15th Scandinavian Symposium and Workshops on Algorithm Theory

SWAT 2016, June 22–24, 2016, Reykjavik, Iceland

Edited by

Rasmus Pagh

LIPIcs – Vol. 53 – SWAT'16 www.dagstuhl.de/lipics

Editor

Rasmus Pagh
IT University of Copenhagen
Copenhagen, Denmark
pagh@itu.dk

ACM Classification 1998
F. Theory of Computation

ISBN 978-3-95977-011-8

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-011-8.

Publication date
June 2016

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/LIPIcs.SWAT.2016.0

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

http://www.dagstuhl.de/lipics

Contents

15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016).
Editor: Rasmus Pagh
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Session 10: Geometry

Invited contributions

Preface

The 15th Scandinavian Symposium and Workshop and Algorithms Theory (SWAT) received 90 submissions by authors from 31 countries, spanning a broad range of areas within design and analysis of algorithms. The program committee, consisting of 23 members from across the world, worked with 167 subreviewers to review the papers. With the exception of a single paper that was withdrawn during the review process, all papers received at least 3 independent reviews. I would like to thank the program committee and subreviewers for a great effort. For example, the average length of reviews exceeded 8300 characters per paper, and during the subsequent discussion phase, program committee members posted more than 250 additional comments to papers.

In the end we selected 30 papers for inclusion in the conference proceedings and presentation at the conference in Reykjavik, Iceland. The selected papers confirm SWAT's strong reputation within important areas such as graph algorithms and computational geometry, while at the same time spanning algorithms theory broadly with contributions within e.g. data structures, data streaming, string algorithms, algorithmic game theory, and on-line algorithms.

The program committee decided to give the best student paper to Adam Karczmarz for his paper *A Simple Mergeable Dictionary*. In addition to presentations of regular papers, SWAT will feature invited talks by Julia Chuzhoy, Dániel Marx, and Christos Papadimitriou. Accompanying invited contributions can be found in the proceedings.

The present proceedings is the first after SWAT's move to publish in the LIPIcs series. I would like to thank Thomas Dybdahl Ahle at IT University of Copenhagen and Marc Herbstritt at Dagstuhl for their contributions to editing the proceedings.

Rasmus Pagh
Program committee chair for SWAT 2016

15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016).
Editor: Rasmus Pagh
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committee

Christian Sohler	Technische Universität Dortmund
Christian Wulff-Nilsen	University of Copenhagen
Dimitris Fotakis	National Technical University of Athens
Djamal Belazzougui	Research Center on Scientific and Technical Information (CERIST)
Ely Porat	Bar-Ilan University
Fabio Vandin	University of Padova
Faith Ellen	University of Toronto
Francois Le Gall	Kyoto University
Gerhard Woeginger	Technical University of Eindhoven
Gonzalo Navarro	University of Chile
Kasper Green Larsen	Aarhus University
Marek Karpinski	University of Bonn
Marina Papatriantafilou	Chalmers University of Technology
Nodari Sitchinava	University of Hawaii at Manoa
Ola Svensson	École Polytechnique Fédérale de Lausanne
Petteri Kaski	Aalto University
Pinar Heggernes	University of Bergen
Rasmus Pagh	IT University of Copenhagen
Rob van Stee	University of Leicester
Seth Pettie	University of Michigan
Stefan Langerman	Université Libre de Bruxelles
Suresh Venkatasubramanian	University of Utah
Therese Biedl	University of Waterloo

15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016).
Editor: Rasmus Pagh

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Subreviewers

Abbas Bazzi, Adrian Dumitrescu, Alex Zelikovsky, Alexandru Popa, Amer Krivosija, Amer Mouawad, Amihood Amir, Amir Abboud, André van Renssen, Andrew Winslow, Andrzej Lingas, Antonios Antoniadis, Ariel Shiftan, Asaf Levin, Ashkan Norouzi-Fard, Avery Miller, Avinatan Hassidim, Avivit Levy, Babak Farzad, Balazs Patkos, Boaz Patt-Shamir, Brian Dean, Charalampos Stylianopoulos, Charis Papadopoulos, Chris Schwiegelshohn, Christoph Durr, Christoph Gladisch, Christos Kalaitzis, Christos Levcopoulos, Clark Thomborson, Dan Vilenchik, Daniel Lokshtanov, Daniel Valenzuela, Darren Strash, David Eppstein, David Rosenbaum, Deeparnab Chakrabarty, Diego Seco, Eduardo Rivera-Campo, Elias Dahlhaus, Elmar Langetepe, Emanuele Giaquinta, Erik Jan van Leeuwen, Fabrizio Montecchiani, Faisal Abu-Khzam, Falk Hüffner, Francesco Silvestri, Frank Staals, Gábor Braun, Gawiejnowicz Stanislaw, Ge Xia, Gerth Stølting Brodal, Grammateia Kotsialou, Guohui Lin, Guyslain Naves, Hendrik Fichtenberger, Hjalte Wedel Vildhøj, Ian Munro, Ignasi Sau, Ignaz Rutter, Iosif Salem, Irina Kostitsyna, Iyad Kanj, Jakub Tarnawski, Jamie Morgenstern, Jan Kratochvil, Jean-Florent Raymond, Jean-Lou De Carufel, Jesper Nederlof, Joanna Berlinska, Joe Halpern, John Hershberger, Joshimar Cordova, Kanstantsin Pashkovich, Kevin Schewior, Kim Thang Nguyen, Kim-Manuel Klein, Klaus Kriegel, Kyriakos Axiotis, Leah Epstein, Linda Farczadi, Ljubomir Perkovic, Luis Barba, Maarten Löffler, Mamadou Moustapha Kanté, Marc Bury, Marc Renault, Marcin Wrochna, Marco Bressan, Maria Paola Bianchi, Markus Chimani, Markus Sortland Dregi, Martin Fürer, Martin Milanic, Mathias Bæk Tejs Knudsen, Mathias Hauptmann, Matthias Englert, Matthias Mnich, Matthias Westermann, Maximilian Wötzel, Melanie Schmidt, Michał Pilipczuk, Michal Wlodarczyk, Michele Schimd, Michele Zito, Michiel Smid, Mina Ghashami, Mohammad Ali Abam, Morteza Monemizadeh, Moshe Lewenstein, Nick Arnosti, Pablo Pérez-Lantero, Paolo Serafino, Pat Morin, Paul Renaud Goud, Pavlos Eirinakis, Peter Kling, Peyman Afshani, Philipp Kindermann, Rahul Shah, Raphael Clifford, René Van Bevern, Rico Zenklusen, Riko Jacob, Rodrigo Silveira, Rolf Fagerberg, Ross McConnell, Saeed Akhoondian Amiri, Saeed Mehrabi, Saket Saurabh, Samira Daruki, Samuel McCauley, Sander Verdonschot, Sándor Fekete, Sebastian Pokutta, Seffi Naor, Shahin Kamali, Sharma Thankachan, Simon Puglisi, Siwei Yang, Sören Riechers, Spyros Kontogiannis, Stephane Durocher, Subhas Nandy, Sudeshna Kolay, Sunil Arya, Tandy Warnow, Thomas Dueholm Hansen, Thomas Erlebach, Tomasz Kociumaka, Toshihiro Fujito, Travis Gagie, Tsvi Kopelowitz, Uri Zwick, Van Bang Le, Vasileios-Orestis Papadigenopoulos, Vincent Cohen-Addad, Vincent Froese, Wing-Kai Hon, Wolfgang Mulzer, Xian Qiu, Yakov Nekrich, Yiannis Nikolakopoulos, Yixin Cao, Yoichi Iwata, Zachary Friggstad, Zhang Fu.

15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016).
Editor: Rasmus Pagh

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Approximating Connected Facility Location with Lower and Upper Bounds via LP Rounding

Zachary Friggstad[*1], **Mohsen Rezapour**[2], and **Mohammad R. Salavatipour**[†3]

1 **Department of Computing Science, University of Alberta, Edmonton, Canada**
zacharyf@ualberta.ca
2 **Department of Computing Science, University of Alberta, Edmonton, Canada**
rezapour@ualberta.ca
3 **Department of Computing Science, University of Alberta, Edmonton, Canada**
mrs@ualberta.ca

Abstract

We consider a lower- and upper-bounded generalization of the classical facility location problem, where each facility has a capacity (upper bound) that limits the number of clients it can serve and a lower bound on the number of clients it must serve if it is opened. We develop an LP rounding framework that exploits a Voronoi diagram-based clustering approach to derive the first bicriteria constant approximation algorithm for this problem with non-uniform lower bounds and uniform upper bounds. This naturally leads to the the first LP-based approximation algorithm for the lower bounded facility location problem (with non-uniform lower bounds).

We also demonstrate the versatility of our framework by extending this and presenting the first constant approximation algorithm for some connected variant of the problems in which the facilities are required to be connected as well.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Facility Location, Approximation Algorithm, LP Rounding

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.1

1 Introduction

We study the *lower- and upper-bounded facility location* (**LUFL**) problem, a natural generalization of the well-known *capacitated facility location* (**CFL**) and *lower bounded facility location* (**LBFL**) problems. We are given a complete graph $G = (V, E)$, with metric edge lengths $c_e \in \mathbb{Z}_{\geq 0}, e \in E$ containing a set of potential facilities $F \subseteq V$ and a set of demand points (clients) $D \subseteq V$. Each facility $i \in F$ has an opening cost $\mu_i \in \mathbb{Z}_{\geq 0}$ and a capacity (upper bound) $U_i \in \mathbb{Z}_{>0}$, which limits the amount of demand it can serve. Moreover, each facility i has a lower bound $L_i \in \mathbb{Z}_{\geq 0}$ on the amount of demand it must serve if it is opened.

A feasible solution to **LUFL** consists of a set of facilities $I \subseteq F$ to open, and a *valid* assignment $\sigma : D \to I$ of clients to the open facilities: an assignment is valid if it satisfies the lower and upper bounds

$$L_i \leq |\sigma^{-1}(i)| \leq U_i \qquad \forall i \in I. \tag{1}$$

* Supported by NSERC and funding from the Canada Research Chairs program.
† Supported by NSERC.

The goal is to minimize the total cost, i.e., $\sum_{i\in I}\mu_i+\sum_{j\in D}c_{\sigma(j)j}$.

In many real-world applications, particularly in telecommunications, there is an additional requirement to connect the open facilities via high bandwidth *core cables*. This leads to a variant of **LUFL** in which open facilities are connected via a tree-like *core network* that consists of infinite capacity cables. We model this variant as a *connected lower- and upper-bounded facility location* (**C-LUFL**) problem. Let us introduce a parameter $M\geq 1$ which reflects the cost per unit length of core cables. A feasible solution to **C-LUFL** is given by a set of facilities $I\subseteq F$, an assignment $\sigma: D\to I$ of clients to the open facilities that is valid, and a Steiner tree of $T\subseteq E$ connecting all facilities I via core cables. The objective of **C-LUFL** is to minimize the total cost, i.e., $\sum_{i\in I}\mu_i+\sum_{j\in D}c_{\sigma(j)j}+M\sum_{e\in T}c_e$.

Both the **CFL** and **LBFL** problems have been well-studied in the literature. However, there is not much work in studying these problems in a complementary way.[1] To address this gap of knowledge, in this paper, we develop a framework that combines LP rounding techniques for facility location problems with a Voronoi diagram-based clustering approach in order to obtain the first (biceriteria) approximation algorithms for several variants of the problems.

▶ **Definition 1.** An (ρ,α,β)-approximation algorithm for LUFL (C-LUFL, resp.) is an algorithm that computes in polynomial time a solution (I,σ) satisfying $\lfloor L_i/\alpha\rfloor\leq|\sigma^{-1}(i)|\leq\lceil\beta U_i\rceil$, $\forall i\in I$, with cost at most $\rho\cdot OPT$, where OPT denotes the minimum cost of a solution to LUFL (C-LUFL, resp.) satisfying (1).

We often loosely refer to a (ρ,α,β)-approximation for **LUFL** or **C-LUFL** when ρ,α,β are constants as a *relaxed constant-factor approximation*.

Related Work. The **CFL** problem is the special case of **LUFL** when $L_i=0$ for all $i\in F$. There are several approximation algorithms for CFL based on local search techniques. For the case of uniform capacities, Korupolu et al. [13] gave the first constant factor approximation algorithm, with ratio 8. This was later improved to 5.83 [6] and 3 [2]. The first constant factor approximation for the case of non-uniform capacities was proposed by [17] who gave an 9-approximation, which was eventually improved to 5 [5]. An LP-based approach to CFL was employed by Shmoys et al. [18] who gave the first bicriteria approximation for uniform capacities; this was extended to non-uniform capacities [1]. Levi et al. [14] obtained an LP-based 5-approximation algorithm when facilities opening costs are uniform. For a long time it was an open question to prove a constant factor approximation for CFL based on LP-rounding. This was recently answered by An et al. [4] who gave an LP-based 288-approximation algorithm for CFL which works for the general case.

The **LBFL** problem is another special case of **LUFL** when $U_i=\infty$ for all $i\in F$. This problem was introduced independently by Guha et al. [9] and Karger et al. [12] who gave a bicriteria approximation. The first true approximation algorithm for **LBFL** was given by Svitkina [19] with ratio 448. The factor was then improved to 82.6 by [3] by applying a modified variant of the algorithm of [19], combined with a more careful analysis. We note that the approaches of both papers work only if all lower bounds are uniform. Finding a true approximation for **LBFL** when the lower bounds are non-uniform remains an open problem. To the best of our knowledge, there have been no LP-based approximation (even bicriteria) algorithms for **LBFL** in the literature.

[1] In an earlier version of [1] there was an attempt to study **LUFL** but there seemed to be an error in the proof. After checking with the authors the claim about **LUFL** is retracted in the current version of [1].

The *Connected Facility Location* (**ConFL**) problem is an obvious special case of **C-LUFL** (when $U_i = \infty$ & $L_i = 0$ for all $i \in F$.) The **ConFL** problem was first introduced by Gupta et al. [10], in the context of reserving bandwidth for *virtual private networks*, where they gave the first constant-factor approximation algorithm for **ConFL**. Using the primal-dual technique, the factor was then improved to 8.55 by [20], and to 6.55 by [11]. Applying sampling techniques, the guarantee was later reduced to 4 by [7], and to 3.19 by [8].

Our Results and Techniques. We explore LP-based approaches to obtain bicriteria approximations for many combinations of lower/upper/connected facility location. Our first main result is the first constant-factor (bicriteria) approximation algorithm for **LUFL**.

▶ **Theorem 2.** *There is a relaxed constant-factor approximation for instances of **LUFL** with uniform upper bounds (and non-uniform lower bounds).*

To prove this theorem we start by presenting an LP-based bicriteria approximation for **LBFL** with non-uniform lower bounds. Such approximations were known before, but ours is the first one whose cost can be compared to an LP relaxation. We emphasize that such bounds may be useful to obtain stronger results. For example, the LP-based **CFL** bicritera approximation by [1] was a key component in devising the true LP-based approximation in [4]. Perhaps our result could be used in an analogous result for **LBFL**.

Next, we incorporate the connectivity requirement. We obtain the first constant-factor bicriteria approximation for the *connected lower-bounded facility location* problem with non-uniform lower bounds. We then extend this to a relaxed constant-factor approximation for **C-LUFL** when the upper bounds U are uniform and the core cable multiplier M is $O(U)$. Some remarks on the difficulty of extending our approach to the case $M = \omega(U)$ are presented in the conclusion. Our second main result is the following.

▶ **Theorem 3.** *There is a relaxed constant-factor approximation for instances of **C-LUFL** with uniform upper bounds where $M = O(U)$.*

A key ingredient in our approach is a clustering step to avoid the standard "*filtering*" steps. That is, in classic facility location and **CFL** rounding algorithms a popular approach is to consider a ball around each client j whose radius is roughly the fractional cost of serving j. Values x_{ij} where i lies far outside this ball are set to 0 and the remaining $x_{i'j}$ values are rounded up by a small constant factor in order to get a solution that is "concentrated" around each client. This approach fails when lower bounds are present. We develop a clustering procedure to find a set of cluster *centers* $\mathcal{C}$ using a *Voronoi diagram* which is inspired by approches to *capacitated k-median* problem that was considered in [15, 16].

2 LP Relaxations and Starting steps

We present LP relaxations for **LUFL** as well as **C-LUFL**. For each $i \in F$, y_i indicates if facility i is opened. For each $i \in F$ and $j \in D$, x_{ij} indicates if client j is assigned to facility i.

$$\min \quad \sum_{i\in F} \mu_i y_i + \sum_{j\in D}\sum_{i\in F} c_{ij} x_{ij} \qquad \text{(LP-LUFL)}$$

$$\sum_{i\in F} x_{ij} = 1 \qquad \forall j \in D \qquad (2)$$

$$x_{ij} \le y_i \qquad \forall i \in F, j \in D \qquad (3)$$

$$\sum_{j\in D} x_{ij} \le U_i y_i \qquad \forall i \in F \qquad (4)$$

$$L_i y_i \le \sum_{j\in D} x_{ij} \qquad \forall i \in F \qquad (5)$$

$$x_{ij}, y_i \in [0,1] \qquad \forall i \in F, j \in D$$

Constraints (2) and (3) are standard *facility location constraints* saying that any client has to be assigned to an open facility in an integer solution. Constraints (4) and (5) ensure the lower and upper bounds are satisfied at the open facilities.

Extending LP-LUFL to model a relaxation for **C-LUFL**, we let z_e indicate if edge $e \in E$ is used by the core Steiner tree. We first guess one particular facility r that is open in the optimum solution and we called r the *root*. LP-C-LUFL is a linear programming relaxation of **C-LUFL**.

$$\min \quad \sum_{i\in F} \mu_i y_i + \sum_{j\in D}\sum_{i\in F} c_{ij} x_{ij} + M \sum_{e\in E} c_e z_e \qquad \text{(LP-C-LUFL)}$$

$$(2) - (5)$$

$$\sum_{e\in\delta(S)} z_e \ge \sum_{i\in S} x_{ij} \qquad \forall S \subseteq V \setminus \{r\}, j \in D \qquad (6)$$

$$y_r = 1 \qquad (7)$$

$$x_{ij}, y_i, z_e \in [0,1] \qquad \forall i \in F, j \in D, e \in E$$

Constraints (6) guarantee that (in the optimal solution) all open facilities are connected to facility r via core links, where Constraint (7) forces facility r to be opened.

Note that while (6) introduces exponentially many constraints, they can easily be separated by an efficient minimum-cut algorithm. Thus we can solve both (LP-LUFL) and (LP-C-LUFL) in polynomial time using the ellipsoid method.

2.1 Reduction Lemmas

In this section we present two lemmas that are used in the algorithms we present. The first lemma is a general clustering step that is applied as a first step of our LP rounding and reduces the facility location problem on hand to solving the problem on a specific cluster of clients facilities. This clustering step has similarities to a Voronoi diagram and for that reason we call it Voronoi clustering (inspired by [15, 16]). The second lemma shows how one can then extend the results obtained via this reduction step to the case where connectivity (with core cables) is required between open facilities.

Let (x, y, z) be a feasible solution to the LP relaxation of (LP-C-LUFL). Let L^j be the connection cost of client j in the LP, i.e. $L^j = \sum_{i\in F} c_{ij} x_{ij}$. The general idea is to select clients in increasing order of their L^j values and selecting them as centers if they are far from all centers so far. We then define a Voronoi cell with center j to be the set of all

facilities for which j is the closest center. This Voronoi clustering will be an important tool in decomposition of an LP solution in our rounding algorithms.

The following algorithm finds a set of clients $\mathcal{C}$ that will act as the centers in the Voronoi diagram and a partition $\{\mathcal{P}_j\}_{j\in\mathcal{C}}$ of F where $i\in P_j$ means j is a closest center to i. Here, λ is some parameter that we can specify. Larger values mean the centers are further apart. The algorithm also records a cluster center $\delta(j)\in\mathcal{C}$ for each $j\in D$: if $j\in\mathcal{C}$ then $\delta(j)=j$ and if $j\notin\mathcal{C}$ then $\delta(j)$ is the center that caused j to not be included in $\mathcal{C}$ (it may not be the closest center to j).

Algorithm 1. Voronoi Clustering algorithm (λ)

$\mathcal{C}\leftarrow\{j^*\}$ where $j^*=\arg\min_j L^j$;
for *each* $j'\in D-\{j^*\}$ *in increasing order of* $L^{j'}$ **do**
  if $c_{jj'}>2\lambda\cdot L^{j'}$ *for all* $j\in\mathcal{C}$ **then**
    $\mathcal{C}\leftarrow\mathcal{C}\cup\{j'\}$;
    $\delta(j')\leftarrow j'$;
  else
    let $j\in\mathcal{C}$ be some center with $c_{jj'}\le 2\lambda\cdot L^{j'}$;
    $\delta(j')\leftarrow j$;
  end
end
for *each* $j\in\mathcal{C}$ **do**
  $P_j\leftarrow\{i\in F: c_{ij}\le c_{ik}$ *for all* $k\in\mathcal{C}, k\neq j\}$;
  Comment: break ties arbitrarily so each $i\in F$ lies in exactly one P_j.
end
return $(\mathcal{C},\mathcal{P},\delta)$

Note that by construction of δ we have that $c_{\delta(j)j}\le 2\lambda L^j$ for each $j\in D$.

In Lemma 4 we show that for each center $j\in\mathcal{C}$, there is a facility i that is close to j whose opening cost can be paid for by the fractional opening cost paid by the LP for facilities near i. Furthermore, this facility i has a small enough lower bound that we can approximately satisfy by assigning to it all fractional client demand that was sent to some facility in $\mathcal{P}_j$.

For each client j and positive radius R, we let $B(j,R)=\{v\in V: c_{jv}\le R\}$ be a ball centered at j.

▶ **Lemma 4.** *Let (x,y) be values satisfying constraints (2), (3), and (5). Suppose $(\mathcal{C},\mathcal{P},\delta)$ is returned by calling Algorithm 1 with some given λ. Let $\widehat{X}_j=\sum_{i\in P_j}\sum_{j'\in D}x_{ij'}$ and let $\eta\in(1,\lambda]$. For each $j\in\mathcal{C}$, there exists some $i\in B^j:=B(j,\eta L^j)$ fulfilling: (i) $\mu_i\le\frac{2\eta}{\eta-1}\sum_{i'\in B^j}\mu_{i'}y_{i'}$ and (ii) $L_i\le\frac{2\eta}{\eta-1}\widehat{X}_j$.*

Proof. First observe that $\sum_{i\in B^j}y_i\ge 1-\frac{1}{\eta}$. For each $i\in B^j$, let $\overline{y}^j_i=\frac{x_{ij}}{\sum_{i'\in B^j}x_{i'j}}$. Note that $\forall i\in B^j$,

$$\overline{y}^j_i\le\frac{\eta}{\eta-1}x_{ij}\le\frac{\eta}{\eta-1}y_i, \tag{8}$$

holds by Constraints (3) and the fact that at least $\frac{\eta-1}{\eta}$ portion of j's demand is served within B^j (using Markov's inequality).

Now think of $\overline{y}^j$ as a probability distribution over facilities in B^j (note that $\sum_{i\in B^j}\overline{y}^j_i=1$). Suppose we sample a facility i from this distribution.

▶ **Claim 5.** $\Pr[\mu_i>\frac{2\eta}{\eta-1}\sum_{i'\in B^j}\mu_{i'}y_{i'}]<1/2$.

Proof. Observe that $\sum_{i' \in B^j} \mu_{i'} \overline{y}^j_{i'} \leq \frac{\eta}{\eta-1} \sum_{i' \in B^j} \mu_{i'} y_{i'}$. This, with Markov's inequality, implies: $\Pr[\mu_i > \frac{2\eta}{\eta-1} \sum_{i' \in B^j} \mu_{i'} y_{i'}] \leq \Pr[\mu_i > 2 \sum_{i' \in B^j} \mu_{i'} \overline{y}^j_{i'}] < 1/2$. ◀

▶ **Claim 6.** $\Pr[L_i > \frac{2\eta}{\eta-1} \widehat{X}_j] < 1/2$.

Proof. Using (5) and (8) and the fact that by choice of $\eta \in (1, \lambda]$, $B^j \cap F \subseteq P_j$ we have

$$\widehat{X}_j \geq \sum_{i \in B^j} \sum_{j' \in D} x_{ij'} \geq \sum_{i \in B^j} y_i L_i \geq \frac{\eta - 1}{\eta} \sum_{i \in B^j} L_i \overline{y}^j_i \tag{9}$$

This implies: $\Pr[L_i > \frac{2\eta}{\eta-1} \widehat{X}_j] \leq \Pr[L_i > 2 \sum_{i \in B^j} \overline{y}^j_i L_i] < 1/2$. ◀

The above two claims immediately imply that with positive probability, there is a facility that satisfies both conditions in inequalities (i) and (ii), respectively. Hence the lemma holds. ◀

Our next lemma demonstrates the utility of our clustering algorithm even in the presence of the connectivity requirements. We show below that if we find a (lower/upper bounded) facility location solution within each cluster and if we can connect those open facilities to the center of the clusters using core cables cheaply then we can connect the centers using core cables cheaply. This helps us to reduce the problem to solving each Voronoi cell separately.

▶ **Lemma 7.** *Let (x, y, z) be values satisfying (2)–(3) and (6)–(7) and $(\mathcal{C}, \mathcal{P}, \delta)$ be returned by Algorithm 1 with x, y, and some given λ. Let $\eta \in (1, \lambda)$. Then we can efficiently find a Steiner tree that connects $\mathcal{C}$ with cost at most $\frac{\lambda}{\lambda-\eta} \cdot \frac{2\eta}{\eta-1} \cdot M \cdot \sum_e c_e z_e$.*

Proof. Note that we require $\eta < \lambda$. We assume that facility $r \in B(j, \eta L^j)$ for some $j \in \mathcal{C}$. The other case where $r \notin B(j, \eta L^j)$ for any $j \in \mathcal{C}$ is nearly identical and results in the same bound. We also observe that $\{B(j, \eta L^j) : j \in \mathcal{C}\}$ consists of disjoint sets: if $B(j, \eta L^j) \cap B(j', \eta L^{j'}) \neq \emptyset$ for distinct $j, j' \in \mathcal{C}$ then $c_{jj'} \leq 2\lambda \cdot \max\{L^j, L^{j'}\}$ so both j and j' could not be cluster centers.

Note that $\sum_{i \in B(j, \eta L^j)} x_{ij} \geq \frac{\eta-1}{\eta}$ holds for any $j \in \mathcal{C}$, using of Markov's inequality. This, together with (6), implies that vector $\frac{\eta}{\eta-1} z$ is a feasible fractional solution to the standard cut based LP relaxation of the Steiner tree problem with terminals being balls $B(j, \eta L^j)$ contracted at their centers. Thus, we can efficiently find a Steiner tree $\hat{T}$ over these contracted balls (on the resulting graph after contracting balls) with cost at most $\frac{2\eta}{\eta-1} \sum_e c_e z_e$.

Now we have to convert this tree $\hat{T}$ into a Steiner tree over centers $\mathcal{C}$. When we uncontract the balls, each edge of $\hat{T}$ between two balls around centers j, j' can be replaced with the edge between two closest nodes, say $u \in B(j, \eta L^j)$ and $v \in B(j', \eta L^{j'})$. We add edges ju and vj' for each such $uv \in \hat{T}$ to complete the Steiner tree. To bound the cost of these new edges, observe that $\eta < \lambda$ and not only balls $B(j, \eta L^j)$ and $B(j', \eta L^{j'})$ are disjoint, but also balls $B(j, \lambda L^j)$ and $B(j', \lambda L^{j'})$ are disjoint as well by the same argument. So we can "charge" the cost of two new edges ju and vj' to the section of edge uv that falls between the two nested balls as follows. Let $\alpha = \max\{L^j, L^{j'}\}$. Since $u \in B(j, \eta L^j)$ and $v \in B(j, \eta L^{j'})$ then $c_{uj} + c_{j'v} \leq 2\eta\alpha$. Furthermore, $2\lambda\alpha \leq c_{jj'} \leq c_{uj} + c_{uv} + c_{vj'} \leq c_{uv} + 2\eta\alpha$. Therefore,

$$c_{ju} + c_{vj'} \leq 2\eta\alpha = \frac{2\eta}{\lambda - \eta} \cdot (\lambda - \eta) \cdot \alpha \leq \frac{\eta}{\lambda - \eta} c_{uv}.$$

Thus, the total cost of this tree is at most $\left(1 + \frac{\eta}{\lambda-\eta}\right) \cdot \frac{2\eta}{\eta-1} \cdot M \cdot \sum_e c_e z_e$. ◀

3 An LP-Based Approximation Algorithm for LUFL

In this section we present a rounding bicriteria approximation algorithm for **LUFL**. We start with the simpler case where we only have lower bounds and then show how to extend the algorithm to work for when there are both upper and lower bounds for facility loads.

3.1 Lower-Bounded Facility Location

We first consider the case where all facilities have infinite capacities. An LP to this case can be written as follows. We let (x, y) and OPT_{LP} be an optimal solution and the optimum cost of LP-LFL, respectively.

$$\begin{aligned} \min \quad & \sum_{i \in F} \mu_i y_i + \sum_{j \in D} \sum_{i \in F} c_{ij} x_{ij} \\ & (2),(3),(5) \\ & x_{ij}, y_i \geq 0 \end{aligned} \tag{LP-LFL}$$

It is easy to see that LP-LFL has unbounded integrality gap: Consider a small instance of **LBFL** consisting of $2(L-1)$ clients (with unit demands), two zero-cost facilities each collocated with $L-1$ clients, and an edge of length L between these two facilities. While the optimal cost to IP is $L(L-1)$, LP can manage to pay only $2(L-1)$ by opening both facilities to the extent of $\frac{L-1}{L}$, and thereby only sending $\frac{1}{L}$ demand of each client to its far facility. Hence, the integrality gap can be made arbitrarily large by increasing L. Therefore bicriteria approximation is unavoidable if we use this LP.

Let $\eta > 1$ be a parameter we may choose, larger values result in more expensive solutions with smaller violations in the lower bound. Our algorithm for **LBFL** has two steps and works as follows. We first find a Voronoi clustering using Algorithm 1 and then for each cluster center j we open one facility in the cell as guaranteed by Lemma 4. All demand $\widehat{X}_j$ that is fractionally assigned to $\mathcal{P}_j$ is assigned to this open facility. To turn this into an integer assignment of clients to facilities, we then compute the minimum-cost integer flow that satisfies the relaxed lower bounds. The fact that this is cheap is witnessed by the fractional assignment we find in the first part of the algorithm.

Algorithm 2: LBFL rounding

Step 1: Construct a Voronoi clustering $(\mathcal{C}, \mathcal{P}, \delta)$ by running Algorithm 1 with the given x, y, and $\lambda = \eta$.

Step 2: Let $I = \{i(j) \colon j \in \mathcal{C}\}$, where $i(j) \in P_j$ is the facility described in Lemma 4. Open facilities I and find the cheapest assignment of clients to them such that each open facility i serves at least $\frac{\eta-1}{2\eta} L_i$ demand.

▶ **Theorem 8.** *Algorithm 2 computes in polynomial time a solution to **LBFL** with the following properties:*

(i) *The solution cost is at most* $\max\{4(\eta+1), \frac{2\eta}{\eta-1}\} \cdot OPT_{LP}$.

(ii) *Each open facility $i \in I$ is serving at least $\lfloor \frac{\eta-1}{2\eta} L_i \rfloor$ clients.*

Proof. We provide a solution as described in Step 2 fulfilling the claimed properties. Consider $(\mathcal{C}, \mathcal{P}, \delta)$ costructed at Step 1. Recall $\widehat{X}_j = \sum_{i \in P_j} \sum_{j' \in D} x_{ij'}$.

By Lemma 4 and the fact that P_j cells are disjoint, the total opening cost is bounded as follows.

$$\mu(I) \leq \sum_{j\in\mathcal{C}} \mu_{i(j)} \leq \frac{2\eta}{\eta-1} \sum_{j\in\mathcal{C}} \sum_{i\in P^j} \mu_i y_i \leq \frac{2\eta}{\eta-1} \sum_{i\in F} \mu_i y_i. \tag{10}$$

Assigning the fractional demands $\widehat{X}_j$ aggregated at j to each $i(j) \in I$ guarantees the second property; see Lemma 4. Hence, we only need to show this assignment is cheap and how to turn it into an integer assignment of no more cost.

Consider some client $j' \in D$ and some facility $i \in F$. In what follows we show that $x_{ij'}$ units of demand travel a distance of at most $4(\eta L^{j'} + c_{ij'})$. Say that $i \in P_j$ and let $B^j := B(j, \eta L^j)$. Thus, in this assignment the $x_{ij'}$-fraction of demand travels distance $c_{j'i(j)}$. We consider two cases:

Case $j' \notin B^j$: We have

$$\begin{aligned}
c_{j'i(j)} &\leq c_{i(j)j} + c_{jj'} && \text{(by the triangle inequality)}\\
&\leq \eta L^j + c_{jj'} && \text{(using the fact that } i(j) \in B^j)\\
&\leq 2c_{jj'} && \text{(using the fact that } j' \notin B^j)\\
&\leq 2(c_{ij} + c_{ij'}) && \text{(by the triangle inequality)}\\
&\leq 2(c_{i\delta(j')} + c_{ij'}) && \text{(using the fact that } i \in P_j)\\
&\leq 2(c_{j'\delta(j')} + 2c_{ij'}) && \text{(by the triangle inequality)}\\
&\leq 2(2\eta L^{j'} + 2c_{ij'}) && \text{(from the clustering procedure)}
\end{aligned}$$

Case $j' \in B^j$: In this case $c_{j'i(j)} \leq 2\eta L^j$ (by the triangle inequality). Below we show that $L^j \leq 2L^{j'}$, which immediately implies $c_{j'i(j)} \leq 4\eta L^{j'}$.

▶ **Claim 9.** $L^j \leq 2L^{j'}$.

Proof. Assume, for the sake of contradiction, that $L^j > 2L^{j'}$. First observe that by the ordering clients are selected as centers in $\mathcal{C}$, $j' \notin \mathcal{C}$: note that $j \in \mathcal{C}$, and since we assumed ($2L^{j'} < L^j$ and so) $L^{j'} < L^j$, and because $c_{jj'} \leq 2\eta L^j$ (recall $j' \in B^j$), if $j' \in \mathcal{C}$ then it would have prevented j from being added to $\mathcal{C}$ in the first step. Now, consider $\delta(j') \in \mathcal{C}$. Note that $L^{\delta(j')} \leq L^{j'}$ and $c_{j'\delta(j')} \leq 2\eta L^{j'}$. This implies

$$\begin{aligned}
c_{j\delta(j')} &\leq c_{j'j} + c_{\delta(j')j'} && \text{(by the triangle inequality)}\\
&\leq \eta L^j + 2\eta L^{j'} && \text{(by } j' \in B^j \text{ and clustering procedure)}\\
&\leq \eta L^j + \eta L^j && \text{(using the assumption that } L^j > 2L^{j'})\\
&\leq 2\eta L^j
\end{aligned}$$

which is a contradiction because then $\delta(j')$ would also have blocked j from being added to $\mathcal{C}$. The claim follows. ◀

This completes the proof of this case that $c_{j'i(j)} \leq 4\eta L^{j'}$.

In either case, $x_{ij'}$ travels a distance of at most $4(\eta L^{j'} + c_{ij'})$. Thus, the total assignment cost of this fractional solution is bounded by

$$\begin{aligned}
4\sum_{i\in F}\sum_{j'\in D} x_{ij'}\left(\eta L^{j'} + c_{ij'}\right) &= 4\eta \sum_{j'\in D} L^{j'} \sum_{i\in F} x_{ij'} + 4\sum_{i\in F}\sum_{j'\in D} c_{ij'}x_{ij'}\\
&= 4\eta \sum_{j'\in D} L^{j'} + 4\sum_{i\in F}\sum_{j'\in D} c_{ij'}x_{ij'} \quad \text{using (2)}\\
&= 4(\eta+1)\sum_{i\in F}\sum_{j'\in D} x_{ij'}c_{ij'} \quad \text{(by def. of } L^{j'})
\end{aligned}$$

Together with (10), this implies the claimed bound.

Finally, because of the integrality of flows with integer lower bounds and because we have explicitly described a cheap fractional flow from the clients to the open facilities that satisfies the integer lower bounds $\lfloor \frac{\eta-1}{2\eta} L_i \rfloor$, then there is an integer assignment $\sigma : D \to I$ that also satisfies these lower bounds with no greater cost. ◀

For example, by choosing $\eta = 1.28$ we get a solution of cost at most $9.12 OPT_{LP}$ and the load of each open facility i is at least $\lfloor \frac{L_i}{9.12} \rfloor$.

3.2 The general case with lower and upper bounds

We now consider the case where each facility has capacity U (uniform across all facilities) as well as a given lower bound L_i. Let (x, y) be an optimal solution to (LP-LUFL).

As before, we first use Algorithm 1 to obtain a Voronoi clustering. We then decide to open a number of facilities in each cell to route the clients demand to be served at them while satisfying the upper and lower bounds on the facility loads (approximately). The algorithm consists of two steps and works as follow.

Algorithm 3: LUFL rounding

Step 1: Construct a Voronoi clustering $(\mathcal{C}, \mathcal{P}, \delta)$ by running Algorithm 1 with the given x, y, and $\lambda = \eta$.

Step 2: For each $j \in \mathcal{C}$, we open a subset $I_j \subseteq P_j$ of facilities and send the demand served by facilities in $\mathcal{P}_j$ (namely $\widehat{X}_j = \sum_{j'} \sum_{i \in \mathcal{P}_j} x_{ij'}$) to those facilities as described below, depending on the value of $\widehat{X}_j$:

Case 1. $\widehat{X}_j \geq U$: In this case, inspired by ideas from [16], we formulate the described subproblem as another (simpler) facility location (inside the cell) using a simpler (sparse) LP.

We firstly move demand $\widehat{X}_j$ to center j as follows. For each client $j' \in D$ and each facility $i \in \mathcal{P}_j$, we send $x_{ij'}$ demand from j' to i (this is what the LP is doing). Let $\hat{d}^i = \sum_{j' \in D} x_{ij'}$ be the demand sent to i. Next, for each facility $i \in P_j$, we send $\hat{d}^i$ demand from i to j. Obviously, the total cost of this moving is bounded by $\sum_{i \in P_j} \sum_{j' \in D} x_{ij'} \big(c_{ij'} + c_{ij}\big)$.

We now ignore the facility lower bounds and write an LP to solve the subproblem. Solving and then rounding this LP helps us to decide which facilities in P_j to open and how to assign the $\widehat{X}_j$ demand (already aggregated at j) to them. We shall show how the cost of this LP can be bounded by the cost of the original LP restricted to this cell and an optimum solution to this LP satisfies the lower bounds on almost all facilities.

In this LP, we have a variable ω_i for each $i \in \mathcal{P}_j$ indicating how much of the $\widehat{X}_j$ is assigned to i.

$$\begin{aligned} \min \sum_{i \in P_j} \omega_i \big(\frac{\mu_i}{U} + c_{ij}\big) & \\ \sum_{i \in P_j} \omega_i = \widehat{X}_j & \\ 0 \leq \omega_i \leq U & \qquad \forall i \in P_j \end{aligned}$$

Note that setting $\omega_i := \sum_{j' \in D} x_{ij'}$ is a feasible solution with cost at most $\sum_{i \in P_j} \mu_i y_i + \sum_{i \in P_j} \sum_{j' \in D} x_{ij'} c_{ij}$ because $\sum_j x_{ij} \leq U y_i$.

Note that there is only one constraint apart from constraints $0 \le \omega_i \le U$. Thus, for all but one $i \in P_j$ we have $\omega_i^* \in \{0, U\}$, where ω^* indicates an optimum extreme point solution to this LP.

To round this solution ω_i^*, let $\zeta \in (1, 1.6)$ be a parameter we get to choose. Let $I_j = \{i \in P_j : \omega_i^* = U\}$. If there is some $i' \in P_j$ such $0 < \omega_{i'}^* < U$ then add i' to I_j if $\omega_{i'}^* \ge \frac{U}{\zeta}$. In this case, the upper bound is satisfied for every $i \in I_j$ and the lower bound is violated by no more than a ζ-factor. Assign precisely ω_i^* units of demand to each $i \in I_j$. The cost of this assignment plus the cost of opening I_j is at most $\zeta \sum_{i \in P_j} \mu_i y_i + \zeta \sum_{i \in P_j} \sum_{j' \in D} x_{ij'} c_{ij}$.

Otherwise, if $\omega_{i'}^* < \frac{U}{\zeta}$ then let i'' be the facility in I_j closest to j and increase $\omega_{i''}^*$ by $\omega_{i'}^*$. Note that such a facility i'' exists because we are assuming $\widehat{X}_j \ge U$. In this case, no lower bounds are violated at any $i \in I_j$ and the upper bound is violated by at most a $\left(1 + \frac{1}{\zeta}\right)$-factor. The assignment and opening cost in this case are bounded by $\frac{\zeta+1}{\zeta} \sum_{i \in P_j} \mu_i y_i + \frac{\zeta+1}{\zeta} \sum_{i \in P_j} \sum_{j' \in D} x_{ij'} c_{ij}$.

In either case, we have opened I_j and assigned demand to each $i \in I_j$ to satisfy the relaxed lower bounds L_i/ζ and the relaxed upper bounds $\frac{\zeta+1}{\zeta} U$. Since $\frac{\zeta+1}{\zeta} > \zeta$ holds for any $\zeta \in (1, 1.6)$, the cost of assigning $\widehat{X}_j$ units of demand from j to I_j in this manner is at most $\frac{\zeta+1}{\zeta} \sum_{i \in P_j} \mu_i y_i + \frac{\zeta+1}{\zeta} \sum_{i \in P_j} \sum_{j' \in D} x_{ij'} c_{ij}$. Altogether, the total cost (of moving the $\widehat{X}_j$ demand to center j plus the cost of assigning it from j to facilities I_j) is bounded by

$$\frac{\zeta+1}{\zeta} \sum_{i \in P_j} \mu_i y_i + \frac{\zeta+1}{\zeta} \sum_{i \in P_j} \sum_{j' \in D} x_{ij'} c_{ij} + \sum_{i \in P_j} \sum_{j' \in D} x_{ij'} \left(c_{ij'} + c_{ij}\right) =$$
$$\frac{\zeta+1}{\zeta} \sum_{i \in P_j} \mu_i y_i + \frac{2\zeta+1}{\zeta} \sum_{i \in P_j} \sum_{j' \in D} x_{ij'} c_{ij} + \sum_{i \in P_j} \sum_{j' \in D} x_{ij'} c_{ij'}.$$

▶ **Lemma 10.** *The total cost (over all cells of Voronoi clustering) incurred due to Case 1 of Step 2 of the algorithm is at most* $\frac{\zeta+1}{\zeta} \sum_{i \in F} \mu_i y_i + \frac{(2\zeta+1)(2\eta+1)+\zeta}{\zeta} \sum_{i \in F} \sum_{j \in D} x_{ij} c_{ij}$.

Proof. The total cost is bounded by

$$\sum_{j \in \mathcal{C}} \Big(\frac{\zeta+1}{\zeta} \sum_{i \in P_j} \mu_i y_i + \frac{2\zeta+1}{\zeta} \sum_{i \in P_j} \sum_{j' \in D} x_{ij'} c_{ij} + \sum_{i \in P_j} \sum_{j' \in D} x_{ij'} c_{ij'} \Big)$$
$$= \frac{\zeta+1}{\zeta} \sum_{i \in F} \mu_i y_i + \frac{2\zeta+1}{\zeta} \sum_{j \in \mathcal{C}} \sum_{i \in P_j} \sum_{j' \in D} x_{ij'} c_{ij} + \sum_{i \in F} \sum_{j' \in D} x_{ij'} c_{ij'}, \tag{11}$$

using the fact that P_j cells are disjoint.

