)-c_j v_{xj}))}{c_j}$. By Lemma 32 (recall that $v\left(\binom{j}{i}\right) = c_j v_{ij}$), we have

$$\rho_x \geq \frac{v(\Theta_{-y})-(v(\Theta)-c_j v_{yj}))}{c_j} = \frac{v(\Theta_{-y})-v(\Theta)}{c_j} + v_{yj}. \quad (2)$$

Note that $v(\Theta_{-y}) - v(\Theta) \leq 0$, $c_j \geq c_{j'}$ and $v_{yj} \geq v_{yj'}$, (2) can be rewritten as $\rho_x \geq \frac{v(\Theta_{-y}-v(\Theta))}{c_{j'}} + v_{yj'} = \frac{v(\Theta_{-y})-(v(\Theta)-c_{j'}v_{yj'})}{c_{j'}}$, which is the price-per-click that VCG charges y.

4 Equilibria of GSP for Range Auction

We are now ready to analyze the equilibria of GSP for range auction. We show that it has an envy-free Nash equilibrium whose allocation and pricing is identical to those of VCG, and among all envy-free Nash equilibria, this one is bidder-optimal.

To simplify the notation, we rename the advertisers if necessary and assume that advertiser j is assigned to slot j for $1 \leq j \leq k$ in the VCG allocation Θ. Suppose that VCG charges j a total price of p_j, and thus a price-per-click $\rho_j = p_j/c_j$. The following procedure determines a bidding Γ of the advertisers that will lead us the desired equilibrium.

1 Initialize all b_{ij} as 0, set all advertisers' stated cutoffs to be their true cutoffs and set $j = 1$.
2 If $\tau_j < j$, set $b_{j(j-1)} = b_{jj} = \rho_{j-1}$.
3 If $\tau_j = j$, set $b_{jj} = \rho_j + \epsilon$, where ϵ is a very small positive real number.
4 If $j = k$ or $\tau_{j+1} = j + 1$, find the $y \in \{j + 1, j + 2, ..., n\}$ with the largest v_{yj}. Set $b_{yj} = \rho_j$.
5 If $j < k$, set $j = j + 1$ and go to Step 2.

Theorem 41. *The allocation and pricing resulted by applying GSP for range auctions to Γ is identical to those given by VCG, i.e., each advertiser is allocated the same slot and charged the same price-per-click. Furthermore, this allocation and pricing is at envy-free Nash equilibrium.*

Proof. For any $1 \leq j \leq k$, since advertiser j is assigned to slot j in Θ, we have $\tau_j \leq j \leq \kappa_j$. We show below that GSP will also assign advertiser j to slot j and charge him a price-per-click ρ_j. Consider two cases.

Case i. Suppose that $\tau_j < j$. In Step 2 of our procedure, we set $b_{jj} = \rho_{j-1}$.

- If $j < k$ and $\tau_{j+1} < j + 1$, then $b_{(j+1)j} = \rho_j$. Since $\tau_j < j$, advertiser j is also interested in slot $j - 1$, and by Lemma 33, price-per-click for slot $j - 1$ is no lower than that for j, i.e., $\rho_{j-1} \geq \rho_j$. Hence $b_{jj} = \rho_{j-1} \geq \rho_j = b_{(j+1)j}$, and together with the fact that other advertisers' bids for slot j are all 0, j would win slot j, and is charged price-per-click of the second highest bid, which is $b_{(j+1)j} = \rho_j$.

- Suppose that $j = k$ or $\tau_{j+1} = j + 1$. By the procedure, there is one advertiser y with $b_{yj} = \rho_j$, and all the other bids for the slot j are 0. Again, we have $\rho_{j-1} \geq \rho_j$ because j is also interested in slot $j - 1$ (Lemma 33); thus, $b_{jj} = \rho_{j-1} \geq \rho_j = b_{yj}$, and advertiser j wins slot j and is charged a price-per-click ρ_j.

Case ii. Suppose that $\tau_j = j$. Then Step 3 of our procedure sets $b_{jj} = \rho_j + \epsilon$.

- If $j < k$ and $\tau_{j+1} < j + 1$, then the procedure sets $b_{(j+1)j} = \rho_j$, and all other bids for j are 0. Obviously, $b_{jj} = \rho_j + \epsilon > \rho_j = b_{(j+1)j}$ and advertiser j wins slot j, and is charged a price-per-click of the second highest bid $b_{(j+1)j} = \rho_j$.
- Suppose that $j = k$ or $\tau_{j+1} = j + 1$. There is one advertiser y with $b_{yj} = \rho_j$, and all other bids for j are 0. Then, $b_{jj} = \rho_j + \epsilon > \rho_j = b_{yj}$, and advertiser j will win slot j, and is charged a price-per-click ρ_j.

Thus, the allocation and pricing of GSP for Γ is identical to those given by VCG.

Now we show that the allocation and pricing is at envy-free equilibrium. We have proved above that for any $1 \leq i \leq k$, advertiser i is assigned to slot i by GSP and is charged the VCG price $p_i = c_i\rho_i = v(\Theta_{-i}) - (v(\Theta) - c_i v_{ii})$. Consider any such advertiser i. We now show that i will not envy any other advertiser j assigned to slot j; or more precisely, we show that the current net profit of i, which is $c_i v_{ii} - p_i$, is no smaller than $c_j v_{ij} - p_j$, the net profit he would make if i is assigned to slot j and pays a price of p_j. We assume that i is interested in j (otherwise, $v_{ij} = 0$ and i does not envy j obviously). From Lemma 32, we have

$$\begin{array}{rcl} & v(\Theta_{-j}) & \geq v(\Theta_{-i}) - v(\binom{j}{j}) + v(\binom{j}{i}) \\ \Leftrightarrow v(\Theta_{-j}) - (v(\Theta) - c_j v_{jj}) & & \geq v(\Theta_{-i}) - (v(\Theta) - c_j v_{ij}) \\ \Leftrightarrow & p_j & \geq p_i - c_i v_{ii} + c_j v_{ij} \\ \Leftrightarrow & c_i v_{ii} - p_i & \geq c_j v_{ij} - p_j, \quad \text{as required.} \end{array}$$

Now consider some advertiser i that is not assigned to any slot. Then, i has zero net profit. To show that he does not envy any slot j that he is interested, we need to show $c_j v_{ij} - p_j \leq 0$. Replacing j by i at slot j in Θ, we have an allocation without j, and by the optimality of Θ_{-j} over these allocations, we have $v(\Theta_{-j}) \geq v(\Theta) - c_j v_{jj} + c_j v_{ij}$. Hence, $v(\Theta_{-j}) - v(\Theta) + c_j v_{jj} \geq c_j v_{ij} \Rightarrow p_j \geq c_j v_{ij} \Rightarrow c_j v_{ij} - p_j \leq 0$.

The following lemma studies the net profit of the advertisers.

Lemma 42. *The allocation and pricing determined by GSP for Γ is bidder optimal, i.e., each advertiser makes the maximum net profit that he can make over any envy-free equilibrium.*

Proof. Recall that the allocation and pricing of GSP for Γ is identical to the VCG allocation Θ and its price is the VCG price, and we have also assumed that in this allocation, advertiser i is assigned to slot i. Hence, the total value of this GSP allocation equals $v(\Theta)$ and for any slot i, its net profit in this GSP allocation is $c_i v_{ii} - p_i = c_i v_{ii} - (v(\Theta_{-i}) - (v(\Theta) - c_i v_{ii})) = v(\Theta) - v(\Theta_{-i})$. We

need to prove that this net profit is no smaller than what i will make in any other envy-free equilibrium.