Note that for any $i \in P_j$ and any $j' \in D$ we have

$$\begin{aligned} c_{ij} &\le c_{i\delta(j')} && \text{(using the fact that } i \in P_j\text{)} \\ &\le c_{ij'} + c_{j'\delta(j')} && \text{(by the triangle inequality)} \\ &\le c_{ij'} + 2\eta L^{j'} && \text{(from Step 1)} \end{aligned}$$

Hence, we have:

$$\frac{2\zeta+1}{\zeta} \sum_{j \in \mathcal{C}} \sum_{i \in P_j} \sum_{j' \in D} x_{ij'} c_{ij} \le \frac{2\zeta+1}{\zeta} \sum_{j \in \mathcal{C}} \sum_{i \in P_j} \sum_{j' \in D} x_{ij'} \left(c_{ij'} + 2\eta L^{j'}\right)$$
$$= \frac{2\zeta+1}{\zeta} \sum_{i \in F} \sum_{j' \in D} c_{ij'} x_{ij'} + \frac{(2\zeta+1)(2\eta)}{\zeta} \sum_{j' \in D} L^{j'} \quad \text{(by (2))}$$
$$= \frac{(2\zeta+1)(2\eta+1)}{\zeta} \sum_{i \in F} \sum_{j' \in D} c_{ij'} x_{ij'}.$$

This, together with (11), implies the claimed bound. ◀

Case 2. $\widehat{X_j} < U$: Observe that in this case we can simply ignore the upper bound. So (similar to that for **LBFL**) we open facility $i(j)$ described in Lemma 4 and send the demand to that facility as follows: For each client $j' \in D$ and each facility $i \in \mathcal{P}_j$, we send $x_{ij'}$ demand from j' (directly) to $i(j)$. Let $I_j = \{i(j)\}$ in this case. Note that facility $i(j)$ serves at least $\frac{\eta-1}{2\eta} L_i$.

The following bound can be obtained using the exact same arguments used to bound that in the proof of Theorem 8.

▶ **Lemma 11.** *The total cost incurred due to Case 2 of Step 2 is at most* $\frac{2\eta}{\eta-1} \sum_{i\in F} \mu_i y_i + 4(\eta+1) \sum_{i\in F} \sum_{j\in D} x_{ij} c_{ij}$.

Let $I = \cup_{j\in\mathcal{C}} I_j$ be the set of facilities opened over all Voronoi cells. Observe that each of the two cases above finds a solution to **LBFL** in which each open facility $i \in I$ serves at least $\min\{\frac{1}{\zeta}, \frac{\eta-1}{2\eta}\} L_i$ (based on the two cases above) and at most $\frac{\zeta+1}{\zeta} U$ demand.

Summing our bounds on the cost of the solutions found in each Voronoi diagram (see Lemmas 10 and 11), we see the cost of opening I is at most

$$\left(\frac{\zeta+1}{\zeta} + \frac{2\eta}{\eta-1}\right) \sum_{i\in F} \mu_i y_i, \tag{12}$$

and the cost of assigning demands is at most:

$$\left(\frac{(2\zeta+1)(2\eta+1)+\zeta}{\zeta} + 4(\eta+1)\right) \sum_{j\in D} \sum_{i\in F} c_{ij} x_{ij} \tag{13}$$

Together, (12) and (13) and using integrality of flows with integer lower and upper bounds, imply the main results of this section.

▶ **Theorem 2** (restated). *Algorithm 3 is a polynomial time (ρ, α, β)-approximation for instances of **LUFL** with uniform capacities where* $\rho = \max\{\frac{(2\zeta+1)(2\eta+1)+\zeta}{\zeta} + 4(\eta+1), \frac{2\eta}{\eta-1} + \frac{\zeta+1}{\zeta}\}$, $\alpha = \max\{\zeta, \frac{2\eta}{\eta-1}\}$, $\beta = \frac{\zeta+1}{\zeta}$.

4 An LP-Based Approximation Algorithm for C-LUFL

In this section we show that our rounding framework for **LUFL** extends to connected variants. In the light of Lemma 7, we observe that our framework works for the connected variants too, as long as we can bound the cost of connecting facilities opened in each Voronoi cell to the center it belongs to.

We begin with the case where all facilities have infinite capacities (denoted by **C-LBFL**). We let (x, y, z) and OPT_{LP} be the optimal solution and the optimum cost of the LP relaxation for this case, respectively.

Let $\lambda > \eta > 1$ be constant parameters. Following the same general ideas of that for **LBFL** and using our observation described in Lemma 7, we present our algorithm for **C-LBFL** which has three stages and works as follows.

Algorithm 4: C-LBFL rounding

Step 1: Construct a Voronoi clustering $(\mathcal{C}, \mathcal{P}, \delta)$ by running Algorithm 1 with the given x, y, λ.

Step 2: Open facilities $I = \{i(j) \colon\ j \in \mathcal{C}\}$ and assign clients to them as described in Step 2 of Algorithm 2. Connect each facility $i(j) \in I$ to the center it belongs to via core cables.

Step 3: Compute a core Steiner tree over centers $\mathcal{C}$ as described in Lemma 7.

One can simply adapt the proof of Lemma 7 to bound the extra cost of connecting each center j to the facility $i(j)$ by losing a constant factor. Apart from this the proof of the following theorem is analogous to that for Theorem 8.

▶ **Theorem 12.** *Algorithm 4 computes in polynomial time a solution to* ***C-LBFL*** *with the following properties:*

(i) *The solution cost is at most* $\max\{4(\eta+1), \frac{2\eta}{\eta-1}, \frac{2\cdot(\lambda+\eta)\eta}{(\lambda-\eta)(\eta-1)}\} \cdot OPT_{LP}$.

(ii) *Each open facility* $i \in I$ *is serving at least* $\lfloor \frac{\eta-1}{2\eta} L_i \rfloor$ *clients.*

We now consider the **C-LUFL** problem. First we show that one can convert an optimum solution of **C-LUFL** to an approximate solution in which each open facility (say) i is assigned a sufficiently large number of clients comparable not only to U and L_i but also to M (core cable cost per unit length). This property of a near optimal solution will help use to compute approximate solutions to **C-LUFL**. Let $\Delta = \min\{M, U\}$. Let $\mathrm{OPT}_{\text{C-LU}}$ be the cost of an optimal solution to **C-LUFL**. Observe that when the number of clients is less than $\frac{\Delta}{2}$, selecting only the cheapest facility to be opened and then assigning all clients to that open facility returns the optimal solution. We hence assume that the number of clients is at least $\frac{\Delta}{2}$. The proof of the following theorem is omitted due to lack of space.

▶ **Theorem 13.** *There is a feasible solution of cost at most* $3OPT_{C\text{-}LU}$ *to* ***C-LUFL*** *in which each open facility* i *is assigned at least* $\max\{\frac{\Delta}{2}, L_i\}$ *units of demand.*

In what follows (instead of approximating **C-LUFL**) we approximate the near optimal solution described above whose property is needed for our analysis to work. We write a modification of LP-C-LUFL to model the approximate solution described above.

$$\begin{aligned} \min \quad & \sum_{i\in F} \mu_i y_i + \sum_{j\in D}\sum_{i\in F} c_{ij}x_{ij} + M\sum_{e\in E} c_e z_e \\ & (2)\text{-}(4), (6)\text{-}(7) \\ & \Delta y_i \le 2\sum_{j\in D} x_{ij} \qquad \forall i \in F \qquad (14) \\ & x_{ij}, y_i, z_e \ge 0 \end{aligned}$$

We let (x, y, z) be the optimal solution of this LP relaxation. Let $\lambda > \eta > 1$ be constant parameters. Following the algorithm for **LBFL** and using Lemma 7, we extend the algorithm for **C-LBFL** to work for the more general case where each facility has a capacity U and $M = O(U)$. Our algorithm has three steps and works as follows.

Algorithm 5: C-LUFL rounding

Step 1: Construct a Voronoi clustering $(\mathcal{C}, \mathcal{P}, \delta)$ by running Algorithm 1 with the given x, y, λ.

Step 2: Open facilities $I = \cup_{j\in\mathcal{C}} I_j$ and assign clients to them as described in Step 3 of Algorithm 3. Then, connect each facility $i \in I$ to the center of the cell it belongs to using core cables.

Step 3: Compute a core Steiner tree over centers $\mathcal{C}$ as described in Lemma 7.

▶ **Theorem 3** (restated). *Algorithm 5 computes in polynomial time a* $\left(O(1), \max\{\zeta, \frac{2\eta}{\eta-1}\}, \frac{\zeta+1}{\zeta}\right)$*-bicriteria approximation for instances of* ***C-LUFL*** *with uniform capacities (and non-uniform lower bounds) and with* $M = O(U)$*.*

Due to lack of space, the proof is deferred to the full version.

5 Conclusion

It would be nice to extend our approximations for C-LUFL to include the case $M = \omega(U)$. As M gets larger, the cost of connecting core cables becomes so large that an optimum solution would open the fewest possible facilities, namely $k := \lceil |D|/U \rceil$. This case resembles the well-studied k-MST problem where it is well-known that even getting a constant-factor bicriteria approximation is not possible using the natural cut-based relaxation. So, this case poses additional difficulties.

Also open is the problem of getting constant-factor biceriteria approximations for **LUFL** when both lower and upper bounds are not necessarily uniform.

References

1. Zoë Abrams, Adam Meyerson, Kamesh Munagala, and Serge Plotkin. The integrality gap of capacitated facility location. *Technical Report CMU-CS-02-199, Carnegie Mellon University*, 2002.
2. Ankit Aggarwal, Anand Louis, Manisha Bansal, Naveen Garg, Neelima Gupta, Shubham Gupta, and Surabhi Jain. A 3-approximation algorithm for the facility location problem with uniform capacities. *Mathematical Programming*, 141, 2013.
3. Sara Ahmadian and Chaitanya Swamy. Improved approximation guarantees for lower-bounded facility location. *In proceedings of WAOA 2012*, pages 257–271, 2012.
4. Hyung-Chan An, Monika Singh, and Ola Svensson. LP-based algorithms for capacitated facility location. *In proceedings of FOCS 2014*, pages 256–265, 2014.
5. Manisha Bansal, Naveen Garg, and Neelima Gupta. A 5-approximation for capacitated facility location. *In proceedings of ESA 2012*, pages 133–144, 2012.
6. Fabián A Chudak and David P Williamson. Improved approximation algorithms for capacitated facility location problems. *Mathematical programming*, 102, 2005.
7. Friedrich Eisenbrand, Fabrizio Grandoni, Thomas Rothvoß, and Guido Schäfer. Connected facility location via random facility sampling and core detouring. *Journal of Computer and System Sciences*, 76(8):709–726, 2010.
8. Fabrizio Grandoni and Thomas Rothvoß. Approximation algorithms for single and multi-commodity connected facility location. *In proceedings of IPCO 2011*, pages 248–260, 2011.
9. Sudipto Guha, Adam Meyerson, and Kamesh Munagala. Hierarchical placement and network design problems. *In proceedings of FOCS 2000*, pages 603–612, 2000.
10. Anupam Gupta, Jon Kleinberg, Amit Kumar, Rajeev Rastogi, and Bulent Yener. Provisioning a virtual private network: a network design problem for multicommodity flow. *In proceedings of STOC 2001*, pages 389–398, 2001.
11. Hyunwoo Jung, Mohammad Khairul Hasan, and Kyung-Yong Chwa. A 6.55 factor primal-dual approximation algorithm for the connected facility location problem. *Journal of combinatorial optimization*, 18(3):258–271, 2009.
12. DR Karget and Maria Minkoff. Building steiner trees with incomplete global knowledge. *In proceedings of FOCS 2000*, pages 613–623, 2000.
13. Madhukar R Korupolu, C Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. *Journal of algorithms*, 2000.

14 Retsef Levi, David B Shmoys, and Chaitanya Swamy. LP-based approximation algorithms for capacitated facility location. *Mathematical programming*, 131, 2012.

15 Shi Li. On uniform capacitated k-median beyond the natural lp relaxation. *In proceedings of SODA 2015*, pages 696–707, 2015.

16 Shi Li. Approximating capacitated k-median with $(1 + \epsilon)$k open facilities. *In proceedings of SODA 2016*, 2016.

17 Martin Pal, Eva Tardos, and Tom Wexler. Facility location with nonuniform hard capacities. *In proceedings of FOCS 2001*, pages 329–338, 2001.

18 David B Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems. *In proceedings of STOC 1997*, pages 265–274, 1997.

19 Zoya Svitkina. Lower-bounded facility location. *ACM Transactions on Algorithms (TALG)*, 6(4):69, 2010.

20 Chaitanya Swamy and Amit Kumar. Primal–dual algorithms for connected facility location problems. *Algorithmica*, 40(4):245–269, 2004.

Approximation Algorithms for Node-Weighted Prize-Collecting Steiner Tree Problems on Planar Graphs*

Jarosław Byrka[1], Mateusz Lewandowski[2], and Carsten Moldenhauer[3]

1 **University of Wrocław, Wrocław, Poland**
2 **University of Wrocław, Wrocław, Poland**
3 **EPFL, Lausanne, Switzerland**

Abstract

We study the prize-collecting version of the node-weighted Steiner tree problem (NWPCST) restricted to planar graphs. We give a new primal-dual Lagrangian-multiplier-preserving (LMP) 3-approximation algorithm for planar NWPCST. We then show a 2.88-approximation which establishes a new best approximation guarantee for planar NWPCST. This is done by combining our LMP algorithm with a threshold rounding technique and utilizing the 2.4-approximation of Berman and Yaroslavtsev [6] for the version without penalties. We also give a primal-dual 4-approximation algorithm for the more general forest version using techniques introduced by Hajiaghay and Jain [17].

1998 ACM Subject Classification G.2.2 [Graph Theory] Graph algorithms

Keywords and phrases approximation algorithms, Node-Weighted Steiner Tree, primal-dual algorithm, LMP, planar graphs

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.2

1 Introduction

In Steiner problems we aim at connecting certain specified vertices (called terminals) by buying edges or nodes of the given graph. The classic edge-weighted setting is well known to have many applications in areas like electronic circuits, computer networking, and telecommunication. The expressive power of the node weighted variants is used to model various settings common to bioinformatics [11], maintenance of electric power networks [16], and computational sustainability [10].

The node weighted setting is a generalization of the edge weighted case. In particular, one may cast the Set Cover problem as an instance of the Node-weighted Steiner Tree problem, which proves hardness of approximation of the general node-weighted setting. In this paper we study a natural special case, namely planar graphs, for which constant factor approximation algorithms are possible.

In the prize-collecting (penalty-avoiding) setting we are given an option not to satisfy a certain connectivity requirement, but to pay a fixed penalty instead. The main focus of this work is to develop efficient primal-dual approximation algorithms for prize-collecting versions of the node-weighted Steiner problems.

* Supported by NCN 2015/18/E/ST6/00456 grant.

Table 1 Summary of best known approximation ratios for Steiner problems. Results of this paper are highlighted.

		edge-weighted		node-weighted	
		tree	**forest**	**tree**	**forest**
	general	1.39 [7]	2 [15]	O(log k) [5]	O(log k) [5]
	planar	PTAS [4]	PTAS [4]	2.4 [6]	2.4 [6]
prize-collecting	**general**	$2-\epsilon$ [2]	3 2.54 (LP) [17]	O(log k) [5, 19]	O(log k) [5]
	planar	PTAS [3]	APX-HARD [3]	**3** **2.88 (LP)**	**4**

1.1 Previous work

The Steiner tree problem is NP-hard even in planar graphs [12]. The most studied version is the standard edge-weighted Steiner tree, for which the best known approximation ratio 1.39 is obtained via a randomized iterative rounding technique [7]. By contrast, the best approximation algorithms for Steiner forest have the so far unbreakable ratio of 2 [1, 18].

For the prize-collecting Steiner tree problem there exists a primal-dual 2-approximation algorithm [15]. It can be shown that it is also Lagrangian-preserving, *i.e.*, that it achieves a 1-approximation on the penalty term. This property was used by Archer et al. to design the currently best $2-\epsilon$ approximation algorithm for PCST [2].

For the prize-collecting Steiner forest problem there is a 3 approximation primal-dual algorithm [17], which introduces a general technique to handle prize-collecting problems. In the same paper the authors use a threshold rounding technique with randomized analysis to obtain a ≈ 2.54 approximation.

There are optimal (up to a constant factor) algorithms for node-weighted Steiner problems. One example is the recent $O(\ln n)$ approximation algorithm for NWPCSF by Bateni et al [5]. Könemann et al [19] gave a Lagrangian-multiplier-preserving (LMP) approximation that achieves the same guarantee. Establishing the LMP property is of crucial importance for the construction of approximation algorithms for quota and budgeted versions of the NWST problem.

Planarity helps significantly in both the edge and node weighted setting. Both ST and SF admit PTAS in planar graphs [4]. Planar PCST can be also approximated with any constant, but PCSF is APX-HARD already on planar graphs [3].

In the case of the node-weighted setting, planarity helps to achieve constant factor approximations. The $NWSF$ can be expressed as the Hitting Set problem for some uncrossing family of cycles and hence solved as a feedback problem. This was exploited by Berman and Yaroslavtsev in [6] where they obtained a 2.4 approximation for $NWSF$ and other problems on planar graphs.

In [21] it was observed that using a threshold rounding technique together with the 2.4-approximation of Berman and Yaroslavtsev [6] for the version without penalties gives a 2.93-approximation algorithm for NWPCST on planar graphs. This was the best approximation guarantee up to date. However, such an algorithm requires solving an LP.

We summarize the current best known results in Table 1.

1.2 Our contribution

We propose a new LMP 3-approximation algorithm for NWPCST on planar graphs. The algorithm is an adaptation of the original technique developed by Goemans and Williamson in [15] for PCST to the node-weighted version. However, we change the pruning phase of the algorithm. This enables us to analyze the connection and penalty costs separately which is the key ingredient. In particular, we can directly charge the penalty costs to a part of the dual solution yielding Langrangian-multiplier-preservation. Further, the connection costs can be bounded using a slightly adapted analysis from [20] for NWSF. The approximation ratio of 3 is slightly higher than the previously best approximation ratio but the primal-dual algorithm does not require solving an LP.

Next, we establish a new best approximation ratio by exploiting the asymmetry of our primal-dual algorithm. A combination of the new LMP algorithm with a threshold rounding technique with the underlying 2.4-approximation from [6] yields a 2.88-approximation for NWPCST on planar graphs.

Furthermore, we obtain an efficient, direct primal-dual 4-approximation algorithm for NWPCSF on planar graphs building up on ideas for edge-weighted PCSF from [17]. This approach was previously indicated by Demaine et al. [9], but we give a better constant.

2 The LMP primal-dual 3-approximation algorithm

Consider an undirected graph $G = (V, E)$ with non-negative cost function and penalties on the vertices denoted by $w : V \to Q_+$ and $\pi : V \to Q_+$, respectively. In the NWPCST problem we are allowed to purchase a *connected* subgraph F of G that connects vertices to a prespecified root $r \in V$. Every bought vertex induces a cost according to w. Every vertex that is not included induces a penalty according to π. The objective is to minimize the sum of the purchase and penalty costs, *i.e.*, $\sum_{v \in F} w_v + \sum_{v \notin F} \pi_v$.

By a standard transformation we can assume that for every vertex v either its cost or its penalty is zero. To see this consider a single vertex v with both strictly positive cost and penalty. Add an additional vertex v', set its cost to zero and penalty to π_v, add an edge from v' to v and set the penalty of v to zero. Now, any solution in the original graph can be transformed to a solution of the same cost in the modified graph and vice-versa.

In the sequel, we call a vertex with a positive penalty a *terminal*. Terminals and the root can be purchased for free. Other vertices do not have a penalty and we call them non-terminals or Steiner vertices.

Let $\Gamma(S)$ denote the set of neighbors of S, *i.e.*, the set of vertices in $V \setminus S$ incident to vertices from $S \subseteq V$. Let also $\Pi(X) = \sum_{v \in X} \pi_v$. Thus, NWPCST is the following problem:

$$\min \sum_{v \in V} w_v x_v + \sum_{X \subseteq V \setminus \{r\}} \Pi(X) z_X \qquad (IP_{PCST})$$

$$s.t.$$

$$\sum_{v \in \Gamma(S)} x_v + \sum_{X : S \subseteq X} z_X \geq 1 \qquad \forall S \subseteq V \setminus \{r\}$$

$$x_v \in \{0, 1\} \qquad \forall v \in V$$

$$z_X \in \{0, 1\} \qquad \forall X \subseteq V \setminus \{r\}$$

By relaxing the integrality constraints to non-negativity constraints we obtain the standard

linear relaxation. The dual of this relaxation is

$$\max \sum_{S \subseteq V \setminus \{r\}} y_S \qquad (DLP_{PCST})$$

$$s.t.$$

$$\sum_{S: v \in \Gamma(S)} y_S \leq w_v \qquad \forall v \in V \qquad (1)$$

$$\sum_{S \subseteq X} y_S \leq \Pi(X) \qquad \forall X \subseteq V \setminus \{r\} \qquad (2)$$

$$y_S \geq 0 \qquad \forall S \subseteq V \setminus \{r\}$$

2.1 Algorithm

Now we shortly describe our primal-dual algorithm which is an adaptation of the generic moat-growing approach of Goemans and Williamson [15]. In each iteration i we maintain a set of already bought nodes F. We say that some vertex was bought at time i if it was bought in iteration i[1]. At the beginning F contains all terminals (including the root). We maintain also the set of connected components C of subgraph $G[F]$ induced by the vertices bought so far. We call each of these connected components a moat. Moats can be active or inactive. The moat containing root r is always inactive. In each iteration we increase (grow) dual variables corresponding to *all active* moats uniformly until one of the following two events happen:

- a vertex v goes tight (constraint (1) becomes equality), or
- a set X goes tight (constraint (2) becomes equality).

In the first case we buy vertex v and possibly merge moats incident to v. If we merge to a moat containing the root r, this moat becomes inactive, otherwise it is declared active.

In the second case, we declare the moat corresponding to set X inactive. Moreover, we mark all unmarked terminals inside X with the current time.

The growth phase terminates when there are no more active moats. After that, we have a pruning phase. In the pruning phase we restrict to the connected component of F containing the root and discard everything else. Let $F^{(r)}$ denote this component. Then, we consider vertices in $F^{(r)}$ in the reverse order of purchase. We delete vertex v (bought at time t) if it does not disconnect from r any terminal which was unmarked at time t. When we delete v, we further discard all vertices that become disconnected from r. As a result we output the set of bought vertices F' that survived pruning.

Our algorithm can be implemented with a notion of so-called potentials. Let $P(X) = \Pi(X) - \sum_{S \subseteq X} y_S$ be the potential of set X. Intuitively, we pay for the growth of moats (increase of dual variables) with potentials of these moats. If the potential of a moat goes to zero, the corresponding constraint becomes tight, so we have to make this moat inactive. When we merge moats to a new moat S by buying a vertex, we compute the potential of S by summing the potentials of the old moats.

[1] When we refer to time we always have in mind the number of the current iteration. Note that it implies that the speed of the uniform growth of dual budgets is not constant across iterations, but it does not affect our description of the algorithm.

2.2 Analysis

▶ **Theorem 1** (Lagrangian Multiplier Preservation). *Let G be planar. The algorithm described in the previous section outputs a set of vertices F' such that*

$$\sum_{v \in F'} w_v + 3\Pi(V \setminus F') \leq 3 \sum_{S \subseteq V \setminus \{r\}} y_S \leq 3 \; OPT \, .$$

In the proof we want to use the obtained dual solution y to account for the connection costs and penalties of the primal solution F'. We will partition the y_S into two sets. The first set will yield a bound on the connection costs and the second a bound on the penalties.

The key ingredient in the analysis is the partition that is based on the following lemma. Consider any iteration i and the active moats A_i before this iteration. Let $S \in A_i$ be an active moat that was not included in the final solution, *i.e.*, $S \cap F' = \emptyset$. Then, the dual variable of S did not contribute to buying any vertex in F'. This means that y_S does not contribute to the left-hand-side of the constraints (1) for any $v \in F'$. More formally, this means that S does not have a neighbor in F'.

▶ **Lemma 2.** *Let $S \in A_i$ be such that $S \subseteq V \setminus F'$. Then, the moat S does not have any neighbor in the solution, i.e. $F' \cap \Gamma(S) = \emptyset$.*

Proof of Lemma 2. Note that $S \in A_i$ means that S is active in iteration i and therefore there is an unmarked (before time i) terminal in S. Now, assume for a contradiction that $F' \cap \Gamma(S) \neq \emptyset$ and let $U \subseteq S$ be the set of vertices having a neighbor in F'. Note that all vertices in U were bought before iteration i because S is a connected component of the vertices bought before iteration i and $U \subseteq S$. Since S is not part of F', all the vertices in U must have been deleted in the pruning phase. A contradiction, since this would disconnect the unmarked (before time i) terminal in S. ◀

Following Lemma 2, we can partition all dual variables into the variables that contributed to buying the vertices of F' and the dual variables that account for the penalties induced by F'. Let CC be the set of all moats $S \subseteq V \setminus \{r\}$ that include a vertex of F' or have a neighbor in F', *i.e.*, $(S \cup \Gamma(S)) \cap F' \neq \emptyset$ and $y_S > 0$. Let PC be the set of all other moats, *i.e.*, sets S with $y_S > 0$ but $S \notin CC$. We will show that

$$\sum_{v \in F'} w_v \leq 3 \sum_{S \in CC} y_S \qquad \text{and} \qquad \Pi(V \setminus F') = \sum_{S \in PC} y_S$$

which yields Theorem 1.

To show the bound on the connection cost we make a degree counting argument. Here, we can leverage the analysis of the primal-dual algorithm for node-weighted Steiner forest given in [20]. Recall that our algorithm can also deactivate moats due to the penalty constraints. However, this fact does not generate problems. Intuitively, deactivating a moat corresponds to satisfying a demand pair in the forest problem. The proof of the following lemma only requires a minor change to the analysis.

▶ **Lemma 3** (Analog of Analysis in [20]). *Let F' be the output of the algorithm and A_i be the set of active moats before running iteration i. Then,*

$$\sum_{S \in A_i \cap CC} |F' \cap \Gamma(S)| \leq 3|A_i \cap CC|.$$

Proof. We outline the proof of Lemma 3. As indicated this proof is, except for a minor change, analogous to the proof used in [20] to show that the generic primal-dual algorithm for node-weighted Steiner forest on planar graphs has an approximation guarantee of 3.

Let F' be the output of the algorithm and A_i be the set of active moats before running iteration i. We want to show that

$$\sum_{S \in A_i \cap CC} |F' \cap \Gamma(S)| \leq 3|A_i \cap CC|. \tag{3}$$

In (3) we count the adjacencies between active moats at iteration i and vertices from F'. Let F_i be the set of vertices bought by the algorithm before iteration i. Consider a graph G' obtained from G in the following way:

1. take the subgraph of G induced by vertices from $F_i \cup F'$
2. keep only the connected component containing root r
3. contract each inactive moat (at iteration i) in this subgraph with a neighboring vertex (excluding the moat containing root)
4. contract each active moat in this component
5. contract the moat containing the root

Next, color the vertices of G' with three colors:

- white color for vertices obtained from contracting active moats
- blue color for the single vertex representing the moat containing the root
- black color for all other vertices, i.e. $F' \setminus F_i$

Observe now that deleting a black vertex in G' disconnects some white vertex from the blue vertex, because otherwise it would be deleted in the pruning phase. G' remains planar, since deletions and contractions preserve planarity. Moreover, it is easy to see that the number of adjacencies $\sum_{S \in A_i} |F' \cap \Gamma(S)|$ in G is the same as the number of edges between white and black vertices in G'.

To bound this number we will use the following result that is implicit in [20].

▶ **Lemma 4.** *Consider a simple connected planar graph $H = (V, E)$ in which vertices are colored with two colors: black and white, i.e. $V = B \cup W$. If for this graph the two following conditions hold*

- *there is no edge between any two white vertices*
- *removing any black vertex disconnects the graph*

then the number of edges between black and white vertices ($|E'|$) is at most 3 times greater than the number of white vertices, i.e., $|E'| \leq 3(|W| - 1)$

Before we prove Lemma 4, let us remark how it yields the claim. Consider for a moment the color of the blue vertex in G' to be white (resulting in graph H). Now removing a black vertex clearly splits the graph into multiple components, since it disconnects at least two white vertices (one of them is this recolored blue vertex). All other conditions of the lemma are satisfied. Applying Lemma 4 finishes the proof of Lemma 3, since $|A_i| = |W| - 1$. ◀

Proof of the Lemma 4. We follow the proof given in [21]. Consider the following operation on the graph H. Take any edge $e = (u, v)$ between two black vertices u and v in H.

- If u and v share a common white neighbor, then delete edge e.
- Otherwise contract u and v.

Observe that this operation preserves conditions of the lemma. Moreover it does not change the number of adjacencies between black and white vertices. Consider now the graph H' obtained by performing as many above operations as possible. The H' is bipartite since we contracted or deleted all edges between any two black vertices. The goal is now to bound

the number of edges in H'. The idea is to use the Euler's formula for planar graphs. But first we have to show a few claims about H'.

Let W and B denote the set of white and black vertices of H', respectively.

▶ **Fact 5.** $|B| \leq |W| - 1$.

Proof. Consider a breadth-first search tree T in H' rooted at any white vertex r_w. Since removing a black vertex splits the graph, all leaves of T are white. Recall that H' is bipartite. Thus each black vertex has at least one unique white child in T. Furthermore, r_w is the only white vertex that does not have a parent. This concludes the proof of Fact 5. ◀

Now, using Fact 5 instead of Claim 1.4 of [21] in the proof of Lemma 1.3 of [21] yields Lemma 4 ◀

To conclude the upper bound on the connection costs, note that constraint (1) is tight for all vertices $v \in F'$. This gives

$$\sum_{v \in F'} w_v = \sum_{v \in F'} \sum_{S : v \in \Gamma(S)} y_S = \sum_{S \subseteq V \setminus \{r\}} |F' \cap \Gamma(S)|\ y_S = \sum_{S \in CC} |F' \cap \Gamma(S)|\ y_S.$$

We will show that $\sum_{S \in CC} |F' \cap \Gamma(S)|\ y_S \leq 3 \sum_{S \in CC} y_S$ by induction on the number of iterations. At the beginning all dual variables are equal to 0 and the inequality holds. In iteration i we grow each active moat from $A_i \cap CC$ by ϵ_i. This increases the left-hand side by $\epsilon_i \sum_{S \in A_i \cap CC} |F' \cap \Gamma(S)|$ and the right-hand side by $3\epsilon_i |A_i \cap CC|$. Then, Lemma 3 concludes the proof of the bound on the connection costs.

In order to prove the bound on the penalties we employ the following lemma.

▶ **Lemma 6.** *Let F' and y_S be the primal and dual solution constructed by the algorithm. The set of vertices $X = V \setminus F'$ not spanned by the final solution can be partitioned into sets $X_1, X_2, \ldots X_l$ such that the potential of each set is 0, i.e., $P(X_k) = 0$ for each k.*

Proof. Observe that there are two ways for a vertex v to be in X: either it was never a part of the root component ($v \in V \setminus F^{(r)}$) or it was deleted in the pruning phase ($v \in F^{(r)}$). It is easy to see that $P(V \setminus F^{(r)}) = 0$. Each vertex in $V \setminus F^{(r)}$ was at the end a part of some inactive component not containing the root and hence the potentials of these components were 0. Or, it was never in any moat.

It remains to show that the set S of vertices disconnected from F' by pruning a vertex v can be partitioned into sets X_k for which $P(X_k) = 0$. Let t be the time when v was bought. Observe that every vertex u in the neighborhood $\Gamma(S)$ of S has been bought after time t or was not bought at all. Now, S contains only marked terminals at time t, otherwise v would not have been pruned. Hence, S is a union of inactive moats at time t. This gives the desired partition. ◀

Observe that PC is the set of all $S \subseteq X_i$ with $y_S > 0$. To conclude the bound on the penalties note that since all X_k have zero potential we have

$$\Pi(V \setminus F') = \sum_{k=1}^{l} \Pi(X_k) = \sum_{k=1}^{l} \sum_{S \subseteq X_k} y_S = \sum_{S \in PC} y_S.$$

3 Combination with threshold rounding

A standard technique to generalize primal-dual algorithms from Steiner tree problems to their price-collecting variations is to use *threshold rounding* (see Section 5.7 of [22] or [13]). Here, in the first step an LP formulation for the price-collecting version is solved over fractional variables. Then, we pick a threshold α and consider the vertices that are bought with value at least α to be terminals. In the second step, the primal-dual algorithm for the original Steiner tree problem is run on this set of terminals to obtain the final solution. We note that the resulting algorithm is deterministic because we can try all possible thresholds (at most one for every vertex). However, the analysis uses a randomization argument.

We observed in [21] that using threshold rounding in combination with the primal-dual 2.4-approximation for node-weighted Steiner forest by Berman and Yaroslavtsev [6] yields a 2.93-approximation for NWPCST on planar graphs.

In this section, we combine the previous LMP algorithm with the threshold rounding technique to gain an improved approximation factor of 2.88. Our approach is inspired by an idea of Goemans [14]. Intuitively, such an improvement is possible because the LMP approximation improves over the factor of 3 if the optimal solution induces a high penalty cost. In contrast, if the penalties are only a small part of the optimal solution's cost, threshold rounding can leverage the robustness of the underlying 2.4-approximation. Thus, by combining the two algorithms we can hedge their weaknesses.

3.1 Threshold rounding

We use the standard threshold rounding technique (cf. [22]). Consider the following LP

$$\min \sum_{v \in V} w_v x_v + \sum_{u \in V \setminus \{r\}} \pi_u y_u \qquad (LP_{thr})$$

$$s.t.$$

$$\sum_{v \in \Gamma(S)} x_v + y_u \geq 1 \qquad \forall S \subseteq V \setminus \{r\}, \quad u \in S$$

$$x_v \geq 0 \quad \forall v \in V \qquad y_u \geq 0 \quad \forall u \in V$$

This LP is equivalent to the LP used in the construction of the primal-dual LMP 3-approximation from Section 2. This was shown by Williamson for the edge-weighted variant (see section 7.4.1 of [23]), however arguments are identical in our case. This is due to the fact, that the mapping between feasible solutions leaves variables related to connection costs unchanged and constructs variables z based solely on y and vice-versa.

In the sequel, let (x^*, y^*) be the optimum solution to LP_{thr} with objective value OPT_{LP}. Further, if T is a solution to NWPCST, let $w(T)$ be the total connection and $\pi(V \setminus T)$ be the total penalties of T. We also use this notation for (fractional) solutions: $w(x)$, $\pi(z)$ and $\pi(y)$.

Let $\beta \in (0, 1)$ be a constant to be determined later. For every possible value α of y^* that is at most β, let $Q = \{u : y^*_u \leq \alpha\}$. Consider the instance I_{NWST_Q} of the $NWST$ problem which is derived from I_{NWPCST} by keeping only terminals from Q. Let LP_{NWST_Q} be the

following linear program

$$\begin{aligned} \min \sum_{v\in V} w_v x_v & & (LP_{NWST_Q}) \\ \sum_{v\in\Gamma(S)} x_v \geq 1 & & \forall S \subseteq V\setminus\{r\},\quad Q\cap S\neq\emptyset \\ x_v \geq 0 & & \forall v\in V \end{aligned}$$

Let OPT_{LP_Q} be the optimum objective function value of LP_{NWST_Q}. We run the 2.4-approximation algorithm for I_{NWST_Q} by Berman and Yaroslavtsev [6] which returns a solution F such that its cost is no greater than $2.4\cdot OPT_{LP_Q}$. Finally, return the best of all obtained solutions F (due to different values of α).

Though the algorithm is deterministic its analysis is based on a randomized argument. Instead of trying all possible values of α, consider α to be chosen uniformly at random from $[0,\beta]$. Consider $x' = \frac{1}{1-\alpha}x^*$. It follows that x' is a feasible solution to LP_{NWST_Q}. We bound the expected connection and penalty costs of F.

$$\begin{aligned} \mathbb{E}\left[\sum_{v\in F} w_v\right] &\leq \mathbb{E}\left[2.4\cdot OPT_{LP_Q}\right] \leq \mathbb{E}\left[2.4\sum_{v\in V} x'_v\cdot w_v\right] \leq \mathbb{E}\left[\frac{2.4}{1-\alpha}\right]\sum_{v\in V} x^*_v\cdot w_v \\ &= \left(\int_0^\beta \frac{1}{\beta}\cdot\frac{2.4}{1-\alpha}d\alpha\right) w(x^*) \\ &= \frac{2.4}{\beta}\ln\left(\frac{1}{1-\beta}\right) w(x^*) \end{aligned}$$

$$\begin{aligned} \mathbb{E}\left[\sum_{u\notin Q}\pi_u\right] &= \mathbb{E}\left[\sum_{u:y^*_u>\alpha}\pi_u\right] \leq \sum_u \pi_u Pr\left[y^*_u\geq\alpha\right] \leq \sum_u \pi_u\int_0^{y^*_u}\frac{1}{\beta}d\alpha \\ &= \sum_u \pi_u\frac{1}{\beta}y^*_u = \frac{1}{\beta}\pi(y^*) \end{aligned}$$

3.2 Combining the two algorithms

To combine the LMP approximation with threshold rounding we require a slight modification of the instance submitted to the LMP approximation.

Recall that for an instance I the LMP 3-approximation returns a solution T such that $w(T)+3\pi(V\setminus T)\leq 3OPT_{LP}$. Consider now instance I' with has its penalties scaled by 1/3, *i.e.*, $\pi'_v=\frac{1}{3}\pi_v$. Run the LMP approximation on I' to obtain a solution T' satisfying $w(T')+\pi(V\setminus T') = w(T')+3\pi'(V\setminus T')\leq 3OPT'_{LP}$, where OPT'_{LP} is the value of the optimum solution to program LP' derived from LP_{thr} by taking scaled penalties π'. Observe that (x^*,y^*) is also feasible to LP', because this program differs only in the objective function. Hence we have that

$$w(T')+\pi(V\setminus T')\leq 3OPT'_{LP}\leq 3\left(w(x^*)+\pi'(y^*)\right) = 3w(x^*)+\pi(y^*)$$

Now, our final algorithm returns the best solution among T' and the solution produced by the threshold rounding technique in the previous section. Note that this is a deterministic procedure. However, the analysis uses a randomized argument inspired by Goemans [14]:

pick one solution with probability p and the other with probability $1-p$. Let SOL be the returned solution.

$$\begin{aligned}\mathbb{E}\left[SOL\right] &\leq \left[3p + (1-p)\frac{2.4}{\beta}\ln\left(\frac{1}{1-\beta}\right)\right] w(x^*) + \left[p + (1-p)\frac{1}{\beta}\right]\pi(y^*) \\ &\leq \left[\left(3p+(1-p)\frac{2.4}{\beta}\ln\left(\frac{1}{1-\beta}\right)\right) w(x^*) + \left(p+(1-p)\frac{1}{\beta}\right)\pi(y^*)\right]\end{aligned}$$

Finally, optimizing constants we obtain for $\beta = 1 - e^{-\frac{5}{36}}$ and $p = \frac{1}{4-3e^{-5/36}}$ the claimed result

$$\begin{aligned}\mathbb{E}\left[SOL\right] &\leq \frac{4}{4-3e^{-5/36}}\left(w(x^*) + \pi(y^*)\right) \\ &\leq \frac{4}{4-3e^{-5/36}} OPT \approx 2.8797 \cdot OPT\end{aligned}$$

4 The primal-dual 4-approximation for forest

In this section we use a general combinatorial approach for solving prize-collecting problems introduced by Hajiaghayi and Jain [17]. In their work they obtained the primal-dual 3-approximation algorithm for edge-weighted prize-collecting Steiner forest problem. We adapt their argumentation to the planar node-weighted setting resulting in the 4-approximation algorithm. We provide here only a sketch - the more detailed description and proofs can be found in the full version of the paper [8].

Consider a graph $G = (V, E)$ with a non-negative cost function on nodes $w : V \to Q_+$, a set of pairs of vertices (demands) $D = (s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$ and a non-negative penalty function $\pi : D \to Q_+$. In the node-weighted prize-collecting Steiner forest problem we are asked to find a set of vertices $F \subseteq V$ which minimizes the sum of costs of vertices in F plus penalties for pairs of vertices which are not connected in a subgraph of G induced by F.

Note that we can give an equivalent definition of demands and penalties by specifying penalties for each unordered pair of vertices. Simply set penalties for pairs of vertices which are not in D to 0. From now on we will use values π_{ij} to denote penalties. Let also $\Gamma(S)$ denote the set of vertices in $V \setminus S$ incident to vertices from $S \subseteq V$ and let $S \odot (i,j)$ means that $|(i,j) \cap S| = 1$ (i.e., S separates vertices i and j) Using this notation, we can formulate our problem with the following integer program

$$\min \sum_{v \in V} w_v x_v + \sum_{(i,j) \in V \times V} \pi_{ij} z_{ij} \qquad (IP_{SF})$$

$$s.t.$$

$$\sum_{v \in \Gamma(S)} x_v + z_{i,j} \geq 1 \qquad \forall S \subseteq V, \quad \forall (i,j) \in V \times V : \quad S \odot (i,j)$$

$$x_v \in \{0,1\} \qquad \forall v \in V$$

$$z_{i,j} \in \{0,1\} \qquad \forall (i,j) \in S \times S$$

Setting $x_v = 1$ corresponds to buying a vertex v (including v into solution F) and setting $z_{i,j} = 1$ corresponds to paying a penalty instead of connecting vertices i and j.

Unfortunately, the dual of the linear relaxation of IP_{SF} is not suitable for obtaining a primal-dual algorithm. However, following the framework in [17], we can replace it with the

following LP:

$$\max \sum_{S \subset V} y_S \qquad (DLP_{SF}4)$$

$$s.t.$$

$$\sum_{S: v \in \Gamma(S)} y_S \leq w_v \qquad \forall v \in V \qquad (4)$$

$$\sum_{S \in \mathbb{S}} y_S \leq \sum_{(i,j) \in V \times V, \mathbb{S} \odot (i,j)} \pi_{i,j} \qquad \forall \mathbb{S} \in 2^{2^V} \qquad (5)$$

$$y_S \geq 0 \qquad \forall S \subset V$$

where $\mathbb{S} \odot (i,j)$ denotes that there exists $S \in \mathbb{S}$ such that $S \odot (i,j)$ (we say that family $\mathbb{S}$ separates vertices i and j if and only if there exists at least one set $S \in \mathbb{S}$ which separates vertices i and j).

This new formulation allows us to obtain a natural primal-dual algorithm which is described below.

The algorithm starts with an initial solution F in which there are all vertices of cost 0 (hence all terminals). In each iteration the algorithm maintains moats which are the connected components of graph G induced by the vertices of the current solution F. Demands can be marked (meaning that we decide to pay a penalty for them) or unmarked. At the beginning all demands are unmarked. Once demand is marked, it stays marked forever. A moat (denoted by the corresponding set $S \subseteq V$) is active in the current iteration if and only if there is at least one unmarked demand (i,j) such that $S \odot (i,j)$. Now in each iteration we simultaneously grow each active moat until one of the following two events occur:

- a vertex v goes tight (constraint (4) becomes equality), or
- a family $\mathbb{S}$ goes tight (constraint (5) becomes equality).

In the first case we simply add v to our solution F (which may make some moats inactive) and continue to the next iteration.

In the second case, we mark each demand (i,j) such that $\mathbb{S} \odot (i,j)$. Hence in the following iterations all moats from $\mathbb{S}$ will be inactive, and we will not violate any constraint during the growth process. We repeat this process until all moats become inactive.

After that we have an additional pruning phase in which we process all vertices of F in the reverse order of buying. We remove a vertex v from F if after its removal from F, all unmarked demands are still connected in the graph induced by F. We output this pruned set of vertices as F' which is our final solution.

Obtaining ϵ_1 and a tight vertex in line 7 is straightforward. On the other hand obtaining ϵ_2 in line 8 and a tight family $\mathbb{S}$ seems to be much harder, since the number of corresponding constraint is doubly exponential. Fortunately Hajiaghayi and Jain in section 4 of [17] gave a polynomial time algorithm for computing ϵ_2 and the corresponding tight family $\mathbb{S}$.

Since the algorithm terminates after at most $2|V| - 1$ iterations (in each iteration the number of active moats or the number of connected components decreases), the running time of this algorithm is polynomial.

We can combine proofs from [20] and [17] in order to obtain the following result.

▶ **Theorem 7.** *The algorithm outputs a set of vertices F' and a set of demands Q' which are not connected via F' such that*

$$\sum_{v \in F'} w_v + \sum_{(i,j) \in Q'} \pi_{ij} \leq 4 \sum_{S \subseteq V} y_S \leq 4 \; OPT \, .$$

Input : A planar graph $G = (V, E)$ with non-negative weights w_i on the nodes and non-negative penalties π_{ij} between each pair of vertices such that if $\pi_{ij} > 0$ then $w_i = 0$ and $w_j = 0$

Output : A set of vertices F' representing a forest and a set of pairs Q' representing not connected demands

begin
  $F \leftarrow \{v_i \in V : w_i = 0\}$;
  $Q \leftarrow \emptyset$ `// set all demands unmarked`
  $y_S \leftarrow 0$ `// implicitly`
  $AM \leftarrow \left\{ S \subseteq V : S \in SCC\left(G[F]\right) \wedge \underset{(i,j)\in V\times V-Q}{\exists} \pi_{ij} > 0 \wedge S \odot (i,j) \right\}$;
  `// identify active moats as components of subgraph of` G `induced by vertices` F `for which there is at least one unmarked demand` (i,j) `which is separated by the corresponding set`
  while $AM \neq \emptyset$ **do**
    find minimum ϵ_1 s.t if we increase y_S for each $S \in AM$ by ϵ_1 we get a new tight vertex v;
    find minimum ϵ_2 s.t if we increase y_S for each $S \in AM$ by ϵ_2 we get a new tight family $\mathbb{S}$;
    $\epsilon \leftarrow min(\epsilon_1, \epsilon_2)$;
    $y_S \leftarrow y_S + \epsilon$ for all $S \in AM$;
    if $\epsilon = \epsilon_1$ **then**
      $F \leftarrow F \cup \{v\}$;
    else
      $Q \leftarrow Q \cup \{(i,j) \in V \times V : \mathbb{S} \odot (i,j)\}$
    end
    $AM \leftarrow \left\{ S \subseteq V : S \in SCC\left(G[F]\right) \wedge \underset{(i,j)\in V\times V-Q}{\exists} \pi_{ij} > 0 \wedge S \odot (i,j) \right\}$;
  end
  `// pruning phase`
  Derive F' from F by removing vertices in reverse order of purchase so that every unmarked demand is connected in F'.
  Let Q' be all demands not connected via F'
end

Algorithm 1: Primal-dual algorithm for NWPCSF on planar graphs.

We need to show that:

$$\sum_{v \in F'} w_v \leq 3 \sum_{S \subseteq V} y_S \qquad \text{and} \qquad \sum_{(i,j) \in Q'} \pi_{ij} \leq \sum_{S \subseteq V} y_S$$

The bound on the connection cost is shown in a similar way as in the tree version, *i.e.*, using a degree counting argument for each iteration. This is captured by the lemma below.

For a set of nodes F and the set of unmarked demands $R = D - Q$ define a minimal feasible augmentation F_{aug} of F with respect to R to be a set of vertices F_{aug} containing F as a subset such that every pair of vertices from R is connected in the subgraph of G induced by F_{aug} and such that removal of any $v \in F_{aug} \setminus F$ from F_{aug} disconnects some pair from R.

▶ **Lemma 8** (Analog of Analysis in [20]). *Let G be planar, R be the set of unmarked demands after running the above algorithm, F_j be the set of bought vertices before running iteration j and F_{aug} be a minimal feasible augmentation of F_j with respect to R. Let also A_j be the set of active moats before running iteration j. Then*

$$\sum_{S \in A_j} |F_{aug} \cap \Gamma(S)| \leq 3|A_j| \,.$$

The proof of this lemma is conducted in a similar way as the proof of Lemma 3 and the analysis is essentially the same as in [20].

In turn, the bound on penalties is shown exactly in the same way as in the edge-weighted version [17]. When we mark a pair it belongs to a tight family. It is observed that the union of those tight families is also tight, hence the corresponding constraint gives the bound.

The more detailed proofs of these bounds can be found in in the full version of the paper [8].

Note that we cannot separate dual variables like in the tree version, hence we obtain a factor of 4 instead of 3 as in Section 2. This is essentially due to the same difficulty as in the standard edge-weighted variant for the prize-collecting Steiner forest problem.

References

1. Ajit Agrawal, Philip N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995.
2. Aaron Archer, MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Howard J. Karloff. Improved approximation algorithms for prize-collecting steiner tree and TSP. *SIAM J. Comput.*, 40(2):309–332, 2011.
3. MohammadHossein Bateni, Chandra Chekuri, Alina Ene, Mohammad Taghi Hajiaghayi, Nitish Korula, and Dániel Marx. Prize-collecting steiner problems on planar graphs. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1028–1049, 2011.
4. MohammadHossein Bateni, Mohammad Taghi Hajiaghayi, and Dániel Marx. Approximation schemes for steiner forest on planar graphs and graphs of bounded treewidth. *J. ACM*, 58(5):21, 2011.
5. MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Vahid Liaghat. Improved approximation algorithms for (budgeted) node-weighted steiner problems. In *Automata, Languages, and Programming – 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 81–92, 2013.

6 Piotr Berman and Grigory Yaroslavtsev. Primal-dual approximation algorithms for node-weighted network design in planar graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques – 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, pages 50–60, 2012.

7 Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. An improved lp-based approximation for steiner tree. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 583–592, 2010.

8 Jarosław Byrka, Mateusz Lewandowski, and Carsten Moldenhauer. Approximation algorithms for node-weighted prize-collecting steiner tree problems on planar graphs. *CoRR*, abs/1601.02481, 2016.

9 Erik D. Demaine, Mohammad Taghi Hajiaghayi, and Philip N. Klein. Node-weighted steiner tree and group steiner tree in planar graphs. *ACM Trans. Algorithms*, 10(3):13:1–13:20, 2014.

10 Bistra N. Dilkina and Carla P. Gomes. Solving connected subgraph problems in wildlife conservation. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 7th International Conference, CPAIOR 2010, Bologna, Italy, June 14-18, 2010. Proceedings*, pages 102–116, 2010.

11 Karoline Faust, Pierre Dupont, Jérôme Callut, and Jacques van Helden. Pathway discovery in metabolic networks by subgraph extraction. *Bioinformatics*, 26(9):1211–1218, 2010.

12 M. R. Garey and David S. Johnson. The rectilinear steiner tree problem in NP complete. *SIAM Journal of Applied Mathematics*, 32:826–834, 1977.

13 Joseph Geunes, Retsef Levi, H. Edwin Romeijn, and David B. Shmoys. Approximation algorithms for supply chain planning and logistics problems with market choice. *Math. Program.*, 130(1):85–106, 2011.

14 Michel X. Goemans. Combining approximation algorithms for the prize-collecting TSP. *CoRR*, abs/0910.0553, 2009.

15 Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.

16 Sudipto Guha, Anna Moss, Joseph Naor, and Baruch Schieber. Efficient recovery from power outage (extended abstract). In *Proc. of the 31st Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 574–582, 1999.

17 Mohammad Taghi Hajiaghayi and Kamal Jain. The prize-collecting generalized steiner tree problem via a new approach of primal-dual schema. In *Proc. of the 7th Annual ACM-SIAM Symp. on Discrete Algorithms, SODA'06*, pages 631–640, 2006.

18 Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

19 Jochen Könemann, Sina Sadeghian Sadeghabad, and Laura Sanità. An LMP o(log n)-approximation algorithm for node weighted prize collecting steiner tree. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 568–577, 2013.

20 Carsten Moldenhauer. Primal-dual approximation algorithms for node-weighted steiner forest on planar graphs. *Inf. Comput.*, 222:293–306, 2013.

21 Carsten Moldenhauer. *Node-weighted network design and maximum sub-determinants*. PhD thesis, EPFL, 2014.

22 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

23 David Paul Williamson. *On the design of approximation algorithms for a class of graph problems*. PhD thesis, MIT, Cambridge, MA, September 1993.

A Logarithmic Integrality Gap Bound for Directed Steiner Tree in Quasi-bipartite Graphs*

Zachary Friggstad[1], Jochen Könemann[2], and Mohammad Shadravan[3]

1 **Department of Computing Science, University of Alberta, Edmonton, Canada**
zacharyf@ualberta.ca
2 **Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Canada**
jochen@uwaterloo.ca
3 **Department of Industrial Engineering and Operations Research, Columbia University, New York, USA**
ms4961@columbia.edu

Abstract

We demonstrate that the integrality gap of the natural cut-based LP relaxation for the directed Steiner tree problem is $O(\log k)$ in quasi-bipartite graphs with k terminals. Such instances can be seen to generalize set cover, so the integrality gap analysis is tight up to a constant factor. A novel aspect of our approach is that we use the primal-dual method; a technique that is rarely used in designing approximation algorithms for network design problems in directed graphs.

1998 ACM Subject Classification G.1.6 Optimization, G.2.2 Graph Theory, I.1.2 Artificial Intelligence

Keywords and phrases Approximation algorithm, Primal-Dual algorithm, Directed Steiner tree

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.3

1 Introduction

In an instance of the *directed Steiner tree* (DST) problem, we are given a directed graph $G = (V, E)$, non-negative costs c_e for all $e \in E$, *terminal* nodes $X \subseteq V$, and a root $r \in V$. The remaining nodes in $V - (X \cup \{r\})$ are the *Steiner nodes*. The goal is to find the cheapest collection of edges $F \subseteq E$ such that for every terminal $t \in X$ there is an r, t-path using only edges in F. Throughout, we let n denote $|V|$ and k denote $|X|$.

If $X \cup \{r\} = V$, then the problem is simply the *minimum-cost arborescence* problem which can be solved efficiently [5]. However, the general case is well-known to be NP-hard. In fact, the problem can be seen to generalize the *set-cover* and *group Steiner tree* problems. The latter cannot be approximated within $O(\log^{2-\epsilon}(n))$ for any constant $\epsilon > 0$ unless NP $\subseteq$ DTIME$(n^{\text{polylog}(n)})$ [11].

For a DST instance G, let OPT_G denote the value of the optimum solution for this instance Say that an instance $G = (V, E)$ of DST with terminals X is ℓ-*layered* if V can be partitioned as $V_0, V_1, \ldots, V_\ell$ where $V_0 = \{r\}, V_\ell = X$ and every edge $uv \in E$ has $u \in V_i$

* This work was in part completed while the first author was a postdoctoral fellow and the third author was a graduate student at the University of Waterloo. The work of all three authors is supported by NSERC's Discovery grant program. The second author greatfully acknowledges the support of the Hausdorff Institute and the Institute for Discrete Mathematics in Bonn, Germany.

and $v \in V_{i+1}$ for some $0 \leq i < \ell$. Zelikovsky showed for any DST instance G and integer $\ell \geq 1$ that we can compute an ℓ-layered DST instance H in poly(n, ℓ) time such that $OPT_G \leq OPT_H \leq \ell \cdot k^{1/\ell} \cdot OPT_G$ and that a DST solution in H can be efficiently mapped to a DST solution in G with the same cost [2, 17].

Charikar et al. [3] exploited this fact and presented an $O(\ell^2 \cdot k^{1/\ell} \cdot \log k)$-approximation with running time poly(n, k^ℓ) for any integer $\ell \geq 1$. In particular, this can be used to obtain an $O(\log^3 k)$-approximation in quasi-polynomial time and a polynomial-time $O(k^\epsilon)$-approximation for any constant $\epsilon > 0$. Finding a polynomial-time polylogarithmic approximation remains an important open problem.

For a set of nodes S, we let $\delta^{in}(S) = \{uv \in V : u \notin S \text{ and } v \in S\}$ be the set of edges entering S. The following is a natural linear programming (LP) relaxation for directed Steiner tree.

$$\min \quad \sum_{e \in E} c_e x_e \qquad \textbf{(DST-Primal)}$$

$$\text{s.t.} \quad x(\delta^{\text{in}}(S)) \geq 1 \quad \forall\, S \subseteq V - r, S \cap X \neq \emptyset \qquad (1)$$

$$x_e \geq 0 \quad \forall e \in E$$

This LP is called a *relaxation* because of the natural correspondence between feasible solutions to a DST instance G and feasible $\{0, 1\}$-integer solutions to the corresponding LP (**DST-Primal**). Thus, if we let OPT_{LP} denote the value of an optimum (possibly fractional) solution to LP (**DST-Primal**) then we have $OPT_{LP} \leq OPT_G$. For a particular instance G we say the *integrality gap* is OPT_G / OPT_{LP}; we are interested in placing the smallest possible upper bound on this quantity.

Interestingly, if $|X| = 1$ (the shortest path problem) or $X \cup \{r\} = V$ (the minimum-cost arborescence problem), the extreme points of (**DST-Primal**) are integral so the integrality gap is 1 ([13] and [5], respectively). However, in the general case Zosin and Khuller showed that (**DST-Primal**) is not useful for finding polylog(k)-approximation algorithms for DST [18]. The authors showed that the integrality gap of (**DST-Primal**) relaxation can, unfortunately, be as bad as $\Omega(\sqrt{k})$, even in instances where G is a 4-layered graph. In their examples, the number of nodes n is exponential in k so the integrality gap may still be $O(\log^c n)$ for some constant c.

On the other hand, Rothvoss recently showed that applying $O(l)$ rounds of the semidefinite programming Lasserre hierarchy to (the flow-based extended formulation of) (**DST-Primal**) yields an SDP with integrality gap $O(\ell \cdot \log k)$ for ℓ-layered instances [15]. Subsequently, Friggstad et al. [7] showed similar results for the weaker Sherali-Adams and Lovász-Schrijver linear programming hierarchies.

In this paper we consider the class of *quasi-bipartite* DST instances. An instance of DST is quasi-bipartite if the Steiner nodes $V \setminus (X \cup \{r\})$ form an independent set (i.e., no directed edge has both endpoints in $V \setminus (X \cup \{r\})$). Such instances still capture the set cover problem, and thus do not admit an $(1 - \epsilon) \ln k$-approximation for any constant $\epsilon > 0$ unless P = NP [4, 6]. Furthermore, it is straightforward to adapt known integrality gap constructions for set cover (e.g. [16]) to show that the integrality gap of (**DST-Primal**) can be as bad as $(1 - o(1)) \cdot \ln k$ in some instances. Hibi and Fujito [12] give an $O(\log k)$-approximation for quasi-bipartite instances of DST, but do not provide any integrality gap bounds.

Quasi-bipartite instances have been well-studied in the context of *undirected* Steiner trees. The class of graphs was first introduced by Rajagopalan and Vazirani [14] who studied the integrality gap of (**DST-Primal**) for the *bidirected* map of the given undirected Steiner

tree instances. Currently, the best approximation for quasi-bipartite instances of undirected Steiner tree is $\frac{73}{60}$ by Goemans et al. [8] who also bound the integrality gap of the bidirected cut relaxation by the same quantity. This is the same LP relaxation as (**DST-Primal**), applied to the directed graph obtained by replacing each undirected edge $\{u, v\}$ with the two directed edges uv and vu. This is a slight improvement over a prior $(\frac{73}{60} + \epsilon)$-approximation for any constant $\epsilon > 0$ by Byrka et al. [1].

The best approximation for general instances of undirected Steiner tree is $\ln(4) + \epsilon$ for any constant $\epsilon > 0$ [1]. However, the best known upper bound on the integrality gap of the bidirected cut relaxation for non-quasi-bipartite instances is only 2; it is an open problem to determine if this integrality gap is a constant-factor better than 2.

1.1 Our contributions

Our main result is the following. Let $H_n = \sum_{i=1}^{n} 1/i = O(\log n)$ be the nth harmonic number.

▶ **Theorem 1.** *The integrality gap of LP (**DST-Primal**) is at most $2\,H_k = O(\log k)$ in quasi-bipartite graphs with k terminals. Furthermore, a Steiner tree with cost at most $2\,H_k \cdot OPT_{LP}$ can be constructed in polynomial time.*

As noted above, Theorem 1 is asymptotically tight since any of the well-known $\Omega(\log k)$ integrality gap constructions for set cover instances with k items translate directly to an integrality gap lower bound for (**DST-Primal**), using the usual reduction from set cover to 2-layered quasi-bipartite instances of directed Steiner tree.

This integrality gap bound asymptotically matches the approximation guarantee proven by Hibi and Fujito for quasi-bipartite DST instaces [12]. We remark that their approach is unlikely to give any integrality gap bounds for (**DST-Primal**) because they iteratively choose *low-density full Steiner trees* in the same spirit as [3] and give an $O(\ell \cdot \log k)$-approximation for finding the optimum DST solution T that does not contain a path with $\geq \ell$ Steiner nodes $V \setminus (X \cup \{r\})$. In particular, their approach will also find an $O(\log k)$-approximation to the optimum DST solution in 4-layered graphs and we know the integrality gap in some 4-layered instances is $\Omega(\sqrt{k})$ [18].

We prove Theorem 1 by constructing a directed Steiner tree in an iterative manner. An iteration starts with a *partial* Steiner tree (see Definition 2 below), which consists of multiple directed components containing the terminals in X. Then a set of arcs are purchased to *augment* this partial solution to one with fewer directed components. These arcs are discovered through a primal-dual moat growing procedure; a feasible solution for the dual (**DST-Primal**) is constructed and the cost of the purchased arcs can be bounded using this dual solution.

While the primal-dual technique has been very successful for *undirected* network design problems (e.g., see [9]), far fewer success stories are known in *directed* domains. Examples include a primal-dual interpretation of Dijkstra's shortest path algorithm (e.g., see Chapter 5.4 of [13]), and Edmonds' [5] algorithm for minimum-cost arborescences. In both cases, the special structure of the problem is instrumental in the primal-dual construction. One issue arising in the implementation of primal-dual approaches for directed network design problems appears to be a certain *overlap* in the *moat* structure maintained by these algorithms. We are able to handle this difficulty here by exploiting the quasi-bipartite nature of our instances.

2 The integrality gap bound

2.1 Preliminaries and definitions

We now present an algorithmic proof of Theorem 1. As we will follow a primal-dual strategy, we first present the LP dual of (**DST-Primal**).

$$\max \quad \sum_{S} y_S \qquad \text{(DST-Dual)}$$

$$\text{s.t.} \quad \sum_{S: e \in \delta^{\text{in}}(S)} y_S \leq c_e \ \forall e \in E \qquad (2)$$

$$y \geq 0$$

In (**DST-Dual**), the sums range only over sets of nodes S such that $S \subseteq V - r$ and $S \cap X \neq \emptyset$.

Our algorithm builds up partial solutions, which are defined as follows.

▶ **Definition 2.** A *partial Steiner tree* is a tuple $\mathcal{T} = (\{B_i, h_i, F_i\}_{i=0}^{\ell}, \bar{B})$ where, for each $0 \leq i \leq \ell$, B_i is a subset of nodes, $h_i \in B_i$, and F_i is a subset of edges with endpoints only in B_i such that the following hold.

- The sets $B_0, B_1, \ldots, B_\ell, \bar{B}$ form a partition V.
- $\bar{B} \subseteq V - X - r$ (i.e. $\bar{B}$ is a subset of Steiner nodes).
- $h_0 = r$ and $h_i \in X$ for each $1 \leq i \leq \ell$.
- For every $0 \leq i \leq \ell$ and every $v \in B_i$, F_i contains an h_i, v-path.

We say that $\bar{B}$ is the set of *free Steiner nodes* in $\mathcal{T}$ and that h_i is the *head* of B_i for each $0 \leq i \leq \ell$. The edges of $\mathcal{T}$, denoted by $E(\mathcal{T})$, are simply $\cup_{i=0}^{\ell} F_i$. We say that $B_0, \ldots, B_\ell$ are the *components* of $\mathcal{T}$ where B_0 is the *root component* and $B_1, \ldots, B_\ell$ are the *non-root components*.

Figure 1 illustrates a partial Steiner tree. Note that if $\mathcal{T}$ is a partial Steiner tree with $\ell = 0$ non-root components, then $E(\mathcal{T})$ is in fact a feasible DST solution.

Finally, for a subset of edges F we let $\text{cost}(F) = \sum_{e \in F} c_e$.

2.2 High-level approach

Our algorithm builds up partial Steiner trees in an iterative manner while ensuring that the cost does not increase by a significant amount between iterations. Specifically, we prove the following lemma in Section 3. Recall that OPT_{LP} refers to the optimum solution value for (**DST-Primal**).

▶ **Lemma 3.** *Given a partial Steiner tree $\mathcal{T}$ with $\ell \geq 1$ non-root components, there is a polynomial-time algorithm that finds a partial Steiner tree $\mathcal{T}'$ with $\ell' < \ell$ non-root components such that*

$$\text{cost}(E(\mathcal{T}')) \leq \text{cost}(E(\mathcal{T})) + 2 \cdot OPT_{LP} \cdot \frac{\ell - \ell'}{\ell}.$$

Theorem 1 follows from Lemma 3 in a standard way.

Proof of Theorem 1. Initialize a partial Steiner tree $\mathcal{T}_k$ with k non-root components as follows. Let $\bar{B}$ be the set of all Steiner nodes, $B_0 = \{r\}$, and $F_0 = \emptyset$. Furthermore, label the terminals as $t_1, \ldots, t_k \in X$ and for each $1 \leq i \leq k$ let $B_i = \{t_i\}, h_i = t_i$ and $F_i = \emptyset$. Note that $\text{cost}(E(\mathcal{T}_k)) = 0$.

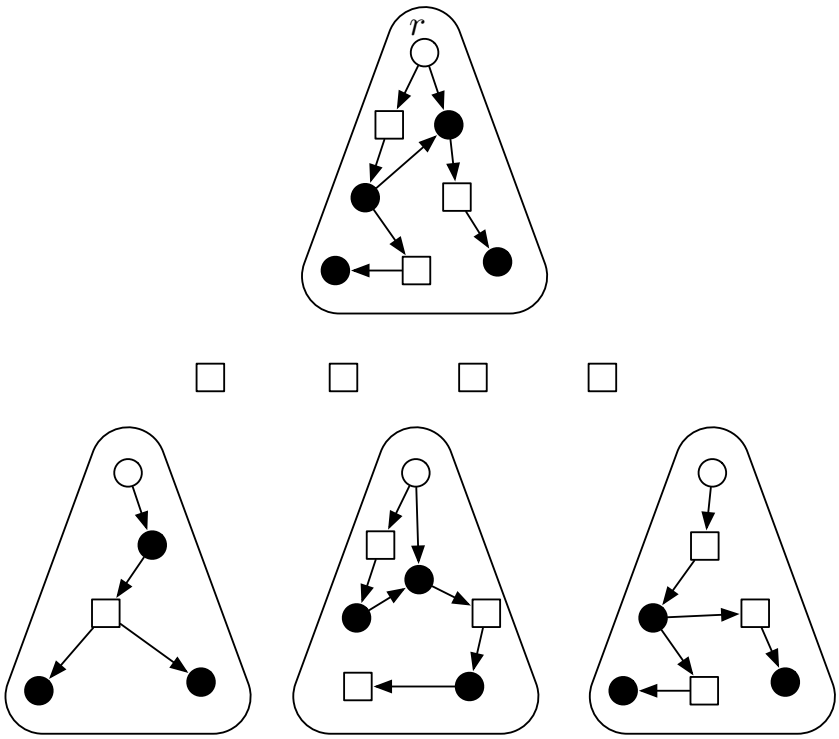


Figure 1 A partial Steiner tree with $\ell = 3$ non-root components (the root is pictured at the top). The only edges shown are those in some F_i. The white circles are the heads of the various sets B_i and the black circles are terminals that are not heads of any components. The squares outside of the components are the free Steiner nodes $\bar{B}$. Note, in particular, that each head can reach every node in its respective component. We do not require each F_i to be a minimal set of edges with this property.

Iterate Lemma 3 to obtain a sequence of partial Steiner trees $\mathcal{T}_{\ell_0}, \mathcal{T}_{\ell_1}, \mathcal{T}_{\ell_2}, \ldots, \mathcal{T}_{\ell_a}$ where $\mathcal{T}_{\ell_i}$ has ℓ_i non-root components such that $k = \ell_0 > \ell_1 > \ldots > \ell_a = 0$ and

$$\text{cost}(E(\mathcal{T}_{i+1})) \leq \text{cost}(E(\mathcal{T}_i)) + 2 \cdot OPT_{LP} \cdot \frac{\ell_i - \ell_{i+1}}{\ell_i}$$

for each $0 \leq i < a$. Return $E(\mathcal{T}_{\ell_a})$ as the final Steiner tree.

That $E(\mathcal{T}_a)$ can be found efficiently follows simply because we are iterating the efficient algorithm from Lemma 3 at most k times. The cost of this Steiner tree can be bounded as follows.

$$\begin{aligned}
\text{cost}(E(\mathcal{T}_a)) &\leq 2 \cdot OPT_{LP} \cdot \sum_{i=0}^{a-1} \frac{\ell_i - \ell_{i+1}}{\ell_i} = 2 \cdot OPT_{LP} \cdot \sum_{i=0}^{a-1} \sum_{j=\ell_{i+1}+1}^{\ell_i} \frac{1}{\ell_i} \\
&\leq 2 \cdot OPT_{LP} \cdot \sum_{i=0}^{a-1} \sum_{j=\ell_{i+1}+1}^{\ell_i} \frac{1}{j} = 2 \cdot OPT_{LP} \cdot \sum_{j=1}^{k} \frac{1}{k} \\
&= 2 \cdot OPT_{LP} \cdot H_k.
\end{aligned}$$

◀

The idea presented above resembles one proposed by Guha et al. [10] for bounding the integrality gap of a natural relaxation for *undirected* node-weighted Steiner tree by $O(\log k)$ [10]. Like our approach, Guha et al. also build a solution incrementally. In each *phase* of the algorithm, the authors reduce the number of connected components of a partial solution by adding vertices whose cost is charged carefully to the value of a dual LP solution that the algorithm constructs simultaneously.

3 A primal-dual proof of Lemma 3

Consider a given partial Steiner tree $\mathcal{T} = (\{B_i, h_i, F_i\}_{i=0}^{\ell}, \bar{B})$ with $\ell \geq 1$ non-root components. Lemma 3 promises a partial Steiner tree $\mathcal{T}'$ with $\ell' < \ell$ non-root components with $\text{cost}(E(\mathcal{T}')) \leq \text{cost}(E(\mathcal{T})) + 2 \cdot OPT_{LP} \cdot \frac{\ell - \ell'}{\ell}$. In this section we will present an algorithm that *augments* forest $\mathcal{T}$ in the sense that it computes a set of edges to *add* to $\mathcal{T}$. The proof presented here is constructive: we will design a *primal-dual* algorithm that maintains a feasible dual solution for (**DST-Dual**), and uses the structure of this solution to guide the process of adding edges to $\mathcal{T}$.

3.1 The algorithm

For any two nodes $u, v \in V$, let $d(u, v)$ be the cost of the cheapest u, v-path in G. More generally, for a subset $\emptyset \subsetneq S \subseteq V$ and a node $v \in V$ we let $d(S, v) = \min_{u \in S} d(u, v)$. We will assume that for every $0 \leq i \leq \ell$ and $1 \leq j \leq \ell, j \neq i$ that $d(B_i, h_j) > 0$ as otherwise, we could merge B_i and B_j by adding the 0-cost B_i, h_j-path to $\mathcal{T}$.

The usual conventions of primal-dual algorithms will be adopted. We think of such an algorithm as a continuous process that increases the value of some dual variables over time. At time $t = 0$, all dual variables are initialized to a value of 0. At any point in time, exactly ℓ dual variables will be raised at a rate of one unit per time unit. We will use Δ for the time at which the algorithm terminates. As is customary, we will say that an edge e *goes tight* if the dual constraint for e becomes tight as the dual variables are being increased. When an edge goes tight, we will perform some updates to the various sets being maintained by the algorithm. Again, the standard convention applies that if multiple edges go tight at the same time, then we process them in any order.

Algorithm 1 describes the main subroutine that augments the partial Steiner tree $\mathcal{T}$ to one with fewer components. It maintains a collection of *moats* $M_i \subseteq V - \{r\}$ and edges F_i' for each $1 \leq i \leq \ell$, while ensuring that the dual solution y it grows remains feasible. Mainly to aid notation, our algorithm will maintain a so called *virtual body* β_i for all $0 \leq i \leq \ell$ such that $B_i \subseteq \beta_i \subseteq B_i \cup \bar{B}$. We will ensure that each $v \in \bar{B} \cap \beta_i$ has a *mate* $u \in B_i$ such that the edge uv has cost no more than Δ. For notational convenience, we will let $\beta_0 = B_0$ be the virtual body of the root component. The algorithm will not grow a moat around the root since dual variables do not exist for sets containing the root.

Our algorithm will ensure that moats are pairwise *terminal-disjoint*. In fact, we ensure that any two moats may only intersect in $\bar{B}$. Terminal-disjointness together with the quasi-bipartite structure of the input graph will allow us to charge the cost of arcs added in the augmentation process to the duals grown.

An intuitive overview of our process is the following. At any time $t \geq 0$, the moats M_i will consist of all nodes v with $d(v, h_i) \leq t$. The moats M_i will be grown until, at some time Δ, for at least one pair i, j with $i \neq j$, there is a tight path connecting β_j to h_i. At this point the algorithm stops, and adds a carefully chosen collection of tight arcs to the partial Steiner tree that merges B_j and B_i (and potentially other components). Crucially, the cost

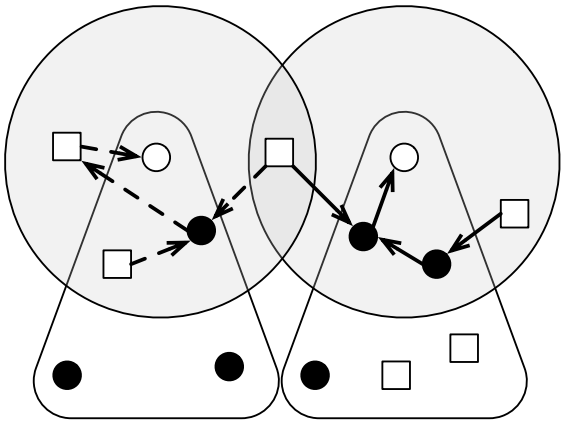


Figure 2 The moats around the two partial Steiner trees are depicted by the gray circles. The dashed edges are those bought by the first moat and the solid edges are those bought by the second moat. Note the moats only intersect in $\overline{B}$ (in particular, v is the only lying in both moats). Also, u lies in the virtual body for the left partial Steiner tree and the dashed arc entering u is coming from its mate. The edges F_i from the original partial Steiner trees are not shown.
Observe that if any edge entering v goes tight then it must be from either r or some terminal (because G is quasi-bipartite). This would allow us to merge at least one partial Steiner tree into the body of another.

of the added arcs will be *charged* to the value of the dual solution grown around the merged components.

Due the structure of quasi-bipartite graphs, we are able to ensure that in each step of the algorithm the active moats pay for at most one arc that is ultimately bought to form $\mathcal{T}'$. Also, if $\mathcal{T}'$ has $\ell' < \ell$ non-root components then each arc was paid for by moats around at most $\ell - \ell' + 1 \leq 2(\ell - \ell')$ different heads. So, the total cost of all purchased arcs is at most $2(\ell - \ell') \cdot \Delta$. Finally, the total dual grown is $\ell \cdot \Delta$, which is $\leq OPT_{LP}$ due to feasibility, so the cost of the edges bought can be bounded by $2\frac{\ell-\ell'}{\ell} \cdot OPT_{LP}$.

3.2 Algorithm and invariants

Now we will be more precise. The primal-dual procedure is presented in Algorithm 1. The following invariants will be maintained at any time $0 \leq t \leq \Delta$ during the execution of Algorithm 1.

1. For each $1 \leq i \leq \ell$, $h_i \in M_i$ and $M_i \subseteq V - \{r\}$ (so there is a variable y_{M_i} in the dual).
2. For each $1 \leq i \leq \ell$, $M_i = \{v \in V : d(v, h_i) < t\} \cup S$ where $S \subseteq \{v \in V : d(v, h_i) = t\}$.
3. $M_i \cap M_j \subseteq \bar{B}$ and both $\beta_i \cap \beta_j = M_i \cap \beta_j = \emptyset$ for distinct $0 \leq i, j \leq \ell$.
4. For each $1 \leq i \leq \ell$ we have $B_i \subseteq \beta_i \subseteq B_i \cup \bar{B}$. Furthermore, for each each $v \in \beta_i - B_i$ there is a *mate* $u \in B_i$ such that $uv \in E$ and $c_{uv} \leq t$.
5. y is feasible for LP (**DST-Dual**) with value exactly $\ell \cdot t$.

These concepts are illustrated in Figure 2.

3.3 Invariant analysis

▶ **Lemma 4.** *Invariants 1–5 are maintained by Algorithm 1 until the condition in the* ***if*** *statement in Step* (5) *is true. Furthermore, the algorithm terminates in* $O(n \cdot k)$ *iterations.*

Algorithm 1 Dual Growing Procedure

1: $M_i \leftarrow \{v \in V : d(v, h_i) = 0\}, 1 \leq i \leq \ell$
2: $\beta_i \leftarrow B_i$ for $0 \leq i \leq \ell$
3: $y \leftarrow \mathbf{0}$
4: Raise $y_{M_{i'}}$ uniformly for each moat $M_{i'}$ until some edge uv goes tight
5: **if** $u \in \beta_j$ for some $0 \leq j \leq \ell$ **and** $v \in M_{i'}$ for some $i' \neq j$ **then**
6: **return** the partial Steiner tree $\mathcal{T}'$ described in Lemma 6.
7: **else**
8: Let M_i be the unique moat with $uv \in \delta^{in}(M_i)$ ▷ cf. Proposition 5
9: $M_i \leftarrow M_i \cup \{u\}$
10: **if** $u \in \beta_i$ **then**
11: $\beta_i \leftarrow \beta_i \cup \{v\}$
12: **go to** Step (4)

Proof. Clearly the invariants are true after the initialization steps (at time $t = 0$), given that $d(B_i, h_j) > 0$ for any $i \neq j$. To see why Algorithm 1 terminates in a polynomial number of iterations, note that each iteration increases the size some moat by 1 and does not decrease the size of any moats. So after at most kn iterations some moat will grow to include the virtual body of another moat, at which point the algorithm stops.

Assume now that the invariants are true at some point just before Step (4) is executed and that the condition in Step (5) is false after Step (4) finishes. We will show that the invariants continue to hold just before the next iteration starts. We let uv denote the edge that went tight that is considered in Step (4). We also let t denote the total time the algorithm has executed (i.e. grown moats) up to this point.

Before proceeding with our proof, we exhibit the following useful fact. In what follows, let $M_j^{t'}$ be the moat around h_j at any time $t' \leq t$ during the algorithm. This proposition demonstrates how we control the overlap of the moats by exploiting the quasi-bipartite structure.

▶ **Proposition 5.** *If $uv \in \delta^{in}(M_i)$, then $uv \notin \delta^{in}(M_j^{t'})$ for any $j \neq i$, and for any $t' \leq t$.*

Proof. Suppose, for the sake of contradiction, that $uv \in \delta^{in}(M_j^{t'})$ for some $j \neq i$ and $t' \leq t$. Since $M_j^{t'}$ is a subset of M_j, the moat containing h_j at time t, we must have $v \in M_j \cap M_i$. Invariant 3 now implies that $v \in \bar{B}$. Since G is quasi-bipartite, then $u \in X$. Therefore $u \in B_{j'}$ for some j'. Since $j' \neq i$ or $j' \neq j$, then the terminating condition in Step (5) would have been satisfied as $u \in \beta_{j'}$. A contradiction. ◀

Following Proposition 5, we let i be the unique index such that $uv \in \delta^{in}(M_i)$ as in Step (8).

Invariant 1

First note that M_i never loses vertices during the algorithm's execution, and it therefore always contains head vertex h_i. Also, vertex u is not part of B_0 as otherwise the algorithm would have terminated in Step (5). Hence $M_i \cup \{u\}$ also does not contain the root node r.

Invariant 2

This is just a reinterpretation of Dijkstra's algorithm in the primal-dual framework (e.g. Chapter 5.4 of [13]), coupled with the fact that no edge considered in Step (4) in some iteration crosses more than one moat at any given time (Proposition 5).

Invariant 3

Suppose $(M_i \cup \{u\}) \cap M_j \not\subseteq \bar{B}$ for some $i \neq j$. This implies $u \in M_j \setminus \bar{B}$, and hence $u \in B_j \subseteq \beta_j$. Thus, the termination condition in Step (5) was satisfied and the algorithm should have terminated; contradiction.

If v is not added to β_i, and thus β_i remains unchanged, $\beta_i \cap \beta_j = M_j \cap \beta_i = \emptyset$ continues to hold for $j \neq i$. We also must have that $(M_i \cup \{u\}) \cap \beta_j = \emptyset$ for $i \neq j$, as otherwise $u \in \beta_j$ and this would violate the termination condition in Step (5).

Now suppose that v is added to β_i. Then for $j \neq i$ we still have $(\beta_i \cup \{v\}) \cap \beta_j = \emptyset$ as otherwise $v \in \beta_j$ which contradicts $v \in M_i$ and the fact that Invariant 3 holds at the start of this iteration. We also have that $M_j \cap (\beta_i \cup \{v\}) = \emptyset$ as otherwise $v \in M_j$. But this would mean that $u \in M_j$ as well by Proposition 5. We established above that $(M_i \cup \{u\}) \cap M_j \subseteq \overline{B}$. However, $\{u, v\} \subseteq (M_i \cup \{u\}) \cap M_j \subseteq \overline{B}$ contradicts the fact that G is quasi-bipartite.

Invariant 4

That $B_i \subseteq \beta_i$ is clear simply because we only add nodes to the sets β_i. Suppose now that v is added to β_i. In this case, $v \notin B_i$ as $B_i \subseteq \beta_i$ from the start. We claim that v can also not be part of B_j for some $j \neq i$, since otherwise $\emptyset \neq B_j \cap M_i \subseteq \beta_j \cap M_i$, contradicting Invariant 3. Hence $v \in \bar{B}$. Note that the quasi-bipartiteness of G implies that $u \in X$, and hence $u \in B_i$. Proposition 5 finally implies that only the moats crossed by uv are moats around i, so since the algorithm only grows one moat around i at any time we have $c_{uv} \leq t$, and this completes the proof of Invariant 4.

Invariant 5

The Step (4) stops the first time a constraint becomes tight, so feasibility is maintained. In each step, the algorithm grows precisely ℓ moats simultaneously. Because the objective function of (**DST-Dual**) is simply the sum of the dual variables, then the value of the dual is just ℓ times the total time spent growing dual variables. ◀

3.4 Augmenting $\mathcal{T}$

To complete the final detail in the description of the algorithm, we now show how to construct the partial Steiner tree after Step (5) has been reached. Lemma 4 shows that Invariants 1 through 5 hold just before Step (4) in the final iteration. Say the final iteration executes for δ time units and that uv is the edge that goes tight and was considered in Step (5).

▶ **Lemma 6.** *When Step* (6) *is reached in Algorithm 1, we can efficiently find a partial Steiner tree $\mathcal{T}'$ with $\ell' < \ell$ non-root components such that $cost(E(\mathcal{T}')) \leq cost(E(pt)) + 2\frac{\ell-\ell'}{\ell} \cdot OPT_{LP}$.*

Proof. Let j be the unique index such that $u \in \beta_j$ at time Δ. There is exactly one such j because $\beta_i \cap \beta_j = \emptyset$ for $i \neq j$ is ensured by the invariants. Next, let $J = \{i' \neq j : v \in M_{i'}\}$ and note that J consists of all indices i' (except, perhaps, j) such that $uv \in \delta^{in}(M_{i'})$. By the termination condition, $J \neq \emptyset$. Vertex u lies in β_j by definition. If $u \notin B_j$ then we let w be the mate of u as defined in Invariant 4. Otherwise, if $w \in B_j$, we let $w = u$.

For notational convenience, we let P_j be the path consisting of the single edge wu (or just the trivial path with no edges if $w = u$). In either case, say cost of P_j is $\Delta - \epsilon_j$ where $\epsilon_j \geq 0$ (cf. Invariant 4). For each $i' \in J$, let $P_{i'}$ be a shortest $v, h_{i'}$-path. Invariant 2 implies that

$$c(P_{i'}) = \Delta - \epsilon_{i'}, \tag{3}$$

for some $\epsilon_{i'} \geq 0$. Observe also that the tightness of uv at time Δ and the definition of J imply that

$$\sum_{i' \in J \cup \{j\}} \epsilon_{i'} \geq c_{uv}. \tag{4}$$

In fact, precisely a $\epsilon_{i'}$-value of the dual variables for $i' \neq j$ contribute to c_{uv}; the contribution of j's variables to c_{uv} is at most $\epsilon_{i'}$.

Construct a partial Steiner tree $\mathcal{T}'$ obtained from $\mathcal{T}$ and Algorithm 1 as follows.

- The sets $B_{j'}, F_{j'}$ and head $h_{j'}$ are unchanged for all $j' \notin J \cup \{j\}$.
- Replace the components $\{B_{i'}\}_{i' \in J \cup \{j\}}$ with a component $B := \bigcup_{i' \in J \cup \{j\}} (B_{i'} \cup V(P_{i'}))$ having head $h := h_j$. The edges of this component in $\mathcal{T}'$ are $F := \bigcup_{i' \in J \cup \{j\}} (F_{i'} \cup E(P_{i'})) \cup \{uv\}$.
- The free Steiner nodes $\bar{B}'$ of $\mathcal{T}'$ are the Steiner nodes not contained in any of these components.

Namely, $\bar{B}'$ consists of those nodes in $\bar{B}$ that are not contained on any path $P_{i'}, i' \in J \cup \{j\}$.

We show that Steiner tree $\mathcal{T}'$ as constructed above satisfies the conditions stated in Lemma 3. We first verify that $\mathcal{T}'$ as constructed above is indeed a valid partial Steiner tree. Clearly the new sets $\bar{B}', \{B_i\}_{i \notin J+j}$ and B partition V and $\bar{B}'$ is a subset of Steiner nodes.

Note that if $0 \in J \cup \{j\}$ in the above construction, then $j = 0$ because no moat contains r. Thus, if B_0 is replaced when B is constructed, then r is the head of this new component.

Next, consider any $b \in B$. If $b \in B_j$ then there is an h_j, b-path in $F_j \subseteq F$. If $b \in B_{i'}, i' \neq j$ then it can be reached from h_j in (B, F) as follows. Follow the h_j, w-path in F_j, then the w, u path P_j, cross the edge uv, follow $P_{i'}$ to reach $h_{i'}$, and finally follow the $h_{i'}, b$-path in $F_{i'}$. Finally, if $b \notin B_{i'}$ for any $i' \in J + j$ then b lies on some path $P_{i'}$, in which case it can be reached in a similar way.

It is also clear that $E(\mathcal{T}) \subseteq E(\mathcal{T}')$ and that the number of non-root components in $\mathcal{T}'$ is $\ell - |J| < \ell$. Also, $\text{cost}(E(\mathcal{T}')) - \text{cost}(E(\mathcal{T}))$ is at most the cost of the the paths $\{P_{i'}\}_{i' \in J+i}$ plus c_{uv}.

It now easily follows from (3) and (4) that

$$\sum_{i' \in J \cup \{j\}} \text{cost}(E(P_{i'})) + c_{uv} \leq \sum_{i' \in J \cup \{j\}} (\Delta - \epsilon_{i'}) + c_{uv} \leq (|J| + 1)\Delta \leq \frac{|J| + 1}{\ell} \cdot OPT_{LP}.$$

The last bound follows because the feasible dual we have grown has value $\ell \cdot \Delta \leq OPT_{LP}$. Let $\ell' = \ell - |J|$ be the number of nonroot components in $\mathcal{T}'$. Conclude by observing $|J| + 1 = \ell - \ell' + 1 \leq 2(\ell - \ell')$. ◀

To wrap things up, executing Algorithm 1 and constructing the partial Steiner tree as in Lemma 6 yields the partial Steiner tree that is promised by Lemma 3.

4 Conclusion

We have shown that the integrality gap of LP relaxation (**DST-Primal**) is $O(\log k)$ in quasi-bipartite instances of directed Steiner tree. The gap is known to be $\Omega(\sqrt{k})$ in 4-layered

instances [18] and $O(\log k)$ in 3-layered instances [7]. Since quasi-bipartite graphs are a generalization 2-layered instances, it is natural to ask if there is a generalization of 3-layered instances which has an $O(\log k)$ or even $o(\sqrt{k})$ integrality gap.

One possible generalization of 3-layered graphs would be when the subgraph of G induced by the Steiner nodes does not have a node with both positive indegree and positive outdegree. None of the known results on directed Steiner tree suggest such instances have a bad gap.

Even when restricted to 3-layered graphs, a straightforward adaptation of our algorithm that grow moats around the partial Steiner tree heads until some partial Steiner trees absorbs another fails to grow a sufficiently large dual to pay for the augmentation within any reasonable factor. A new idea is needed.