Consider an arbitrary envy-free equilibrium with allocation E and pricing q. Suppose that for $1 \le i \le k$, advertiser x_i is assigned to slot i in E and is charged a price of q_i. To avoid unnecessary complication, assume that the remaining $n-k$ unassigned advertisers are assigned to $n-k$ dummy slots j with $c_j = 0$ and are charged with zero price. First, we argue that $v(E) = v(\Theta)$. By the envy-free property of E, we have the net-profit of x_i is no smaller than that he will make if we reassign him to slot x_i, i.e., $c_i v_{x_i i} - q_i \ge c_{x_i} v_{x_i x_i} - q_{x_i}$. It follows that

$$\begin{array}{rc} & \sum_{1\le i\le n}(c_i v_{x_i i} - q_i) \ge \sum_{1\le i\le n}(c_{x_i} v_{x_i x_i} - q_{x_i}) \\ \Leftrightarrow & \sum_{1\le i\le n} c_i v_{x_i i} \ge \sum_{1\le i\le n} c_{x_i} v_{x_i x_i} \\ \Leftrightarrow & v(E) \ge v(\Theta). \end{array}$$

(Recall that we have assumed that in the VCG allocation Θ, advertiser x is assigned to slot x.) But Θ has the maximum value, thus, $v(E) = v(\Theta)$.

Now we show that for any advertiser x_{i_0}, the net profit he makes in E is not greater than what he makes in GSP, i.e.,

$$c_{i_0} v_{x_{i_0} i_0} - q_{i_0} \le v(\Theta) - v(\Theta_{-x_{i_0}}). \tag{3}$$

Since $v(E) = v(\Theta)$, E is also an optimal allocation. Let $x_{i_0} \leftarrow x_{i_1} \leftarrow \ldots \leftarrow x_{i_\ell}$ be the chain that converts E to $\Theta_{-x_{i_0}}$. Since E is envy-free, for $j \in [0, \ell-1]$, we have $c_{i_{j+1}} v_{x_{i_{j+1}} i_{j+1}} - q_{i_{j+1}} \ge c_{i_j} v_{x_{i_{j+1}} i_j} - q_{i_j}$. It follows that

$$\begin{array}{l} \sum_{0\le j\le \ell-1} q_{i_j} \ge \sum_{0\le j\le \ell-1}(c_{i_j} v_{x_{i_{j+1}} i_j} - c_{i_{j+1}} v_{x_{i_{j+1}} i_{j+1}} + q_{i_{j+1}}). \\ \Leftrightarrow q_{i_0} \ge c_{i_0} v_{x_{i_1} i_0} - c_{i_1} v_{x_{i_1} i_1} + \ldots + c_{i_{\ell-1}} v_{x_{i_\ell} i_{\ell-1}} - c_{i_\ell} v_{x_{i_\ell} i_\ell} + q_{i_\ell} \\ \Rightarrow q_{i_0} \ge (c_{i_0} v_{x_{i_1} i_0} + \ldots + c_{i_{\ell-1}} v_{x_{i_\ell} i_{\ell-1}}) - (c_{i_1} v_{x_{i_1} i_1} + \ldots + c_{i_\ell} v_{x_{i_\ell} i_\ell}) \\ \Leftrightarrow q_{i_0} \ge v(\Theta_{-x_{i_0}}) - v(E) + c_{i_0} v_{x_{i_0} i_0} = v(\Theta_{-x_{i_0}}) - v(\Theta) + c_{i_0} v_{x_{i_0} i_0}, \end{array}$$

and (3) follows.

5 Social Welfare

Our result that GSP for range auction has Nash equilibrium equal to the outcome of the VCG mechanism is encouraging because VCG makes truthful behavior the dominant strategy and its outcome has the maximum total valuation or social welfare. However, we may not be able to reach this particular equilibrium. In fact, Abrams, Ghosh and E. Vee [1] have shown that even for the less restrictive position-based auction, there are input values for the v_{ij}'s and c_j's such that for any safe bidding (which means that every advertiser i bids a price b_{ij} for slot j no higher than the value-per-click v_{ij}, i.e., $b_{ij} \le v_{ij}$), GSP cannot return the VCG outcomes for this input. It is natural to ask how bad, in terms of social welfare, a Nash equilibrium of GSP can be. In this section, we prove that, under the assumption that every advertiser i only submits safe bids, i.e., $b_{ij} \le v_{ij}$ for any slot j, any Nash equilibrium of GSP for range auction has social welfare

at least $\frac{1}{2}$ times that of VCG, and we also show that this bound is tight. Note that this bound holds for any Nash equilibrium, not just the envy-free Nash equilibrium. To be precise, we give the definition of (general) Nash equilibrium here: Consider an allocation and pricing given by GSP for some bidding b. We say that this allocation and pricing is at Nash equilibrium if for each advertiser x, if he changes his bid, and the bids of all the other advertisers remain the same, then the new profit of x will not increase. In other words, suppose that advertiser x is originally assigned to slot j and charged a price of p, and is reassigned to slot j' and charged a new price p' after only x changes his bid, then

$$c_j v_{xj} - p \geq c_{j'} v_{xj'} - p'. \tag{4}$$

To derive the $\frac{1}{2}$ bound, we consider a Nash equilibrium given by GSP for some bidding b in the rest of this section. We assume that b is safe, i.e., $b_{ij} \leq v_{ij}$ for any advertiser i and slot j. Suppose that for any slot j, advertiser π_j is assigned to j by GSP, and charged a price-per-click ρ_j (and hence a price of $p_j = c_j \rho_j$). We have the following lemma.

Lemma 51. *For any slots $1 \leq i, j \leq k$, we have $c_i v_{\pi_i i} + c_j v_{\pi_j j} \geq c_i v_{\pi_j i}$.*

Proof. First, consider the case when $i < j$. Since $i < j$, $b_{\pi_i i} \geq b_{\pi_j i}$ (or else GSP will not assign π_i to slot i). Note that if π_j increases his bid $b_{\pi_j i}$ for slot i to $b_{\pi_i i} + \epsilon$, and the other advertisers make no change to their bids, then π_j will win slot i, and pay the price $b_{\pi_i i}$, the second highest price for slot i after the change. Since the allocation and pricing is at Nash equilibrium, by (4), we have $c_j v_{\pi_j j} - c_j \rho_j \geq c_i v_{\pi_j i} - c_i b_{\pi_i i} \Rightarrow c_j v_{\pi_j j} \geq c_i v_{\pi_j i} - c_i v_{\pi_i i} \Rightarrow c_i v_{\pi_i i} + c_j v_{\pi_j j} \geq c_i v_{\pi_j i}$, where the second inequality holds because $\rho_j \geq 0$ and $b_{\pi_i i} \leq v_{\pi_i i}$ (safe bid).

For the case when $j \leq i$, we have $c_j \geq c_i$ and $v_{\pi_j j} \geq v_{\pi_j i}$, and obviously $c_i v_{\pi_i i} + c_j v_{\pi_j j} \geq c_j v_{\pi_j j} \geq c_i v_{\pi_j i}$.

We are now ready to compare the social welfare of GSP with that of VCG, whose social welfare is optimal. To simplify the analysis, we assume again there are $n-k$ dummy slots $k+1, k+2, \ldots n$ such that the click-through rate of dummy slot j is $c_j = 0$, and $v_{ij} = 0$ for all advertisers i. Furthermore, we assume that in VCG, advertiser i is assigned to slot i. Then, the allocation of VCG is a perfect matching, and its social welfare is $\sum_{1 \leq h \leq n} c_h v_{hh}$. For GSP, we assume that after GSP decides the allocation and pricing for the top k slots, we assign the unassigned advertisers to the unallocated slots arbitrary, and each of them is charged a price of 0. (Note that for any unassigned advertiser i, if he is assigned to unallocated slot j, then its value is $c_j v_{ij} = 0$.) Then, the social welfare of GSP becomes $\sum_{1 \leq h \leq n} c_h v_{\pi_h h}$. It is easy to verify that Lemma 51 is still true if either i or j is dummy slot.

Theorem 52. *The social welfare of GSP is at least half that of VCG, i.e., $\sum_{1 \leq h \leq n} c_h v_{\pi_h h} \geq \frac{1}{2} \sum_{1 \leq h \leq n} c_h v_{hh}$.*

Proof. By Lemma 51, we have, for any $1 \leq i, j \leq n$, $c_i v_{\pi_i i} + c_j v_{\pi_j j} \geq c_i v_{\pi_j i}$. Consider any slot h. After substituting $j = h$, $i = \pi_h$ in the inequality, we have $c_{\pi_h} \cdot$

$v_{\pi_{\pi_h}\pi_h} + c_h \cdot v_{\pi_h h} \geq c_{\pi_h} \cdot v_{\pi_h \pi_h}$. It follows that $\sum_{1 \leq h \leq n} \left(c_{\pi_h} \cdot v_{\pi_{\pi_h}\pi_h} + c_h \cdot v_{\pi_h h} \right) \geq \sum_{1 \leq h \leq n} c_{\pi_h} \cdot v_{\pi_h \pi_h} \Leftrightarrow \sum_{1 \leq t \leq n} c_t \cdot v_{\pi_t t} + \sum_{1 \leq h \leq n} c_h \cdot v_{\pi_h h} \geq \sum_{1 \leq t \leq n} c_t \cdot v_{tt} \Leftrightarrow \sum_{1 \leq h \leq n} c_h \cdot v_{\pi_h h} \geq \frac{1}{2} \sum_{1 \leq h \leq n} c_h \cdot v_{hh}$, the theorem follows.

We can show that the bound $\frac{1}{2}$ is tight by an example. Assume that there are 2 slots and 2 advertisers. $v_{11} = 100 + \epsilon$, $v_{12} = 100 + \epsilon$, $v_{21} = 100$, $v_{22} = 0$; $c_1 = 1 + \sigma$, $c_2 = 1$. $(0 < \epsilon < 1, 0 < \sigma < 1)$. The bidding b is: $b_{11} = 100 + \epsilon$, $b_{12} = 100 + \epsilon$, $b_{21} = 0$, $b_{22} = 0$. The allocation and pricing given by GSP for b is at Nash equilibrium. Advertiser 1 is assigned to slot 1 and advertiser 2 to slot 2. Each of them is charged a price of 0. Advertiser 1 has no incentive to lower his bid and advertiser 2 cannot win slot 1 by changing his bid. The social welfare of GSP is $v(b) = c_1 \cdot v_{11} + c_2 \cdot v_{22} = (1 + \sigma) \cdot (100 + \epsilon) + 0$. In VCG, advertiser 2 is assigned to slot 1 and advertiser 1 to slot 2; its social welfare is $v(\Theta) = c_1 \cdot v_{21} + c_2 \cdot v_{12} = (1 + \sigma) \cdot 100 + 1 \cdot (100 + \epsilon)$. Therefore, $\frac{v(b)}{v(\Theta)} = \frac{(1+\sigma)(100+\epsilon)}{(1+\sigma)\cdot 100 + 1 \cdot (100+\epsilon)} = \frac{1}{2} + \frac{100\sigma + \epsilon + 2\sigma\epsilon}{2(200+100\sigma+\epsilon)}$.

References

1. Abrams, Z., Ghosh, A., Vee, E.: Cost of conciseness in sponsored search auctions. Internet and Network Economics, 326–334 (2007)
2. Aggarwal, G., Feldman, J., Muthukrishnan, S.: Bidding to the top: VCG and equilibria of position-based auctions. Approximation and Online Algorithms, 15–28 (2007)
3. Aggarwal, G., Feldman, J., Muthukrishnan, S., Pál, M.: Sponsored search auctions with markovian users. Internet and Network Economics, 621–628 (2008)
4. Aggarwal, G., Goel, A., Motwani, R.: Truthful auctions for pricing search keywords. In: Proceedings of the 7th ACM Conference on Electronic Commerce, pp. 1–7. ACM (2006)
5. Blumrosen, L., Hartline, J., Nong, S.: Position auctions and non-uniform conversion rates. In: ACM EC Workshop on Advertisement Auctions (2008)
6. Caragiannis, I., Kanellopoulos, P., Kaklamanis, C., Kyropoulou, M.: On the efficiency of equilibria in generalized second price auctions. In: Proceedings of the 12th ACM Conference on Electronic Commerce, pp. 81–90. ACM (2011)
7. Clarke, E.H.: Multipart pricing of public goods. Public Choice 11(1), 17–33 (1971)
8. Even-Dar, E., Feldman, J., Mansour, Y., Muthukrishnan, S.: Position auctions with bidder-specific minimum prices. Internet and Network Economics, 577–584 (2008)
9. Leme, R.P., Tardos, E.: Pure and bayes-nash price of anarchy for generalized second price auction. In: 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, pp. 735–744. IEEE (2010)
10. Ostrovsky, M., Schwarz, M., Edelman, B.G.: Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. NBER Working Paper (2005)
11. Varian, H.R.: Position auctions. International Journal of Industrial Organization 25(6), 1163–1178 (2007)
12. Vickrey, W.: Counterspeculation, auctions, and competitive sealed tenders. The Journal of Finance 16(1), 8–37 (1961)

Stretch in Bottleneck Games

Costas Busch and Rajgopal Kannan

Louisiana State University, Baton Rouge, LA 70803, USA
{busch,rkannan}@csc.lsu.edu

Abstract. In bottleneck congestion games the social cost is the worst congestion (bottleneck) on any resource, and each player selfishly minimizes the worst resource congestion in its strategy. We examine the price of anarchy with respect to the *stretch* which is a measure of variation in the resource utilization in the strategy sets of the players. The stretch is particularly important in routing problems since it compares the chosen path lengths with the respective shortest path lengths. We show that the price of anarchy in general bottleneck games is bounded by $O(sm)$, where s is the stretch and m is the total number of resources. In linear bottleneck games, where the resource latencies are linearly proportional to the players' workloads, the price of anarchy is improved to $O(\sqrt{sm})$. These bounds are asymptotically tight. For constant stretch linear games we obtain a $\Theta(\sqrt{m})$ improvement over the previously best known bound.

Keywords: algorithmic game theory, congestion games, bottleneck games, price of anarchy.

1 Introduction

We consider non-cooperative atomic congestion games with n players, where each player's *pure strategy* consists of a subset from a set of shared resources R. Access to the common resources causes congestion which is expressed with a *delay function* on each resource. We consider the utility cost of a player to be the largest congestion (bottleneck) experienced in any of the resources in its chosen strategy. Players prefer smaller bottleneck congestion. The social cost, denoted C, is the largest bottleneck congestion that appears in any resource. *Bottleneck congestion games* capture the effect that the bottleneck player utility costs have on the bottleneck social cost.

Bottleneck congestion games have been studied in the literature primarily in the context of routing games [1,2,3] where resources correspond to network links (or nodes) and player strategies are paths over the links. In [1] the authors observe that bottleneck routing games are important in networks for various practical reasons. In wireless networks, the maximum congested link affects the lifetime of the network since adjacent nodes transmit large number of packets resulting to fast energy depletion. High congestion links also induce hot-spots which slow-down the network throughput. This may further expose the network to malicious attacks which aim to increase the congestion in the hope to bring down the network.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 592–603, 2012.