References

1. J. Byrka, F. Grandoni, T. Rothvoss, , and L. Sanita. Steiner tree approximation via iterative randomized rounding. *Journal of the ACM*, 60(1), 2013.
2. G. Calinescu and G. Zelikovsky. The polymatroid steiner problems. *J. Combinatorial Optimization*, 9(3):281–294, 2005.
3. M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha, , and M. Li. Approximation algorithms for directed steiner problems. *J. Algorithms*, 33(1):73–91, 1999.
4. I. Dinur and D. Steurer. Analytic approach to parallel repetition. In *In proceedings of STOC*, 2014.
5. J. Edmonds. Optimum branchings. *J. Res. Natl. Bur. Stand.*, 71:233–240, 1967.
6. U. Feige. A threshold of $\ln n$ for approximating set-cover. *Journal of the ACM*, 45(4):634–652, 1998.
7. Z. Friggstad, A. Louis, Y. K. Ko, J. Könemann, M. Shadravan, and M. Tulsiani. Linear programming hierarchies suffice for directed steiner tree. In *In proceedings of IPCO*, 2014.
8. M. X. Goemans, N. Olver, T. Rothvoss, and R. Zenklusen. Matroids and integrality gaps for hypergraphic steiner tree relaxations. In *In proceedings of STOC*, 2012.
9. M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.
10. S. Guha, A. Moss, J. Naor, and B. Scheiber. Efficient recover from power outage. In *In proceedings of STOC*, 1999.
11. E. Halperin and R. Krauthgamer. Polylogarithmic inapproximability. In *In proceedings of STOC*, 2003.
12. T. Hibi and T. Fujito. Multi-rooted greedy approximation of directed steiner trees with applications. In *In proceedings of WG*, 2012.
13. C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.
14. S. Rajagopalan and V. V. Vazirani. On the bidirected cut relaxation for the metric steiner tree problem. In *In proceedings of SODA*, 1999.
15. T. Rothvoss. Directed steiner tree and the lasserre hierarchy. Technical report, CORR abs/1111.5473, 2011.
16. V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2003.
17. A. Zelikovsky. A series of approximation algorthms for the acyclic directed steiner tree problem. *Algorithmica*, 18:99–110, 1997.
18. L. Zosin and S. Khuller. On directed steiner trees. In *In proceedings of SODA*, 2002.

A Linear Kernel for Finding Square Roots of Almost Planar Graphs*

Petr A. Golovach[1], Dieter Kratsch[2], Daniël Paulusma[3], and Anthony Stewart[4]

1 **Department of Informatics, University of Bergen, Bergen, Norway**
petr.golovach@ii.uib.no
2 **Laboratoire d'Informatique Théorique et Appliquée, Université de Lorraine, Metz, France**
dieter.kratsch@univ-lorraine.fr
3 **School of Engineering and Computing Sciences, Durham University, Durham, United Kingdom**
daniel.paulusma@durham.ac.uk
4 **School of Engineering and Computing Sciences, Durham University, Durham, United Kingdom**
a.g.stewart@durham.ac.uk

Abstract

A graph H is a square root of a graph G if G can be obtained from H by the addition of edges between any two vertices in H that are of distance 2 from each other. The Square Root problem is that of deciding whether a given graph admits a square root. We consider this problem for planar graphs in the context of the "distance from triviality" framework. For an integer k, a planar+kv graph is a graph that can be made planar by the removal of at most k vertices. We prove that a generalization of Square Root, in which some edges are prescribed to be either in or out of any solution, has a kernel of size $O(k)$ for planar+kv graphs, when parameterized by k. Our result is based on a new edge reduction rule which, as we shall also show, has a wider applicability for the Square Root problem.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases planar graphs, square roots, linear kernel

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.4

1 Introduction

Squares and square roots are well-known concepts in graph theory with a long history. The *square* $G = H^2$ of a graph $H = (V_H, E_H)$ is the graph with vertex set $V_G = V_H$, such that any two distinct vertices $u, v \in V_H$ are adjacent in G if and only if u and v are of distance at most 2 in H. A graph H is a *square root* of G if $G = H^2$. It is easy to check that there exist graphs with no square root, graphs with a unique square root as well as graphs with many square roots. The corresponding recognition problem, which asks whether a given graph admits a square root, is called the Square Root problem. Motwani and Sudan [21] showed that Square Root is NP-complete.

* This paper received support from EPSRC (EP/G043434/1), ERC (267959) and ANR project AGAPE.

1.1 Existing Results

In 1967, Mukhopadhyay [22] characterized the graphs that have a square root. In line with the aforementioned NP-completeness result of Motwani and Sudan, which appeared in 1994, this characterization does not lead to a polynomial-time algorithm for Square Root. Later results focussed on the following two recognition questions ($\mathcal{G}$ denotes some fixed graph class):

- How hard is it to recognize squares of graphs of $\mathcal{G}$?
- How hard is it to recognize graphs of $\mathcal{G}$ that have a square root?

Note that the second question corresponds to the Square Root problem restricted to graphs in $\mathcal{G}$, whereas the first question is the same as asking whether a given graph has a square root in $\mathcal{G}$.

Ross and Harary [24] characterized squares of a tree and proved that if a connected graph has a tree square root, then this root is unique up to isomorphism. Lin and Skiena [18] gave a linear-time algorithm for recognizing squares of trees; they also proved that Square Root can be solved in linear time for planar graphs. Le and Tuy [16] generalized the above results for trees [18, 24] to block graphs. Nestoridis and Thilikos [23] proved that Square Root is not only polynomial-time solvable for the class of planar graphs but for any non-trivial minor-closed graph class, that is, for any graph class that does not contain all graphs and that is closed under taking vertex deletions, edge deletions and edge contractions.

Lau [12] gave a polynomial-time algorithm for recognizing squares of bipartite graphs; note that Square Root is trivial for bipartite graphs, and even for K_4-free graphs, or equivalently, graphs of clique number at most 3, as square roots of K_4-free graphs must have maximum degree at most 2. Milanic, Oversberg and Schaudt [19] proved that line graphs can only have bipartite graphs as a square root. The same authors also gave a linear-time algorithm for Square Root restricted to line graphs.

Lau and Corneil [13] gave a polynomial-time algorithm for recognizing squares of proper interval graphs and showed that the problems of recognizing squares of chordal graphs and squares of split graphs are both NP-complete. The same authors also proved that Square Root is NP-complete even for chordal graphs. Le and Tuy [17] gave a quadratic-time algorithm for recognizing squares of strongly chordal split graphs. Le, Oversberg and Schaudt [14] gave polynomial algorithms for recognizing squares of ptolemaic graphs and 3-sun-free split graphs. In a more recent paper [15], the same authors extended the latter result by giving polynomial-time results for recognizing squares of a number of other subclasses of split graphs. Milanic and Schaudt [20] proved that Square Root can be solved in linear time for trivially perfect graphs and threshold graphs. They posed the complexity of Square Root restricted to split graphs and cographs as open problems. Recently, we proved that Square Root is linear-time solvable for 3-degenerate graphs and for (K_r, P_t)-free graphs for any two positive integers r and t [8].

Adamaszek and Adamaszek [1] proved that if a graph has a square root of girth at least 6, then this square root is unique up to isomorphism. Farzad, Lau, Le and Tuy [7] showed that recognizing graphs with a square root of girth at least g is polynomial-time solvable if $g \geq 6$ and NP-complete if $g = 4$. The missing case $g = 5$ was shown to be NP-complete by Farzad and Karimi [6].

In a previous paper [2] we proved that Square Root is polynomial-time solvable for graphs of maximum degree 6. We also considered square roots under the framework of parameterized complexity [3, 2]. We proved that the following two problems are fixed-parameter tractable with parameter k: testing whether a connected n-vertex graph with m edges has a square root with at most $n - 1 + k$ edges and testing whether such a graph has a

square root with at least $m - k$ edges. In particular, the first result implies that the problem of recognizing squares of tree+ke graphs, that is, graphs that can be modified into trees by removing at most k edges, is fixed-parameter tractable when parameterized by k.

1.2 Our Focus

We are interested in developing techniques that lead to new polynomial-time or parameterized algorithms for Square Root for special graph classes. In particular, there are currently very few results on the parameterized complexity, which is the main focus of our paper.

The graph classes that we consider fall under the "distance from triviality" framework, introduced by Guo, Hüffner and Niedermeier [10]. For a graph class $\mathcal{G}$ and an integer p we define four classes of "almost $\mathcal{G}$" graphs, that is, graphs that are editing distance k apart from $\mathcal{G}$. To be more precise, the classes $\mathcal{G} + ke$, $\mathcal{G} - ke$, $\mathcal{G} + kv$ and $\mathcal{G} - kv$ consist of all graphs that can be modified into a graph of $\mathcal{G}$ by deleting at most k edges, adding at most k edges, deleting at most k vertices and adding at most k vertices, respectively. Taking k as the natural parameter, these graph classes have been well studied from a parameterized point of view for a number of problems. In particular this is true for the vertex coloring problem restricted to (subclasses of) almost perfect graphs (due to the result of Grötschel, Lovász, and Schrijver [9], who proved that vertex coloring is polynomial-time solvable on perfect graphs). We consider $\mathcal{G}$ to be the class of *planar graphs*. As planar graphs are closed under taking edge and vertex deletions, classes of planar$-kv$ graphs and planar$-ke$ graphs coincide with planar graphs. Hence, we only need to consider planar$+kv$ graphs and planar$+ke$ graphs, that is, graphs that can be made planar by at most k vertex deletions or at most k edge deletions, respectively.

1.3 Our Results

Our main contribution is showing a linear kernel result for Square Root. In fact, we consider a more general version of Square Root, called Square Root with Labels, that takes as input a graph G with two subsets R and B of prespecified edges: the edges of R need to be included in a solution (square root) and the edges of B are forbidden in the solution. We prove that Square Root with Labels has a kernel of size $O(k)$ for planar$+kv$ graphs, when parameterized by k. Note that this immediately implies the same result for planar$+ke$ graphs. Square Root with Labels was introduced in a previous paper [3], but in this paper we introduce a new reduction rule, which we call the *edge reduction rule*.

The edge reduction rule is used to recognize, in polynomial time, a certain local substructure that graphs with square roots must have. As such, our rule can be added to the list of known and similar polynomial-time reduction rules for recognizing square roots. To give a few examples, the reduction rule of Lin and Skiena [18] is based on recognizing pendant edges and bridges of square roots of planar graphs, whereas the reduction rule of Farzad, Le and Tuy [7] is based on the fact that squares of graphs with large girth can be recognized to have a unique root. In contrast, our edge reduction rule, which is based on detecting so-called recognizable edges whose neighbourhoods have some special property (see Section 3 for a formal description) is tailored for graphs with no unique square root, just as we did in [3]; in fact our new rule, which we explain in detail in Section 4, can be seen as an improved and more powerful variant of the rule used in [3]. For squares with no unique square root, not all the root edges can be recognized in polynomial time. Hence, removing certain local substructures, thereby reducing the graph to a smaller graph, and keeping track of the compulsory edges (the recognized edges) and forbidden edges is the best we can do.

However, after the reduction, the connected components of the remaining graph might be dealt with further by exploiting the properties of the graph class under consideration. This is exactly what we do for planar+kv graphs to obtain the linear kernel in Section 5.

In Section 6 we show, besides giving some directions for future work, that the edge rule can also be used to obtain other polynomial-time results for Square Root, namely for graphs of maximum average degree smaller than $\frac{46}{11}$.

2 Preliminaries

We only consider finite undirected graphs without loops or multiple edges. We refer to the textbook by Diestel [5] for any undefined graph terminology.

We denote the vertex set of a graph G by V_G and the edge set by E_G. The subgraph of G induced by a subset $U \subseteq V_G$ is denoted by $G[U]$. The graph $G-U$ is the graph obtained from G after removing the vertices of U. If $U = \{u\}$, we also write $G - u$. Similarly, we denote the graph obtained from G after deleting an edge e by $G - e$. A vertex u is a *cut vertex* of a connected graph G with at least two vertices if $G - u$ is disconnected. An inclusion-maximal subgraph of G that has no cut vertices is called a *block*. A *bridge* of a connected graph G is an edge e such that $G - e$ is disconnected.

In the remainder of this section let G be a graph. We say that G is planar+kv if G can be made planar by removing at most k vertices. The *distance* $\mathrm{dist}_G(u, v)$ between a pair of vertices u and v of G is the number of edges of a shortest path between them. The diameter $\mathrm{diam}(G)$ of G is the maximum distance between any two vertices of G. The distance between a vertex $u \in V_G$ and a subset $X \subseteq V_G$ is denoted by $\mathrm{dist}_G(u, X) = \min\{\mathrm{dist}_G(u, v) \mid v \in X\}$. The distance between two subsets X and Y of V_G is denoted by $\mathrm{dist}_G(X, Y) = \min\{\mathrm{dist}_G(u, v) \mid u \in X, v \in Y\}$. Whenever we speak about the distance between a vertex set X and a subgraph H of G, we mean the distance between X and V_H.

The *open neighbourhood* of a vertex $u \in V_G$ is defined as $N_G(u) = \{v \mid uv \in E_G\}$ and its *closed neighbourhood* is defined as $N_G[u] = N_G(u) \cup \{u\}$. For $X \subseteq V_G$, let $N_G(X) = \bigcup_{u \in X} N_G(u) \setminus X$. Two (adjacent) vertices u, v are said to be *true twins* if $N_G[u] = N_G[v]$. The degree of a vertex $u \in V_G$ is defined as $d_G(u) = |N_G(u)|$. The maximum degree of G is $\Delta(G) = \max\{d_G(v) \mid v \in V_G\}$. A vertex of degree 1 is said to be a *pendant* vertex. If v is a pendant vertex, then we say the unique edge incident to u is a *pendant* edge.

The framework of parameterized complexity allows us to study the computational complexity of a discrete optimization problem in two dimensions. One dimension is the input size n and the other one is a parameter k. We refer to the recent textbook of Cygan et al. [4] for further details and only give the definitions for those notions relevant for our paper here. A parameterized problem is *fixed parameter tractable* (FPT) if it can be solved in time $f(k) \cdot n^{O(1)}$ for some computable function f. A *kernelization* of a parameterized problem Π is a polynomial-time algorithm that maps each instance (x, k) with input x and parameter k to an instance (x', k'), such that (i) (x, k) is a yes-instance if and only if (x', k') is a yes-instance of Π, and (ii) $|x'| + k'$ is bounded by $f(k)$ for some computable function f. The output (x', k') is called a *kernel* for Π. The function f is said to be a *size* of the kernel. It is well known that a decidable parameterized problem is FPT if and only if it has a kernel. A logical next step is then to try to reduce the size of the kernel. We say that (x', k') is a *linear* kernel if f is linear.

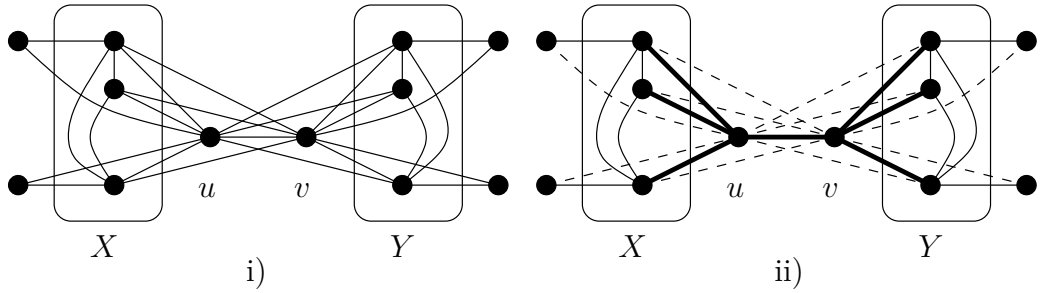


Figure 1 (i) An example of a graph G with a recognizable edge uv and a corresponding (u,v)-partition (X,Y). (ii) A square root of G. In this figure, the edges of the square root are shown by thick lines and the edges of G not belonging to the square root are shown by dashed lines. Edges which may or may not belong to the square root are shown by neither thick nor dashed lines.

3 Recognizable Edges

In this section we introduce the definition of a recognizable edge, which plays a crucial role in our paper, together with the corresponding notion of a (u,v)-partition. We also prove some important lemmas about this type of edges. See Fig. 1(i) for an example of a recognizable edge and a corresponding (u,v)-partition (X,Y).

▶ **Definition 1.** An edge uv of a graph G is said to be *recognizable* if the following four conditions are satisfied:

- **(a)** $N_G(u) \cap N_G(v)$ has a partition (X,Y) where $X = \{x_1,\ldots,x_p\}$ and $Y = \{y_1,\ldots,y_q\}$, $p,q \geq 1$, are (disjoint) cliques in G;
- **(b)** $x_iy_j \notin E_G$ for $i \in \{1,\ldots,p\}$ and $j \in \{1,\ldots,q\}$;
- **(c)** for any $w \in N_G(u) \setminus N_G[v]$, $wy_j \notin E_G$ for $j \in \{1,\ldots,q\}$, and symmetrically, for any $w \in N_G(v) \setminus N_G[u]$, $wx_i \notin E_G$ for $i \in \{1,\ldots,p\}$;
- **(d)** for any $w \in N_G(u) \setminus N_G[v]$, there is an $i \in \{1,\ldots,p\}$ such that $wx_i \in E_G$, and symmetrically, for any $w \in N_G(v) \setminus N_G[u]$, there is a $j \in \{1,\ldots,q\}$ such that $wy_j \in E_G$.

We also call such a partition (X,Y) a *(u,v)-partition* of $N_G(u) \cap N_G(v)$.

Notice that due to (c) and (d), (X,Y) is an ordered pair defined for an ordered pair (u,v); if $N_G(u) \setminus N_G(v) \neq \emptyset$ or $N_G(v) \setminus N_G(u) \neq \emptyset$ then (Y,X) is not a *(u,v)-partition*, as condition (c) is violated (and in some instances, condition (d) as well).

In the next lemma we give a necessary condition of an edge of a square root H of a graph G to be recognizable in G. In particular, this lemma implies that any non-pendant bridge of H is a recognizable edge of G.

▶ **Lemma 2.** *Let H be a square root of a graph G. Let uv be an edge of H that is not pendant and such that any cycle in H containing uv has length at least 7. Then uv is a recognizable edge of G and $(N_H(u) \setminus \{v\}, N_H(v) \setminus \{u\})$ is a (u,v)-partition in G.*

Proof. Let H be a square root of a graph G and let uv be an edge of H such that uv is not a pendant edge of H and any cycle in H containing uv has length at least 7. Let $X = \{x_1,\ldots,x_p\} = N_H(u) \setminus \{v\}$ and $Y = \{y_1,\ldots,y_q\} = N_H(v) \setminus \{u\}$. Because uv is not a pendant edge and any cycle in H that contains uv has length at least 7, it follows that $X \neq \emptyset$, $Y \neq \emptyset$ and $X \cap Y = \emptyset$. We show that (X,Y) is a (u,v)-partition of $N_G(u) \cap N_G(v)$ in G by proving that conditions (a)–(d) of Definition 1 are fulfilled.

First we prove (a). Let $z \in N_G(u) \cap N_G(v)$. We will show that $z \in X \cup Y$. If $uz \in E_H$ then $z \in X$, and if $vz \in E_H$ then $z \in Y$. Suppose that $z \notin X$ and $z \notin Y$. Since $uz \in E_G$,

there is a vertex $w \in V_G$ such that $uw, wz \in E_H$. Since $vz \notin E_H$ it follows that $w \neq v$. It follows due to symmetry that there exists $w' \in V_G$ such that $vw', w'z \in E_H$ and $w' \neq u$. Then either $wuvw'$ is a cycle in H if $w = w'$, otherwise, $zwuvw'z$ is a cycle of H. In both cases we have a contradiction since any cycle in H containing uv has length at least 7. This proves that $z \in X \cup Y$ and therefore, $N_G(u) \cap N_G(v) \subseteq X \cup Y$. Since $vx_i \in E_G$ and $uy_j \in E_G$ for all $i \in \{1, \ldots, p\}$ and $j \in \{1, \ldots, q\}$, we see that $X \cup Y \subseteq N_G(u) \cap N_G(v)$. Because $X, Y \neq \emptyset$ and $X \cap Y = \emptyset$, (X, Y) is a partition of $N_G(u) \cup N_G(v)$. It remains to observe that X and Y are cliques in G because any two vertices of X and any two vertices of Y have u or v, respectively, as common neighbour in H.

To prove (b), assume that there are $i \in \{1, \ldots, p\}$ and $j \in \{1, \ldots, q\}$ such that $x_iy_j \in E_G$. Because H has no cycle of length 4 containing uv, $x_iy_j \notin E_H$. Hence, there is $z \in V_H$ such that $x_iz, zy_j \in E_H$. Because H has no cycles of length 3 containing uv, we find that $z \notin \{u, v\}$. We conclude that zx_iuvy_jz is a cycle of length 5 in H that contains uv; a contradiction.

To prove (c), it suffices to show that for any $w \in N_G(u) \setminus N_G[v]$, $wy_j \notin E_G$ for $j \in \{1, \ldots, q\}$, as the second part is symmetric. To obtain a contradiction, assume that there are vertices $w \in N_G(u) \setminus N_G[v]$ and y_j for some $j \in \{1, \ldots, q\}$ such that $wy_j \in E_G$. By (a), (X, Y) is a partition of $N_G(u) \cap N_G(v)$. Hence, $w \notin X$ and $w \notin Y$. Because $w \notin X$ and $w \in N_G(u)$, there is $x \in V_G$ such that $ux, xw \in E_H$. As $ux \in E_H$, we have $x \in X$. If $wy_j \in E_H$, then the cycle $uxwy_jvu$ containing uv has length 5; a contradiction. Hence, $wy_j \notin E_H$. Because $wy_j \in E_G$, there is a vertex $z \in V_H$ such that $wz, zy_j \in E_H$. Since $w \in N_G(u) \setminus N_G[v]$, we have $w \notin \{u, v\}$. If $x = z$, then uvy_jxu is a cycle of length 4 containing uv, a contradiction. If $x \neq z$, then uvy_jzwxu is a cycle of length 6 containing uv, another contradiction.

To prove (d) we consider some $w \in N_G(u) \setminus N_G[v]$. We note that since $X \subseteq N_G(u) \cap N_G(v)$, $w \notin X$ and thus $uw \notin E_H$. Since $uw \in E_G$ by definition, there must be some $x \in V_G$ such that $ux, xw \in E_H$. Because w is not adjacent to v, we find that $x \neq v$. Since $ux \in E_H$ and $X = N_H(u) \setminus \{v\}$, this means that $x \in X$. The second condition in (d) follows by symmetry. ◀

The following corollary follows immediately from Lemma 2.

▶ **Corollary 3.** *Let H be a square root of a graph with no recognizable edges. Then every non-pendant edge of H lies on a cycle of length at most* 6.

In Lemma 4 we show that recognizable edges in a graph G can be used to identify some edges of a square root of G and also some edges that are not included in any square root of G; see Fig. 1(ii) for an illustration of this lemma.

▶ **Lemma 4.** *Let G be a graph with a square root H. Additionally let uv be a recognizable edge of G with a (u, v)-partition (X, Y) where $X = \{x_1, \ldots, x_p\}$ and $Y = \{y_1, \ldots, y_q\}$. Then:*

- **(i)** *$uv \in E_H$;*
- **(ii)** *for every $w \in N_G(u) \setminus N_G[v]$, $wu \notin E_H$, and for every $w \in N_G(v) \setminus N_G[u]$, $wv \notin E_H$.*
- **(iii)** *if u, v are true twins in G, then either $ux_1, \ldots, ux_p \in E_H$, $vy_1, \ldots, vy_q \in E_H$ and $uy_1, \ldots, uy_q \notin E_H$, $vx_1, \ldots, vx_p \notin E_H$ or $ux_1, \ldots, ux_p \notin E_H$, $vy_1, \ldots, vy_q \notin E_H$ and $uy_1, \ldots, uy_q \in E_H$, $vx_1, \ldots, vx_p \in E_H$;*
- **(iv)** *if u, v are not true twins in G, then $ux_1, \ldots, ux_p \in E_H$, $vy_1, \ldots, vy_q \in E_H$ and $uy_1, \ldots, uy_q \notin E_H$, $vx_1, \ldots, vx_p \notin E_H$.*

Proof. The proof uses conditions (a)–(d) of Definition 1.

To prove (i), suppose that $uv \notin E_H$. Then there is a vertex $z \in N_G(u) \cap N_G(v)$ such that $zu, zv \in E_H$. Assume without loss of generality that $z \in X$. Because of (b), $zy_1 \notin E_G$, which implies, together with $zv \in E_H$, that $vy_1 \notin E_H$. Because $vy_1 \in E_G$, this means that there is a vertex w with $vw, wy_1 \in E_H$. Because we assume $uv \notin E_H$, we observe that $w \neq u$. By (b), $w \notin X$ and, therefore, $w \in N_G(v) \setminus N_G(u)$. As $zv, vw \in E_H$, we obtain $wz \in E_G$. However, as $z \in X$, this contradicts (c). We conclude that $uv \in E_H$.

To prove (ii), it suffices to consider the case in which $w \in N_G(u) \setminus N_G[v]$, as the other case is symmetric. If $wu \in E_H$, then because $uv \in E_H$, we have $wv \in E_G$ contradicting $w \notin N_G(v)$.

We now prove (iii) and (iv). First suppose that there exist vertices x_i and x_j (with possibly $i = j$) for some $i, j \in \{1, \ldots, p\}$ such that $x_iu, x_jv \in E_H$. Then, as $x_iy_1, x_jy_1 \notin E_G$ by (b), we find that $y_1u, y_1v \notin E_H$. As $y_1u \in E_G$, the fact that $y_1u \notin E_H$ means that there exists a vertex $w \in V_H \setminus \{u\}$ such that $wu, wy_1 \in E_H$. As $y_1v \notin E_H$, we find that $w \neq v$, so $w \in V_H \setminus \{u, v\}$. As $x_iu, uw \in E_H$, we find that $x_iw \in E_G$, consequently $w \notin Y$ due to (b). Because $wy_1 \in E_H$ we obtain $w \notin X$, again due to (b). Hence, $w \notin X \cup Y = N_G(u) \cap N_G(v)$. Therefore, as $uw \in E_G$ and $w \neq v$, we have $w \in N_G(u) \setminus N_G[v]$, but as $wy_1 \in E_G$ this contradicts (c). Hence, this situation cannot occur.

Suppose that there a vertex x_i for some $i \in \{1, \ldots, p\}$ such that $x_iu, x_iv \notin E_H$. Then, as $x_iv \in E_G$, there exists a vertex $w \in V_H \setminus \{u, v\}$, such that $wv, wx_i \in E_H$. By (b), $w \notin Y$. As $uv \in E_H$ due to statement (i) and $vw \in E_H$, we find that $uw \in E_G$. Hence, as $w \notin Y$, we obtain $w \in X$. As $x_iu \in E_G \setminus E_H$ and $x_iv \notin E_H$, there is a vertex $z \in V_H \setminus \{u, v\}$ such that $zu, zx_i \in E_H$. As $uv \in E_H$ due to statement (i), this implies that $zv \in E_G$. Hence, $z \in X \cup Y$. As $zx_i \in E_H$, we find that $z \notin Y$ due to (b). Consequently, $z \in X$. This means that we have vertices $w, z \in X$ (possibly $w = z$) and edges $zu, wv \in E_H$. However, we already proved above that this is not possible.

We obtain that either $ux_1, \ldots, ux_p \in E_H$ and $vx_1, \ldots, vx_p \notin E_H$, or $ux_1, \ldots, ux_p \notin E_H$ and $vx_1, \ldots, vx_p \in E_H$. Symmetrically, either $uy_1, \ldots, uy_q \in E_H$ and $vy_1, \ldots, vy_q \notin E_H$, or $uy_1, \ldots, uy_q \notin E_H$ and $vy_1, \ldots, vy_q \in E_H$. By (b), it cannot happen that $ux_1, uy_1 \in E_H$ or $vx_1, vy_1 \in E_H$. Hence, either $ux_1, \ldots, ux_p \in E_H$, $vy_1, \ldots, vy_q \in E_H$ and $uy_1, \ldots, uy_q \notin E_H$, $vx_1, \ldots, vx_p \notin E_H$ or $ux_1, \ldots, ux_p \notin E_H$, $vy_1, \ldots, vy_q \notin E_H$ and $uy_1, \ldots, uy_q \in E_H$, $vx_1, \ldots, vx_p \in E_H$. In particular, this implies (iii).

To prove iv), assume without loss of generality that $N_G(u) \setminus N_G[v] \neq \emptyset$. For contradiction, let $ux_1, \ldots, ux_p \notin E_H$, $vy_1, \ldots, vy_q \notin E_H$ and $uy_1, \ldots, uy_q \in E_H$, $vx_1, \ldots, vx_p \in E_H$. Let $w \in N_G(u) \setminus N_G[v]$. By (d), there is a vertex x_i for some $i \in \{1, \ldots, p\}$ such that $wx_i \in E_G$. Then $wx_i \notin E_H$, as otherwise our assumption that $vx_i \in E_H$ will imply that $w \in N_G(v)$, which is not possible. Since $wx_i \in E_G \setminus E_H$, there exists a vertex $z \in V_H$, such that $zw, zx_i \in E_H$. Because $x_iu \notin E_H$, we find that $z \neq u$, and because $w \notin N_G(v)$, we find that $z \neq v$. Because $zx_i, x_iv \in E_H$, we obtain $zv \in E_G$. As $w \notin N_G(v)$ and $vx_j \in E_H$ for all $j \in \{1, \ldots, p\}$, we have $wx_j \notin E_H$ for all $j \in \{1, \ldots, p\}$. Hence, as $zw \in E_H$, we find that $z \notin X$. As $zx_i \in E_H$, we find that $z \notin Y$ due to (b). Hence, $z \notin X \cup Y = N_G(u) \cap N_G(v)$. As $zv \in E_G$, this implies that $z \in N_G(v) \setminus N_G[u]$ (recall that $z \neq u$). Because $zx_i \in E_G$, this is in contradiction with (c). ◀

▶ **Remark 1.** *If the vertices u and v of the recognizable edge of the square G in Lemma 4 are true twins, then by statement (iii) of this lemma and the fact that the vertices u and v are interchangeable, G has at least two isomorphic square roots: one root containing $ux_1, \ldots, ux_p$, $vy_1, \ldots, vy_q$ and excluding $uy_1, \ldots, uy_q$, $vx_1, \ldots, vx_p$, and another one containing $ux_1, \ldots, ux_p$, $vy_1, \ldots, vy_q$ and excluding $uy_1, \ldots, uy_q$, $vx_1, \ldots, vx_p$.*

4 The Edge Reduction Rule

In this section we present our edge reduction rule. As mentioned in Section 1.3, we solve a more general problem than Square Root. Before discussing the edge reduction rule, we first formally define this problem.

Square Root with Labels
Input: a graph G and two sets of edges $R, B \subseteq E_G$.
Question: is there a graph H with $H^2 = G$, $R \subseteq E_H$ and $B \cap E_H = \emptyset$?
Note that Square Root is indeed a special case of Square Root with Labels: choose $R = B = \emptyset$.

We say that a graph H is a *solution* for an instance (G, R, B) of Square Root with Labels if H satisfies the following three conditions:

(i) $H^2 = G$;
(ii) $R \subseteq E_H$; and
(iii) $B \cap E_H = \emptyset$.

We use Lemmas 2 and 4 to preprocess instances of Square Root with Labels. Our edge reduction algorithm takes as input an instance (G, R, B) of Square Root with Labels and either returns an equivalent instance with no recognizable edges or answers no.

Edge Reduction

1. Find a recognizable edge uv together with corresponding (u, v)-partition (X, Y), $X = \{x_1, \ldots, x_p\}$ and $Y = \{y_1, \ldots, y_q\}$. If such an edge uv does not exist, then return the obtained instance of Square Root with Labels and stop.
2. If $uv \in B$ then return `no` and stop. Otherwise let $B_1 = \{wu \mid w \in N_G(u) \setminus N_G[v]\} \cup \{wv \mid w \in N_G(v) \setminus N_G[u]\}$. If $R \cap B_1 \neq \emptyset$, then return `no` and stop.
3. If u and v are not true twins then set $R_2 = \{ux_1, \ldots, ux_p\} \cup \{vy_1, \ldots, vy_q\}$ and $B_2 = \{uy_1, \ldots, uy_q\} \cup \{vx_1, \ldots, vx_p\}$. If $R_2 \cap B \neq \emptyset$ or $B_2 \cap R \neq \emptyset$, then return `no` and stop.
4. If u and v are true twins then do as follows:
   a. If $(\{uy_1, \ldots, uy_q\} \cup \{vx_1, \ldots, vx_p\}) \cap R \neq \emptyset$ or $(\{ux_1, \ldots, ux_p\} \cup \{vy_1, \ldots, vy_q\}) \cap B \neq \emptyset$ then set $R_2 = \{uy_1, \ldots, uy_q\} \cup \{vx_1, \ldots, vx_p\}$ and $B_2 = \{ux_1, \ldots, ux_p\} \cup \{vy_1, \ldots, vy_q\}$. If $R_2 \cap B \neq \emptyset$ or $B_2 \cap R \neq \emptyset$, then return `no` and stop.
   b. If $(\{uy_1, \ldots, uy_q\} \cup \{vx_1, \ldots, vx_p\}) \cap R = \emptyset$ and $(\{ux_1, \ldots, ux_p\} \cup \{vy_1, \ldots, vy_q\}) \cap B = \emptyset$ then set $R_2 = \{ux_1, \ldots, ux_p\} \cup \{vy_1, \ldots, vu_q\}$ and $B_2 = \{uy_1, \ldots, uy_q\} \cup \{vx_1, \ldots, vx_p\}$. (Note that $R_2 \cap B = \emptyset$ and $B_2 \cap R = \emptyset$.)
5. Delete the edge uv and the edges of B_1 from G, set $R := (R \setminus \{uv\}) \cup R_2$ and $B := (B \setminus B_1) \cup B_2$, and return to Step 1.

▶ **Lemma 5.** *For an instance (G, R, B) of* Square Root with Labels *where G has n vertices and m edges,* **Edge Reduction** *in time $O(n^2m^2)$ either correctly answers* no *or returns an equivalent instance (G', R', B') with the following property: for any square root H of G', every edge of H is either a pendant edge of H or is included in a cycle of length at most 6 in H. Moreover, (G', R', B') has a solution H if and only if (G, R, B) has a solution that can be obtained from H by restoring all recognizable edges.*

Proof. It suffices to consider one iteration of the algorithm to prove its correctness. If we stop at Step 1 and return the obtained instance of Minimum Square Root with Labels, then by Lemma 2, for any square root H of G', every non-pendant edge of H is included in a cycle of length at most 6 in H.

To show the correctness of Step 2, we note that by Lemma 4(i), uv is included in any square root and the edges of B_1 are not included in any square root. Hence, if what we do in Step 2 is not consistent with R and B, there is no square root of G that includes the edges of R and excludes the edges of B, thus returning output `no` is correct.

To show the correctness of Step 3, suppose u and v are not true twins. Then by Lemma 4 iv) it follows that $ux_1,\ldots,ux_p \in E_H$, $vy_1,\ldots,vy_q \in E_H$, $uy_1,\ldots,uy_q \notin E_H$ and $vx_1,\ldots,vx_p \notin E_H$ for any square root H. Hence, we must define R_2 and B_2 according to this lemma. If afterwards we find that $R_2 \cap B \neq \emptyset$ or $B_2 \cap R \neq \emptyset$, then R_2 or B_2 is not consistent with R or B, respectively, and thus, retuning `no` if this case happens is correct.

To show the correctness of Step 4, suppose that u and v are true twins. Then by Lemma 4 iv) we have two options. First, if $(\{uy_1,\ldots,uy_q\} \cup \{vx_1,\ldots,vx_p\}) \cap R \neq \emptyset$ or $(\{ux_1,\ldots,ux_p\} \cup \{vy_1,\ldots,vy_q\}) \cap B \neq \emptyset$, then we are forced to go for the option as defined in Step 4(a). If afterwards $R_2 \cap B \neq \emptyset$ or $B_2 \cap R \neq \emptyset$, then we still need to return `no` as in Step 3. Second, if $(\{uy_1,\ldots,uy_q\} \cup \{vx_1,\ldots,vx_p\}) \cap R = \emptyset$ and $(\{ux_1,\ldots,ux_p\} \cup \{vy_1,\ldots,vy_q\}) \cap B = \emptyset$, then we may set without loss of generality (cf. Remark 1) that $R_2 = \{ux_1,\ldots,ux_p\} \cup \{vy_1,\ldots,vu_q\}$ and $B_2 = \{uy_1,\ldots,uy_q\} \cup \{vx_1,\ldots,vx_p\}$. Note that in this case $R_2 \cap B = \emptyset$ and $B_2 \cap R = \emptyset$.

Finally, to show the correctness of Step 5, let G' be the graph obtained from G after deleting the edge uv and the edges of B_1. Let $R' = (R \setminus \{uv\}) \cup R_2$ and $B' = (B \setminus B_1) \cup B_2$. Then the instances (G,R,B) and (G',R',B') are equivalent: a graph H is readily seen to be a solution for (G,R,B) if and only if $H - uv$ is a solution for (G',R',B'). This completes the correctness proof of our algorithm.

It remains to evaluate the running time. We can find a recognizable edge uv together with the corresponding (u,v)-partition (X,Y) in time $O(mn^2)$. This can be seen as follows. For each edge uv, we find $Z = N_G(u) \cap N_G(v)$. Then we check conditions (a) and (b) of Definition 1, that is, we check whether Z is the union of two disjoint cliques with no edges between them. Finally, we check conditions (c) and (d) of Definition 1. For a given uv, this can all be done in time $O(n^2)$. As we need to check at most m edges, one iteration takes time $O(mn^2)$. As the total number of iterations is at most m, the whole algorithm runs in time $O(n^2m^2)$. ◀

5 The Linear Kernel

For proving that Square Root with Labels restricted to planar+kv graphs has a linear kernel when parameterized by k, we will use the following result of Harary, Karp and Tutte as a lemma.

▶ **Lemma 6** ([11]). *A graph H has a planar square if and only if*

(i) *every vertex $v \in V_H$ has degree at most 3,*
(ii) *every block of H with more than four vertices is a cycle of even length, and*
(iii) *H has no three mutually adjacent cut vertices.*

We need the following additional terminology. A block is *trivial* if it has exactly one vertex; note that this vertex must have degree 0. A block is *small* if it has exactly two vertices and *big* otherwise. We say that a block is *pendant* if it is a small block with a vertex of degree 1.

We need two more structural lemmas. We first show the effect of applying our **Edge Reduction Rule** on the number of vertices in a connected component of a planar graph.

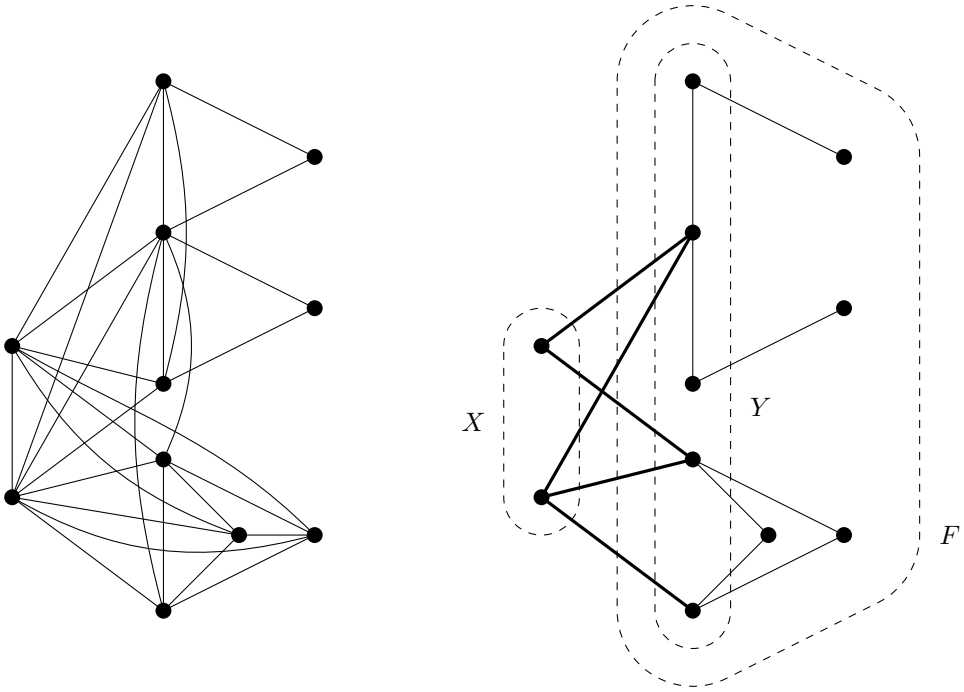


Figure 2 An example of a planar+$2v$ graph $G = H^2$ (left side) and a square root H of G (right side). The thick edges in H denote the edges of A.

▶ **Lemma 7.** *Let G be a planar graph with a square root. If G has no recognizable edges, then every connected component of G has at most 12 vertices.*

Proof. Let G be a planar square with no recognizable edges. We may assume without loss of generality that G is connected and $|V_G| \geq 2$. Let H be a square root of G. Recall that H is a connected spanning subgraph of G. Hence, it suffices to prove that H has at most 12 vertices.

First suppose that H does not have a big block, in which case every edge of H is a bridge. As G has no recognizable edges, Corollary 3 implies that every block of H is pendant. By Lemma 6, every vertex of H degree at most 3. Hence, H has at most four vertices.

Now suppose that H has a big block F. If F contains no cut vertices of H, then $H = F$ has at most six vertices due to Corollary 3 and Lemma 6. Assume that F contains a cut vertex v of H. Lemma 6 tells us that $d_H(v) \leq 3$; therefore v is a vertex of exactly two blocks, namely F and some other block S. Because F is big, v has two neighbours in F. Hence, v can only have one neighbour in S, thus S is small. As G has no recongizable edges, Corollary 3 implies that S is a pendant block. Hence, we find that $|V_G| \leq 2|V_F|$ (with equality if and only if each vertex of F is a cut vertex).

If F has at least seven vertices, then it follows from Lemma 6 that F is a cycle of even length at least 8, which is not possible due to Corollary 3. We conclude that $|V_F| \leq 6$ and find that $|V_G| = |V_H| \leq 2|V_F| \leq 12$. ◀

We now prove our second structural lemma.

▶ **Lemma 8.** *Let G be a planar+kv graph with no recognizable edges, such that every connected component of G has at least 13 vertices. If G has a square root, then $|V_G| \leq 137k$.*

Proof. Let H be a square root of G. By Lemma 7, G cannot have any planar connected components (as these would have at most 12 vertices). Hence, every connected component of G is non-planar.

Since G is planar+kv, there exists a subset $X \subseteq V_G$ of size at most k such that $G - X$ is planar. Let $F = H - X$. Note that F is a spanning subgraph of $G - X$ and that F^2 is a (spanning) subgraph of $G - X$; hence F^2 is planar. Let Y be the set that consists of all those vertices of F that are a neighbour of X in H, that is $Y = N_H(X) \cap V_F$. Since every connected component of $G - X$ is non-planar, every connected component of F contains at least one vertex of Y. Let A be the set that consists of all edges between X and Y in H, that is, $A = \{uv \in E(H) \mid u \in X, v \in Y\}$. See Figure 2 for an example.

Consider a vertex $v \in X$. By Kuratowski's Theorem, the (planar) graph $G - X$ has no clique of size 5. Since $N_H(v) \cap (V_G \setminus X)$ is a clique in $G - X$, we find that $|N_H(v) \cap (V_G \setminus X)| \leq 4$. Hence, $|Y| \leq 4|X| \leq 4k$.

We now prove three claims about the structure of blocks of F.

▶ **Claim A.** *If R is a block of F that is not a pendant block of H, then V_R is at distance at most 1 from Y in F.*

Proof. We prove Claim A as follows. Let R be a block of F that is not a pendant block of H. To obtain a contradiction, assume that V_R is at distance at least 2 from Y in F. Let u be a vertex of R such that $\mathrm{dist}_F(u, Y) = \min\{\mathrm{dist}_F(u, v) \mid v \in V_R\}$, so u is a cut vertex of F that is of distance at least 2 from Y in F. Note that R is not a trivial block of F, since all trivial blocks are isolated vertices of F that are vertices of Y.

First suppose that R is a small block of F and let v be the second vertex of R. Then the edge uv is a bridge of F. Since R is not pendant, it follows from Corollary 3 that uv is in a cycle of length C at most 6 in H. Observe that C must contain at least two edges of A, which implies that u or v is at distance at most 1 from Y. This is a contradiction.