Table 1. Price of anarchy (PoA) bounds for bottleneck games

PoA Upper Bound	Kind of Bottleneck Game	Theorem
sm	general game	Theorem 1
$\sqrt{sm}+1$	uniform linear game ($\lambda = 1$)	Theorem 2
$\sqrt{\frac{8}{3}sm}$	arbitrary linear game (any λ)	Theorem 3
$\sqrt{2sm}+o(\sqrt{sm})+1$	special linear game with $1 < \lambda \leq 2^{o(\sqrt{sm})}$	Theorem 4

Bottleneck games are also important to machine scheduling problems, where each resource corresponds to a machine and a player's strategy is a sequence of jobs that are to be scheduled on respective machines. The seminal result of Leighton *et al.* [10] demonstrates the existence of (coordinated) scheduling algorithms that dispatch the jobs in time very close to $C+D$, where C is the social bottleneck congestion and D is the maximum strategy size. When $C \gg D$, the congestion becomes the dominant factor in the scheduling performance. Thus, smaller bottleneck congestion immediately implies improved task throughput. Same observation also applies to packet scheduling problems.

An important aspect of bottleneck games is the stretch on the players' strategies. The *stretch* expresses the variability on the resource utilization in the strategy sets of the players. For a player the stretch is the ratio of the maximum versus the minimum resource utilization considering all of the player's strategy set. The stretch of the game, denoted s, is simply the maximum stretch of any player. Note that $s \geq 1$. Stretch is an important metric of performance. In networks it expresses how much longer are path lengths compared to shortest paths. Constant stretch paths are typically preferred since routed packets can be delivered very fast under low traffic conditions. In machine scheduling problems low stretch implies smaller total resource utilization which is beneficial for improved job throughput.

1.1 Contributions

We examine the consequence of the players' selfish behavior to the social welfare in pure *Nash equilibria* in which no player can unilaterally improve its situation. The impact of selfishness is quantified with the *price of anarchy* (PoA) [9,13], which expresses how much larger is the worst social cost in a Nash equilibrium compared to the social cost in the optimal coordinated solution. Ideally, the price of anarchy should be small.

We explore the impact of stretch on the price of anarchy in bottleneck games. We summarize our results in Table 1. We first consider *general bottleneck games* where each player can apply arbitrary weights on the resources that it uses. The congestion on a resource is simply the sum of the weights of the players that use the resource. We show that the price of anarchy is sm, where s is the stretch and m is the number of resources ($m = |R|$). We demonstrate that this bound is asymptotically tight by giving a matching worst case lower bound.

We then turn to the class of *linear bottleneck games* [6] where a user applies the same weight on each resource that it uses (different users may have different weights). The congestion of a resource is a linear function on the sum of the weights of the players that use the resource, as expressed by a linear coefficient associated with the resource. Linear games represent many natural problems in network routing and machine scheduling, as for example problems where a player has the same perception of cost for each network link or system machine. We show that the price of anarchy in linear bottleneck games is $\sqrt{\frac{8}{3}sm}$. We also demonstrate that this bound is tight within a constant factor. Therefore, linear bottleneck games have a significant benefit compared to general bottleneck games. In [6] is is shown that in linear bottleneck games the price of anarchy is $O(m)$. For constant stretch $s = O(1)$ we obtain price of anarchy $O(\sqrt{m})$. Thus, we improve significantly the previously known bound for linear games by a factor of $\Theta(\sqrt{m})$, and there is always an asymptotic benefit for $s = o(m)$.

The constant factor of $\sqrt{\frac{8}{3}}$ in the price of anarchy bound can be improved when we consider special cases of linear bottleneck games. Let λ denote the ratio of the largest linear coefficient versus the smallest linear coefficient on any resource. In *uniform* games, where $\lambda = 1$, the price of anarchy bound is $\sqrt{sm} + 1$. This is within an additive constant of 1 away from a worst case lower bound. When $1 < \lambda \leq 2^{o(\sqrt{sm})}$, which allows for a wide range of different values in the linear coefficients of the resources, the price of anarchy becomes $\sqrt{2sm} + o(\sqrt{sm}) + 1$.

Our analysis relies on the observation that in a Nash equilibrium the average congestion of a set of resources Q can be related with the optimal congestion and the stretch s. In a uniform linear game, if a player uses x resources in a Nash equilibrium, then it is guaranteed to use at least x/s resources in the optimal solution. These resources are either in Q or in $Q' = R \setminus Q$. Thus, by taking one of the sets Q or Q' with the largest use of resources in it, we can manage to relate the average and optimal congestion with the sizes of Q and R though the stretch s. We also identify appropriate *support sets* of resources which have high congestion in the equilibrium and are also used in the optimal solution. These observations can be extended to arbitrary linear games to provide the aforementioned price of anarchy bounds.

1.2 Related Work

Congestion games were introduced and studied in [12,14]. Rosenthal [14] proves that congestion games have always pure Nash equilibria. Koutsoupias and Papadimitriou [9] introduced the notion of price of anarchy in the specific *parallel link networks* model in which they provide the bound $PoA = 3/2$. Roughgarden and Tardos [16] provided the first result for splittable flows in general networks in which they showed that $PoA \leq 4/3$ for a player cost which reflects to the sum of congestions of the resources of a path. Pure equilibria with atomic flow have been studied in [3,5,11,18] (our work fits into this category), and with splittable flow in [15,16,17]. Most of the work in the literature uses a player cost

metric related to the aggregate sum of congestions on all the edges of the player's path; and the social cost metric is also an aggregate expression of all the edge congestions [5,15,16,17,18].

Banner and Orda [1] study bottleneck routing games in general networks for splittable and atomic flow, where they show the existence and non-uniqueness of equilibria. They prove that finding the best Nash equilibrium that minimizes the social cost is a NP-hard problem. They also establish that if the resource congestion delay function is bounded by some polynomial with degree $k \geq 1$ then $PoA = O(m^k)$, where m is the number of links in the underlying network graph. Further, they show that the price of anarchy may be unbounded for specific resource congestion functions. Busch and Magdon-Ismail [3] consider atomic bottleneck routing games where they prove that $\ell \leq PoA \leq c(\ell^2 + log^2|V|)$, where ℓ is the size of the largest edge-simple cycle in the network raph and c is a constant. In [3] it is also shown that if $k = 1$ there are game instances with $PoA = \Omega(m)$.

Bottleneck games with the $C + D$ metric are studied in [2]. In [7] a proof for the existence of strong Nash equilibria is given (which concern coalitions of players) for games with the lexicographic improvement property; such games include the bottleneck games that we consider here. Variations of bottleneck games with player coalitions are studied in [6]. In [4] a polynomial time algorithm is presented for computing Nash Equilibria in bottleneck routing games where there is a single sink or source and linear delay functions. In [8], the authors study games with the bottleneck social cost, which achieve low price of anarchy when the players use a cost function which is an aggregate exponential expression of the congestions of the edges in their selected paths.

Outline of Paper: In Section 2 we give basic definitions. In Section 3 we present the price of anarchy result for the general bottleneck games and also prove the worst case lower bound. In Section 4 we present the price of anarchy bounds for uniform linear bottleneck games. In Section 5 we present the price of anarchy bounds for arbitrary linear bottleneck games.

2 Basic Definitions

A (general) *bottleneck congestion game* with n players and m resources is a strategic game $G = (\Pi, R, \mathcal{W}, \mathbb{S})$ with: Players $\Pi = \{\pi_1, \ldots, \pi_n\}$; Resources $R = \{r_1, \ldots, r_m\}$; Weights $\mathcal{W} = \{w_{\pi,r} : \pi \in \Pi \wedge r \in R\}$, where each $w_{\pi,r} \geq 0$ is a weight for player π on resource r; Strategy space $\mathbb{S} = \mathbb{S}_{\pi_1} \times \cdots \times \mathbb{S}_{\pi_n}$, where $\mathbb{S}_\pi \subseteq 2^R$ is a (pure) strategy set for player π.

A *game state* (*strategy profile*) is an n-tuple $S = (S_{\pi_1}, \ldots, S_{\pi_n}) \in \mathbb{S}$, where each strategy S_{π_i} is a set of resources. Each resource $r \in R$ has a *delay cost* $C_r(S)$ equal to the sum of weights of the players that use the resource r: $C_r(S) = \sum_{(\pi \in \Pi) \wedge (r \in S_\pi)} w_{\pi,r}$. In any game state $S \in \mathbb{S}$, each player $\pi \in \Pi_G$ has a *player (utility) cost* $pc_\pi(S) = \max_{r \in S_\pi} C_r(S)$ (player bottleneck congestion).

For any state S, we use the standard notation $S = (S_{\pi_i}, S_{-\pi_i})$ to emphasize the dependence on player π_i. Player π_i is *locally optimal* (or *stable*) in state S

if $pc_{\pi_i}(S) \leq pc_{\pi_i}((S'_{\pi_i}, S_{-\pi_i},))$ for all strategies $S'_{\pi_i} \in \$_{\pi_i}$. A greedy move by a player π_i is any change of its strategy from S'_{π_i} to S_{π_i} which improves the player's cost, that is, $pc_{\pi_i}((S_{\pi_i}, S_{-\pi_i})) < pc_{\pi_i}((S'_{\pi_i}, S_{-\pi_i}))$. A state S is in a *Nash Equilibrium* if every player is locally optimal; in other words, no greedy move is available to any player. Nash Equilibria quantify the notion of a stable selfish outcome. In [7] it is shown that bottleneck congestion games have always at least one Nash equilibrium.

For any game G and state S, we will consider a *social cost* (or *global cost*) function, denoted $C(S)$, which measures the state's impact to the global welfare. In bottleneck congestion games $C(S) = \max_{r \in R} C_r(S)$. A state S^* is called *optimal* if it has minimum attainable social cost: for any other state S, $C(S^*) \leq C(S)$. We quantify the quality of the states which are Nash Equilibria with the *price of anarchy* (PoA) (sometimes referred to as the coordination ratio). Let $\mathcal{P}$ denote the set of distinct Nash Equilibria. Then, the price of anarchy of game G is:

$$PoA = \sup_{S \in \mathcal{P}} \frac{C(S)}{C(S^*)}.$$

We continue to define the stretch of a game. Given a bottleneck congestion game, for any set of resources Q and player π, let $\sigma_\pi(Q) = \sum_{r \in Q} w_{\pi,r}$ (and for the linear games that we define below $\sigma_\pi(Q) = \sum_{r \in Q} a_r w_\pi$) denote the resource utilization for player π on set Q. Let $\sigma_\pi^{\min} = \min_{S \in \$_\pi} \sigma_\pi(S)$ and $\sigma_\pi^{\max} = \max_{S \in \$_\pi} \sigma_\pi(S)$, denote the respective minimum and maximum resource utilization for player π among all available strategies to π. For a player π we define its *stretch* s_π as: $s_\pi = \sigma_\pi^{\max}/\sigma_\pi^{\min}$. The *stretch* s of the game is simply the maximum stretch of the players: $s = \max_{\pi \in \Pi} s_\pi$.

We will examine the special case of *linear bottleneck games* which have the following two restrictions:

- *Single player weight:* Each player $\pi \in \Pi$ applies the same weight $w_\pi > 0$ on each resource that it uses (different players may have different weights).
- *Linear coefficients:* Each resource $r \in R$ has a coefficient $a_r > 0$ so that its delay cost is evaluated as: $C_r(S) = a_r \sum_{(\pi \in \Pi) \wedge (r \in S_\pi)} w_\pi$.

Note that linear bottleneck games are special cases of general bottleneck games, where each player's resource weight is represented with a fixed player weight multiplied by the resource's coefficient. For a linear bottleneck game, we denote $a_{\min} = \min_{r \in R} a_r$ and $a_{\max} = \max_{r \in R} a_r$. We define the *coefficient ratio* as $\lambda = a_{\max}/a_{\min}$. The special case of *uniform* linear games is when $\lambda = 1$, where all there resources have the same coefficient.

3 General Bottleneck Games

Let G be a general bottleneck game. Consider a Nash equilibrium $S = (S_{\pi_1}, \ldots, S_{\pi_n})$ which has the maximum possible congestion among all Nash equilibria in G. Let $S^* = (S^*_{\pi_1}, \ldots, S^*_{\pi_n})$ denote an optimal state. Denote $C(S) = C$ and $C(S^*) = C^*$.

Theorem 1 (*PoA* for General Bottleneck Games). *Any general bottleneck game with m resources and stretch s has $PoA \leq sm$.*

Proof. Consider a resource $r' \in R$ with $C_{r'} = C$. Let X denote the set of players that use r' in state S. Since the stretch factor is s, we have that for each player $\pi \in \Pi$, $\sum_{r \in S_\pi} w_{\pi,r} \leq s \sum_{r \in S^*_\pi} w_{\pi,r}$. We denote by $W = \sum_{\pi \in X} \sum_{r \in S^*_\pi} w_{\pi,r}$ the total utilization of the resources in R by the optimal strategies of the players X. We have:

$$C = \sum_{\pi \in X} w_{\pi,r'} \leq \sum_{\pi \in X} \sum_{r \in S_\pi} w_{\pi,r} \leq \sum_{\pi \in X} s \sum_{r \in S^*_\pi} w_{\pi,r} = s \sum_{\pi \in X} \sum_{r \in S^*_\pi} w_{\pi,r} = sW.$$

We also have that in state S^* at least one resource experiences congestion at least W/R: $C^* \geq W/|R| = Wm$. Consequently, $PoA \leq C/C^* \leq sm$.

In order to demonstrate the asymptotic tightness of the result in Theorem 1, we give a game instance with price of anarchy $\Omega(sm)$, for any $s \geq 1$. This game uses $m > 0$ resources (for simplicity assume that m is a multiple of 2), and involves two players π_1 and π_2. There are two disjoint sets of resources $R = R_1 \cup R_2$, each of size $m/2$. There are two special resources $r_1 \in R_1$ and $r_2 \in R_2$. Player π_1 has two strategies, strategy $S^1_{\pi_1}$ which contains all resources in R_1 with $w^r_{\pi_1} = 1$ for each $r \in R_1$, and strategy $S^2_{\pi_1}$ which contains only $r_2 \in R_2$ with $w^{r_2}_{\pi_1} = sm/2$. Symmetrically, player π_2 has two strategies, strategy $S^1_{\pi_2}$ which contains all resources in R_2 with $w^r_{\pi_2} = 1$ for each $r \in R_2$, and strategy $S^2_{\pi_2}$ which contains only $r_1 \in R_1$ with $w^{r_1}_{\pi_2} = sm/2$.