Now suppose that R is a big block of G. Let v be the neighbour of u in a shortest path between u and Y in F. By Lemma 6, u has degree at most 3 in F. As R is big, u has at least two neighbours in F. Hence, uv is a bridge of F. As v has at least two neighbours in F as well, uv is not a pendant edge of H. Then it follows from Corollary 3 that uv is in a cycle C of length at most 6 in H. Observe that C must contain at least two edges of A and at least one edge uw of R for some vertex $w \neq u$ in R. Hence, w is at distance at most 1 from Y, which is a contradiction. This completes the proof of Claim A. ◀

By Lemma 6, every vertex of F has degree at most 3 in F. Hence the following holds:

▶ **Claim B.** *For every $u \in Y$, F has at most three big blocks at distance at most 1 from u.*

Let Z be the set of vertices of F at distance at most 3 from X in H.

▶ **Claim C.** *If R is a block of F with $V_R \setminus Z \neq \emptyset$, then $|V_R| \leq 6$.*

Proof. We prove Claim C as follows. Suppose R is a block of F with $V_R \setminus Z \neq \emptyset$. For contradiction, assume that $|V_R| \geq 7$. Then, by Lemma 6, R is a cycle of F of even size. As $V_R \setminus Z \neq \emptyset$ and R is connected, there exists an edge uv of F with $u \notin Z$. By Corollary 3, we find that uv is in a cycle C of H of length at most 6. Since u is at distance at least 4 from X in H, we find that C contains no vertex of X and therefore, C is a cycle of F. Then $R = C$ must hold, which is a contradiction as $|V_R| \geq 7 > 6 \geq |V_C|$. This completes the proof of Claim C. ◀

We will now show that the diameter of F is bounded. We start with proving the following claim.

▶ **Claim D.** *Every vertex of every block R of F that is non-pendant in H is at distance at most 5 from X in H. Moreover,*

(i) *if R has a vertex at distance at least 4 from X in H, then R is a big block,*

(ii) *R has at most three vertices at distance at least 4 and at most one vertex at distance 5 from X in H.*

Proof. We prove Claim D as follows. Let R be a block of F that is non-pendant in H. Claim A tells us that V_R is at distance at most 1 from Y in F.

If R is a small block, then every vertex of R is at distance at most 2 from Y. Hence, every vertex of R is at distance at most 3 from X in H and the claim holds for R.

Let R be a big block. If R has at most four vertices, then the vertices of R are at distance at most 3 from Y in F and at most one vertex of R is at distance exactly 3. Hence, the vertices of R are at distance at most 4 from X in H and at most one vertex of R is at distance exactly 4. Assume that $|V_R| > 4$. Then either $V_R \subseteq Z$, that is, all the vertices are at distance at most 3 from X in H, or, by Lemma 6 and Claim C, we find that R has at most six vertices. As $|V_R| > 4$, we find that R is a cycle on six vertices by Lemma 6. Hence, in the latter case every vertex of R is at distance at most 4 from Y, that is, at distance at most 5 from X in H. Moreover, at most three vertices are at distance at least 4 and at most one vertex is at distance 5 from X in H as R is a cycle. This completes the proof of Claim D. ◀

By combining Claim B with the fact that $|Y| \leq 4k$, we find that F has at most $12k$ big blocks at distance at most 1 from Y. By Claims A and D, this implies that H has at most $36k$ vertices of non-pendant blocks at distance at least 4 from X in H and at most $12k$ vertices at distance at least 5 from X in H. Let v be a vertex H of degree 1 in H. If v is at distance at least 5 from X, then v is adjacent to a vertex u of a non-pendant block and u is at distance at least 4 from X in H. Notice that v is a unique vertex of degree 1 adjacent to u, because by Claim D, u is in a big block and $d_F(u) \leq 3$ by Lemma 6. Since H has at most $36k$ vertices of non-pendant blocks at distance at least 4 from X in H, the total number of vertices of degree 1 at distance at least 5 from X in H is at most $36k$. Taking into account that there are at most $12k$ vertices at distance at least 5 from X in H in non-pendant blocks, we see that there are at most $48k$ vertices in H at distance at least 5 from X and all other vertices in F are at distance at most 4 from X. Using the facts that $|Y| \leq 4k$ and that $d_F(v) \leq 3$ for $v \in V_F$ by Lemma 6, we observe that H has at most $k + 4k + 12k + 24k + 48k = 89k$ vertices at distance at most 4 from X. It then follows that $|V_G| = |V_H| \leq 48k + 89k = 137k$. ◀

We are now ready to prove our main result.

▶ **Theorem 9.** Square Root with Labels *has a kernel of size $O(k)$ for planar+kv graphs when parameterized by k.*

Proof. Let (G, R, B) be an instance of Square Root with Labels. First we apply **Edge Reduction**, which takes polynomial time due to Lemma 5. By the same lemma we either solve the problem in polynomial time or obtain an equivalent instance (G', R', B') with the following property: for any square root H of G', every edge of H is either a pendant edge of H or is included in a cycle of length at most 6 in H. In the latter case we apply the following reduction rule exhaustively, which takes polynomial time as well.

Component Reduction. If G' has a connected component F with $|V_F| \leq 12$, then use brute force to solve Square Root with Labels for $(F, R \cap V_F, B \cap V_F)$. If this yields a `no`-answer, then return `no` and stop. Otherwise, return $(G' - V_F, R' \setminus V_F, B' \setminus V_F)$ or if $G' = F$, return `yes` and stop.

It is readily seen that this rule either solves the problem correctly or returns an equivalent instance. Assume we obtain an instance (G'', R'', B''). Our reduction rules do not increase the deletion distance, that is, G'' is a planar+kv graph. Then by Lemma 8, if G'' has more than $137k$ vertices then G'', and thus G, has no square root. Hence, if $|V_G''| > 137k$, we have a no-instance, in which case we return a `no`-answer and stop. Otherwise, we return the kernel (G'', R'', B''). ◀

6 Conclusions

We proved a linear kernel for Square Root with Labels, which generalizes the Square Root problem, for planar+kv graphs using a new edge reduction rule. It would be interesting to research whether our edge reduction rule can be used to obtain other results for Square Root. We could prove that this rule can be used to show the known result [2] that Square Root is polynomial-time solvable for graphs of maximum degree at most 6. We conclude our paper by showing that there exists at least one other application.

The *average degree* of a graph G is $\mathrm{ad}(G) = \frac{1}{|V_G|}\sum_{v \in V_G} d_G(v) = \frac{2|E_G|}{|V_G|}$. Then the *maximum average degree* of G is defined as $\mathrm{mad}(G) = \max\{\mathrm{ad}(H) \mid H \text{ is a subgraph of } G\}$. We use our rule **Edge Reduction** to prove the following result (proof omitted).

▶ **Theorem 10.** Square Root *can be solved in time* $O(n^4)$ *for* n*-vertex graphs* G *with* $\mathrm{mad}(G) < \frac{46}{11}$.

We pose the problem as to whether Theorem 10 can be strengthened to hold for graphs of higher maximum average degree as an open problem.

References

1 Anna Adamaszek and Michal Adamaszek. Uniqueness of graph square roots of girth six. *Electronic Journal of Combinatorics*, 18, 2011.

2 Manfred Cochefert, Jean-François Couturier, Petr A. Golovach, Dieter Kratsch, and Daniël Paulusma. Sparse square roots. In Andreas Brandstädt, Klaus Jansen, and Rüdiger Reischuk, editors, *Graph-Theoretic Concepts in Computer Science – 39th International Workshop, WG 2013, Lübeck, Germany, June 19-21, 2013, Revised Papers*, volume 8165 of *Lecture Notes in Computer Science*, pages 177–188. Springer, 2013.

3 Manfred Cochefert, Jean-François Couturier, Petr A. Golovach, Dieter Kratsch, and Daniël Paulusma. Parameterized algorithms for finding square roots. *Algorithmica*, 74:602–629, 2016.

4 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

5 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate Texts in Mathematics*. Springer, 2012.

6 Babak Farzad and Majid Karimi. Square-root finding problem in graphs, a complete dichotomy theorem. *CoRR*, abs/1210.7684, 2012.

7 Babak Farzad, Lap Chi Lau, Van Bang Le, and Nguyen Ngoc Tuy. Complexity of finding graph roots with girth conditions. *Algorithmica*, 62:38–53, 2012.

8 Petr A. Golovach, Dieter Kratsch, Daniël Paulusma, and Anthony Stewart. Squares of low clique number. In *14th Cologne Twente Workshop 2016 (CTW 2016), Gargnano, Italy, June 6-8, 2016*, Electronic Notes in Discrete Mathematics, to appear, 2016.

9 Martin Grötschel, László Lovász, and Alexander Schrijver. Polynomial algorithms for perfect graphs. *Annals of Discrete Mathematics*, 21:325–356, 1984.

10 Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A structural view on parameterizing problems: distance from triviality. In Rodney G. Downey, Michael R. Fellows, and Frank K. H. A. Dehne, editors, *Parameterized and Exact Computation, First International Workshop, IWPEC 2004, Bergen, Norway, September 14-17, 2004, Proceedings*, volume 3162 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 2004.

11 Frank Harary, Richard M. Karp, and William T. Tutte. A criterion for planarity of the square of a graph. *Journal of Combinatorial Theory*, 2:395–405, 1967.

12 Lap Chi Lau. Bipartite roots of graphs. *ACM Transactions on Algorithms*, 2:178–208, 2006.

13 Lap Chi Lau and Derek G. Corneil. Recognizing powers of proper interval, split, and chordal graphs. *SIAM Journal on Discrete Mathematics*, 18:83–102, 2004.

14 Van Bang Le, Andrea Oversberg, and Oliver Schaudt. Polynomial time recognition of squares of ptolemaic graphs and 3-sun-free split graphs. *Theoretical Computer Science*, 602:39–49, 2015.

15 Van Bang Le, Andrea Oversberg, and Oliver Schaudt. A unified approach for recognizing squares of split graphs. *Manuscript*, 2015.

16 Van Bang Le and Nguyen Ngoc Tuy. The square of a block graph. *Discrete Mathematics*, 310:734–741, 2010.

17 Van Bang Le and Nguyen Ngoc Tuy. A good characterization of squares of strongly chordal split graphs. *Information Processing Letters*, 111:120–123, 2011.

18 Yaw-Ling Lin and Steven Skiena. Algorithms for square roots of graphs. *SIAM Journal on Discrete Mathematics*, 8:99–118, 1995.

19 Martin Milanic, Andrea Oversberg, and Oliver Schaudt. A characterization of line graphs that are squares of graphs. *Discrete Applied Mathematics*, 173:83–91, 2014.

20 Martin Milanic and Oliver Schaudt. Computing square roots of trivially perfect and threshold graphs. *Discrete Applied Mathematics*, 161:1538–1545, 2013.

21 Rajeev Motwani and Madhu Sudan. Computing roots of graphs is hard. *Discrete Applied Mathematics*, 54:81–88, 1994.

22 A. Mukhopadhyay. The square root of a graph. *Journal of Combinatorial Theory*, 2:290–295, 1967.

23 Nestor V. Nestoridis and Dimitrios M. Thilikos. Square roots of minor closed graph classes. *Discrete Applied Mathematics*, 168:34–39, 2014.

24 Ian C. Ross and Frank Harary. The square of a tree. *Bell System Technical Journal*, 39:641–647, 1960.

Linear-Time Recognition of Map Graphs with Outerplanar Witness*

Matthias Mnich[†1], **Ignaz Rutter**[2], **and Jens M. Schmidt**[3]

1 **University of Bonn, Bonn, Germany**
mmnich@uni-bonn.de
2 **Karlsruhe Institute of Technology, Karlsruhe, Germany**
ignaz.rutter@kit.edu
3 **TU Ilmenau, Ilmenau, Germany**
jens.schmidt@tu-ilmenau.de

Abstract

Map graphs generalize planar graphs and were introduced by Chen, Grigni and Papadimitriou [STOC 1998, J.ACM 2002]. They showed that the problem of recognizing map graphs is in NP by proving the existence of a planar witness graph W. Shortly after, Thorup [FOCS 1998] published a polynomial-time recognition algorithm for map graphs. However, the run time of this algorithm is estimated to be $\Omega(n^{120})$ for n-vertex graphs, and a full description of its details remains unpublished.

We give a new and purely combinatorial algorithm that decides whether a graph G is a map graph having an outerplanar witness W. This is a step towards a first combinatorial recognition algorithm for general map graphs. The algorithm runs in time and space $O(n+m)$. In contrast to Thorup's approach, it computes the witness graph W in the affirmative case.

1998 ACM Subject Classification G.2.2 Graph algorithms

Keywords and phrases Algorithms and data structures, map graphs, recognition, planar graphs

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.5

1 Introduction

Consider the adjacency graph of the states of the USA, where two states are adjacent if their borders intersect. Since Arizona, Colorado, New Mexico and Utah meet pairwise at a single common point, the adjacency graph will not be planar; however, it will be a *map graph*. In (much) more detail, a *map* of a graph $G = (V, E)$ is a function $\mathcal{M}$ that maps each vertex $v \in V$ to a disc homeomorph $\mathcal{M}(v)$ on the sphere (the states) such that, for any two distinct vertices $v, w \in V$, the interiors of $\mathcal{M}(v)$ and $\mathcal{M}(w)$ are disjoint, and v and w are adjacent in G if and only if the boundaries of $\mathcal{M}(v)$ and $\mathcal{M}(w)$ intersect. A graph G is a *map graph* if a map of G exists.

By definition, map graphs contain and exceed the class of planar graphs. They have applications in graph drawing, circuit board design and topological inference problems [4]. Chen, Grigni and Papadimitriou [2] characterized map graphs as the *half-squares* of sufficiently small planar bipartite graphs called *witnesses* (we give precise definitions for both terms in the next section). This result allows, similar to Kuratowski's Theorem for planar graphs, to use purely combinatorial arguments for an object that has been originally defined by

* The authors thank Alexander Wolff from the University of Würzburg for hosting them.
† This research was partially supported by ERC Starting Grant 306465 (BeyondWorstCase).

topological properties. Since such witnesses can always be chosen small in size ($O(n)$ vertices for map graphs on n vertices), the recognition problem for map graphs is in NP. In 1998, Chen et al. therefore raised the question whether recognizing map graphs is in P.

This problem was resolved shortly after by Thorup [18], whose solution is based on a carefully designed topological treatment. However, a full version of the extended abstract [18] has, to the best of our knowledge, not yet appeared. The algorithm is complicated; its run time is not given explicitly, but estimated to be at least $\Omega(n^{120})$. Moreover, driven by topological arguments, the algorithm does not produce a witness if the graph is indeed a map graph, although a combinatorial description of this witness is at hand. In this view, an important question left open is whether there is a polynomial-time *certifying* algorithm in the sense of McConnell et al. [16], where a good candidate for a certificate would be the witness mentioned above.

Our Contribution. We give a purely combinatorial recognition algorithm for map graphs that have an outerplanar witness (rather than a planar witness). Map graphs with an outerplanar witness are general enough that they can have unbounded treewidth; in particular, cliques of any size may belong to this class of graphs. Our algorithm runs in time and space $O(n + m)$ and is certifying. This is the first non-trivial step towards a combinatorial and efficient recognition algorithm for general map graphs. Although the restriction to outerplanar witnesses is somewhat specific compared to the general case of planar witnesses, we will show structural properties for certain classes beyond (e.g. for $K_{2,k}$-free witnesses, and for graphs with small separators), that might be important for solving the general case.

We remark that the main algorithmic task is to compute a witness W, or to decide that none exists. Creating a map from W is a simple task that can be accomplished in linear time [2]. The crucial part of computing a witness W is that we know only a *subset* of the vertices of W; we need to do non-trivial algorithmic work in order to compute the remaining vertices of W. This is the reason why recognizing graphs that are *half-squares* of planar graphs is more challenging than recognizing graphs that are squares of planar graphs [15].

Related Work. By definition, planar graphs are an important subclass of map graphs, and planar graphs have been known since the 1970s to be recognizable in time $O(n)$ [12]. Nowadays, several other linear-time algorithms for planar graph recognition exist , and so it is natural to ask whether they can be generalized to the much wider class of map graphs. Let a *d-map* graph be a map graph that has a witness in which every intersection point has at most d neighbors (states). The planar graphs are exactly the map graphs for which at most three states meet at each single point; thus, by the well-known linear-time recognition algorithms for planar graphs, 3-map graphs can be recognized in $O(n)$ time.

An intricate cubic-time recognition algorithm for a subclass of 4-map graphs was given by Chen et al. [3]; here, the 4-map graphs are required to be *hole-free*, meaning that there is at most one connected region of the plane that is not covered by states or borders. However, even efficiently recognizing general 4-map graphs in polynomial time remains an open problem; Thorup's algorithm does not necessarily give an embedding minimizing the maximum degree of the intersection points, so it cannot be used to recognize 4-map graphs.

Another motivation for d-map graphs is the study of *1-planar* graphs, which are the graphs that can be embedded in the plane such that each edge crosses at most one other edge. Recognizing 1-planar graphs is NP-complete [10, 14]. Brandenburg [1] characterizes "fully triangulated" 1-planar graphs as hole-free 4-map graphs, which by Chen et al.'s algorithm [3] are hence efficiently recognizable. It would be interesting to know where exactly the recognition problem becomes NP-complete between these two graph classes.

Further interest stems from the parameterized complexity community: Generalizing earlier algorithms for problems on planar graphs, Demaine et al. [5] gave fixed-parameter algorithms for combinatorial optimization problems such as minimum dominating set in map graphs. Fomin et al. [9] gave PTAS's for optimization problems on map graphs; they later improved these to EPTAS's [8].

2 Preliminaries

All graphs considered in this paper are finite, simple, and undirected. For a graph G, let $V(G)$ and $E(G)$ denote its vertex set and edge set, and let $n := |V(G)|$ and $m := |E(G)|$. For a vertex $v \in V(G)$, let $N_G(v)$ be the set of neighbors of v in G. For a subset $V' \subseteq V(G)$, let $G[V']$ denote the subgraph of G induced by V'. For a graph G, its *square* G^2 is the graph on vertex set $V(G)$ in which two vertices are adjacent if their distance in G is at most two.

Witnesses. A *witness* of a map graph $G = (V, E)$ is a bipartite planar graph $W = (V \uplus I, E_W)$ with $E_W \subseteq V \times I$ such that $W^2[V] = G$. The graph $W^2[V]$ is also called the *half-square* of W, as it is the square of W restricted to the side V of the bipartition. The vertices in I are called *intersection points*, the vertices in V *real vertices*. We say that a witness W is a *tree witness* if it is a tree; analogously, *outerplanar witnesses* are outerplanar and the usual witnesses, which are planar, are sometimes called *planar witnesses*.

▶ **Proposition 1** ([2]). *A graph G is a map graph if and only if it has a* witness. *If so, there is a witness with at most $3n - 6$ intersection points.*

A direct consequence of this result is that the recognition problem for map graphs is in NP. Let G be a map graph with witness $W = (V \uplus I, E_W)$. Throughout this paper, we assume, without loss of generality, that every intersection point in I has degree at least two.

Let G be a map graph with witness W and let $P = v_1, v_2, \ldots, v_k$ be a path in G. A path P_W in W *corresponds* to P if $P_W = v_1, x_1, v_2, x_2, \ldots, x_{k-1}, v_k$ such that x_i is an intersection point that is adjacent to v_i and v_{i+1}, for $i = 1, \ldots, k-1$. Observe that any path P in G has some corresponding path in W, and any induced path P in G has a corresponding induced path P' in W, as every chord of P' in the bipartite witness W would join an intersection point with a real vertex and therefore generate a chord of P. More generally, for a subgraph of G that is induced by some vertex subset $V' \subseteq V(G)$, we specify the corresponding part in a witness of G:

▶ **Definition 2.** Let G be a map graph with witness W and let $U \subseteq V(G)$. A vertex $w \in W$ is *touched* by U if either $w \in U$ or w is an intersection point with at least two neighbors in U. The *touched set* $T(U)$ of U is the set of all vertices in W touched by U. The *touched subgraph* of U is $W[T(U)]$.

By using half-squares, we can get back from an induced subgraph $W[U]$ of W for some $U \subseteq V(G) \cup I$ to the *original subgraph* $W^2[U \cap V(G)]$ in G. Clearly, $W[U]$ witnesses $W^2[U \cap V]$. We will often use the following observation.

▶ **Observation 3.** *For every $U \subseteq V$, $W[T(U)]$ is a witness of $G[U]$. Moreover, $G[U]$ is connected if and only if $W[T(U)]$ is connected.*

Outerplanar Graphs. The following characterizations of planar and outerplanar graphs in terms of forbidden minors are well-known.

▶ **Proposition 4** (Wagner [19]). *A graph is planar if and only if it neither contains a K_5-minor nor a $K_{3,3}$-minor.*

▶ **Proposition 5.** *A graph is outerplanar if and only if it neither contains a K_4-minor nor a $K_{2,3}$-minor.*

▶ **Proposition 6** (Sysło [17]). *A triangle-free graph is outerplanar if and only if it does not contain a $K_{2,3}$-minor.*

Connectivity and SPQR trees. A graph is *connected* if every two of its vertices are connected by a path; the maximal connected subgraphs of G are called *components* of G. A *separator* S of a graph G is a subset of V such that $G - S$ has more components than G. For an integer $c \in \mathbb{N}$, a connected graph is *c-connected* if it either has at most c vertices or removing any set of less than c vertices leaves a connected subgraph. A 2-connected resp. 3-connected graph is sometimes called *biconnected* resp. *triconnected*.

For a graph G, an *SPQR tree* [6, 7] is a tree T for which each node $x \in V(T)$ has an associated multigraph G_x, called *skeleton* of x, and one of the following four types:

- S-node: then G_x is a cycle on at least three vertices.
- P-node: then G_x is a multigraph with two vertices and at least three edges.
- Q-node: then G_x is a multigraph with two vertices and two parallel edges.
- R-node: then G_x is a 3-connected graph.

Each edge xy between two nodes of T is associated with two directed *virtual* edges, one in G_x and one in G_y. Each edge in G_x can be virtual for at most one edge of T. All edges of S-, P- and R-nodes are virtual for some edge of T, and we simply call them *virtual edges*. An edge that is not virtual for any edge of T is *real*. Only skeletons of Q-nodes contain real edges and every Q-node skeleton contains exactly one real edge.

An SPQR tree T represents a biconnected graph G_T, formed as follows. Whenever an edge $xy \in E(T)$ associates the virtual edge of G_x with the virtual edge of G_y, form a larger graph as the *2-clique-sum* of G_x and G_y: We identify the endpoints of the virtual edge of G_x with that of G_y, and then delete the resulting edge. Applying this step to each edge of T (in any order) produces the graph G_T.

We assume throughout that T is *minimal*, which implies that its S- and P-nodes are pairwise non-adjacent. Under this assumption, T is uniquely determined from G. The graphs G_x associated with the nodes of T are called the *triconnected components* of G.

While the above definition coincides with the classical definition of SPQR trees, it is often more convenient to omit the Q-nodes from the tree as they carry little information. To this end, we simply remove each Q-node and replace the corresponding virtual edge in the skeleton of the neighboring node by a real edge. In the following we will use this modified version of SPQR trees.

3 Reduction along Small Separators

Clearly, every separator S of W that contains only vertices of $V(G)$ and for which at least two components of $W - S$ contain vertices in $V(G)$ is also a separator in G, as no edge that is generated by the half-square can cross S.

▶ **Lemma 7.** *Let G be a map graph with witness W and let $S \subseteq V(G)$. Then $C \mapsto W[T(C)]$ is a bijection from the vertex sets C of the components of $G - S$ to the components of $W - S$ that contain a vertex of $V(G)$. In particular, every separator S of G is a separator of W and,*

conversely, every separator $S \subseteq V(G)$ of W such that each component of $W - S$ contains a vertex of V is a separator of G.

Proof. If S is not a separator of G or of W, the statement follows from Observation 3. Hence, assume that S separates both G and W. Let A and B be the vertex sets of two arbitrary components of $G - S$. By Observation 3, the touched subgraphs $W[T(A)]$ and $W[T(B)]$ are connected in $W - S$. Assume to the contrary that some vertices $a \in A$ and $b \in B$ are contained in the same component of $W - S$. Then $W - S$ contains a shortest path from a to b, whose original subgraph in G must be a path from a to b on the same real vertices, i.e., disjoint from S. This contradicts that A and B are different components of $G - S$; hence, the components of $G - S$ partition V in exactly the same way as the components of $W - S$. In order to show that every $W[T(C)]$ is a component of $W - S$, it remains to prove that no intersection point is contained in two touched subgraphs $W[T(A)]$ and $W[T(B)]$. However, in that case, A and B would be connected in $G - S$. ◀

▶ **Lemma 8.** *A map graph G has a planar (outerplanar, tree) witness if and only if all of its biconnected components have planar (outerplanar, tree) witnesses, respectively.*

Proof of Lemma 8. Assume G has a planar (outerplanar, tree) witness and let C be the vertex set of any biconnected component of G. Then $W[T(U)]$ is a planar (outerplanar, tree) witness for $G[C]$ by Observation 3, as trees, planar and outerplanar graphs are closed under taking induced connected subgraphs.

If, on the other hand, each biconnected component of G has a planar (outerplanar, tree) witness, we can identify these witnesses along the cutvertices of G, obtaining a planar (outerplanar, tree) witness of G. ◀

We will thus assume that G is biconnected throughout the paper. Lemma 8 can be generalized to separators of size two as follows (a similar generalization exists for separators of size three). Consider a separator $S = \{u, v\}$ of size two in a biconnected graph. An *S-bridge* is either the edge uv, or the graph that is obtained from a component C of $G - S$ by adding the edges of G that join C with S, as well as their endpoints.

▶ **Lemma 9.** *Let G be a biconnected map graph that is not triconnected, and let $S = \{u, v\}$ be a separator of G. If uv is an edge of G, let $G' = G[C \cup S]$ for some component C of $G - S$, otherwise let $G' = (V', E')$ be the graph obtained from G by contracting some S-bridge B of G to a single edge. Then G' is a map graph, and any witness of G contains some witness of G' as a minor.*

Proof. If uv is an edge of G, then G' is an induced subgraph of G, and is therefore a map graph. Hence, any witness of G' is an induced subgraph of some witness of G, and so in this case the statement of the lemma holds.

Now assume that G does not contain the edge S. Then there exists a shortest path P in B connecting u and v. Obtain the graph $G'' = G - (V(B) \setminus V(P))$. Then G'' is an induced subgraph of G and hence is a map graph. Hence, any witness of G'' is an induced subgraph of some witness of G. Now contract the path P (whose interior vertices have degree 2) to a single edge, and call the resulting graph G'. Then any witness of G'' contains a path P'' realizing the path P. The internal vertices of P'' all have degree 2, and so contracting this path P'' to a path of length 2 yields a corresponding witness for G'. Hence any witness of G contains as a minor a witness of G'', which in turn contains as a minor a witness of G'. ◀

Lemma 9 will allow us to reduce along separators of size two.

4 Map Graphs with a Tree Witness

We characterize the map graphs that admit a tree witness. The characterization implies immediately a linear-time recognition algorithm for such graphs.

▶ **Lemma 10.** *A biconnected map graph has a tree witness if and only if it is a clique.*

Proof. Clearly a clique has a tree witness (the star). Conversely, assume that G is a biconnected graph with tree witness W and assume that G is not a clique. Then W is not a star, and it hence has two adjacent non-leaf vertices u and v. Since intersection points are pairwise non-adjacent, one of them, without loss of generality v, is not an intersection point. Then v is a cutvertex in W and, by Lemma 7, a cutvertex in G. This contradicts the assumption that G is biconnected. ◀

Lemma 8 and Lemma 10 immediately imply the following characterization of map graphs with a tree witness.

▶ **Theorem 11.** *A map graph has a tree witness if and only if each of its biconnected components is a clique.*

▶ **Corollary 12.** *Map graphs with a tree witness can be recognized in $O(n+m)$ time.*

5 Map Graphs with an Outerplanar Witness

In this section we study the problem of recognizing map graphs with an outerplanar witness. Due to Lemma 8, we can assume that the input graph G is biconnected. As bipartite planar graphs are triangle-free, we know with Proposition 6 that G has an outerplanar witness if and only if G has a $K_{2,3}$-minor free witness. Thus, all of the following proofs work for recognizing map graphs admitting witnesses that are $K_{2,3}$-minor free.

The next result states that triconnected map graphs G have witnesses with a very simple structure. For $k \geq 2$, a set of paths $\Pi_1, \ldots, \Pi_k$ in a witness $W = (V(G) \cup I, E_W)$ of G is *internally $V(G)$-disjoint* if no two paths Π_i and Π_j share an internal vertex in $V(G)$.

▶ **Lemma 13.** *For $k \geq 3$, a k-connected map graph G has a $K_{2,k}$-minor free witness if and only if it is a clique.*

Proof. If G is a clique, it has a tree witness that is a star whose center is an intersection point, and this witness is $K_{2,k}$-minor free. If G is not a clique, two vertices, say $u, v \in V(G)$, are not adjacent; let W be a witness of G. Since G is k-connected, G contains k internally vertex-disjoint paths from u to v. Let $P_1, \ldots, P_k$ denote such internally vertex-disjoint uv-paths of minimum total length; in particular, each of the P_i is an induced path. Denote by u_i the neighbor of u in P_i for $i = 1, \ldots, k$; see Fig. 1a for an example for $k = 3$. Let Π_i denote a path in W corresponding to P_i for $i = 1, \ldots, k$. Clearly the Π_i are internally $V(G)$-disjoint and each of them is an induced path. For a path Π containing vertices a and b, let $\Pi[a, b]$ denote the subpath of Π from a to b. Let $A = \bigcup_{i=1}^{k} V(\Pi_i[u, u_i]) \setminus \{u_1, \ldots, u_k\}$ and $B = \bigcup_{i=1}^{k} V(\Pi_i[u_i, v]) \setminus \{u_1, \ldots, u_k\}$; see Fig. 1b. Note that A consists exactly of u and the neighbors of u on the paths Π_i.

We claim that

(i) $W[A]$ and $W[B]$ are connected,
(ii) $A \cap B = \emptyset$, and
(iii) each vertex u_i has a neighbor in A and a neighbor in B.

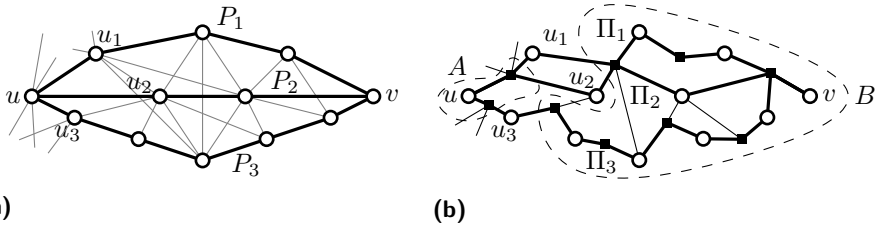


Figure 1 Illustration for the proof of Lemma 13 for $k = 3$. Vertices of G are empty disks, intersection points are small black squares.

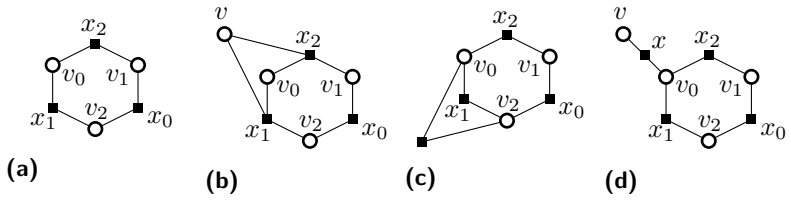


Figure 2 Illustration of the cases in the proof of Lemma 14.

Assume the claim holds and consider the graph $W' := W[A \cup B \cup \{u_1, \ldots, u_k\}]$. Since $W[A] \subseteq W'$ and $W[B'] \subseteq W'$ are connected (Claim (i)) and disjoint (Claim (ii)), we can contract these subgraphs into distinct vertices v_A and v_B, respectively. By Claim (iii), it follows that each of the u_i is adjacent to both v_A and v_B. Omitting a possible edge $v_A v_B$ yields a $K_{2,k}$-minor in W.

We now prove the claim. Statements (i) and (iii) follow immediately from the definitions of A and B via the paths Π_i. For (ii), assume that $A \cap B \neq \emptyset$ and let $x \in A \cap B$. Since the paths $\Pi_1, \ldots, \Pi_k$ are internally $V(G)$-disjoint and each of them is induced, x must be an intersection point. Since $x \in A$, x is adjacent to u. Since $x \in B$, it follows that x is adjacent to a vertex $w \in V(P_i) \cap B$ for some $i \in \{1, \ldots, k\}$, say without loss of generality $w \in V(P_1) \cap B$. Since x is adjacent to u and w, G contains the edge uw. Moreover, $w \neq u_1$, since $u_1 \notin B$. But then replacing the subpath from u to w in P_1, which contains u_1 in its interior, by the edge uw yields a shorter k-tuple of internally vertex-disjoint uv-paths in G. This contradicts the minimality of $P_1, \ldots, P_k$. ◀

In particular, a triconnected map graph with outerplanar witness, which is $K_{2,3}$-minor free, must be a clique. Hence, it suffices to investigate separators of size 2 in G.

In general map graphs, every two adjacent vertices have at least one neighboring intersection point in the witness, according to the definition of half-squares. Let G be a map graph with an outerplanar witness H. The intuition for the next lemma is that then every three vertices of a clique in G have a common neighboring intersection point in H. Note that this property is not true for arbitrary planar witnesses, as each of the pizza with crust, hamantash and riceball [2] contains three nations without any common intersection.

▶ **Lemma 14.** *Let G be a map graph with an outerplanar witness W and let v_0, v_1, v_2 be vertices of a clique of size at least 4. Then some intersection point in W is adjacent to v_0, v_1, v_2.*

Proof. Assume the contrary. Then there exist distinct intersection points x_0, x_1, x_2 such that x_i is adjacent to v_{i+1}, v_{i+2} but not to v_i, where indices are taken modulo 3; see Fig. 2a.

Now consider a vertex v of the clique that is distinct from the v_i. There is a path of length two from v to each of the v_i in W. If v is adjacent to two (or more) of the x_i, we

(a) **(b)**

Figure 3 Illustration of the proofs of Lemma 16(a) and Lemma 17(b).

immediately have a $K_{2,3}$-minor (Fig. 2b with branch vertices $\{x_1, x_2\}$); likewise, if there is an intersection point distinct from the x_i adjacent to two of the v_i ($\{v_0, v_2\}$ in Fig. 2c). It follows that v must reach one of the v_i, without loss of generality v_0, via an intersection point x distinct from the x_i, and x_i is not adjacent to v_1 and v_2 (Fig. 2d). But now, to reach v_1 and v_2, v either has to be adjacent to x_0, or it must use a new intersection point y adjacent to v_1 or v_2. In both cases, we obtain a $K_{2,3}$-minor. ◀

The proof of Lemma 14 shows that the bound "4" on the clique size is as small as possible.

▶ **Definition 15.** A clique C in a map graph G with witness W is *represented by an intersection point* if W contains at least one intersection point whose neighborhood is $V(C)$.

Cliques that are represented by exactly one intersection point are called "pizzas" by Chen et al. [2]. We now show that in outerplanar witnesses all cliques of size at least 4 must be represented in this way, thus significantly reducing the possible representations.

▶ **Lemma 16.** *Let G be a map graph with outerplanar witness W. Each maximal clique C of size at least 4 is represented by an intersection point.*

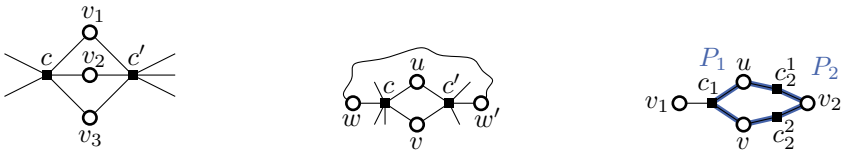
Proof. We first show that W contains an intersection point x that is adjacent to all vertices of C. This readily implies that x has no other neighbor, as any such neighbor would contradict the maximality of C. Let x be an intersection point in W with a maximum number of neighbors in C and assume to the contrary that C contains a vertex v_0 that is not adjacent to x. By Lemma 14, x has at least three neighbors v_1, v_2, v_3 in C. Let $V' = \{v_0, v_1, v_2, v_3\}$.

If there were a single intersection point $y \neq x$ adjacent to $\{v_1, v_2, v_3\}$, this would result in a $K_{2,3}$ with branch vertices x and y. This implies that any intersection point different from x can be adjacent to at most three of the vertices in V' (omitting at least one of $\{v_1, v_2, v_3\}$). On the other hand, by Lemma 14, for any such subset a corresponding intersection point exists. Thus, there exist intersection points $y \neq x$ and $z \neq x$ with $N(y) \cap V' = \{v_0, v_1, v_2\}$ and $N(z) \cap V' = \{v_0, v_2, v_3\}$; see Fig. 3a. Contracting the two edges yv_0 and zv_0 yields a $K_{2,3}$-minor in W. This contradicts outerplanarity.

◀

According to the definition of witnesses, maximal cliques of size 2 must also be represented by intersection points. Thus, the only cliques for which the representation is unclear are maximal cliques of size 3. In the following we show that cliques that cannot be represented by an intersection point in an outerplanar witness induce a special structure. Namely, any two of its vertices form a separator, and we further describe the way in which these separators decompose the graph.

▶ **Lemma 17.** *Let G be a map graph with outerplanar witness and let W be an outerplanar witness of G that maximizes the number of maximal cliques of size 3 that are represented by*



(a) **(b)** **(c)**

Figure 4 Illustration for the proof of Lemma 18. The wiggly line in (b) indicates an arbitrary path from w to w' avoiding u and v. The paths P_1 and P_2 are drawn bold (and blue) in (c).

intersection points. Let v_0 and v_1 be any two vertices of a maximal clique $C = \{v_0, v_1, v_2\}$ that is not represented by an intersection point. Then $\{v_0, v_1\}$ is a separator in G that separates v_2 from every other maximal clique of G containing v_0 and v_1.

Proof. Since C is not represented by an intersection point, there exist witness points x_0, x_1, x_2 such that x_i is adjacent to v_{i+1} and v_{i+2} but not to v_i (indices modulo 3). The cycle containing the x_i and v_i does not contain any vertex inside because of outerplanarity of W; in addition, any interior edge would contradict that C is not represented by an intersection point. Thus the x_i and v_i form the boundary of a face of W; see Fig. 3b.

If some x_i has degree 2, we can add the edge x_iv_i, which would result in an intersection point for the clique, contradicting the maximality of W. It follows that each x_i is adjacent to some $u_i \in V(G) - C$. We claim that $\{v_0, v_1\}$ separates u_2 from v_2 in W. Otherwise, there exists a path from u_2 to v_2 in $W - v_0 - v_1$. Together with the cycle formed by the v_i and x_i, this yields a $K_{2,3}$-minor, contradicting the outerplanarity. Thus, $\{v_0, v_1\}$ is also a separator in G that separates u_2 and v_2. ◀

▶ **Lemma 18.** *For a map graph G with outerplanar witness, the following statements hold.*

(i) *Any two maximal cliques of G share at most two vertices.*

(ii) *Any two vertices that are shared by two maximal cliques of G form a separator of G separating these cliques.*

(iii) *Any two vertices are shared by at most two maximal cliques.*

Proof. For (i), assume that C, C' are maximal cliques sharing vertices v_1, v_2, v_3. Since C, C' are distinct, they have size at least 4. By Lemma 16, they are represented by distinct intersection points c, c'; see Fig. 4a. Then v_1, v_2, v_3, c, c' induce a $K_{2,3}$; a contradiction.

For (ii), let $\{u, v\}$ be two vertices that are shared by two maximal cliques C and C'. Consider an outerplanar witness W of G that maximizes the number of cliques of size 3 that are represented by intersection points. If C and C' are realized by intersection points c and c', respectively, consider w and w' in $C \setminus C'$ and $C' \setminus C$, respectively. A path between these two vertices avoiding u and v yields a $K_{2,3}$-minor with branch vertices c and c', contradicting outerplanarity; see Fig. 4b. Hence, assume that one of the cliques is not realized as an intersection point. With Lemma 16, this clique has at most three vertices and the statement follows from Lemma 17.

For (iii), assume that two vertices u and v are shared by at least three maximal cliques C_0, C_1 and C_2. Note that each C_i has size at least three, as otherwise the C_i would not be distinct. According to (ii), every C_i contains a vertex v_i that is not in $C_{i+1} \cup C_{i+2}$ (indices taken modulo 3). If C_i is represented by an intersection point c_i, then let P_i denote the path uc_iv in W (path P_1 in Fig. 4c). If C_i is not represented by an

intersection point, then C_i has size 3, and W contains a path uc_i^1, v_i, c_i^2, v, where c_i^1 and c_i^2 are intersection points. We define P_i to be this path (path P_2 in Fig. 4c). Paths P_i and P_j for $i \neq j$ are internally disjoint by the definition of the v_i, and so three internally disjoint paths from u to v in W yield a $K_{2,3}$-minor. This contradicts the outerplanarity of W. ◀

▶ **Lemma 19.** *Let G be a biconnected map graph with an outerplanar witness W. Every separator $S = \{u, v\}$ of G of size two separates exactly two components.*

Proof. Assume to the contrary that $G - S$ contains at least three components C_i, $1 \leq i \leq 3$. By Lemma 7, the touched subgraphs $W[T(V(C_i))]$ are different components of $W - S$. For every i, there is a path P_i in G from u to v that contains a vertex of C_i as inner vertex, since S is minimal in G. In W, each P_i corresponds to a path from u to v that contains a vertex of $W[T(V(C_i))]$ as inner vertex (this may be either an intersection point or a real vertex). Since each P_i has length at least two, W contains a $K_{2,3}$-minor with branch vertices u, v. ◀

5.1 Structural Properties of Map Graphs with Outerplanar Witness

To obtain an efficient recognition algorithm for map graphs with outerplanar witness, two things remain to be done. First, we need to better understand the structure of those maximal cliques for which the representation in the witness is not already decided by the previous results. Second, we need to find a way to quickly enumerate all the relevant cliques in order to decide upon their representation in the witness.

As we have seen, the maximal cliques for which the representation cannot be an intersection point induce separating pairs in the input graph. This, together with the fact that certainly all cliques of size at least 4 belong to a single triconnected component of the input graph motivates the study of the triconnected components of the input graph. Essentially, we show:

1. Every maximal clique of size at least three shows up as a triconnected component of G.
2. A description of the maximal cliques of size three that cannot be represented by an intersection point.

The first item allows us to quickly compute all maximal cliques by exploiting the SPQR tree, which can be computed in linear time [11], rather then by a maximal clique enumeration algorithm, which might be much slower. The second item is used to determine the correct intersection points for all maximal cliques.

The following corollary follows immediately from applying Lemma 9 along the recursive definition of SPQR trees. Afterwards we derive further structural results on the triconnected components of a map graphs with outerplanar witnesses.

▶ **Corollary 20.** *Let G be a biconnected map graph with an outerplanar witness. Then each skeleton of the SPQR tree of G is a map graph with an outerplanar witness.*

▶ **Lemma 21.** *Let G be a biconnected map graph with an outerplanar witness. Then the SPQR tree of G satisfies the following properties:*

(i) *Every P-node skeleton consists of three parallel edges of which one is a real edge.*
(ii) *Every R-node skeleton is a clique.*

Proof. For (i), observe that, according to Lemma 19, there are exactly two components in $G - S$ for every separator $S = \{u, v\}$ of G of size two. Thus, every parallel P-node in the SPQR tree has at most three parallel edges (and at least three by definition of SPQR trees): two virtual ones and one edge from G.

For (ii), observe that the skeleton of an R-node is a triconnected graph. According to Corollary 20, this skeleton is a map graph with an outerplanar witness. Applying Lemma 13 with $k = 3$ implies that the skeleton is a clique. ◀

In fact, not only is every R-node skeleton a clique, but any such clique is a subgraph of G.