Note that the stretch for each player is s. The optimal state is $S^* = (S^1_{\pi_1}, S^1_{\pi_2})$ with $C^* = 1$. The state $S = (S^1_{\pi_1}, S^1_{\pi_2})$ is a Nash equilibrium with $C = sm/2$. (The example can be adjusted appropriately for arbitrary values of m which may not be powers of 2). Therefore, we have the following lemma:

Lemma 1 (Lower Bound for General Bottleneck Games). *For any $m \geq 1$ and $s \geq 1$, there is a general bottleneck game instance with m resources and stretch s, such that $PoA = \Omega(sm)$.*

4 Uniform Linear Bottleneck Games

We continue to examine the price of anarchy in uniform linear bottleneck games, where $\lambda = 1$. We start with defining the notion of a support set which identifies a set of resources used in the optimal state but with high utility costs in the Nash equilibrium. We express the price of anarchy with respect to the size of a support set. We also show that the average congestion in a resource set is related to its size and the game stretch. Combining these observations we obtain a theorem that relates the price of anarchy with the stretch.

Consider for now G to be an arbitrary linear bottleneck game (with $\lambda \geq 1$). Let $S = (S_{\pi_1}, \ldots, S_{\pi_n})$ be a Nash equilibrium which has the maximum congestion among all Nash equilibria in G. Let $S^* = (S^*_{\pi_1}, \ldots, S^*_{\pi_n})$ denote an optimal state. Denote $C(S) = C$ and $C(S^*) = C^*$. We denote with $\overline{C}(Q) = \sum_{r \in Q} C_r(S)/|Q|$ the average congestion of a set of resources $Q \subseteq R$ in state S. Denote $\breve{C}(S) = \min_{r \in Q} C_r(S)$ the smallest congestion of any resource in Q and state S.

4.1 Support Sets

For any resource r let $X(r)$ denote the set of players that use r in state S. Let $\pi \in X(r)$. The *support of player* π *for resource* r in state S, denoted $F(r,\pi)$, contains all resources $r' \in R$ such that: (i) r' belongs to the optimal strategy of π, that is, $r' \in S^*_\pi$, and (ii) π would experience higher cost if it switched to its optimal strategy because of r', that is, $C_{r'} + a_{r'} w_\pi \geq C_r$. We define the *support of resource* r in state S as the union of the respective supports of the players that use r: $F(r) = \bigcup_{\pi \in X(r)} F(r,\pi)$. The *support for a set of resources* $Q \subseteq R$ is defined as $F(Q) = \bigcup_{r \in Q} F(r)$.

Lemma 2. *For any set of resources* $Q \in R$, $\breve{C}(F(Q)) \geq \breve{C}(Q) - C^*$.

Proof. From the definition of support $F(r,\pi)$, for each resource $r' \in F(r,\pi)$ there is a player $\pi \in X(r)$ such that r' belongs to the optimal strategy of π, $r' \in S^*_\pi$. Consequently, $a_{r'} w_\pi \leq C^*$. Moreover, $C_{r'} + a_{r'} w_\pi \geq C_r$, which gives $C_{r'} \geq C_r - a_{r'} w_\pi \geq C_r - C^*$. Now considering the union of all the support sets of all the players that use r, we obtain $\breve{C}(F(r)) \geq C_r - C^*$. Therefore,

$$\breve{C}(F(Q)) \geq \min_{r \in Q} \breve{C}(F(r)) \geq \min_{r \in Q}(C_r - C^*) = \breve{C}(Q) - C^*.$$

Lemma 3. *For any resource* r *with* $C_r = C$, $PoA \leq \lambda |F(r)|$.

Proof. From the way we selected r we have $C = C_r = \sum_{\pi \in X(r)} a_r w_\pi$. From the definition of $F(r)$, every player $\pi \in X(r)$ has at least one resource $r_\pi \in S^*_\pi \cap F(r)$. Since $a_{r_\pi} \geq a_r/\lambda$, we obtain:

$$C^* \geq \frac{\sum_{\pi \in X(r)} a_{r_\pi} w_\pi}{|F(r)|} \geq \frac{1}{|F(r)|} \sum_{\pi \in X(r)} \frac{a_r w_\pi}{\lambda} = \frac{1}{\lambda |F(r)|} \sum_{\pi \in X(r)} a_r w_\pi = \frac{C}{\lambda |F(r)|}.$$

Thus, $PoA = C/C^* \leq \lambda |F(r)|$.

4.2 Price of Anarchy and Stretch

In the next result we take an arbitrary set of resources Q and we bound the optimal congestion with respect to the average congestion in Q, the size of Q, the stretch s, and the total number of resources m in the game. This result is crucial for later relating the price of anarchy with the stretch.

Lemma 4. *For any linear bottleneck game and any set of resources* $Q \subseteq R$, $C^* \geq \overline{C}(Q)|Q|/(sm)$.

Proof. Let Y denote the set of players that use resources in Q. Let $Z = \sum_{r \in Q} C_r$ be the total cost contributed by the players in Y in the edges Q in state S. From the definition of $\overline{C}(Q)$, we have $Z = \overline{C}(Q)|Q|$.

For any set of resources $T \subseteq R$ we denote with $W(T) = \sum_{\pi \in Y} \sum_{r \in S^*_\pi \cap T} a_r w_\pi$ the total cost contributed in T by the players in Y when they use their optimal strategies. Clearly, $W(R) = W(Q) + W(Q')$, where $Q' = R \setminus Q$.

Since the stretch factor is s, we have that for each player $\pi \in \Pi$, $\sum_{r\in S_\pi} a_r w_\pi \le s \sum_{r\in S^*_\pi} a_r w_\pi$. Therefore,

$$\begin{aligned} Z &= \sum_{\pi\in Y} \sum_{r\in S_\pi \cap Q} a_r w_\pi \le \sum_{\pi\in Y} \sum_{r\in S_\pi} a_r w_\pi \le \sum_{\pi\in Y} s \sum_{r\in S^*_\pi} a_r w_\pi \\ &= s \sum_{\pi\in Y} \sum_{r\in S^*_\pi \cap R} a_r w_\pi = sW(R). \end{aligned}$$

Let $W(Q') = (1-\epsilon)W(R)$, where $0 \le \epsilon \le 1$. For $\epsilon < 1$ we have that $|Q'| > 0$ and we obtain:

$$C^* \ge \frac{W(Q')}{|Q'|} = \frac{(1-\epsilon)W(R)}{|Q'|} \ge \frac{(1-\epsilon)Z}{s|Q'|} = \frac{(1-\epsilon)\overline{C}(Q)|Q|}{s|Q'|}. \tag{1}$$

For $\epsilon > 0$ we have that $|Q| > 0$ and we obtain (recall that $s \ge 1$):

$$C^* \ge \frac{W(Q)}{|Q|} = \frac{\epsilon W(R)}{|Q|} \ge \frac{\epsilon Z}{s|Q|} = \frac{\epsilon\overline{C}(Q)|Q|}{s|Q|} = \frac{\epsilon\overline{C}(Q)}{s}. \tag{2}$$

Consequently, from Equations 1 and 2 we obtain for $0 < \epsilon < 1$:

$$C^* \ge \frac{\overline{C}(Q)}{s} \cdot \max\left\{\frac{(1-\epsilon)|Q|}{|Q'|}, \epsilon\right\} \tag{3}$$

A lower bound in Equation 3 is obtained when we set $(1-\epsilon)|Q|/|Q'| = \epsilon$, which gives $\epsilon = |Q|/(|Q'| + |Q|) = |Q|/|R| = |Q|/m$. Therefore, $C^* \ge \overline{C}(Q)|Q|/(sm)$. This bound holds also in the boundary values $\epsilon = 0$ or $\epsilon = 1$: if $\epsilon = 0$ then Equation 1 gives $C^* \ge \overline{C}(Q)|Q|/(s|Q'|) \ge \overline{C}(Q)|Q|/(sm)$; if $\epsilon = 1$ then Equation 2 gives $C^* \ge \overline{C}(Q)/s \ge \overline{C}(Q)|Q|/(sm)$.