▶ **Lemma 22.** *Let G be a map graph with outerplanar witness. Then every R-node skeleton is a maximal clique that is a subgraph of G.*

Proof of Lemma 22. Consider an R-node skeleton S, which is a clique by Lemma 21(ii), and let uv be an edge of S that is not in G (thus, a virtual edge). Let G' be the graph that is obtained from G by contracting the subgraph corresponding to each remaining virtual edge into a single edge. Let G'' be the subgraph obtained from G' by removing all vertices in the subgraph that corresponds to the virtual edge uv, except for u, v and a shortest path between them. The graph G' is a map graph with outerplanar witness by Lemma 9, and G'' is a map graph with outerplanar witness by Observation 3, as G'' is an induced subgraph of G'. Thus, G'' contains the two cliques with vertex sets $V_1 = V(S) - \{u\}$ and $V_2 = V(S) - \{v\}$, which are maximal, as uv is not in G''. But then $|V_1 \cap V_2| \geq 2$, since $|V(S)| \geq 4$ and $|V_1 \cap V_2| \leq 2$ by Lemma 18(i). Hence $|V_1 \cap V_2| = 2$, $|S| = 4$ and thus, G'' is the graph obtained from K_4 by replacing an edge with a path of length at least two. This contradicts that, according to Lemma 18(ii), $V_1 \cap V_2$ is a separator of G''. ◀

▶ **Lemma 23.** *Let G be a biconnected map graph with outerplanar witness and let $C = \{u, v, w\}$ be a maximal clique in G. Then there is an S-node skeleton with vertex set $\{u, v, w\}$.*

Proof. By definition, the induced subgraph $G[C]$ is triconnected, hence there is a skeleton of a node q in the SPQR tree of G that contains all three vertices of C. However, q cannot be a P-node (as it contains only two vertices) and it cannot be an R-node, whose skeletons are well-known to contain at least four vertices. Hence, it must be an S-node. The skeleton of q cannot contain any other vertex than those in C, as it then would not be a cycle. ◀

It follows from Lemma 22 and Lemma 23 that we find all maximal cliques by considering the skeletons of the SPQR tree. In particular, this allows us to enumerate all maximal cliques in linear time. Recall that by Lemma 16 each maximal clique of size at least 4, which are precisely the cliques corresponding to R-nodes, must be represented by an intersection point. It remains to understand which maximal cliques of size three may not be represented by an intersection point.

▶ **Lemma 24.** *Let G be a biconnected map graph with outerplanar witness, let $C = \{u, v, w\}$ be a maximal clique in G, and let W be a witness that maximizes the number of cliques of size 3 represented by an intersection point. Then C is not represented by an intersection point in W if and only if the skeleton of the S-node corresponding to C has 3 virtual edges.*

Proof. Let S be the skeleton of the S-node on vertex set $\{u, v, w\}$, which exists by Lemma 23. Suppose, for the sake of contradiction, that C is not represented by an intersection point but one of the edges of S, say uv, is not virtual. Then the edges of C are represented by three distinct intersection points x_1, x_2, x_3, which form a cycle K together with the vertices of $\{u, v, w\}$ in W (cf. Fig. 2a). Assume, without loss of generality, that x_1 is adjacent to u and v. Since the edge uv is not virtual, the separator $\{u, v\}$ has exactly two split components of which one is an edge. As the other split-component is the one containing w, it follows that C is the only maximal clique that contains uv. Hence, x_1 has degree two in W.

We claim that K bounds a face of W in any outerplanar embedding of W. To justify the claim, observe that there cannot be a vertex embedded inside K due to W being outerplanar, and an edge embedded inside K would contradict the assumption that C is not represented by an intersection point. Thus, the claim holds, and the interior of K is empty.

We can then insert the edge x_1w to W, resulting in an outerplanar witness of G where C is represented by an intersection point. This, however, contradicts the maximality of W.

For proving sufficiency, assume to the contrary that C is represented by an intersection point c, but all edges of S are virtual. Then, the witness W contains for each virtual edge ab a path from a to b that avoids c and all inner vertices from each subgraph corresponding to a virtual edge different from ab. Thus, we obtain three internally disjoint paths connecting u to v, v to w and w to u, respectively. These three paths are all vertex-disjoint from $\{c\}$, and thus gives a K_4-minor in W with branching vertices $\{c, u, v, w\}$. This, however, contradicts the outerplanarity of W. ◀

5.2 Recognition Algorithm

Based on our structural observations, we give a linear-time algorithm for recognizing map graphs that admit an outerplanar witness, Algorithm 1.

Algorithm 1 Linear-Time Recognition Algorithm for Map Graphs with Outerplanar Witness

Input: A graph G.
Output: An outerplanar witness W of G if G is a map graph, "no" otherwise.

```
1: Create a candidate bipartite graph W; let V be one side of the bipartition of W.
2: for each biconnected component H_i of G do
3:    Compute an SPQR tree T_i of H_i.
4:    for each R-node R of T_i do
5:       if the skeleton of R is not a clique of non-virtual edges return "no"
6:       Add an intersection point p_R with neighborhood V(R) to W.
7:    for each S-node S of T_i whose skeleton is a clique of size 3 with some real edge do
8:       Add an intersection point p_S with neighborhood V(S) to W.
9: for each edge e = uv in G that is not yet represented by an intersection point do
10:   Add an intersection point p_uv of degree 2 with neighborhood {u, v} to W.
11: Test outerplanarity of W: if "yes", return W, else return "no".
```

The algorithm takes as input an arbitrary graph G. First, it decomposes G into its biconnected components $H_1, \ldots, H_t$. We know that G is a map graph with outerplanar witness if and only if each H_i is a map graph with outerplanar witness (see Lemma 8).

We seek to construct a bipartite witness candidate W of G as follows. Let the vertices of G be one side of the bipartition of W. For each H_i, compute the decomposition into its triconnected components, i.e., its SPQR tree T_i in linear time [11, 13]. For each R-node of T_i, check whether it is a clique of non-virtual edges. If not, then reject the graph H_i (and hence G) as not being a map graph with outerplanar witness. Otherwise, add an intersection point to W that represents that clique.

For each S-node of T_i that forms a clique of size 3 and that has a non-virtual edge, add an intersection point to W representing the clique.

Finally, for each edge of G that is not yet represented by one of the previously constructed intersection points, add a separate intersection point of degree 2 to W representing exactly this edge. Let W be the resulting candidate witness graph. Test whether W is outerplanar.

If W is outerplanar, then output the outerplanar witness W, otherwise reject the input graph G as not being a map graph with outerplanar witness.

▶ **Theorem 25.** *Map graphs with outerplanar witness can be recognized in $O(n+m)$ time.*

Proof. It is not hard to see that the above algorithm can be implemented to run in $O(n+m)$ time. In the following we prove the correctness.

First, assume that the algorithm outputs a witness W in the end. We show that W is a witness of the input graph G. Note that all intersection points we create represent either cliques of various sizes of G, or they only represent a single edge (if they are added in the last step). Thus, $W^2[V(G)] \subseteq G$. On the other hand, the last step ensures that $W^2[V(G)] \supseteq G$, and thus we have $W^2[V(G)] = G$, which shows that indeed G is a map graph with outerplanar witness W.

Conversely, assume that the algorithm rejects G, although G is a map graph with outerplanar witness. Let W^* be an outerplanar witness that minimizes the number of cliques of size 3 that are not represented by an intersection point.

The are only two steps in the algorithm where G may be rejected. First, when an R-node skeleton is not a clique that is subgraph of G. But in this case, G is not a map graph with outerplanar witness by Lemma 22. Second, G may be rejected when W is found not to be outerplanar. In this case, we will give an isomorphism from W to an induced subgraph of W^* whose restriction to V is the identity. This contradicts that W^* is outerplanar.

It suffices to give the mapping for intersection points only, as every witness contains an identical set V of real vertices. Let $w \in W$ be an intersection point of degree at least 4. Then $N(w)$ is a clique of size at least 4 in G. The intersection point w was added due to an R-node clique of that size, and hence $N(w)$ is a maximal clique of size at least 4. By Lemma 16, any outerplanar witness contains an intersection point representing that clique; we thus find an image for w in W^*. Note that a second vertex with the same neighborhood is not created; this would imply the existence of a second R-node skeleton with the same vertex set, which is impossible since any two skeletons share at most two vertices.

Let $w \in W$ be an intersection point of degree 3. Then $N(w)$ is a maximal clique of size 3, and w was created due to an S-node skeleton that contained a non-virtual edge. By Lemma 24, $N(w)$ is represented by an intersection point in W^* as well, and we thus find an image of w in W^*. Again, any intersection point mapped to the same image would imply the existence of a second S-node skeleton with the same vertex set, which is not possible.

Finally, let $w \in W$ be an intersection point of degree 2, which must have been added in the last step, and let $N(w) = \{u, v\}$. Clearly, W^* must contain an intersection point x adjacent to u and v. If x has degree at least 4, then $N(x)$ is a clique of size at least 4, which must show up as an R-node of the SPQR tree. But then the edge uv was already represented in W by an intersection point corresponding to x and the algorithm would not have added w. Similarly, if x has degree 3, then $N(x)$ is a maximal 3-clique. Since it is represented by the intersection point x, it follows that the corresponding S-node contains a virtual edge by Lemma 24. But then, again, the algorithm would have added an intersection point to W that corresponds to x. Thus, the degree of x must be 2, and we can choose it as an image for w. Since all degree-2 intersection points inserted in the last step represent distinct edges of G, no two degree-2 intersection points of W are mapped to the same intersection point of W^*. Hence, we have found an isomorphism from W to a subgraph of W^*. ◀

The correctness proof shows that the algorithm computes a smallest (with respect to subgraph inclusion) outerplanar witness, and that this witness is unique up to isomorphism.

6 Discussion

We gave an $O(n+m)$ time and space recognition algorithm for map graphs with an outerplanar witness. The algorithm is certifying. This result is a first step towards improving Thorup's recognition algorithm for map graphs with planar witness that requires time about $\Omega(n^{120})$.

For map graphs with outerplanar witness, Lemma 18 shows that any two maximal cliques of G intersect in at most two vertices. However, this property does not generalize, as for arbitrary map graphs any $k \geq 2$ maximal cliques can have intersections of unbounded size: namely, one can show that for every pair $k, \ell \in \mathbb{N}$ there exist map graphs $G_{k,\ell}$ with exactly k maximal cliques such that these cliques intersect in exactly ℓ vertices.

References

1 F. J. Brandenburg. On 4-map graphs and 1-planar graphs and their recognition problem. Technical report, ArXiv, 2015. URL: http://arxiv.org/abs/1509.03447.

2 Z.-Z. Chen, M. Grigni, and C. H. Papadimitriou. Map graphs. *J. ACM*, 49(2):127–138, 2002.

3 Z.-Z. Chen, M. Grigni, and C. H. Papadimitriou. Recognizing hole-free 4-map graphs in cubic time. *Algorithmica*, 45(2):227–262, 2006.

4 Z.-Z. Chen, X. He, and M.-Y. Kao. Nonplanar topological inference and political-map graphs. In *Proc. SODA 1999*, pages 195–204, 1999.

5 E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Trans. Algorithms*, 1(1):33–47, 2005.

6 G. Di Battista and R. Tamassia. On-Line Maintenance of Triconnected Components with SPQR-Trees. *Algorithmica*, 15(4):302–318, 1996.

7 G. Di Battista and R. Tamassia. On-Line Planarity Testing. *SIAM J. Comput.*, 25(5):956–997, 1996.

8 F. V. Fomin, D. Lokshtanov, N. Misra, and S. Saurabh. Planar $\mathcal{F}$-deletion: Approximation, kernelization and optimal FPT algorithms. *Proc. FOCS 2012*, pages 470–479, 2012.

9 F. V. Fomin, D. Lokshtanov, and S. Saurabh. Bidimensionality and geometric graphs. In *Proc. SODA 2012*, pages 1563–1575, 2012.

10 A. Grigoriev and H. L. Bodlaender. Algorithms for graphs embeddable with few crossings per edge. *Algorithmica*, 49(1):1–11, 2007. doi:10.1007/s00453-007-0010-x.

11 C. Gutwenger and P. Mutzel. A linear time implementation of SPQR-trees. In *Proc. GD 2000*, volume 1984 of *Lecture Notes Comput. Sci.*, pages 77–90, 2001.

12 J. Hopcroft and R. Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.

13 J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973.

14 V. P. Korzhik and B. Mohar. Minimal obstructions for 1-immersions and hardness of 1-planarity testing. *J. Graph Theory*, 72(1):30–71, 2013.

15 Y. L. Lin and S. S. Skiena. Algorithms for square roots of graphs. *SIAM J. Discrete Math.*, 8(1):99–118, 1995.

16 R. M. McConnell, K. Mehlhorn, S. Näher, and P. Schweitzer. Certifying algorithms. *Comput. Sci. Review*, 5(2):119–161, 2011.

17 M. M. Sysło. Characterizations of outerplanar graphs. *Discrete Math.*, 26(1):47–53, 1979.

18 M. Thorup. Map graphs in polynomial time. In *Proc. FOCS 1998*, pages 396–405, 1998.

19 K. Wagner. Über eine Eigenschaft der ebenen Komplexe. *Math. Ann.*, 114(1):570–590, 1937.

The p-Center Problem in Tree Networks Revisited

Aritra Banik[1], Binay Bhattacharya[2], Sandip Das[3], Tsunehiko Kameda[4], and Zhao Song[5]

1 **Advanced Computing and Microelectronics Unit, Indian Statistical Inst., Kolkata, India**
aritrabanik@gmail.com
2 **School of Computing Science, Simon Fraser University, Vancouver, Canada**
binay@sfu.ca
3 **Advanced Computing and Microelectronics Unit, Indian Statistical Inst., Kolkata, India**
sandipdas@isical.ac.in
4 **School of Computing Science, Simon Fraser University, Vancouver, Canada**
tikokameda@gmail.com
5 **Department of Computer Science, University of Texas, Austin, USA**
zhaos@utexas.edu

Abstract

We present two improved algorithms for weighted discrete p-center problem for tree networks with n vertices. One of our proposed algorithms runs in $O(n \log n + p \log^2 n \log(n/p))$ time. For all values of p, our algorithm thus runs as fast as or faster than the most efficient $O(n \log^2 n)$ time algorithm obtained by applying Cole's speed-up technique [10] to the algorithm due to Megiddo and Tamir [20], which has remained unchallenged for nearly 30 years.

Our other algorithm, which is more practical, runs in $O(n \log n + p^2 \log^2(n/p))$ time, and when $p = O(\sqrt{n})$ it is faster than Megiddo and Tamir's $O(n \log^2 n \log \log n)$ time algorithm [20].

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Facility location, p-center, parametric search, tree network, sorting network

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.6

1 Introduction

Deciding where to locate facilities to minimize the communication or travel costs is known as the *facility location problem*. It has attracted much research interest since the publication of the seminal paper on this topic by Hakimi [14]. For a good review of this subject, the reader is referred to [15]. It can be applied to locate fire stations, distribution centers, etc.

In the *p-center problem*, p centers are to be located in a network $G(V, E)$, so that the maximum (weighted) distance from any demand point to its nearest center is minimized. The simplest version of the problem (V/V/p) allows centers to be located only on vertices (V), and restricts demand points to be vertices. Other variations allow points on edges to be demand points (V/E/p), or points on edges (E) to be centers (E/V/p), or both (E/E/p). The vertices of a network could be weighted, i.e., the vertex weights can be different, or unweighted. In this paper we refer to weighted E/V/p as the *weighted discrete* p-center problem (*WDpC*). The p-center problem in a general network is NP-hard [17]. In this paper, we focus on the tree networks, on which there has been very little progress (for arbitrary p) since the mid-1980s.

1.1 Previous work

Megiddo [19] solved E/V/1 for the tree networks in $O(n)$ time, where n is the number of vertices. Megiddo and Tamir also studied this problem [20]. Kariv and Hakimi [17] presented an $O(m^p n^{2p-1} \log n/(p-1)!)$ time algorithm for WDpC in a general network, where m is the number of edges. Tamir [25] improved the above bound to $O(m^p n^p \log n \alpha'(n))$, where $\alpha'(n)$ is the inverse of Ackerman's function. Recently, Bhattacharya and Shi [7] improved it to $O(m^p n^{p/2} 2^{\log^* n} \log n)$ for $p \geq 3$, where $\log^* n$ denotes the iterated logarithm of n. A recent result on Klee's measure due to Chan [8] implies that this bound can be further improved to $O(m^p n^{p/2} \log n)$.

Frederickson [11, 12] solved the unweighted V/V/p, E/V/p and V/E/p problems in $O(n)$ time, independently of p. For the weighted tree networks, linear time algorithms have been proposed in the case where p is a constant [3, 24]. For arbitrary p, Kariv and Hakimi [17] gave an exhaustive $O(n^2 \log n)$ time algorithm. Megiddo's linear time feasibility test [21] can be parameterized to solve the problem in $O(n^2)$ time, using the idea introduced in [21]. Megiddo and Tamir [20] then provided an $O(n \log^2 n \log \log n)$ time algorithm, which can be made to run in $O(n \log^2 n)$ time using the AKS or similar $n \times O(\log n)$ sorting networks [1, 13, 23], together with Cole's improvement [10]. The $O(pn \log n)$ time algorithm due to Jeger and Kariv [16] is faster than all others if $p = o(\log n)$.

The running time of the algorithm of Megiddo and Tamir [20] is dominated by the time for computing the distance queries in their binary-search based algorithm. Frederickson [11, 12] used parametric search to design optimal algorithms for the unweighted p-center problem in tree networks. In parametric search, one first designs an α-*feasibility* test to see if p centers can be placed in such a way that every vertex is within *cost* (=distance weighted by the weight of the vertex) α from some center. In general, a set of candidate values for α is explicitly or implicitly tested as the algorithm progresses. Eventually, the search will settle on the smallest α value, α^*. The ideas presented in [11, 12] are for the unweighted case only, and therefore cannot be extended easily to WDpC. The question of whether an algorithm which runs faster than $O(n \log^2 n)$ time is possible for the tree networks has been open for a long time since.

To present our basic approach clearly, we first solve WDpC for balanced binary tree networks. We then generalize it to general (unbalanced) tree networks based on *spine tree decomposition* [4, 5].

1.2 Our contributions

Our major contributions in this paper are (i) an $O(p \log(n/p))$ time algorithm for testing α-feasibility for an arbitrary α, with preprocessing that requires $O(n \log n)$ time, (ii) a practical $O(n \log n + p^2 \log^2(n/p))$ time WDpC algorithm, which outperforms the $O(n \log^2 n \log \log n)$ time algorithm proposed in [20] when $p = O(\sqrt{n})$, and iii) an $O(n \log n + p \log^2 n \log(n/p))$ time WDpC algorithm based on AKS-like sorting networks [1, 13, 23], which improves upon the currently best $O(n \log^2 n)$ time algorithm [10, 20].

The rest of the paper is organized as follows. In Section 2 we first define the terms that are used throughout the paper. We then give a rough sketch of our parametric search approach to solving WDpC on balanced tree networks. We also propose our location policy that guides the placement of the centers. Section 3 describes preprocessing that we perform, in particular, the computation of upper envelopes and a preparation for fractional cascading. We then present in Section 4 the details of the feasibility test part of parametric search for balanced tree networks. The optimization part of parametric search is discussed in detail in

Section 5 for balanced tree networks. At the end of the section, we present our results for the general (unbalanced) tree networks.

2 Preliminaries

2.1 Definitions

Let $T=(V,E)$ denote a tree network, where each vertex $v \in V$ has weight $w(v)$ (≥ 0) and each edge $e \in E$ has a non-negative length. We write $x \in T$, if point x lies anywhere in T, be it on an edge or at a vertex. For $a, b \in T$, let $\pi(a,b)$ denote the unique path from a to b, and $d(a,b)$ its length. If a or b is on an edge, its prorated length is used. If T is a binary rooted with root vertex r, for any vertex $v \in V$, the subtree rooted at v is denoted by $T(v)$, and the parent of v ($\neq r$) is denoted by $p(v)$.

For a non-leaf vertex $v \in V$, let v_l (resp. v_r) denote its left (resp. right) child vertex, and define the left (resp. right) *branch* of v by $B(v_l) = T(v_l) \cup (v_l, v)$ (resp. $B(v_r) = T(v_r) \cup (v_r, v)$). We thus have $T(v) = B(v_l) \cup B(v_r)$, and the root of $B(v_l)$ (resp. $B(v_r)$) is v with degree 1 in $B(v_l)$ (resp. $B(v_r)$).

Let $V' \subseteq V$ and $x \in T$. We define the distance between a point x and V' by $d(x,V') \triangleq \min_{v \in V'}\{d(x,v)\}$. The *cost* of a vertex v at point x is given by $d(v,x)w(v)$. We say that point $x \in T$ *α-covers* V' ($\subseteq V$) if $\max_{v \in V'}\{d(x,v)w(v)\} \leq \alpha$. If α is clear from the context, we may simply say that x *covers* V'. A problem instance is said to be *α-feasible* if there exists p centers such that every vertex is α-covered by at least one of the centers. Those p centers are said to form a *p-center* [17]. For a vertex $v \in V$ and points $x \in T \backslash T(v)$, we define the *upper envelope*

$$E_v(x) = \max_{u \in T(v)} \{d(x,u)w(u)\}. \tag{1}$$

If $E_v(x) = d(u,x)w(u) = \alpha$, then vertex u is said to be an *α-critical vertex* in $T(v)$ with respect to $x \in T \backslash T(v)$, and is denoted by $u = cv(x, T(v))$. If α is clear from the context, we may call it just a critical vertex

2.2 Spine tree decomposition and upper envelopes

We give a brief review of *spine tree decomposition* [4, 5]. The materials in this subsection is not needed until Sec. 5.2. We can assume that given T is a binary tree; otherwise we can introduce $O(n)$ vertices of 0 weight and $O(n)$ edges of 0 length to make it binary. Thus each vertex has degree at most 3. Let r be the root of T, which can be chosen arbitrarily. Traverse T, starting on an edge incident to r. At each vertex visited, move to the branch that contains the largest number of leaf vertices, breaking a tie arbitrarily. When a leaf vertex, u, is reached, the path $\pi(v,u)$ is generated, and it is called the *top spine*, denoted by σ_1. We then repeat a similar traversal from each vertex on the generated spine, to generate other spines, until every vertex of T belongs to some spine.

Let $STD(T)$ denote the tree constructed by the spine tree decomposition of tree T, together with the *search tree* τ_{σ_l} for each *spine* σ_l, whose root is denoted by ρ_l [4, 5]. Fig. 1 illustrates a typical structure of spine σ_l and its search tree τ_{σ_l}. The horizontal line represents spine σ_l, and we name the vertices on it $v_1, v_2, \ldots$ from left to right.

The triangles represent subtrees hanging from σ_l. If a hanging subtree t is connected to vertex $v_i \in \sigma_l$, then we call the subgraph consisting of t, v_i, and the edge connecting them a *branch* of σ_l and denote it by B_i. Since we assume that the vertices of T have degree at most 3, there is at most one branch hanging from any vertex on the spine.

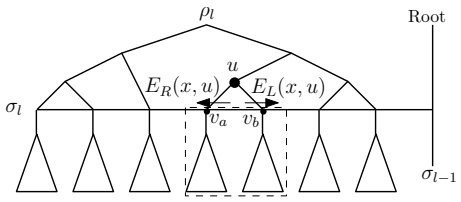


Figure 1 Search tree τ_{σ_l} for spine σ_l. $v_a = v_L(u)$ and $v_b = v_R(u)$.

For a node[1] u in τ_{σ_l}, let $v_L(u)$ (resp. $v_R(u)$) denotes the leftmost (resp. rightmost)[2] vertex on σ_l that belongs to the subtree $\tau_{\sigma_l}(u)$. We introduce upper envelope $E_L(x,u)$ (resp. $E_R(x,u)$) for the costs of the vertices in the branches of σ_l that belong to $\tau_{\sigma_l}(u)$, for point x that lies to the right (resp. left) of vertex $v_R(u)$ (resp. $v_L(u)$). See Fig. 1. Since $E_L(x,u)$ and $E_R(x,u)$ are upper envelopes of linear functions, they are piecewise linear. For each node u of $STD(T)$ we compute $E_L(x,u)$ and $E_R(x,u)$, and store them at u as sequences of bending points (their x and y coordinates). These upper envelopes can be computed in $O(n \log n)$ time by the following lemma.

▶ **Lemma 1** ([4, 5]). *The path from any leaf to the root of $STD(T)$ has $O(\log n)$ nodes on it.*

2.3 Our approach

Except in the last subsection of the paper, we assume that the given tree T is balanced with respect to its root r, so that its height is $O(\log n)$. If not, we can use spine tree decomposition that transforms T in linear time to a structure that has most of the properties of a balanced binary tree. Working on a balanced binary tree network also helps us to explain the essence of our approach, without getting bogged down in details. Our algorithms consist of a lower part and an upper part. In the lower part, we test α-feasibility for a given cost α, and in the upper part we carry out Megiddo's *parametric search* [18]. To perform a feasibility test, we first identify the *α-peripheral centers*, below which no center needs be placed. Once all the q ($< p$) α-peripheral centers are identified, we place $p - q$ additional centers to α-cover the vertices that are not covered by the α-peripheral centers. If no more than p centers are used to α-cover the entire tree T, then the α-feasibility test is successful. Theorem 8 shows that, using fractional cascading, α-feasibility can be tested in $O(p \log(n/p))$ time after preprocessing, which takes $O(n \log n)$ time.

The second part of parametric search finds the smallest α value, α^*. We work on T bottom-up, doing essentially the same thing as in the first part. Whenever α is used in the first part, we need to invoke an α-feasibility test [18]. At each level of T, we need to invoke α-feasibility tests $O(l)$ times at level l. Therefore the total number of invocations is $O(\log^2 n)$, and the total time is $O(p \log^2 n \log(n/p))$ after preprocessing, yielding one of our main results stated in Theorem 10.

2.4 Center location policy

Suppose that we want to place a center c_i in a tree network T to α-cover a subset V_i of vertices that are connected. We propose the following location policy.

---

[1] A 'node' is more general than a vertex of T. A vertex is also a node, because it belongs to τ_{σ_l}, but not every node is a vertex.

[2] Right (resp. left) means towards (resp. away from) the parent spine σ_{l-1} of σ_l.

Root-centric policy: *Place c_i at the point that α-covers all the vertices in V_i and is closest to root r of T.*

It is easy to prove the following lemmas.

▶ **Lemma 2.** *If a set of p centers α-covers all the vertices in V, then there is a partition of vertex set $\{V_i \mid i = 1, \ldots, p\}$, where each V_i is the vertex set of a connected part of T, such that the root-centric location policy locates each center c_i that α-covers V_i.*

▶ **Lemma 3.** *Let $\{c_i \mid i = 1, \ldots, p\}$ be p centers obeying the root-centric policy that together α-cover V. For each center c_i, find a vertex $v \in V_i$ with maximum cost $d(v, c_i)w(v)$ that is the farthest from the root, and name it g_i. Then it satisfies $c_i \in \pi(g_i, r)$.*

Proof. If $c_i \notin \pi(g_i, r)$, then c_i could move closer to r, a contradiction. ◀

3 Preprocessing

3.1 Upper envelopes

According to our definition of upper envelope $E_v(x)$ for subtree $T(v)$ (see (1)), if v is a leaf vertex, we have

$$E_v(x) = d(x, v)w(v), \tag{2}$$

for any $x \in T$. Let v_l (resp. v_r) be the left (resp. right) child vertex of a non-leaf vertex $v \in V$. Then for any $x \in T \setminus T(v)$, we have

$$E_v(x) = \max\{E_{v_l}(x), E_{v_r}(x), d(x, v)w(v)\}. \tag{3}$$

Function $E_v(x)$ is piecewise linear in $x \in \pi(v, r)$ and can be represented by a sequence of bending points. In the sequence representing $E_v(x)$, in addition to the values of $E_v(x)$ at the bending points, we insert the values of $E_v(x)$ evaluated at all the $O(\log n)$ vertices on $\pi(v, r)$.[3]

▶ **Lemma 4.** *If T is balanced, then $\{E_v(x) \mid v \in V, x \in \pi(v, r)\}$ can be computed bottom-up in $O(n \log n)$ time and $O(n \log n)$ space.*

In the rest of this paper we assume that the given tree T is a balanced binary tree. If not we can use spine tree decomposition [4, 5, 6], which shares many useful properties of a balanced tree.

3.2 Fractional cascading

From now on we assume that we have the bending points of $\{E_v(x) \mid v \in V, x \in \pi(v, r)\}$ at our disposal. The second task of preprocessing is to merge the bending points of $\{E_v(x) \mid v \in V, x \in \pi(v, r)\}$ to prepare for fractional cascading [9]. Again we do this bottom up, merge-sorting the two sequences of bending points into one at each vertex. Since each vertex causes at most $O(\log n)$ bending points in $\{E_v(x) \mid v \in V, x \in \pi(v, r)\}$, the total number of bending points is $O(n \log n)$.

[3] We mix those values among the bending points, so that we know on which edges the bending points lie.

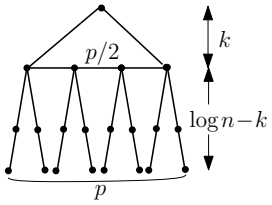


Figure 2 Illustration for the proof of Lemma 5.

4 α-Feasibility

4.1 Peripheral centers

As a result of preprocessing, we have the upper envelopes $\{E_v(x) \mid v \in V, x \in \pi(v,r)\}$. To find the peripheral centers, α-peripheral we carry out *truncated* pre-order DFS (depth-first-search), looking for the vertex-point pairs (v,x) satisfying $E_v(x) = \alpha$, which means v is an α-critical vertex in $T(v)$ with respect to x.

▶ **Procedure 1.** *`Find-Peripheral-Centers`* (α)

Perform pre-order DFS, modified as follows, where v is the vertex being visited.

1. *If $\exists x \in (v, p(v))$ such that $E_v(x) = \alpha$, return x as an α-peripheral center,*[4] *and backtrack.*
2. *If $p+1$ α-peripheral centers have been found, then return `Infeasible` and stop.* ◀

To carry out Step 1 efficiently, we perform binary search with key α in the merged sequence of bending points (of the upper envelopes) stored at the root r, and follow the relevant pointers based on fractional cascading.

▶ **Lemma 5.** *Procedure `Find-Peripheral-Centers`(α) visits $O(p \log(n/p))$ vertices.*

Proof. The number of vertices that Procedure `Find-Peripheral-Centers`(α) visits is the largest when the α-peripheral centers are as low as possible and they separate from each other as high as possible. This extreme case is illustrated in Fig. 2, where $p = 2^k - 1$ for some integer k.

The total number of edges that are traversed is given by

$$O(p(\log n - k) + p) = O(p(\log n - \log p) + p) = O(p \log(n/p)),$$

where the second term, p, is an upper bound on the number of vertices at depth k or shallower. ◀

▶ **Lemma 6.** *If $\{E_v(x) \mid v \in V\}$ are available, all the α-peripheral centers can be found in $O(p \log(n/p))$ time.*

Proof. If fractional cascading is used in Step 1 of Procedure 1, it runs in amortized constant time per vertex. The rest follows from Lemma 5. ◀

[4] We assume that the trivial case, where one center at root r α-covers the entire tree, is dealt with specially, which is straightforward.

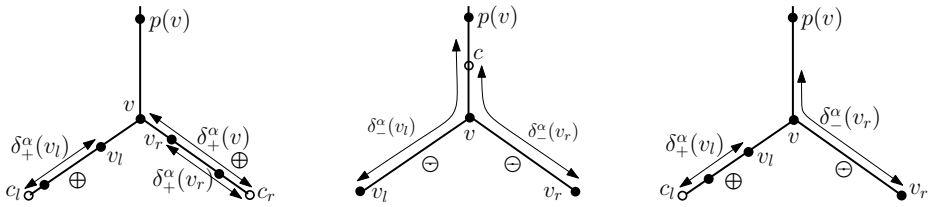


Figure 3 (Left) $c_l \in B(v_l)$ and $c_r \in B(v_r)$; (Middle) A center is needed within $\delta^{\alpha}_{-}(u)$ from u; (Right) $c_l \in B(v_l)$.

4.2 α-Feasibility test

Given an α value, suppose that we have found q ($< p$) α-peripheral centers, following the root-centric location policy. We replace each α-peripheral center by a *dummy vertex*, and define the *trimmed tree* $T'_{\alpha} = (V'_{\alpha}, E'_{\alpha})$. Its vertex set V'_{α} consists of two types of vertices: the *first type* is a vertex that lies on the path between a dummy vertex and root r, inclusive. If any such vertex has only one child vertex among them, then the other child vertex of T (called a vertex of the *second type*) is kept in T' to represent the α-critical vertex in the subtree of T rooted at that vertex. In what follows, we use T' instead of T'_{α} for simplicity, since the implied α will be clear from the context. It is easy to see that tree T' contains $O(q \log n)$ vertices. Without loss of generality, we consider each vertex of the second type as the right child of its parent.

Let u be a vertex of the second type. Then we must have visited u during the execution of `Find-Peripheral-Centers`(α), and no α-peripheral center was placed in subtree $T(u)$. At the time of this visit, we identified the α-critical vertex in $T(u)$, which implies that we can store this α-critical vertex at u as a by-product of `Find-Peripheral-Centers`(α) at no extra cost.

Later, we will be introducing more centers, in addition to α-peripheral centers, working on the trimmed tree T' bottom up. For each vertex in T', its subtrees can be one of the following types:

- $\ominus$-subtree: The centers in it, if any, do not α-cover all the vertices in the subtree.
- $\oplus$-subtree: The centers in it α-cover all the vertices in the subtree, and possibly outside it.

If $T'(v)$ is a $\oplus$-subtree, let $\delta^{\alpha}_{+}(v)$ denote the distance from v to the highest center in $T'(v)$ at or below v. See the leftmost figure of Fig. 3, where v_l (resp. v_r) is the left (resp. right) child vertex of v, and c_l (resp. c_r) is the highest center placed in $T'(v_l)$ (resp. $T'(v_r)$).

If $T'(v)$ is a $\ominus$-subtree, on the other hand, let $\delta^{\alpha}_{-}(v)$ denote the minimum distance from v to a point above $T'(v)$ within which a center must be placed to α-cover the uncovered vertices in $T'(v)$. See the middle figure in Fig. 3.

Let us discuss how to process the trimmed tree T', to introduce additional centers closer to the root in order to α-cover more vertices. We perform post-order DFS on T', always visiting the left child of a vertex first. Assume that we explored $T'(v_l)$ first and then $T'(v_r)$, and we are just back to v, and that $\delta^{\alpha}_{-}(v_l)$ or $\delta^{\alpha}_{+}(v_l)$ (resp. $\delta^{\alpha}_{-}(v_r)$ or $\delta^{\alpha}_{+}(v_r)$) are available at vertex v_l (resp. v_r). For each dummy leaf vertex v of T', we have $\delta^{\alpha}_{+}(v) = 0$. At each vertex v visited, we have one of the following three cases.

(a) [Both are $\oplus$-subtrees]. In the leftmost figure of Fig. 3, c_l (resp. c_r) is the highest center in $T'(v_l)$ (resp. $T'(v_r)$). We compute

$$\delta = \min\{\delta^{\alpha}_{+}(v_l) + d(v, v_l), \delta^{\alpha}_{+}(v_r) + d(v, v_r)\}, \tag{4}$$

which is the distance from v to the nearest center in $T'(v)$. If $\delta \cdot w(v) \leq \alpha$, then v is α-covered by c_l or c_r. Otherwise (i.e., even the center in $T'(v)$ that is nearer to v cannot α-cover v) v must be covered by a center placed above v, and $T'(v)$ $(= \{v\})$ now becomes a $\ominus$-subtree of $p(v)$.

(b) [Both are $\ominus$-subtrees]. See the middle figure of Fig. 3. If $\delta^{\alpha}_{-}(v_l) < d(v, v_l)$, for example, we need to place a center c_l on the edge (v, v_l), and $T'(v)$ now becomes a $\oplus$-subtree, provided v is α-covered by c_l. If both c_l and c_r are placed this way, we set $\delta^{\alpha}_{+}(v) = \min\{d(c_l, v), d(c_r, v)\}$, provided one of them α-covers v. If no center needs to be placed on (v, v_l) or (v, v_r), then we compute

$$\delta = \min\{\delta^{\alpha}_{-}(v_l) - d(v, v_l), \delta^{\alpha}_{-}(v_r) - d(v, v_r)\}. \tag{5}$$

We need a center within $\min\{\delta, \alpha/w(v)\}$ above v. These are some of the typical cases, which illustrate kinds of necessary operations. Procedure `Merge`$(v; \alpha, T)$, given below, deals with the other cases as well, not mentioned here, exhaustively.

(c) [One is a $\ominus$-subtree and the other is a $\oplus$-subtree]. We assume without loss of generality that the left (resp. right) subtree is a $\oplus$-subtree (resp. $\ominus$-subtree), as shown in the rightmost figure of Fig. 3, and c_l is the highest center in $T'(v_l)$. As in Case (b), we first test if $\delta^{\alpha}_{-}(v_r) < d(v, v_r)$, and if so place a center c_r on edge (v, v_r). Then we have case (a). Otherwise, we need to test if c_l α-covers the uncovered vertices in $T'(v_r)$ as well as v. If not, they must be covered by a new center above v.

We now present a formal procedure that deals with all possible cases. We will use it for $T = T'$.

▶ **Procedure 2.** *Merge*$(v; \alpha, T)$

Case (a): $[T(v_l) = \oplus, T(v_r) = \oplus]$ *Compute δ using* (4). *If $\delta \cdot w(v) \leq \alpha$, then set $\delta^{\alpha}_{+}(v) = \delta$. Otherwise, make $T(v)$ a $\ominus$-subtree of $p(v)$ with $\delta^{\alpha}_{-}(v) = \alpha/w(v)$.*

Case (b): $[T(v_l) = \ominus, T(v_r) = \ominus]$ *If $\delta^{\alpha}_{-}(v_l) < d(v, v_l)$ (resp. $\delta^{\alpha}_{-}(v_r) < d(v, v_r)$), place a center c_l (resp. c_r) on the edge (v, v_l), (resp. (v, v_r)) at distance $\delta^{\alpha}_{-}(v_l)$ from v_l (resp. $\delta^{\alpha}_{-}(v_r)$ from v_r). If c_l and/or c_r α-covers v, then make $T(v)$ a $\oplus$-subtree of $p(v)$ with $\delta^{\alpha}_{+}(v) = \min\{d(c_l, v), d(c_r, v)\}$, where $d(c_l, v) = 0$ (resp. $d(c_r, v) = 0$) if c_l (resp. c_r) is not introduced. If neither of them covers v, then make $T(v)$ a $\ominus$-subtree of $p(v)$ with $\delta^{\alpha}_{-}(v) = \alpha/w(v)$. If neither c_l nor c_r is introduced, then compute δ using* (5) *and make $T(v)$ a $\ominus$-subtree of $p(v)$ with $\delta^{\alpha}_{-}(v) = \min\{\delta, \alpha/w(v)\}$.*

Case (c): $[T(v_l) = \oplus, T(v_r) = \ominus]$[5] *If $\delta^{\alpha}_{-}(v_r) < d(v, v_r)$, then place a center c_r on edge (v, v_r) at distance $\delta^{\alpha}_{-}(v_r)$ from v_r, set $\delta^{\alpha}_{+}(c_r) = d(v, c_r) = d(v, v_r) - \delta^{\alpha}_{-}(v_r)$, and go to Case (a). Otherwise,*

[5] The case $[T(v_l) = \ominus, T(v_r) = \oplus]$ is symmetric.

(i) *If c_l covers v (i.e., $\{\delta^\alpha_+(v_l)+d(v_l,v))\}w(v) \le \alpha$), and c_l also covers $T(v_r)$ (i.e., $\delta^\alpha_+(v_l)+d(v_l,v_r) \le \delta^\alpha_-(v_r)$), then let $\delta^\alpha_+(v) = \delta^\alpha_+(v_l)+d(v_l,v)$.*
(ii) *In all the remaining cases, set $\delta^\alpha_-(v) = \min\{\delta^\alpha_-(v_r) - d(v,v_r), \alpha/w(v)\}$.* ◀

It is easy to show that

▶ **Lemma 7.** *After preprocessing, `Merge-I`$(v;\alpha,T)$ runs in constant time.*

We now formally state our algorithm for testing α-feasibility.

▶ **Algorithm 1.** `Feasibility-Test` (α, T)
1. *Call `Find-Peripheral-Centers`(α).*
2. *Construct the trimmed tree T', consisting of the vertices of the first type and those of the second type and the edges connecting them. For each vertex u of the second type, compute the α-critical vertex for $T'(u)$.*
3. *Perform a post-order depth-first traversal on T', invoking `Merge`$(v;\alpha,T')$ on each vertex v visited.*
4. *If a set of no more than p centers covering T has been found, then return `Feasible` and stop. If the p centers found so far do not totally cover T, then return `Infeasible` and stop.* ◀

▶ **Theorem 8.** *For a balanced tree network, `Feasibility-Test`(α,T) runs in $O(p\log(n/p))$ time, excluding the preprocessing time.*

Proof. Step 1 runs in $O(p\log(n/p))$ time by Lemma 6. Step 2 can be carried out at the same time as Step 1 in $O(p\log(n/p))$ time. Step 3 also runs in $O(p\log(n/p))$ time by Lemma 7. Lastly, Step 4 takes constant time. ◀

5 Optimization

We will employ Megiddo's parametric search [18], using the α-feasibility test we developed in Sec. 4.2. We maintain a lower bound $\underline{\alpha}$ and an upper bound $\overline{\alpha}$ on α^*, where $\underline{\alpha} < \alpha^* \le \overline{\alpha}$. Eventually we will end up with $\alpha^* = \overline{\alpha}$. If we succeed (resp. fail) in an α-feasibility test, then it means that $\alpha \ge \alpha^*$ (resp. $\alpha < \alpha^*$), so we update $\overline{\alpha}$ (resp, $\underline{\alpha}$) to α.

5.1 Balanced tree networks

Based on Theorem 8, the main theorem in [18] implies:

▶ **Theorem 9.** *WDpC for the balanced tree networks with n vertices can be solved in $O(n\log n + p^2\log^2(n/p))$ time.*

We propose another algorithm which performs better than the first algorithm referred to in the above theorem for some range of values of p. For this algorithm we will show later that we need to test feasibility $O(\log^2 n)$ times. This fact, together with Theorem 8, leads to the following theorem.

▶ **Theorem 10.** *WDpC for the balanced tree networks with n vertices can also be solved in $O(n\log n + p\log^2 n\log(n/p))$ time.*

In the rest of this subsection we prove Theorem 10. Let $l = 1, 2, \ldots, k$ be the *levels* of T from top to bottom, where the root r is at level 1 and the leaves are at level $k = \lceil \log n \rceil = O(\log n)$. At each vertex, we need to perform a few feasibility tests. Since there are 2^{l-1}

vertices at level l of T, using prune and search, we can know the results of the feasibility tests at all the vertices of level l after actually performing only $O(\log(2^{l-1})) = O(l)$ feasibility tests. The total for all levels is thus $O(\sum_{l=1}^{\log n} l) = O(\log^2 n)$, as claimed above.

It is easy to prove the following lemma.

▶ **Lemma 11.** *Let $v_a, v_b \in V$.*

(a) [17] *Vertices v_a and v_b have the equal cost*

$$\alpha(v_a, v_b) = \frac{d(v_a, v_b)w(v_a)w(v_b)}{w(v_a) + w(v_b)} \tag{6}$$

at a point $c(v_a, v_b) \in \pi(v_a, v_b)$.

(b) *Let $v_a, v_b \in T(v)$, and suppose that $w(v_a) \neq w(v_b)$, and let $w(v_a) < w(v_b)$ without loss of generality. If $d(v_a, v)w(v_a) \geq d(v_b, v)w(v_b)$ holds, then vertices v_a and v_b have the equal cost*

$$\alpha'(v_a, v_b) = \frac{\{d(v_a, v) - d(v_b, v)\}w(v_a)w(v_b)}{w(v_b) - w(v_a)}, \tag{7}$$

at a point $c'(v_a, v_b) \in \pi(v, r)$. If $d(v_a, v)w(v_a) < d(v_b, v)w(v_b)$, then vertex v_b has a higher cost than v_a at all points on $\pi(v, r)$.[6]

If we let $v_b = v$ in Case (b) in the above lemma, v_a and v have the equal cost

$$\alpha'(v_a, v) = \frac{d(v_a, v)w(v_a)w(v)}{w(v) - w(v_a)}, \tag{8}$$

at a point $c'(v_a, v_b) \in \pi(v, r)$.