Lemma 5. *$PoA \le \sqrt{\lambda sm} + 1$.*

Proof. Consider resource r with $C_r = C$. Let $A = F(r)$. From Lemma 3, $PoA \le \lambda|A|$. Further, from Lemma 6, $\overline{C}(A) \ge \breve{C}(A) \ge C_r - C^* = C - C^*$.

From Lemma 4, we obtain $C^* \ge \overline{C}(A)|A|/(sm) \ge (C - C^*)|A|/(sm)$. Thus, $PoA = C/C^* \le 1 + sm/|A|$.

When we combine the above price of anarchy bounds we obtain for $q = |A|$:

$$PoA \le \min\left\{\lambda q, 1 + \frac{sm}{q}\right\} \le 1 + \min\left\{\lambda q, \frac{sm}{q}\right\}.$$

An upper bound on this expression is obtained when $\lambda q = sm/q$ which gives $q = \sqrt{sm/\lambda}$. Therefore, $PoA \le \sqrt{\lambda sm} + 1$.

From Lemma 3, since $|F(r)| \le m$, we have that $PoA \le \lambda m$. By setting $\lambda = 1$, we obtain from Lemma 5:

Theorem 2 (*PoA* for Uniform Linear Bottleneck Games). *For any uniform linear bottleneck game with $\lambda = 1$, $PoA \le \min(\sqrt{sm} + 1, m)$.*

5 Arbitrary Linear Bottleneck Games

We continue to provide a bound to the price of anarchy in arbitrary linear bottleneck games that does not depend on the value of λ. We will start with defining the notion of an extensive support set which is a repetitive expansion of support sets. The extensive support set helps to express the average congestion and the price of anarchy with respect to the size of a set of resources without depending on λ. Then, using Lemma 4 we obtain the main result.

5.1 Extensive Support Sets

We now define the *extensive support* $H^k(Q)$ for set of resources Q and for any integer $k \geq 0$, so that $H^0(Q) = Q$ and for $k \geq 1$, $H^k(r) = H^{k-1}(Q) \cup F(H^{k-1}(Q))$.

Lemma 6. *For any set of resources $Q \subseteq R$ and integer $k \geq 0$, $\breve{C}(H^k(Q)) \geq \breve{C}(Q) - kC^*$.*

Proof. We will prove the main claim by induction on k. Since $\breve{C}(H^0(Q)) = \breve{C}(Q)$, the claim holds for $k = 0$. Suppose that the claim holds up to $k > 0$. We will show it holds for $k + 1$. We have that

$$\breve{C}(H^{k+1}(Q)) = \breve{C}(H^k(Q) \cup F(H^k(Q))) = \min\{\breve{C}(H^k(Q)), \breve{C}(F(H^k(Q)))\}.$$

From induction hypothesis, $\breve{C}(H^k(Q)) \geq \breve{C}(Q) - kC^*$. From Lemma 2,

$$\breve{C}(F(H^k(Q))) \geq \breve{C}(H^k(Q)) - C^* \geq \breve{C}(Q) - (k+1)C^*.$$

Consequently,

$$\breve{C}(H^{k+1}(Q)) \geq \min\{\breve{C}(Q) - kC^*, \breve{C}(Q) - (k+1)C^*\} = \breve{C}(Q) - (k+1)C^*.$$

Let $T = \{r\}$ with $C_r = C$. Denote $T^k = H^k(T)$. Suppose for convenience that $T^{-1} = \emptyset$. Let $\beta \geq 1$ be the smallest integer such that $\breve{a}(T^\beta) = \breve{a}(T^{\beta-1})$ (we denote with $\breve{a}(Q) = \min_{r \in Q} a_r$ the smallest coefficient in any set of resources Q). Observe that for all $0 \leq k \leq \beta - 1$, $|T^k| \subset |T^{k+1}|$ and hence, $|T^k| < |T^{k+1}|$, which implies $|T^k| \geq k+1$. Further $|T^\beta| \geq |T^{\beta-1}| \geq \beta$. There is a set of β distinct resources $A = \{r_{i_0}, \ldots, r_{i_{\beta-1}}\}$, such that $r_{i_j} \in T^j \setminus T^{j-1}$, with $a_{r_{i_j}} = \breve{a}(T^j)$ for all $0 \leq j \leq \beta - 1$. Let $A_k \subseteq A$, where $0 \leq k \leq \beta - 1$, denote the subset with the first k resources: $A_k = \{r_{i_0}, \ldots, r_{i_k}\}$ (note $T = A_0$ and $A = A_{\beta-1}$).

Lemma 7. $\overline{C}(A_k) \geq C - \frac{k}{2}C^*$ *for* $0 \leq k \leq \beta - 1$

Proof. From Lemma 6, $\breve{C}(T^j) \geq \breve{C}(T) - jC^* = C - jC^*$. Therefore, $C_{r_{i_j}} \geq C - jC^*$ and we get:

$$\overline{C}(A_k) = \frac{1}{k+1} \cdot \sum_{j=0}^{k} C_{r_{i_j}} \geq \frac{1}{k+1} \cdot \sum_{j=0}^{k} (C - jC^*) = C - \frac{C^*}{k+1} \cdot \sum_{j=0}^{k} j = C - \frac{k}{2}C^*.$$

Lemma 8. *$PoA \leq |T^\beta| + \beta - 1$.*

Proof. From Lemma 6, we have that $\breve{C}(T^{\beta-1}) \geq \breve{C}(T) - (\beta-1)C^* = C - (\beta - 1)C^*$. Thus, $C_{r_{i_{\beta-1}}} \geq C - (\beta-1)C^*$. Further, $C_{r_{i_{\beta-1}}} = \sum_{\pi \in X(r_{i_{\beta-1}})} a_{r_{i_{\beta-1}}} w_\pi$. From the definition of T^β every player $\pi \in X(r_{i_{\beta-1}})$ has at least one resource in $S^*_\pi \cap T^\beta$. Consequently,

$$C^* \geq \sum_{\pi \in X(r_{i_{\beta-1}})} \frac{\breve{a}(T^\beta) w_\pi}{|T^\beta|} = \frac{1}{|T^\beta|} \sum_{\pi \in X(r_{i_{\beta-1}})} \breve{a}(T^{\beta-1}) w_\pi$$

$$= \frac{1}{|T^\beta|} \sum_{\pi \in X(r_{i_{\beta-1}})} a_{r_{i_{\beta-1}}} w_\pi = \frac{1}{|T^\beta|} C_{r_{i_{\beta-1}}} \geq \frac{1}{|T^\beta|}(C - (\beta-1)C^*).$$

Thus, $PoA = C/C^* \leq |T^\beta| + \beta - 1$.

The following result can be proved using Lemmas 6, 7, and 8.

Lemma 9. *There is a set of resources $Q \subseteq R$ and constants $0 < \delta_1, \delta_2 \leq 2$, with $\delta_1/\delta_2 \leq 8/3$, such that $PoA \leq \delta_1|Q|$ and $\overline{C}(Q) \geq \delta_2 C$.*

Since $PoA \leq \delta_1|Q| \leq \delta_1 m \leq 2m$, Lemma 9 gives:

Corollary 1. *$PoA \leq 2m$.*

Corollary 1 is useful for the cases where s exceeds m, and it provides an alternative upper bound on the price of anarchy. A similar bound as in Corollary 1 was also proven in [6] using different techniques.