We now need to modify the definition of the critical vertex given in Sec. 2.1. With respect to $x \in T \setminus T(v)$, we are interested in the vertex $u \in T(v)$, such that $\alpha(x, u)$ is maximum, We call such a u the *critical vertex* with respect to x and denote it by γ_v. The main difference of the optimization part from the feasibility test part is that we cannot find the exact locations of the centers until the very end. However, making use of critical vertices, it is possible to identify the component of T that is to be α^*-covered by each new center. So, we will isolate/detach them one by one from T, and repeat the process.

Let v_l and v_r be the two child vertices of a vertex v at level l. When we visit v, moving up T, we need to either isolate a subtree to be covered by a center that lies below v, or determine the critical vertex in $T(v)$ to be carried higher. Whenever the result of an α-feasibility test shows that $\alpha \geq \alpha^*$, we update $\overline{\alpha}$ and assume that $\overline{\alpha} > \alpha^*$ holds, and introduce a new center (without an exact location), as necessary. This assumption will be justified if $\overline{\alpha}$ is updated later. If $\overline{\alpha}$ is never updated thereafter,[7] it implies that $\overline{\alpha} = \alpha^*$. See Lemma 12.

Based on (6), if

$$\alpha(v, \gamma_{v_l}) \geq \alpha^* \text{ (resp. } \alpha(v, \gamma_{v_r}) \geq \alpha^*), \tag{9}$$

we assume that $\alpha(v, \gamma_{v_l}) > \alpha^*$ (resp. $\alpha(v, \gamma_{v_r}) > \alpha^*$), and cut the edge (v, v_l) (resp. (v, v_r)) to detach a new component below v to be covered by the new center placed in it.[8] We need

[6] In this case, the equal cost point lies on $\pi(v, v_b)$.
[7] $\underline{\alpha}$ may be updated.
[8] Note that if $\alpha(v, \gamma_{v_l}) = \alpha^*$, for example, we cannot isolate a component.

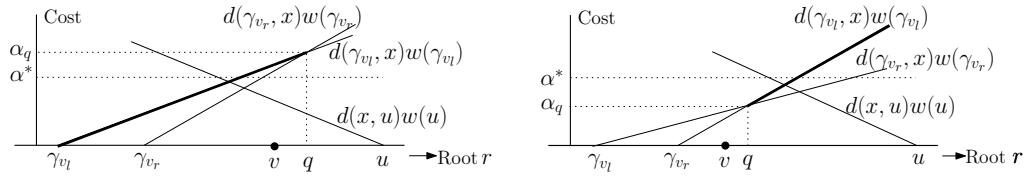


Figure 4 The cost lines of $\gamma_{v_l} \in B(v_l)$ and $\gamma_{v_r} \in B(v_r)$ intersect at q above v: (Left) Cost α_q at intersection q is higher than α^* ($\alpha_q > \alpha^*$); (Right) $\alpha_q < \alpha^*$.

not know the exact position of the new center. If two new centers are introduced this way, vertex v must be α^*-covered by a center placed above v, and v becomes a (tentative) critical vertex for $T(v)$ with respect to x above v. If only one of the inequalities in (9) holds and only (v, v_l) (resp. (v, v_r)) is cut, then either v or γ_{v_r} (resp. γ_{v_l}) becomes a critical vertex for $T(v)$, based on the outcome of $\alpha'(v_a, v)$-feasibility test. See (8).

Consider the remaining case, where neither inequality in (9) holds. We need to determine a critical vertex in $T(v)$ with respect to x above v.

To this end, we first find the intersection $q = c'(\gamma_{v_l}, \gamma_{v_r}) \in \pi[v, r]$ of the two cost lines $d(\gamma_{v_l}, x)w(\gamma_{v_l})$ and $d(\gamma_{v_r}, x)w(\gamma_{v_r})$, and its cost $\alpha_q = \alpha'(\gamma_{v_l}, \gamma_{v_r})$, assuming the condition for (7) is met. We then test α_q-feasibility. If $\alpha_q \geq \alpha^*$, as in the left figure of Fig. 4, then we set $\gamma'_v = \gamma_{v_l}$ (resp. $\gamma_v = \gamma_{v_r}$) if $w(\gamma_{v_l}) \leq w(\gamma_{v_r})$ (resp. $w(\gamma_{v_l}) > w(\gamma_{v_r})$). If $\alpha_q < \alpha^*$, on the other hand, as in the right figure of Fig. 4, then we set $\gamma'_v = \gamma_{v_r}$ (resp. $\gamma_v = \gamma_{v_l}$) if $w(\gamma_{v_l}) \leq w(\gamma_{v_r})$ (resp. $w(\gamma_{v_r}) < w(\gamma_{v_l})$). In order to find the true critical vertex γ_v in place of γ'_v, we need to take v into consideration as well. This time we use $\alpha'(v_a, v)$ of (8) instead of (7). In the future we will be testing vertices $u \notin T(v)$ to see if the cost of the intersection between $d(x, u)w(u)$ and $d(\gamma_v, x)w(\gamma_v)$ is lower than α^* or not. We must choose the critical vertex that gives the highest cost near α^*, which is indicated by a thick line segment in Fig. 4.

In any case, we need to perform a constant number of feasibility tests per vertex visited. Whenever an α-feasibility test in (9) succeeds (resp. fails), we update $\overline{\alpha}$ (resp. $\underline{\alpha}$) to α.

▶ **Lemma 12.** *The optimal cost α^* equals $\overline{\alpha}$ at the end of the above steps.*

Proof. It was shown by Kariv and Hakimi [17] that α^* has the value $d(u, v)/(1/w(u)+1/w(v))$ for some pair of vertices u and v. See Lemma 11(a). Assume that $\alpha^* < \overline{\alpha}$ and there is a pair of vertices u and v in the same partition $V_i \subset V$ (Lemma 2) such that $\alpha^* = d(u, v)/(1/w(u) + 1/w(v))$, but we haven't tested them, a contradiction. ◀

5.2 General tree networks

We use spine tree decomposition (STD), reviewed in Sec. 2.2, for general (unbalanced) tree networks. The counterparts to Theorems 8 and 9 hold with the same complexities.

▶ **Theorem 13.**

(a) *We can test α-feasibility in $O(p \log(n/p))$ time, excluding the preprocessing, which takes $O(n \log n)$ time.*

(b) *WDpC for general tree networks with n vertices can be solved in $O(n \log n + p^2 \log^2(n/p))$ time.*

Proof. Part (a) can be proved in essentially the same way as we proved Theorem 8 in Sec. 4.2. Instead of working directly on the given tree T, we first construct $STD(T)$ and compute upper envelopes at its nodes. The concepts of the $\ominus$-subtree and $\oplus$-subtree can be carried

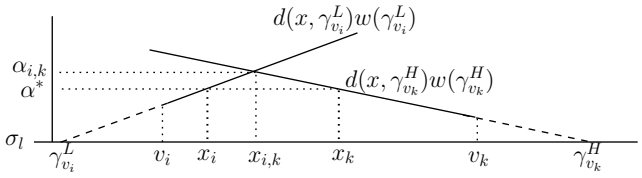


Figure 5 B_i and B_k are each a $\ominus$-branch.

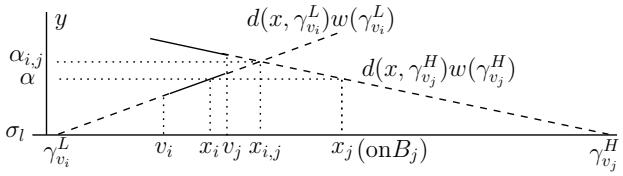


Figure 6 Point x_j is the mapped image onto σ_l of the highest center in B_j.

over to $STD(T)$. One complication is that we need to work on a group of $\ominus$-branches, instead of single $\ominus$-subtrees, but we can process them in the same order of time as in the balanced tree case. Part (b) is implied by part (a) by the main theorem in Megiddo [18]. ◀

As for the counterpart to Theorem 10, we need to use AKS-like sorting networks [1, 13, 22, 23], as in [10].

▶ **Theorem 14.** *WDpC for the general tree networks with n vertices can be solved in $O(n \log n + p \log^2 n \log(n/p))$ time.*

Proof (Informal). Let us first analyze how many times we need to perform feasibility tests when $STD(T)$ is used for a non-balanced tree network. Let n_l be the number of vertices in the spines at level l, so that we have $\sum_{l=1}^{\lambda} n_l = n$, where λ is the number of levels in $STD(T)$. We now consider one particular spine σ_l at level l. Let v_i and v_k be two vertices on σ_l, from which branches B_i and B_k hang. Assume first that both B_i and B_k are $\ominus$-branches, and let γ_{v_i} (resp. γ_{v_k}) be the α^*-critical vertices in B_i (resp. B_k). If γ_{v_i} is at distance d_i from v_i, then we map it onto σ_l at distance d_i from v_i.

There can be up to two such positions on σ_l (or its extension if it is not long enough), and we call the lower (resp. higher)[9] one $\gamma^L_{v_i}$ (resp. $\gamma^H_{v_i}$). Fig. 5 illustrates $\gamma^L_{v_i}$ and $\gamma^H_{v_k}$. In this figure each cost function $d(x,\gamma^L_{v_i})w(\gamma^L_{v_i})$ is represented by a solid and a dashed line, where the solid (resp. dashed) part shows its value on σ_l (in B_i). Similarly for the cost function $d(x,\gamma^H_{v_k})w(\gamma^H_{v_k})$. In this figure, they meet at $x_{i,k}$ on σ_l, and at this point the cost is $\alpha_{i,k} > \alpha^*$. This implies that $x_i \prec x_k$, where x_i (resp. x_k) is the point on σ_l where the cost of $\gamma^L_{v_i}$ (resp. $\gamma^H_{v_k}$) is α^*. This in turn means that a single center cannot α^*-cover both $\gamma^L_{v_i}$ and $\gamma^H_{v_k}$. If we had $\alpha_{i,k} \le \alpha^*$, then a center would cover both of them.

Consider next the case where B_i is a $\ominus$-branch and B_j is a $\oplus$-branch, as shown in Fig. 6.

In this case, the dashed part of the cost function $d(x,\gamma^H_{v_j})w(\gamma^H_{v_j})$ takes the value α^* at $x_j \in B_j$, which means that B_j is a $\oplus$-branch. The two cost functions $d(x,\gamma^L_{v_i})w(\gamma^L_{v_i})$ and $d(x,\gamma^H_{v_j})w(\gamma^H_{v_j})$ intersect at $x_{i,j}$ in their dashed parts, which implies that they meet in B_j. Since the corresponding cost $\alpha_{i,j}$ is larger than α^* in this figure, a center at $x_j \in B_j$ cannot α^*-cover γ_{v_i}.

[9] Lower (resp. higher) means farther (resp. nearer) from/to the root.

The above discussion implies that whether the cost at the intersection of two cost lines is higher or lower than α^*, which can be tested by a feasibility test, determines if an additional center needs to be introduced or not. Each feasibility test determines the relative order of x_i, x_j, x_k, etc., for all vertices on spine σ_l. This is tantamount to sorting x_i, x_j, x_k, etc., which we can do by a sorting network, such as the AKS sorter. By examining the sorted sequence, and scanning σ_l from its lower end, we can determine the number of centers needed on σ_l.

Finally, we need to find the α^*-critical vertex that represents the part of spine σ_l not covered by the centers introduced so far, or the center that could cover additional vertices in the next higher spine. Namely, spine σ_l may become a $\ominus$-branch or a $\oplus$-branch *vis-à-vis* the next higher spine. If it becomes a $\ominus$-branch, there may be several candidates for the α^*-critical vertex. The situation is somewhat to that depicted in the left figure in Fig. 4, where γ_{v_l} and γ_{v_r} are the two candidates. The α^*-critical vertex is whichever candidate whose cost line reaches α^* first, i.e., at the lowest position.

If σ_l becomes a $\oplus$-branch in the next higher spine, we want to find the α^*-critical vertex in σ_l that can cover the "farthest" vertex in the next higher spine. Therefore, among the candidate critical vertices we pick the one whose cost line reaches α^* last, i.e., at the highest position..

Following Megiddo [20], for each spine we employ an AKS sorting network. The number of inputs to the AKS sorting networks employed at level l is thus $2n_l$. Each such AKS sorting network has $O(\log n_l)$ layers of comparators, and their sorted outputs can be computed with $O(\log n_l)$ calls to a feasibility test with Cole's speed up [10]. The total number of calls at all levels $l = 1, 2, \ldots, \lambda$ with Cole's speed up is thus $O(\sum_{l=1}^{\lambda} \log n_l)$. Since $\sum_{l=1}^{\lambda} n_l = O(n)$, we have $\sum_{l=1}^{\lambda} \log n_l \leq \lambda \log(n/\lambda) = O(\log^2 n)$. Since each feasibility test takes $O(p \log(n/p))$ by Theorem 8 (extended to $STD(T)$), the total time spent by the feasibility tests is $O(p \log^2 \log(n/p))$. In addition, we need time to compute the median at each layer of the AKS networks, which is $O(n_l)$ per layer and $O(n_l \log n_l)$ at level l. Summing this for all levels, we get $O(\sum_{l=1}^{\lambda} n_l \log n_l) = O(n \log n)$. ◀

6 Conclusion and Discussion

We have presented an algorithm for the weighted discrete p-center problem for tree networks with n vertices, which runs in $O(n \log n + p \log^2 n \log(n/p))$ time. This improves upon the previously best $O(n \log^2 n)$ time algorithm [10]. The main contributors to this speed up are spine tree decomposition, which enabled us to limit the tree height to $O(\log n)$, and the root-centric location policy, which made locating centers simple. Fractional cascading helped to shave a factor of $O(\log n)$ off the time complexity in Theorem 8. The $O(n \log^2 n)$ time algorithm [10] and ours both make use of the AKS sorting network [1], which is impractically large. However, recently AKS-like sorting networks with orders of magnitude reduced sizes have been discovered [13, 23], and further size reduction in the not-so-distant future may make the above algorithms more practical. We also presented a practical $O(n \log n + p^2 \log^2(n/p))$ time WDpC algorithm, which improves upon the $O(n \log^2 n \log \log n)$ time algorithm [20] when $p = O(\sqrt{n})$.

In Lemma 4 we showed that it takes $O(n \log n)$ time and space to compute the set of bending point sequences for the upper envelopes at all the vertices. Suppose that the weight of a vertex is increased arbitrarily, which could influence the locations of some centers, if the vertex becomes critical for a center. We can test this situation without updating the upper envelopes, and thus without increasing the time requirement. Therefore, every p-center query

with the weight of one vertex arbitrarily increased can be answered in $O(p \log(n/p) \log n)$ time. This result realizes a sub-quadratic algorithm for the minmax regret p-center problem in tree networks [2].

References

1. M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proc. 15th ACM Symp. on Theory of Comput. (STOC)*, pages 1–9, 1983.
2. I. Averbakh and O. Berman. Minimax regret p-center location on a network with demand uncertainty. *Location Science*, 5:247–254, 1997.
3. Boaz Ben-Moshe, Binay Bhattacharya, and Qiaosheng Shi. An optimal algorithm for the continuous/discrete weighted 2-center problem in trees. In *Proc. LATIN 2006*, volume LNCS 3887, pages 166–177, 2006.
4. R. Benkoczi. *Cardinality constrained facility location problems in trees.* PhD thesis, School of Computing Science, Simon Fraser University, Canada, 2004.
5. R. Benkoczi, B. Bhattacharya, M. Chrobak, L. Larmore, and W. Rytter. Faster algorithms for k-median problems in trees. *Mathematical Foundations of Computer Science, Springer-Verlag*, LNCS 2747:218–227, 2003.
6. Binay Bhattacharya, Tsunehiko Kameda, and Zhao Song. Minmax regret 1-center on a path/cycle/tree. In *Proc. 6th Int'l Conf. on Advanced Engineering Computing and Applications in Sciences (ADVCOMP)*, pages 108–113, 2012.
7. Binay Bhattacharya and Qiaosheng Shi. Improved algorithms to network p-center location problems. *Computational Geometry*, 47:307–315, 2014.
8. Timothy M. Chan. Klee's measure problem made easy. In *Proc. Symp. on Foundation of Computer Science (FOCS)*, pages 410–419, 2013.
9. Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1:133–162, 1986.
10. R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34:200–208, 1987.
11. G.N. Frederickson. Optimal algorithms for partitioning trees and locating p centers in trees. Technical Report CSD-TR-1029, Purdue University, 1990.
12. G.N. Frederickson. Parametric search and locating supply centers in trees. In *Proc. Workshop on Algorithms and Data Structures (WADS), Springer-Verlag*, volume LNCS 519, pages 299–319, 1991.
13. Michael T. Goodrich. Zig-zag sort: A simple deterministic data-oblivious sorting algorithm running in $O(n \log n)$ time. arXiv:1403,2777v1 [cs.DS] 11 Mar2014, 2014.
14. S.L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12:450–459, 1964.
15. Trevor S. Hale and Christopher R. Moberg. Location science research: A review. *Annals of Operations Research*, 123:21–35, 2003.
16. M. Jeger and O. Kariv. Algorithms for finding p-centers on a weighted tree (for relatively small p). *Networks*, 15:381–389, 1985.
17. O. Kariv and S.L. Hakimi. An algorithmic approach to network location problems, part 1: The p-centers. *SIAM J. Appl. Math.*, 37:513–538, 1979.
18. N. Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.*, 4:414–424, 1979.
19. N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30:852–865, 1983.
20. N. Megiddo and A. Tamir. New results on the complexity of p-center problems. *SIAM J. Comput.*, 12:751–758, 1983.

21 N. Megiddo, A. Tamir, E. Zemel, and R. Chandrasekaran. An $O(n \log^2 n)$ algorithm for the kth longest path in a tree with applications to location problems. *SIAM J. Comput.*, 10:328–337, 1981.

22 M.S. Paterson. Improved sorting networks with $O(\log n)$ depth. *Algorithmica*, 5:75–92, 1990.

23 Joel Seiferas. Sorting networks of logarithmic depth, further simplified. *Algorithmica*, 53:374–384, 2009.

24 Q. Shi. *Efficient algorithms for network center/covering location optimization problems*. PhD thesis, School of Computing Science, Simon Fraser University, Canada, 2008.

25 A. Tamir. Improved complexity bounds for center location problems on networks by using dynamic structures. *SIAM J. Discrete Mathematics*, 1:377–396, 1988.

A Simple Mergeable Dictionary*

Adam Karczmarz

Insitute of Informatics, University of Warsaw, Poland
a.karczmarz@mimuw.edu.pl

Abstract

A mergeable dictionary is a data structure storing a dynamic subset S of a totally ordered set $\mathcal{U}$ and supporting predecessor searches in S. Apart from insertions and deletions to S, we can both merge two arbitrarily interleaved dictionaries and split a given dictionary around some pivot $x \in \mathcal{U}$. We present an implementation of a mergeable dictionary matching the optimal amortized logarithmic bounds of Iacono and Özkan [11]. However, our solution is significantly simpler. The proposed data structure can also be generalized to the case when the universe $\mathcal{U}$ is dynamic or infinite, thus addressing one issue of [11].

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, E.1 Data Structures

Keywords and phrases dictionary, mergeable, data structure, merge, split

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.7

1 Introduction

Let $\mathcal{U}$ be some totally ordered set. An *ordered dictionary* is a data structure maintaining a set $S \subseteq \mathcal{U}$ and supporting the following operations:

- $S \leftarrow$ Make-Set(): create an empty set S.
- Insert(S, x): add an element $x \in \mathcal{U}$ to the set S.
- Delete(S, x): remove an element $x \in \mathcal{U}$ from the set S.
- $y \leftarrow$ Search(S, x): find the largest $y \in S$ such that $y \leq x$ (if such y exists).

Typically, such dictionaries also allow traversing the stored sets in order in linear time.

We call a data structure a *mergeable dictionary* if it supports two additional operations:

- $C \leftarrow$ Merge(A, B): create a set $C = A \cup B$. The sets A and B are destroyed.
- $(A, B) \leftarrow$ Split(C, x): create two sets $A = \{y \in C : y \leq x\}$ and $B = C \setminus A$, where $x \in \mathcal{U}$. The set C is destroyed.

Note that the operation Merge does not pose any conditions on its arguments. It should not be confused with the commonly used operation Join(A, B) which merges its arguments under the assumption that all the elements of A are no larger than the smallest element of B.

The ordered dictionary problem is well understood and various optimal solutions have been developed, including balanced binary search trees such as AVL trees or red-black trees [9], and skip-lists [14]. Each of these data structures performs the operations Insert, Delete, Search on a set S in $O(\log |S|)$ worst-case time. Most ordered dictionaries can be easily extended to support the operations Join and Split within the same time bounds. Clearly, it is not possible to achieve $o(|A| + |B|)$ worst-case bound for the Merge(A, B) operation, as that would lead to a $o(n \log n)$ comparison-based sorting algorithm. Nevertheless, it is interesting to study the amortized upper bounds of the mergeable dictionary operations.

* Supported by the grant NCN2014/13/B/ST6/01811 of the Polish Science Center.

Iacono and Özkan [11] developed the only data structure to date which provably supports *all* the mergeable dictionary operations in amortized logarithmic time. It is worth noting that their definition of a mergeable dictionary is a bit different: they define it to be a data structure maintaining a partition of a finite universe $\mathcal{U}$ of size n. The set of operations they support is MERGE, SPLIT, SEARCH and FIND, where FIND(x) returns the unique element of the partition containing $x \in \mathcal{U}$. We find it more appropriate to call a data structure supporting such an interface *an ordered union-split-find* data structure instead. This small difference in the definition does not influence the core of the problem. The amortized lower bound of $\Omega(\log n)$ for at least one of the operations MERGE, SPLIT and FIND is an easy consequence of the lower bounds for *partial sums* and *dynamic connectivity* [15].[1]

However, the data structure of Iacono and Özkan has two drawbacks. First, both the methods they used and the analysis are quite involved. Specifically, in order to achieve the goal, they used a highly non-trivial potential function for analysis, extended the *biased skip list* of [2] to support various finger-search-related operations. They also developed an element weighting scheme that allows MERGE to be performed in time proportional to the decrease of the potential. Second, their weighting scheme depends heavily on the differences of ranks [2] of individual elements in $\mathcal{U}$ and as a result it is not clear how to generalize the data structure to work with potentially infinite universes, such as $\mathbb{R}$, in an online fashion. The subtlety of handling such universes lies in the fact that one can always insert a new element between consecutive elements of a stored set.

In this paper we show a very simple data structure that addresses the former issue in the case of a finite universe $\mathcal{U}$. We then generalize our approach and obtain a slightly more involved data structure supporting infinite/dynamic universes.

Techniques. We map the universe $\mathcal{U}$ into a set of $O(\log |\mathcal{U}|)$-bit labels and implement the mergeable dictionary S as a *compressed trie* with leaves corresponding to the elements of S. This resembles the approach used by Willard [18] to obtain an efficient dynamic predecessor search data structure called the *x-fast trie*. However, as we aim at performing the operations in amortized $O(\log |\mathcal{U}|)$ time, the additional components that make up the x-fast-trie are unnecessary. The tight structure of tries allows us to use a fine-grained potential function for analyzing the amortized cost of mergeable dictionary operations. As a result, both the implementation and the analysis are surprisingly simple. In order to obtain linear space, the paths consisting of trie nodes with a single child are replaced with edges labeled with bit strings. As we work in the word-RAM model, each such label can be stored in $O(1)$ machine words.

In order to allow dynamic (or potentially infinite) universes, we look at the used tries from a somewhat different perspective: each trie can be seen as a subtree of a single tree T representing the entire universe. Our method is to maintain such a tree T representing the part of the universe that contains all the elements of the stored sets. We implement T with a *weight-balanced B-tree* of [1] and represent the individual sets as compressed subtrees of T. This in turn enables us to control the behavior of our potential function when inserting previously unseen elements of $\mathcal{U}$.

Related Work. Ordered dictionaries supporting arbitrary merges but no splits have also been studied, although somewhat implicitly. Brown and Tarjan [4] showed how to merge two

[1] See also [13]. For a detailed reduction in the case of the mergerable dictionary operations, see Section 2.2.
[2] The rank of x in $\mathcal{U}$ is defined as the size of the set $\{y \in \mathcal{U} : y \leq x\}$.

AVL trees of sizes n and m, $n \leq m$, in $O(n \log(m/n))$ worst-case time, which they further proved to be optimal. They also showed that using their merging method, any sequence of merges on a set of n singleton sets can be performed in $O(n \log n)$ time. Hence, assuming no Split operations, each of the operations Insert, Delete, Merge can be performed in amortized $O(\log n)$ time.

An alternative method to handle the case of no splits, called *segment merging*, follows as an easy application of *finger search trees* [10]. A finger search tree is capable of joining two ordered dictionaries of sizes n and m in $O(\log \min(n, m))$ time, as well as splitting an ordered dictionary into parts of sizes n, m in $O(\log \min(n, m))$ time. In order to merge two arbitrarily interleaved ordered dictionaries A and B, where $|A| \leq |B|$, we can partition the set $A \cup B$ into a minimal number of *segments* $\{C_1, \ldots, C_l\}$ such that for each i we have either $C_i \subseteq A$ or $C_i \subseteq B$ and $\max\{C_i\} \leq \min\{C_{i+1}\}$. The finger search tree allows to sequentially extract the segments C_i from either A or B in $O(\log |C_i|)$ time. The segments are then joined in $O\left(\sum_i^l \log |C_i|\right)$ time. As $l \leq |A|$, from the concavity of a logarithmic function it follows that this merging algorithm runs in $O\left(|A| \log \frac{|B|}{|A|}\right)$ time, which is no worse than the algorithm of Brown and Tarjan.

The ordered dictionaries supporting both arbitrary merges and splits have been first considered explicitly by Iacono and Özkan [11]. However, as they point out, the need for a data structure supporting a similar set of operations emerged in several prior works, e.g., the *union-split-find* problem [13], the first non-trivial algorithm for pattern matching in a LZ77-compressed text [6], and the data structure for *mergeable trees* [8]. In particular, Farach and Thorup [6] used a potential function argument to prove (somewhat implicitly) that when using the segment merging strategy, any sequence of Merge and Split operations performed on a collection of subsets with n distinct elements has amortized $O(\log n)$ segments per merge. [3] Hence, segment merging can be used to obtain a mergeable dictionary with $O(\log^2 n)$ amortized bounds, even if one uses an ordinary balanced binary search tree in place of a finger search tree.

On the other hand, Lai [13] proved that if we store individual sets as finger search trees and use the segment merging strategy as discussed above, there exist a sequence of merges and splits that leads to $\Omega(\log^2 n)$ amortized time per Merge. This implies that even if we use an optimal merging algorithm, splits may cause the merges to run asymptotically slower.

As we later show, an optimal solution to the ordered union-split-find problem can be easily obtained by extending our simple data structure for a finite universe. However, in the case of mergeable trees and the compressed pattern matching algorithm of Farach and Thorup, an optimal mergeable dictionary does not immediately lead to a better solution. The mergeable trees [8] generalize mergeable dictionaries in a way analogous to how dynamic trees [16] generalize dynamic paths. Thus, employing the main idea of [16], i.e., decomposing a tree into a set of paths and representing each path with a mergeable dictionary, would lead to amortized $O(\log^2 n)$ time per Merge, a bound already achieved by the data structure of Georgiadis et al. [8]. Obtaining a more efficient data structure for mergeable trees would probably require developing some kind of biased version of a mergeable dictionary.

In the algorithm of Farach and Thorup, a somewhat more powerful variant of a mergeable dictionary is needed. For $\mathcal{U} = \{0, \ldots, N\}$ one also needs to support efficient shifting of all the elements of a set $S \subseteq \mathcal{U}$ by a constant. Even though the data structure of Iacono and Özkan

[3] In fact, the very same potential function was used by Iacono and Özkan [11] to analyze their mergeable dictionary data structure.

can be easily augmented to support such shifts, in our data structure, the representation of a set might dramatically change after such a shift. Nevertheless, the algorithm of Farach and Thorup has another bottleneck and it is not clear how to remove it, even equipped with a mergeable dictionary supporting efficient shifts. It is worth noting that a more efficient solution to LZ77-compressed pattern matching was developed recently, using very different methods [7].

Organization of the Paper. In Section 2 we develop a simple solution for finite universes. We generalize the used methods to obtain a data structure for infinite universes in Section 3. In Section 4 we make some concluding remarks and discuss a few further interesting questions concerning the mergeable dictionaries.

2 A Data Structure for a Finite Universe

In this section we assume that $|\mathcal{U}| = n$ and that the entire universe (along with the order of the elements) is known beforehand: in particular, $\mathcal{U}$ is allowed to be preprocessed during the initialization. We treat n as a measure of the problem size and consequently assume that we operate in the word-RAM model, where arithmetic and bitwise operations on $\lceil \log_2 n \rceil$-bit integers are performed in constant time.

Representation. Let D be the smallest integer such that $2^D \geq n$. Each $x \in \mathcal{U}$ is assigned a bit string $\mathrm{bits}(x)$ of length D such that for $y \in \mathcal{U}$, $x \leq y$, $\mathrm{bits}(x)$ is lexicographically not greater than $\mathrm{bits}(y)$.

Recall that a trie T storing a set of strings W is a rooted tree with single-character *labels* on edges. T has a unique node v_p for each distinct prefix p of some of the strings of W. If sz is a prefix of some word of W and z is a character, then $c_z(v_s) = v_{sz}$ is a child of v_s and the edge $v_s - v_{sz}$ is labeled z. If sz is not a prefix of any word of W, we set $c_z(v_s) = \mathbf{nil}$. If some node $v \in T$ corresponds to a prefix p, then we call $p = \ell(v)$ a *label* of the node v. Note that $\ell(v)$ is the string composed of the subsequent characters on the root-to-v path in T. A subtree of T rooted at v, denoted by T_v, can also be seen as a trie, but storing strings (in fact, some suffixes of the words in W) that are $|\ell(v)|$ characters shorter. If $W = \emptyset$, we set the corresponding trie to an empty trie, also denoted by **nil**.

Each set $S \subseteq \mathcal{U}$ is represented as a trie $\mathcal{T}(S)$ storing the strings $\mathcal{B}(S) = \{\mathrm{bits}(s) : s \in S\}$. Note that all the stored strings are of the same length and thus the leaves of $\mathcal{T}(S)$ are at the same depth and correspond to individual elements of $\mathcal{B}(S)$. The tries we use are binary, i.e., each node $v \in \mathcal{T}(S)$ has at most two children $c_0(v)$ and $c_1(v)$. We sometimes call $c_0(v)$ ($c_1(v)$) the left (right respectively) child of v and we call the subtrees $\mathcal{T}(S)_{c_0(v)}$, $\mathcal{T}(S)_{c_1(v)}$ the left and right subtrees of v, correspondingly. Each leaf v stores the value $q(v) \in \mathcal{U}$ such that $\mathrm{bits}(q(v)) = \ell(v)$. See Figure 1 for an example.

Implementing the Operations. We now show how to implement the operations. The operation Make-Set returns **nil**.

To insert an element x into S, we descend down the tree $\mathcal{T}(S)$ to the deepest node v corresponding to a prefix of $\mathrm{bits}(x)$ (if $\mathcal{T}(S) = \mathbf{nil}$, we first create the root node). We then create $D - |\ell(v)|$ new nodes so that the created leaf has label $\mathrm{bits}(x)$. The operation Delete(S, x) is basically a reverse of Insert(S, x): we locate the leaf v with label $\mathrm{bits}(x)$ and sequentially remove its ancestors until we reach a node w with label being a prefix of some string of $\mathcal{B}(S \setminus \{x\})$. Both Delete and Insert take $O(D) = O(\log n)$ time.

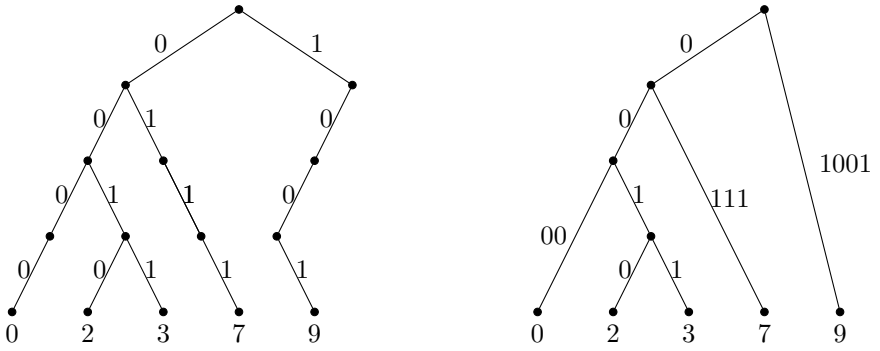


Figure 1 In the left, we have the representation $\mathcal{T}(S)$ (left) of the set $S = \{0, 2, 3, 7, 9\}$. In this example $\mathcal{U} = \{0, \ldots, 15\}$ and thus $D = 4$. We assume that for each $u \in \mathcal{U}$, bits(u) is the 4-bit binary representation of u. In the right, a compressed version $\mathcal{T}^*(S)$ of $\mathcal{T}(S)$ is depicted.

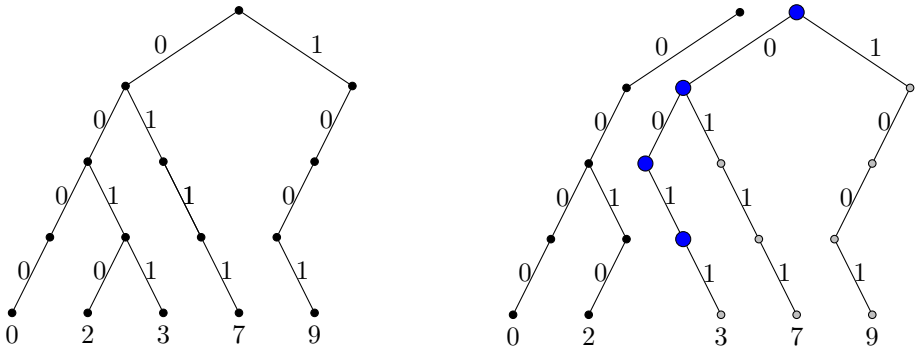


Figure 2 The effect of the call $(A, B) \leftarrow \textsc{Split}(S, 2)$, where $S = \{0, 2, 3, 7, 9\}$. The larger blue nodes denote the only nodes that had to be copied.

To perform $\textsc{Search}(S, x)$, we again descend to the deepest node v such that $\ell(v)$ is a prefix of bits(x). If v is a leaf, then $x \in S$ and we return x. Otherwise, we climb up the tree until we reach a node w such that $c_0(w) \neq \textbf{nil}$ and $\ell(w)1$ is a prefix of bits(x). If no such w exists, we return **nil**. Otherwise, we descend to the rightmost leaf w_R in the subtree $\mathcal{T}(S)_{c_0(w)}$ and return the corresponding element $q(w_R)$. One can easily perform these steps in $O(D) = O(\log n)$ worst-case time.

To split the set S around a pivot x, we need to construct two tries $\mathcal{T}^-$ and $\mathcal{T}^+$ such that $\mathcal{T}^-$ stores the strings $\mathcal{B}^- = \{s \in \mathcal{B}(S) : s \leq \text{bits}(x)\}$ and $\mathcal{T}^+$ stores the strings $\mathcal{B}^+ = \mathcal{B}(S) \setminus \mathcal{B}^-$. Both $\mathcal{T}^-$ and $\mathcal{T}^+$ can be obtained by removing some subtrees of $\mathcal{T}(S)$. If $\mathcal{B}^- = \emptyset$, then $\mathcal{T}^- = \textbf{nil}$ and $\mathcal{T}^+ = \mathcal{T}(S)$. The case when $\mathcal{B}^+ = \emptyset$ is analogous. Now, let v_l be the leaf of $\mathcal{T}(S)$ such that $\ell(v_l) = \max\{\mathcal{B}^-\}$ and let r be the leaf such that $\ell(v_r) = \min\{\mathcal{B}^+\}$. $\mathcal{T}^-$ ($\mathcal{T}^+$) is exactly the part of $\mathcal{T}$ weakly to the left (to the right, resp.) of the path from the root to v_l (v_r resp.). These paths have at most D common nodes in $\mathcal{T}(S)$ – let us call the set of common nodes C. In order to obtain $\mathcal{T}^-$, we remove all the right subtrees of nodes in C, whereas to construct $\mathcal{T}^+$, a copy of each node of C is made with the left subtree removed (Figure 2). Therefore, the operation $\textsc{Split}$ can be implemented in $O(\log n)$ worst-case time.

The implementation of $\textsc{Merge}(A, B)$ is very simple. We use a recursive function `merge` returning a union of two (possibly empty) tries $\mathcal{T}_1$, $\mathcal{T}_2$. Unless some of the tries $\mathcal{T}_1$, $\mathcal{T}_2$ are non-empty, the tries are required to have equal heights in the interval $[0, D]$. `merge` uses parts of $\mathcal{T}_1$ and $\mathcal{T}_2$ to assemble a trie storing exactly the strings that are stored in $\mathcal{T}_1$ or in

$\mathcal{T}_2$. For $i = 1, 2$, denote by $\mathcal{T}_i^L$ and $\mathcal{T}_i^R$ the left and right subtrees of the root node $\texttt{root}(\mathcal{T}_i)$ of $\mathcal{T}_i$. We have

$$\texttt{merge}(\mathcal{T}_1, \textbf{nil}) = \mathcal{T}_1, \tag{1}$$

$$\texttt{merge}(\textbf{nil}, \mathcal{T}_2) = \mathcal{T}_2, \tag{2}$$

$$\texttt{merge}(\mathcal{T}_1, \mathcal{T}_2) = \texttt{trie}(\texttt{root}(\mathcal{T}_1), \texttt{merge}(\mathcal{T}_1^L, \mathcal{T}_2^L), \texttt{merge}(\mathcal{T}_1^R, \mathcal{T}_2^R)). \tag{3}$$

Here, we use assume that the call $\texttt{trie}(v, \mathcal{T}^L, \mathcal{T}^R)$ creates a trie rooted at v with the left and right subtrees $\mathcal{T}^L$ and $\mathcal{T}^R$ respectively, without copying the subtrees. Clearly, after we call $\texttt{merge}(\mathcal{T}(A), \mathcal{T}(B))$, in each recursive step $\texttt{merge}(\mathcal{T}_1, \mathcal{T}_2)$ such that $\mathcal{T}_1 \neq \textbf{nil}$ and $\mathcal{T}_2 \neq \textbf{nil}$, the labels $\ell(\texttt{root}(\mathcal{T}_1))$ in $\mathcal{T}(A)$ and $\ell(\texttt{root}(\mathcal{T}_2))$ in $\mathcal{T}(B)$ are equal. The correctness of this trie merging procedure can be proved in a bottom-up manner with the following simple structural induction argument. The correctness in cases (1) and (2) is trivial. Consider the case (3) and let h be the height of both $\mathcal{T}_1$ and $\mathcal{T}_2$. Then, either one of the tries $\mathcal{T}_1^L$ and $\mathcal{T}_2^L$ is empty, or both $\mathcal{T}_1^L$ and $\mathcal{T}_2^L$ have height $h - 1$. Thus, by the inductive assumption, $\texttt{merge}(\mathcal{T}_1^L, \mathcal{T}_2^L)$ returns the union of $\mathcal{T}_1^L$ and $\mathcal{T}_2^L$. Symmetrically, $\texttt{merge}(\mathcal{T}_1^R, \mathcal{T}_2^R)$ returns the union of $\mathcal{T}_1^R$ and $\mathcal{T}_2^R$. In the final step, the unions of respective subtries are made the new children of $\texttt{root}(\mathcal{T}_1)$.

Note that each time the case (3) arises, the node $\texttt{root}(\mathcal{T}_2)$ is destroyed.

The Amortized Cost of the Operations. Let $\mathcal{S} = \{S_1, S_2, \dots, \}$ be the collection of subsets of $\mathcal{U}$ maintained by our data structure. We define the potential $\phi(\mathcal{S})$ to be the sum of sizes of the tries representing individual sets, i.e., $\phi(\mathcal{S}) = \sum_{S \in \mathcal{S}} |\mathcal{T}(S)|$. It is clear that each operation Make-Set, Insert and Split increases $\phi(\mathcal{S})$ by at most $D = O(\log n)$. The operations Delete and Merge can only decrease the potential. We now show that the worst-case running time of the operation $C \leftarrow \textsc{Merge}(A, B)$ is $O(|\mathcal{T}(A)| + |\mathcal{T}(B)| - |\mathcal{T}(A \cup B)| + 1)$, i.e., it is proportional to the decrease of the potential. Indeed, consider the call $\texttt{merge}(\mathcal{T}_1, \mathcal{T}_2)$ which is not the topmost call $\texttt{merge}(\mathcal{T}(A), \mathcal{T}(B))$. If $\mathcal{T}_1 \neq \textbf{nil}$ and $\mathcal{T}_2 \neq \textbf{nil}$, we can charge the cost of this call (not including the recursive calls) to the destroyed root of $\mathcal{T}_2$. Otherwise, the parent invocation $\texttt{merge}(*, *)$ was of type (3) and thus we can charge this call to the destroyed parent of $\mathcal{T}_2$. Consequently, for each destroyed node, at most 3 calls to `merge` are charged to that node. The total number of destroyed nodes after calling $\texttt{merge}(\mathcal{T}(A), \mathcal{T}(B))$ is $|\mathcal{T}(A)| + |\mathcal{T}(B)| - |\mathcal{T}(A \cup B)|$.

The amortized cost of an operation is defined as its actual cost plus the increase of the potential. Hence, both amortized and worst-case costs of the operations Insert, Delete and Split on $\mathcal{S}$ are $O(\log n)$, whereas the amortized cost of Merge is $O(1)$.

▶ **Theorem 1.** *Let $|\mathcal{U}| = n$. There exists a data structure supporting* Insert, Delete *and* Split *in $O(\log n)$ amortized and worst-case time. The operation* Merge *takes $O(1)$ amortized time. The operation* Search *can be performed in $O(\log n)$ worst-case time.*

The Ordered Union-Split-Find Data Structure. One can easily extend our approach to implement the ordered union-split-find data structure. Assume that the collection $\mathcal{S}$ forms a partition of $\mathcal{U}$, i.e., the elements of $\mathcal{S}$ are disjoint and $\bigcup \mathcal{S} = \mathcal{U}$. For each $u \in \mathcal{U}$ we store a pointer to the leaf of a unique trie $\mathcal{T}(S)$ such that $u \in S$. Additionally, each trie node is accompanied with a parent pointer.

When performing a Split operation, we update the parent pointers of all the newly created (copied) nodes and their children. During Merge, parent pointers are updated each time a node is assigned new children (case (3) of the `merge` procedure). Insert and Delete

can also be easily extended to update the appropriate parent pointers. The maintenance of parent pointers does not influence the asymptotic worst-case and amortized time bounds of the operations.

Answering a FIND(u) query boils down to climbing up the appropriate trie using the parent pointers and returning the root of $\mathcal{T}(S)$, where $u \in S$.

2.1 Obtaining Linear Space

The above construction might incur $\Omega(\log n)$-space overhead per each stored element, e.g., if every set of the collection is a singleton. This can be easily avoided by dissolving all the non-root non-leaf trie nodes having a single child. Now, each edge can be labeled with at most D bits (stored in a single word), whereas the total length of the labels on any root-to-leaf path remains D. As this results in all the nodes having either 0 or 2 children, a compressed trie with t leaves has now at most $2t-1$ nodes in total. Thus, any set S can be stored in $O(|S|)$ machine words. The compressed version of $\mathcal{T}(S)$ obtained this way is denoted by $\mathcal{T}^*(S)$. See Figure 1 for an example.

All the discussed operations can be implemented by introducing a layer of abstraction over $\mathcal{T}^*(S)$, so that we are allowed to operate on $\mathcal{T}(S)$ instead. Each time we access a node $v \in \mathcal{T}^*(S)$, we can "decompress" its outgoing edges by creating at most two additional nodes c_0, c_1 and make them the children of v, so that the labels of the edges (v, c_0) and (v, c_1) have single-bit labels. All nodes of $\mathcal{T}(S)$ "touched" by an operation are processed bottom-up after the operation completes and the non-root nodes of $\mathcal{T}(S)$ with a single child are dissolved back.

2.2 Lower Bound

For completeness, we prove the following lemma, which establishes the optimality of our data structure, as far as the cost of the most expensive operation is concerned.