5.2 Price of Anarchy for Arbitrary Linear Games

Next is a central result for the price of anarchy in linear bottleneck games.

Lemma 10. *$PoA \leq \sqrt{\frac{8}{3}sm}$.*

Proof. From Lemma 9, there is a set of resources Q and constants $0 < \delta_1, \delta_2 \leq 2$, with $\delta_1/\delta_2 \leq 8/3$, such that $PoA \leq \delta_1|Q|$ and $\overline{C}(Q) \geq \delta_2 C$.

When we apply Lemma 4 with Q we obtain: $C^* \geq \overline{C}(Q)|Q|/(sm) \geq \delta_2 C|Q|/(sm)$, which gives $PoA \leq sm/(\delta_2|Q|)$.

When we combine the above two price of anarchy bounds and for $q = |Q|$ we get:

$$PoA \leq \min\left\{\delta_1 q, \frac{sm}{\delta_2 q}\right\}.$$

An upper bound on this expression is obtained when $\delta_1 q = sm/(\delta_2 q)$ which gives $q = \sqrt{\frac{sm}{\delta_1\delta_2}}$. Therefore,

$$PoA \leq \delta_1\sqrt{\frac{sm}{\delta_1\delta_2}} = \sqrt{\frac{\delta_1 sm}{\delta_2}} \leq \sqrt{\frac{8}{3}sm}.$$

By combining Corollary 1 and Lemma 10, we obtain:

Theorem 3 (*PoA* for Arbitrary Linear Bottleneck Games). *For any linear bottleneck game,* $PoA \leq \min\left(\sqrt{\frac{8}{3}sm}, 2m\right)$.

Special Case: There is a special case of linear games for which we can improve by a constant factor the price of anarchy given in Theorem 3.

Theorem 4. *For* $\lambda > 1$ *and any* $\epsilon > 0$, $PoA \leq \sqrt{(1+\epsilon)sm} + \log_{1+\epsilon}\lambda + 1$.

Theorem 4 allows for a parametrization on λ and a constant $\epsilon > 0$. In Table 1 we demonstrate the case where $1 < \lambda \leq 2^{o(\sqrt{sm})}$ and $\epsilon = 1$, which gives the factor $\sqrt{2}$ in front of the term $\sqrt{sm}$ in the price of anarchy bound.

5.3 Lower Bound for Linear Games

We describe a linear game instance G with price of anarchy $\Omega(\sqrt{sm})$ which demonstrates the asymptotic tightness of Theorem 3. This is a uniform game instance where each resource has the same coefficient, that is, $\lambda = 1$. In particular we take $a_r = 1$ for each $r \in R$. The game involves n players $\Pi = \{\pi_1, \ldots, \pi_n\}$, where each player $\pi \in \Pi$ has the same weight $w_\pi = 1$. Under this model the maximum value of stretch s is bounded by the number of resources m.

The set of resources is partitioned into disjoint sets as $R = Z \cup Q_1 \cup \cdots \cup Q_n$. Set Z consists of n resources, $Z = \{r_1, \ldots, r_n\}$, and each Q_i consists of $\lfloor n/s \rfloor - 1$ resources. Note that for $s = n$ each Q_i is empty. The total number of resources is $m = n + n(\lfloor n/s \rfloor - 1) = \Theta(n^2/s)$.

Each player $\pi_i \in \Pi$ has two strategies, strategy $S^1_{\pi_i}$ which consists of resources $Q_i \cup \{r_i\}$, and strategy $S^2_{\pi_i}$ which consists of all resources in Z. Thus, the stretch is at least $n/\lfloor n/s \rfloor = \Omega(s)$.

The optimal strategy is $S^* = (S^1_{\pi_1}, \ldots, S^1_{\pi_n})$ with $C^* = 1$. The state $S = (S^2_{\pi_1}, \ldots, S^2_{\pi_n})$ is a Nash equilibrium with $C = n$. Thus, the price of anarchy is $PoA = n = \Omega(\sqrt{sm})$. Therefore, we have.

Lemma 11 (Lower Bound for Linear Bottleneck Games). *For any* $m \geq 1$ *and* s, $1 \leq s \leq m$, *there is a linear bottleneck game instance with* m *resources and stretch for each player at least* s *such that* $PoA = \Omega(\sqrt{sm})$.

The price of anarchy in the above example is very tight in boundary values of s. For $s = 1$ the game gives $PoA = \sqrt{m} = \sqrt{sm}$, and for $s = n$ (which implies $n = m$) the game gives $PoA = m = \sqrt{sm}$. Since $\lambda = 1$, these results are within an additive constant of 1 from the bound $PoA \leq \sqrt{sm} + 1$ of Theorem 2.

References

1. Banner, R., Orda, A.: Bottleneck routing games in communication networks. IEEE Journal on Selected Areas in Communications 25(6), 1173–1179 (2007); also appears in INFOCOM 2006

2. Busch, C., Kannan, R., Vasilakos, A.V.: Approximating congestion + dilation in networks via 'quality of routing' games. IEEE Transactions on Computers (2011); also appears in WICON 2008
3. Busch, C., Magdon-Ismail, M.: Atomic routing games on maximum congestion. Theoretical Computer Science 410(36), 3337–3347 (2009); also appears in AAIM 2006
4. Caragiannis, I., Galdi, C., Kaklamanis, C.: Network Load Games. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 809–818. Springer, Heidelberg (2005)
5. 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), Baltimore, MD, USA, pp. 67–73. ACM (May 2005)
6. de Keijzer, B., Schäfer, G., Telelis, O.A.: On the Inefficiency of Equilibria in Linear Bottleneck Congestion Games. In: Kontogiannis, S., Koutsoupias, E., Spirakis, P.G. (eds.) SAGT 2010. LNCS, vol. 6386, pp. 335–346. Springer, Heidelberg (2010)
7. Harks, T., Klimm, M., Möhring, R.H.: Strong Nash Equilibria in Games with the Lexicographical Improvement Property. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 463–470. Springer, Heidelberg (2009)
8. 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)
9. Koutsoupias, E., Papadimitriou, C.: Worst-Case Equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
10. Leighton, F.T., Maggs, B.M., Rao, S.B.: Packet routing and job-scheduling in $O(congestion + dilation)$ steps. Combinatorica 14, 167–186 (1994)
11. Libman, L., Orda, A.: Atomic resource sharing in noncooperative networks. Telecomunication Systems 17(4), 385–409 (2001)
12. Monderer, D., Shapely, L.S.: Potential games. Games and Economic Behavior 14, 124–143 (1996)
13. Papadimitriou, C.: Algorithms, games, and the Internet. In: ACM Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), Hersonissos, Crete, Greece, pp. 749–753 (July 2001)
14. Rosenthal, R.W.: A class of games possesing pure-strategy Nash equilibria. International Journal of Game Theory 2, 65–67 (1973)
15. Roughgarden, T.: Selfish routing with atomic players. In: Proc. 16th Symp. on Discrete Algorithms (SODA), pp. 1184–1185. ACM/SIAM (2005)
16. Roughgarden, T., Tardos, É.: How bad is selfish routing. Journal of the ACM 49(2), 236–259 (2002)
17. Roughgarden, T., Tardos, É.: Bounding the inefficiency of equilibria in nonatomic congestion games. Games and Economic Behavior 47(2), 389–403 (2004)
18. Suri, S., Toth, C.D., Zhou, Y.: Selfish load balancing and atomic congestion games. Algorithmica 47(1), 79–96 (2007)

Author Index