▶ **Lemma 2.** *Let $|\mathcal{U}| = \Omega(n^2)$. At least one of the mergeable dictionary operations* SPLIT, MERGE *and* SEARCH *requires $\Omega(\log n^2) = \Omega(\log n)$ time.*

Proof. Let $\mathcal{U} = \{(x, y) : x \in \{0, \ldots, n\}, y \in \{1, \ldots, n\}\}$ and suppose the order of $\mathcal{U}$ is such that $(x_1, y_1) \leq (x_2, y_2)$ if and only if $x_1 < x_2$ or $x_1 = x_2 \wedge y_1 \leq y_2$.

Pătraşcu and Demaine [15] considered the following dynamic permutation composition problem. Let $\pi_1, \ldots, \pi_n$ be the permutations of the set $\{1, \ldots, n\}$. Initially $\pi_i = \mathbf{id}$ for all i. We are to support two operations:

- UPDATE(i, π'): set $\pi_i \leftarrow \pi'$,
- VERIFY(i, π'): check if $\pi_i \circ \pi_{i-1} \circ \ldots \circ \pi_1 = \pi'$.

▶ **Lemma 3** ([15]). *Any data structure requires $\Omega(n^2 \log n)$ expected time to support a sequence of n* UPDATE *operations and n* VERIFY *operations.*

We show how to reduce this problem to maintaining a certain partition of $\mathcal{U}$. In our reduction we maintain n sets $S_1, \ldots, S_n$ so that after each UPDATE operation, for each $j = 1, 2, \ldots, n$ we have

$$S_j = \{(0, j), (1, \pi_1(j)), (2, \pi_2(\pi_1(j))), \ldots, (n, \pi_n(\ldots(\pi_1(j))))\}. \tag{4}$$

Clearly, $S_i \subseteq \mathcal{U}$. Note that for each $k \in [0, n]$ and $j \in [1, n]$ we can find $\pi_k(\ldots(\pi_1(j)))$ with a single SEARCH$(S_j, (k, n))$ query. Thus, VERIFY(i, π') can be implemented with n SEARCH operations: for each $j = 1, \ldots, n$ we check whether SEARCH$(S_j, (i, n)) = (i, \pi'(j))$.

In order to implement UPDATE(i, π'), we first execute $(A_j, B_j) \leftarrow$ SPLIT$(S_j, (i-1, n))$ for each $j = 1, 2, \ldots, n$. Note that $A_j \neq \emptyset$ and $B_j \neq \emptyset$. The last element $(i-1, a_j)$ of each A_j can be found with a single SEARCH operation. Similarly, the first element (i, b_j) of each B_j can be found with a single SEARCH. The values a_j are distinct and so are the values b_j. The last step is to create each set S_j by merging A_j with a unique B_k satisfying $b_k = \pi'(a_j)$. It is easy to see that $S_j =$ MERGE(A_j, B_k) satisfies (4) with $\pi_i = \pi'$.

To conclude, n UPDATE and VERIFY can be implemented with $O(n^2)$ SEARCH, SPLIT and MERGE operations on a mergeable dictionary. Being able to execute each of these mergeable dictionary in $o(\log n)$ amortized time would contradict Lemma 3. ◀

3 Handling Dynamic and Infinite Universes

Overview. In the previous section we have only supported subsets of a finite universe $\mathcal{U}$. The critical idea was that we could assign a $O(\log |\mathcal{U}|)$-bit label bits(x) to each $x \in \mathcal{U}$ so that $x \leq y$ implied bits$(x) \leq$ bits(y). This allowed us to store the sets in trees of small depth and predictable structure, which was consistent among the representations of different sets. If the universe can grow or is infinite, e.g. $\mathcal{U} = \mathbb{R}$, it is not clear how to assign such labels beforehand, during the initialization.

In this section we aim at achieving amortized $O(\log N)$ bounds for all mergeable dictionary operations on the collection $\mathcal{S} = \{S_1, S_2, \ldots, \}$, where $N = \sum_{S \in \mathcal{S}} |S|$. At any time, N is no more than the number of INSERT operations performed.

Imagine a perfect binary tree $\bar{T}$ with 2^B leaves such that each edge to the left child is labeled with 0 and each edge to the right child is labeled with 1. The (uncompressed) tries used in the previous section can be seen as subtrees of $\bar{T}$. More formally, $\mathcal{T}(S)$ can be obtained from $\bar{T}$ by removing all the subtrees $\bar{T}_v$ of $\bar{T}$ such that $\bar{T}_v$ does not contain any leaf corresponding to an element of S.

Our strategy is to maintain a similar "global" tree T, so that the representations of individual sets constitute subtrees of T. We incrementally store all the elements of $\bigcup \mathcal{S}$ in the leaves of a *weight-balanced B-tree* T [1]. As opposed to $\bar{T}$, T is not binary. However, it still allows us to keep all the elements as leaves at the same depth of order $O(\log N)$ and add new elements in logarithmic time. One crucial property of a weight-balanced B-tree allows us to still represent the sets $S \in \mathcal{S}$ as compressed subtrees $\mathcal{T}(S)$ of T, even though T undergoes updates. The potential function ϕ we use to analyze the amortized performance of the operations is exactly the same as previously, i.e., $\phi(\mathcal{S}) = \sum_{S \in \mathcal{S}} |\mathcal{T}(S)|$.

The weight-balanced trees have been previously used in the context of the *monotonic list labeling* problem, which typically asks to maintain a totally ordered set Q and $O(\log |Q|)$-bit labels of the elements of Q subject to insertions of a new element y to Q between two existing elements $x < z$, $x, z \in Q$. Several optimal data structures exist for this problem (e.g. [3, 5, 12]): each supports inserting a new element in $O(\log |Q|)$ amortized time and guarantees that such insertion incurs amortized logarithmic number of relabels of existing elements in Q. In particular, Kopelowitz [12] used the weight-balanced B-tree to obtain optimal worst-case bounds for this problem. However, it is not clear how to use a monotonic list labeling data structure as a black-box in our case. Instead of keeping the number of relabels small, we rather need to keep the *potential increase* per insertion small.

We again assume that we work in the word-RAM model, so that the operations on $O(\log N)$-bit integers take $O(1)$ time and the space is measured in the number of words.

The Weight-Balanced B-tree. A weight-balanced B-tree T with a (constant) branching parameter $a \geq 4$ stores its elements in leaves. For an internal node $v \in T$, we define its weight $w(v)$ to be the number of leaves among the descendants of v. The following are the key invariants that define a weight-balanced B-tree:

1. All the leaves of T are at the same depth.
2. Let *height* of a node $v \in T$ be the number of edges on the path from v to any leaf. An internal node v of height h has weight less than $2a^h$.
3. Except for the root, an internal node of height h has weight greater than $\frac{1}{2}a^h$.

▶ **Lemma 4** ([1]). *Assume T is a weight-balanced B-tree with branching parameter a.*

- *All internal nodes of T have at most $4a$ children.*
- *Except for the root, all internal nodes of T have at least $a/4$ children.*
- *If T contains n elements, then the height of T is $O(\log_a n)$.*

For each internal node v and its two children v_1, v_2 such that v_1 is to the left of v_2, the elements in the subtree of v_1 are no larger than any of the elements in the subtree of v_2. Each internal node stores the minimum and maximum elements stored in its subtree. This information allows us to drive the searches down the tree.

To insert an element e into T, we first descend down T to find an appropriate position for the new leaf corresponding to e. The insertion of a new leaf may result in some nodes getting out of balance. Let $v \in T$ be the deepest node such that $w(v) = 2a^h$ at that point, where h is the height of v. As each child of v has weight less than $2a^{h-1}$, one can split the children of v into two groups of consecutive children C_-, C_+ so that the total weight of nodes in any group is in the interval $(a^h - 2a^{h-1}, a^h + 2a^{h-1})$. We have $a^h - 2a^{h-1} = a^h(1 - 2/a) \geq \frac{1}{2}a^h$ and similarly $a^h + 2a^{h-1} \leq \frac{3}{2}a^h$. v is split into two nodes v_- and v_+ so that the elements of C_- become the children of v_- and the elements of C_+ become the children of v_+. We have $w(v_-), w(v_+) \in (\frac{1}{2}a^h, \frac{3}{2}a^h)$, so both v_- and v_+ satisfy the balance constraints. If v is not the root before the split, nodes v_-, v_+ are made the children of the parent of v in place of v. Otherwise, a new root with children v_-, v_+ is created. The process is repeated until all the nodes are balanced and thus the insertion takes $O(\log n)$ time, where n is the number of elements stored in T.

To delete an element from a weight-balanced B-tree, we mark the corresponding leaf as deleted, which takes $O(\log n)$ time. Once more than a half of the stored elements are marked as deleted, the entire tree is rebuilt (the elements marked as deleted are skipped) in $O(n)$ time. This can be charged to the deletions that left the marked leaves. Thus, the amortized time complexity of a deletion is $O(\log n)$ as well.

The main advantage of a weight-balanced B-tree is the fact that for any newly created node v of height h, at least $\Omega(a^h)$ leaves have to be inserted into the subtree of v to cause the split of v. Therefore, when the node v is split, we can afford to spend $O(a^h)$ time for instance for traversing all the leaf descendants of v or updating some secondary data structure that accompanies v. This work can be charged to $\Omega(a^h)$ insertions into the subtree of v that take place between the creation of v and its split. The total amortized time spent on the "additional maintenance" per insertion is thus proportional to the depth of T, i.e., $O(\log n)$.

Labeling the Tree T. Each time an operation $\textsc{Insert}(S, u)$ (for $u \notin S$) is issued, u is stored as a leaf at an appropriate position of the weight-balanced B-tree T. We stress that there is a separate leaf for each (u, S) pair, where $u \in S$, i.e., multiple leaves may correspond to a single $u \in S$. Such a design decision is explained later on (see Remark 1).

We introduce the labels $\ell(v)$ of the vertices of T such that for any $v_1, v_2 \in T$, where v_1 is an ancestor of v_2, the path $v_1 \to v_2$ in T (i.e., the indices of children of subsequent nodes to

be entered when following the path $v_1 \to v_2$ in T) can be computed based only on $\ell(v_1)$ and $\ell(v_2)$. As the trees $\mathcal{T}(S)$ are stored in a compressed way, the labels will help navigate $\mathcal{T}(S)$ while performing the operations on S. We now define the labels $\ell(v)$ formally.

Let H be the height of T. The label $\ell(v)$ of a node v of height h consists of H blocks of $\lceil \log_2(4a+1) \rceil = O(1)$ bits. Clearly, $\ell(v)$ can be stored in a constant number of machine words. We number the blocks with integers $0, \ldots, H-1$ starting at the block containing the least significant bits. Let $v_H \to \ldots \to v_h = v$ be the root-to-v path in T. We define $z(v_i)$ to be the (0-based) position of v_i among the children of v_{i+1} in the left-to-right order. At any time (even immediately before the split) v_{i+1} has at most $4a+1$ children. Thus, $\lceil \log_2(4a+1) \rceil$ bits suffice to store $z(v_i)$. For $i \in [h, H-1]$, we define the the bits of the i-th block of $\ell(v)$ to contain exactly the value $z(v_i)$. The blocks $h-1, \ldots, 0$ of $\ell(v)$ are filled with zeros. Note that the label $\ell(v)$ can be computed in $O(1)$ time based on the label of its parent in T using standard bitwise operations.

Storing the Individual Sets. Let $S \in \mathcal{S}$. Denote by $L(S)$ the set of leaves of T that correspond to the elements of S. Recall that each $\mathcal{T}(S)$ is in fact the tree T with subtrees containing no leaves of $L(S)$ removed. Again, in the compressed version $\mathcal{T}^*(S)$ we only keep the nodes v of $\mathcal{T}(S)$ such that either v is the root of $\mathcal{T}(S)$, $v \in L(S)$, or for at least two children c_1, c_2 of v, the subtree rooted at c_i $(i = 1, 2)$ contains at least one leaf of $L(S)$. An example tree T along with the compressed and uncompressed representation of a set $S \in \mathcal{S}$ is presented in Figure 3.

Each node v of $\mathcal{T}^*(S)$ is a copy of the corresponding node of T and v stores a pointer to the original node of T. Every node of T also maintains a list of its copies used in the representations $\mathcal{T}^*(S)$ of the sets of $S \in \mathcal{S}$. The pointers between $\mathcal{T}^*(S)$ and T along with the labels $\ell(*)$ allow to temporarily decompress the relevant parts of $\mathcal{T}^*(S)$ when performing the operations Insert, Delete, Merge and Split, analogously as in Section 2.1.

Differences in the Implementation of Operations. In comparison to the data structure of Section 2, the implementations of operations Insert, Delete, Split do not generally change. We basically replace values bits$(*)$ with labels $\ell(v)$. Each operation Insert(S, x) first inserts a leaf into T and thus the new element is given a label before we modify $\mathcal{T}^*(S)$.

When the operation Search(S, x) is performed, we first find in $O(\log N)$ worst-case time a leaf in T that corresponds to a maximum value $y \in \bigcup \mathcal{S}$ such that $y \leq x$. Note that Search(S, y) computes the same value as Search(S, x), but now y corresponds to some leaf of T and it has a label, so we can proceed analogously as in Section 2.

Handling the Splits of the Nodes of T. Suppose a new leaf is added to T at an appropriate position among the children of some height-1 node v. The labels of all the children of the node v might have to be recomputed.

The insertion may also cause the splits of some internal nodes, as described previously. Let v be an internal non-root node of height h that is split into two nodes v_-, v_+. Denote by p the parent of v. After the split, both the values $z(*)$ of the children of p and the values $z(*)$ of the children of v_-, v_+ may change. This implies that for each v of the $O(a^h)$ nodes of the subtree rooted at p, the contents of at most two blocks (namely, the blocks h and $h-1$) of $\ell(v)$ may change. As discussed above, we can afford going through all these nodes without sacrificing our amortized $O(\log n)$ insertion bound.

The split also requires to repair some of the representations $\mathcal{T}^*(S)$. First assume that S is such that there is no copy of v included in $\mathcal{T}^*(S)$. Then, either there is no copy of v in

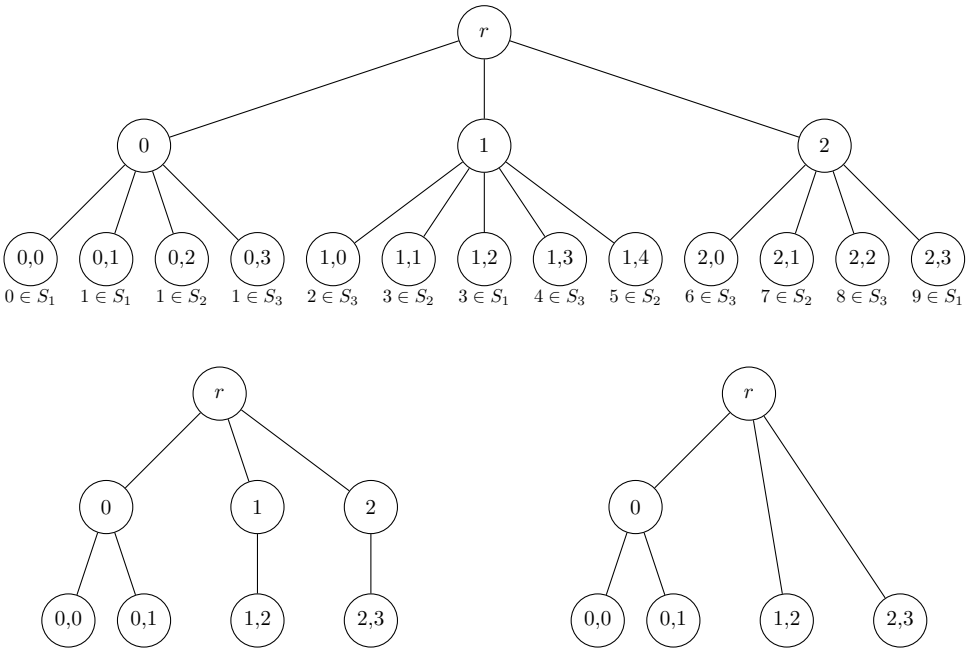


Figure 3 Let $\mathcal{S} = \{S_1, S_2, S_3\}$, where $S_1 = \{0, 1, 3, 9\}$, $S_2 = \{1, 3, 5, 7\}$ and $S_3 = \{1, 2, 4, 6, 8\}$. A weight-balanced B-tree T (with branching factor $a = 4$) that could arise when constructing $\mathcal{S}$ is depicted at the top. The values in the nodes are their labels $\ell(*)$. The trie $\mathcal{T}(S_1)$ can be seen in the bottom left. The compressed version $\mathcal{T}^*(S_1)$ is illustrated in the bottom-right.

$\mathcal{T}(S)$ and thus $\mathcal{T}(S)$ contains no leaves of the subtree of T rooted at v, and we are done, or a copy v_S is a node of $\mathcal{T}(S)$. Then, v_S has a single child c in $\mathcal{T}(S)$ and therefore, after the split v_S should be replaced with a copy of either v_- or v_+. However, as v_- or v_+ would have been dissolved in $\mathcal{T}^*(S)$, we actually do not need to update $\mathcal{T}^*(S)$ at all. Moreover, in this case the size $|\mathcal{T}(S)|$ does not change and neither does the potential ϕ.

Let us now suppose that a copy v_S of v is a node of $\mathcal{T}^*(S)$ and denote by q the parent of v_S in $\mathcal{T}^*(S)$. If all the children of v_S in $\mathcal{T}(S)$ are contained in the subtree of v_- of T after the split, it suffices to replace v in $\mathcal{T}^*(S)$ with a copy of v_- and update the pointers between $\mathcal{T}^*(S)$ and T. The case when all the children of v in $\mathcal{T}(S)$ are contained in the subtree of v_+ of T is similar. Both this cases required $O(1)$ time to process, but ϕ does not change. The last case is when some two children c_-, c_+ of v in $\mathcal{T}(S)$ are contained in the subtrees of v_- and v_+, respectively. Then, copies of both v_- and v_+ have to be introduced in $\mathcal{T}(S)$ in place of v. Thus, the potential ϕ increases by 1 in this case. As far as the compact representation $\mathcal{T}^*(S)$ is concerned, a copy of p has to be included in $\mathcal{T}^*(S)$, if it is not already there. The copies of v_- and v_+ are created in $\mathcal{T}^*(S)$ only if they would not be dissolved afterwards. We skip the description of the case when v is the root, as it is analogous.

We conclude that it takes $O(1)$ time to repair $\mathcal{T}^*(S)$ in any case and the potential ϕ increases by at most 1 per repair. The number of repairs incurred by the split of v is not more than the number of leaves of the subtree rooted in v, i.e., $O(a^h)$, as for each representation of S that actually needs to be repaired, $\mathcal{T}^*(S)$ has to contain (a copy of) some leaf of the subtree T_v. Finally, note that a single leaf of T_v has a copy in at most one representation $\mathcal{T}^*(S)$.

Thus, the repairs made during the maintenance of the tree T increase the potential by amortized $O(\log N)$ per INSERT operation.

▶ Remark 1. Imagine the tree T was allowed to contain only a single leaf for each element $u \in \bigcup \mathcal{S}$. Suppose that for each $S_i \in \mathcal{S} = \{S_1, \ldots, S_m\}$, $S_i = \{u\}$, for some $u \in \mathcal{U}$. Each split of an ancestor of the leaf corresponding to u in T would cause a repair of m set representations. Let $x_0 \in \mathcal{U}$ be such that $x_0 > u$. Now suppose the adversary sequentially performs Insert(S_i, x_i), where $u < x_i < x_{i-1}$ for $i = 1, \ldots, m$. $\Omega(m)$ of such operations would lead to a split of some ancestor of the leaf u, and the total running time of these sequence could be as much as $\Omega(m^2)$.

The weight-balanced B-tree does only guarantees that the total size of split subtrees after m insertions is $O(m \log m)$. However, as the above example shows, the total number of times when some particular leaf is contained in a subtree undergoing a split might be $\Omega(m^2)$. That is why we decided to store duplicate leaves per single value $u \in \mathcal{U}$, if u is a frequent element in the stored sets.

The Amortized Analysis of the Operations. Each of the operations Insert, Delete, Split runs in amortized $O(\log N)$ time, as discussed above. Also it is clear that the (amortized) potential increase per each of this operations is $O(\log N)$.

The operation Merge is implemented almost identically as in Section 2. We only need to make sure that the modified recursive procedure `merge` is always fed two copies of the same node $v \in T$ as arguments. As each node has $O(1)$ children, we can charge a constant amount of work to the nodes of $\mathcal{T}(*)$ destroyed during the merging process. Consequently, Merge runs in time proportional to the decrease of the potential ϕ.

▶ **Theorem 5.** *There exists a data structure supporting all the mergeable dictionary operations on a collection $\mathcal{S}$ of subsets of $\mathcal{U}$ in amortized $O(\log N)$ time, where $N = \sum_{S \in \mathcal{S}} |S|$.*

4 Conclusions and Open Problems

In this paper we developed a simpler solution for the mergeable dictionary problem. We also addressed the issue of supporting dynamic/infinite universes raised in [11].

We can see two interesting further questions about mergeable dictionaries. First, in the finite universe case, the amortized cost of all the operations was logarithmic in the size of the universe. On the other hand, for the infinite case, we only managed to obtain amortized $O\left(\log \sum_{S \in \mathcal{S}} |S|\right) = O\left(\log |\mathcal{S}| + \log |\bigcup \mathcal{S}|\right)$ bounds. We can think of the size of the "used universe" to be $|\bigcup \mathcal{S}|$. Thus, in the infinite universe case, our time bounds are also logarithmic in the number of stored sets, which might be of order much larger than $|\bigcup \mathcal{S}|$. It would be interesting to know if one could remove this dependence.

Second, our solution for infinite universes involves maintaining a "common infrastructure" T in order to limit the potential growth. Is there a way to implement a mergeable dictionary in a dynamic/infinite universe regime without any common infrastructure, so that the representation of a set does not depend on the shapes of other stored sets? In particular, is the splay tree [17] a mergeable dictionary with such a property?

Acknowledgments. We thank the anonymous reviewers for their helpful comments.

References

1 Lars Arge and Jeffrey Scott Vitter. Optimal external memory interval management. *SIAM J. Comput.*, 32(6):1488–1508, 2003. doi:10.1137/S009753970240481X.

2 Amitabha Bagchi, Adam L. Buchsbaum, and Michael T. Goodrich. Biased skip lists. *Algorithmica*, 42(1):31–48, 2005. doi:10.1007/s00453-004-1138-6.

3 Michael A. Bender, Richard Cole, Erik D. Demaine, Martin Farach-Colton, and Jack Zito. Two simplified algorithms for maintaining order in a list. In *Algorithms – ESA 2002, 10th Annual European Symposium, Rome, Italy, September 17-21, 2002, Proceedings*, pages 152–164, 2002. doi:10.1007/3-540-45749-6_17.

4 Mark R. Brown and Robert Endre Tarjan. A fast merging algorithm. *J. ACM*, 26(2):211–226, 1979. doi:10.1145/322123.322127.

5 Paul F. Dietz and Daniel Dominic Sleator. Two algorithms for maintaining order in a list. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 365–372, 1987. doi:10.1145/28395.28434.

6 Martin Farach and Mikkel Thorup. String matching in lempel-ziv compressed strings. *Algorithmica*, 20(4):388–404, 1998. doi:10.1007/PL00009202.

7 Paweł Gawrychowski. Pattern matching in lempel-ziv compressed strings: Fast, simple, and deterministic. In *Algorithms – ESA 2011 – 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, pages 421–432, 2011. doi:10.1007/978-3-642-23719-5_36.

8 Loukas Georgiadis, Haim Kaplan, Nira Shafrir, Robert Endre Tarjan, and Renato Fonseca F. Werneck. Data structures for mergeable trees. *ACM Transactions on Algorithms*, 7(2):14, 2011. doi:10.1145/1921659.1921660.

9 Leonidas J. Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16-18 October 1978*, pages 8–21, 1978. doi:10.1109/SFCS.1978.3.

10 Scott Huddleston and Kurt Mehlhorn. A new data structure for representing sorted lists. *Acta Inf.*, 17:157–184, 1982. doi:10.1007/BF00288968.

11 John Iacono and Özgür Özkan. Mergeable dictionaries. In *Proceedings of the 37th International Colloquium Conference on Automata, Languages and Programming*, ICALP'10, pages 164–175, Berlin, Heidelberg, 2010. Springer-Verlag. doi:10.1007/978-3-642-14165-2_15.

12 Tsvi Kopelowitz. On-line indexing for general alphabets via predecessor queries on subsets of an ordered list. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 283–292, 2012. doi:10.1109/FOCS.2012.79.

13 Katherine Jane Lai. Complexity of union-split-find problems. Master's thesis, Massachusetts Institute of Technology, 2008. URL: http://erikdemaine.org/theses/klai.pdf.

14 William Pugh. Skip lists: A probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, 1990. doi:10.1145/78973.78977.

15 Mihai Pătraşcu and Erik D. Demaine. Lower bounds for dynamic connectivity. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 546–553, 2004. doi:10.1145/1007352.1007435.

16 Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, June 1983. doi:10.1016/0022-0000(83)90006-5.

17 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary trees. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 235–245, 1983. doi:10.1145/800061.808752.

18 Dan E. Willard. Log-logarithmic worst-case range queries are possible in space theta(n). *Inf. Process. Lett.*, 17(2):81–84, 1983. doi:10.1016/0020-0190(83)90075-3.

Cuckoo Filter: Simplification and Analysis

David Eppstein*

Computer Science Department, University of California, Irvine, USA

Abstract

The cuckoo filter data structure of Fan, Andersen, Kaminsky, and Mitzenmacher (CoNEXT 2014) performs the same approximate set operations as a Bloom filter in less memory, with better locality of reference, and adds the ability to delete elements as well as to insert them. However, until now it has lacked theoretical guarantees on its performance. We describe a simplified version of the cuckoo filter using fewer hash function calls per query. With this simplification, we provide the first theoretical performance guarantees on cuckoo filters, showing that they succeed with high probability whenever their fingerprint length is large enough.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases approximate set, Bloom filter, cuckoo filter, cuckoo hashing

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.8

1 Introduction

Bloom filters [2] are a very widely used data structure for approximately representing sets using low space. At the cost of $O(1)$ bits per element, they can represent any set, with constant-time membership testing, no false negatives, and an arbitrarily low false positive rate controlled by the bits per element. Recently, Fan, Andersen, Kaminsky, and Mitzenmacher [8] proposed an alternative data structure for the same purpose, the *cuckoo filter*. They show experimentally that cuckoo filters are better than Bloom filters in several important ways: they use (up to lower-order terms) 30% less space for the same false positive rate, matching the information-theoretic lower bound. They have better locality of reference, accessing only two contiguous blocks of memory per query rather than the larger numbers of a typical Bloom filter. And, unlike a Bloom filter, they can handle element deletions as well as insertions and queries without any increase in storage. These good features have already led to the use of cuckoo filters in several applications [11, 10]. (For a different and more theoretical replacement for Bloom filters with similar advantages, see Pagh, Pagh, and Rao [13].)

A cuckoo filter uses a hash table to store a small *fingerprint* for each element, and answers queries by testing whether the fingerprint of the queried element is present. Each element has two hash table cells where its fingerprints might be stored, determined by a combination of a hash of the element and a second hash of the fingerprint. As in cuckoo hashing [14], fingerprints already stored in the table may be moved to their second location to make room for a newly inserted fingerprint. The performance of a cuckoo filter is controlled by the number n of elements in the set it represents, together with three design parameters: the table size N (number of cells), block size b (fingerprints that can be stored in a single cell), and fingerprint size f (bits per fingerprint). A good choice of these parameters allows the fingerprints for all elements in the given set to be stored in the table, giving a data structure whose false positive rate ϵ (the probability that an element not in the set is falsely reported

* Supported in part by NSF grant 1228639. The author would like to thank Michael Mitzenmacher for introducing him to cuckoo filters, and for helpful discussions on this work.

to be in the set) can easily be bounded by $\epsilon \leq 2b/(2^f - 1)$. For bad choices of parameters, or unlucky choices of hash function, the data structure may fail, being unable to store all its elements' fingerprints. Therefore, it is important to analyze the likelihood of a failure, and to understand which combinations of parameters have a guaranteed low failure probability.

We may define the *load factor* $\frac{n}{bN}$ to be the ratio of the number of stored fingerprints to the number that could be stored. As this number will be close to one, it is convenient to represent its difference from one as a parameter δ, with $1 - \delta = \frac{n}{bN}$. The experiments of Fan et al. [8], show that the load factor can be made arbitrarily close to one while keeping the failure rate low, by a large enough choice of the block size b. With this choice and a small false positive rate ϵ, the storage cost is $\big(1 + o(1)\big) \log(1/\epsilon)$ bits per element, matching the information-theoretic lower bound on any approximate set data structure. However, this combination of low storage costs and low failure rate has been observed only in experiments. The only theoretical analysis so far, also by Fan et al. [8], is a lower bound showing that f must be $\Omega((\log n)/b)$ to have a low failure probability. They did not provide any matching upper bound showing that some combination of parameters can ensure a low failure probability.

In this paper we provide the first theoretical guarantees on the performance of cuckoo filters. To do so, we describe a simplified version of cuckoo filters, in which we determine the two cells for an element by using its fingerprint directly rather than by using a hash of its fingerprint. This simplification had previously been considered by Fan et al. [8], but they discarded it without publishing any experimental test results for it. Like Fan et al., we do not expect this simplification to be a practical improvement, but it makes the data structure more amenable to analysis. Under this simplification, we show that the cuckoo filter has (up to lower-order terms, for the same choice of block size, with polynomially small failure probability) nearly the same maximum load rate as the blocked cuckoo hash table of Dietzfelbinger and Weidling [5], as long as a constraint that $f = \Omega((\log n)/b)$ is also satisfied. In particular, this analysis allows for the load factor to be arbitrarily close to one, controlled by the block size b. Thus, $f = \Omega((\log n)/b)$ is both a necessary and a sufficient condition for the high-probability success of cuckoo filters. We also describe how to add a stash to the simplified cuckoo filter, allowing the cuckoo filter to take advantage of the improved reliability of cuckoo filtering with a stash [12] without any change in the false positive rate.

Our analysis uses the unrealistic assumption of a uniformly random hash function. However, it uses only two properties of this function: that blocked cuckoo hashing using it succeeds with high probability, and that with high probability it balances the load of a set of balls distributed into a significantly smaller number of bins. Therefore, it is likely that, if the analysis of cuckoo hashing with realistic hash functions [15, 1] is extended to blocked cuckoo hashing, the same analysis can also be extended to cuckoo filters. It also seems likely that the original version of cuckoo filters behaves at least as well as the simplified version, but we leave the problem of proving this as open for future research. Our algorithm for cuckoo filtering with a stash depends in an essential way on the structure of the simplified cuckoo filter, so extending it to the original cuckoo filter also remains open.

2 Preliminaries

We begin by briefly reviewing the Bloom filter, whose operations the cuckoo filter emulates, and the cuckoo hash table on which the organization of a cuckoo filter is based. We then describe the cuckoo filter itself, in the original version given by Fan et al.

As a notational convenience, we use log without a base to refer to the binary logarithm $\log_2$. We will also use the natural logarithm, denoted by ln.

2.1 Bloom filter

A Bloom filter represents a set of n elements by an array of N cells ($N > n$), each containing a single bit of information, together with a hash function mapping the potential elements of the set to k-tuples of cells (for a chosen constant parameter value k). A cell contains a nonzero bit if at least one of the elements is mapped to it by the hash function, and a zero bit otherwise. To insert an element into the set, the hash function is used to find its cells, and all of these cells are set to nonzero. To query whether an element belongs to a set, all of its cells are examined, and the result of the query is positive if and only if they are all nonzero. There is no deletion operation.

A false positive occurs if an element that is not part of the given set coincidentally has all of its cells nonzero. For a given choice of N, and a given set size n, the optimal false positive rate is achieved by setting $k \approx \frac{N \ln 2}{n}$, so that with high probability approximately half of the cells in the table are nonzero. With these choices, the false positive rate is approximately 2^{-k}. Inverting this calculation, the Bloom filter data structure achieves a false positive rate of ϵ using approximately $\frac{1}{\ln 2} \log(1/\epsilon) \approx 1.44 \log(1/\epsilon)$ bits of storage per element [8].

Many extensions of Bloom filters have been studied. For instance, a *counting Bloom filter* [9] stores a counter instead of a bit per cell; it can handle deletions, and can also be used (with a smaller number of cells) as a *count-min sketch* to estimate the frequency of items in a data stream [3]. An *invertible Bloom filter* adds even more information per cell in order to be able to recover the identities of the set elements stored in it, when there are few enough elements; it also allows deletions, and can be used to find stragglers in a data stream [6], or as a sketch to communicate the symmetric difference of two similar sets using an amount of communication proportional to the difference [7]. However, these methods blow up the size of the data structure by a nonconstant factor, and so are less suitable for the original task of the Bloom filter, of representing approximate sets using very little memory.

2.2 Cuckoo hashing

Cuckoo filters are based on *cuckoo hashing*, one of many hashing based techniques for maintaining a collection of key–value pairs and looking up the value associated with a query key [14]. Cuckoo hashing is a form of open addressing, a family of hashing techniques in which each cell of a hash table stores a single key–value pair. In cuckoo hashing, each key has only two locations in which it may be stored, which are determined by a hash function. Thus, answering a query is simple: look in those two cells and test whether either cell contains the query key.

Inserting a key into a cuckoo hash table is more complicated. If one of the two cells for the key is empty, it can be inserted there. But otherwise, one of the other keys occupying one of these two cells must be kicked out, to make way for the new key. The kicked-out key must then be re-inserted into its second location, possibly kicking out another key there, and so on. This process will either eventually terminate with all keys stored in one of their two cells, or it may fail and force the data structure to be rebuilt. A failure may occur, for instance, when some set of q keys is mapped to fewer than q cells, so there is not enough room to store all of these keys in their cells.

In analyzing this structure, we make the standard assumption that the two cells for each key are chosen uniformly at random, independently from each other and from all the other keys. However, there has also been much research on practical hash function algorithms that do not obey this assumption but nevertheless can be made to work with cuckoo hashing [4, 15].

Two of the shortcomings of this basic version of cuckoo hashing are that the failure probability is only moderately small (proportional to $1/n^2$ per insertion, for a hash table with a constant load factor, rather than being adjustable to arbitrary inverse polynomials) and that the load factor it can tolerate while achieving this failure probability is also bounded away from 1 (in fact, bounded below 1/2). Because of these issues, researchers have investigated modifications of cuckoo hashing that can tolerate higher loads with improved failure probabilities. For the results that we report on in this paper, we need to understand two such modifications, blocked cuckoo hashing [5] and cuckoo hashing with a stash [12].

2.3 Blocked cuckoo hashing

Blocked cuckoo hashing was initially developed by Dietzfelbinger and Weidling [5]; we follow here its description by Kirsch et al. [12]. In blocked cuckoo hashing, each cell of the hash table stores a block of up to b different key–value pairs, for a parameter b chosen as part of the implementation or initialization of the data structure. A query may examine all of the pairs in the two cells that it searches; however, the locality of reference of the query is still as good as in the original version of cuckoo hashing.

When a key is inserted, and one of its two cells is not full (has fewer than b keys already stored in it) it may be placed directly in that cell. However, when both of its cells are full, one of the keys already placed in one of those cells must be kicked out, and moved to its other location. As with standard cuckoo hashing, this move may cause another key to move, possibly creating a chain of dislocations. This sequence of moves can alternatively be viewed as an augmenting path in a graph whose vertices are table cells and whose edges are the pairs of cells that each key maps to. Kirsch et al. write that, for the analysis of the failure probability of this algorithm, it is unimportant how the augmenting path is found, but that an analysis of Dietzfelbinger and Weidling [5] shows constant expected time (in the event of no failures) for a breadth-first algorithm for finding these augmenting paths.

In order to achieve a load factor of $1 - \delta$, blocked cuckoo hashing may be used with any block size $b \geq 1 + \frac{\ln(1/\delta)}{1-\ln 2}$. Thus, for constant δ, the block size is also a constant. With this block size and load factor, the failure probability per insertion is $O(1/n^b)$ [12].

2.4 Cuckoo hashing with a stash

A *stash* is a small collection of key–value pairs that have not been included in a cuckoo hash table. In cuckoo hashing with a stash, the stash is used to store key–value pairs whose insertion would otherwise cause the hash table to fail. To perform a query in a cuckoo hash table with a stash (in either the original or blocked form of cuckoo hashing) one first checks the two cells that can contain the query key. Then, if the key is not found in either of those two locations, and both locations are full, the stash is also searched. This causes an additional sequence of memory accesses for unsuccessful searches (or for searches of keys already in the stash), but does not slow down most searches, and in many cases greatly improves the reliability of this structure [12].

To apply this technique to cuckoo filtering, we need an analysis of blocked filtering with a stash. Kirsch et al. [12] claimed that, for blocked cuckoo hashing with block size b and a stash that can hold σ key–value pairs, the failure probability per insertion is $O(1/n^{(\sigma+1)(b-1)+1})$. Unfortunately, this multiplicative improvement in the exponent of failure probability, compared to the version without the stash, is incorrect. As Martin Dietzfelbinger and Michael Rink observed, a failure mode in which $\sigma + 2b + 1$ keys all map to the same pair of cells already causes the failure probability to be much larger than this

bound.[1] Therefore, any improvement to the reliability of blocked cuckoo hashing, obtained by adding a stash, should be considered conjectural.

2.5 Cuckoo filter

A *cuckoo filter* [8] modifies the blocked cuckoo hash table by storing a small *fingerprint* for each element in a set, instead of storing a key–value pair. As in the blocked cuckoo hash table, each cell of the table can store a small number b of (fingerprints of) elements. For each element there are two cells at which its fingerprint could be stored. To test whether an element belongs to the filter, we examine these two cells and report yes when a matching fingerprint is found in one of them.

In order to pack its fingerprints into these cells, the cuckoo filter (like a cuckoo hash table) may sometimes move them to the other location for their key. However, when it does this, it will not know the key from which the fingerprint was generated. Therefore, it must determine the other location for a fingerprint from the fingerprint alone. This limitation means that the two locations for the fingerprint of a key can no longer be chosen independently of each other, complicating the analysis of this data structure.

To keep things simple, we will assume that the number of cells N in the cuckoo filter is a power of two.[2] Thus, if x and y are indices into the table (numbers in the range from 0 to $N-1$ inclusive), we can combine them by a bitwise exclusive or operation giving a number $x \oplus y$ that is also an index into the table. Cuckoo filtering depends on three hash functions, ϕ, h_1, and h_2, assumed (for purposes of analysis) to be independent random functions:

- Function ϕ maps each potential set element to its fingerprint, an f-bit binary number. It is convenient to restrict ϕ to have nonzero values so that a zero fingerprint can be used to mark an unused cell of the cuckoo filter; this has no significant effect on the asymptotic behavior of the structure.
- Function h_1 maps each potential set element to a number from 0 to $N-1$. This gives the location of one of the two hash table cells into which the fingerprint for that element can be placed.
- Function h_2 maps fingerprints to numbers in the range from 1 to $N-1$ (inclusive). This is not the index of a hash table cell, but rather the difference (or more precisely the bitwise exclusive or) of any pair of locations at which that fingerprint should be stored.

Thus, using these functions, the fingerprint $\phi(x)$ for any element x of the given set will be stored either in cell $h_1(x)$ or cell $h_1(x) \oplus h_2(\phi(x))$ of the hash table. If some subset of two or more elements all have the same fingerprint and are all mapped by h_1 to the same cell, then each of the fingerprints for these elements will be stored separately in one of the two cells for these elements, even when that would cause two copies of the same fingerprint to be stored in the same cell. That is, in order to make deletions possible, we do not allow different elements to share a copy of a stored fingerprint.

To query whether a value x is a member of the set represented by a cuckoo filter, we examine the $2b$ fingerprints stored at cells $h_1(x)$ and $h_1(x) \oplus h_2(\phi(x))$, and check whether any of them equal $\phi(x)$; if so we report that x is indeed a member. Thus, this check always answers correctly when x belongs to the set, and may give a false positive with probability at most $2b/2^{f-1}$ when x does not belong to the set. This query may be performed using a

[1] Michael Mitzenmacher, personal communication, February 7, 2016.

[2] It is tempting to try modular addition in place of exclusive ors to extend this method to other choices of N, but this fails because exclusive or is an involution and modular addition isn't. However, the simplified cuckoo filter that we describe later can have a number of subtables that is not a power of two.

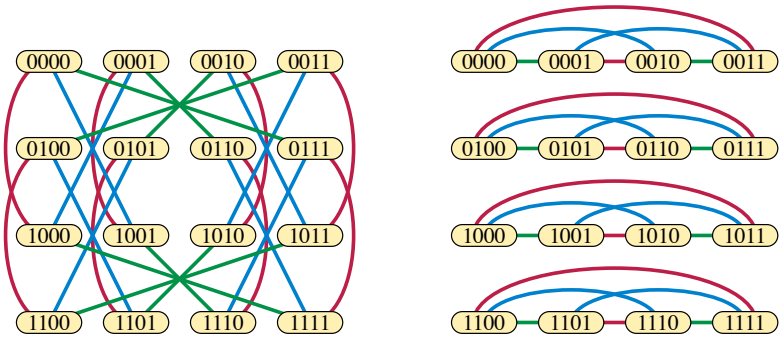


Figure 1 The graph of cuckoo filter cells, and pairs of cells that can be the two locations of any fingerprint, for a cuckoo filter with 16 cells and two-bit fingerprints. The edges for each fingerprint are colored green for fingerprint 01, blue for fingerprint 10, and red for fingerprint 11. Left: the original cuckoo filter, with $h_2(01) = 0111$, $h_2(10) = 1001$, and $h_2(11) = 1000$. Right: the simplified cuckoo filter without h_2.

number of binary word operations (including multiplication of binary numbers) proportional to the number of words needed to store the two cells, rather than being proportional to the larger number of fingerprints stored in these cells; see section 6 for details.

To insert a value x into the set, we compute its fingerprint and place this fingerprint into one of the two cells associated with x, possibly relocating other fingerprints along an augmenting path in the graph of cells and pairs of cells associated with the elements of the set. The second location for each fingerprint can be calculated from its first location and the fingerprint itself, without needing to know the element from which the fingerprint is generated; otherwise, the insertion operation proceeds exactly as in the blocked cuckoo hash table. Following Kirsch et al. [12], we do not specify precisely how the augmenting path is to be found, except to note that one possible choice for finding it is to use a breadth first search.

To remove a value x from the set, we find a matching fingerprint for x, and remove it from its cell. If there are multiple matching fingerprints, we remove only one copy; it does not matter whether that copy is the one created when x was inserted.

3 The simplification and its graph

In the simplified version of cuckoo filters that we analyze here, we omit the second hash function, h_2. Thus, given any set element x, the two locations in which we may store the fingerprint $\phi(x)$ are the cells with index $h_1(x)$ and $h_1(x) \oplus \phi(x)$. Other than this change, the operation of the simplified cuckoo filter remains the same as in the original version.

The effect of this simplification may be visualized using a graph whose vertices are the cells of the hash table, and whose edges connect the possible pairs of locations of a single fingerprint (Figure 1). In the original cuckoo filter, the endpoints of each edge differ (in their exclusive or) by one of the values of the hash function h_2. The resulting graph is regular, as each vertex has one incident edge for each possible hash function value, and has the symmetries of a hypercube. When $2^f > \log n$, the graph has probability $\Omega(1)$ of being connected, in which case it is a Cayley graph of the group generated by exclusive ors of the values of h_2. For smaller values of f, with probability $\Omega(1)$ the hashes of all the fingerprints will be independent vectors over the two-element field, and if so the graph will be a disjoint union of $N/2^{2^f-1}$ hypercube subgraphs with 2^{2^f-1} vertices per hypercube. However, when the second hash function h_2 is omitted, the corresponding graph of cell pairs is much less

well-connected: it is a disjoint union of $N/2^f$ subgraphs, each of which is a clique. The graph of the simplified cuckoo filter is not connected unless $f > \log n$, a much stronger requirement than is needed for the likely connectivity of the graph of the original cuckoo filter.

This simplification had previously been considered by Fan et al. [8]. However, they discarded it without publishing any test results for it. They write:

> "If the alternate location were calculated ... without hashing the fingerprint, the items kicked out from nearby buckets would land close to each other in the table ... Hashing the fingerprints ensures that these items can be relocated to buckets in an entirely different part of the hash table, hence reducing hash collisions and improving the table utilization."

Although we agree with this criticism, the simplification has the advantage that it makes the method more amenable to analysis. In particular, we can bound the failure probability of the simplified cuckoo filter by treating each connected component of the graph described above as an independent cuckoo hash table. Within each component, all pairs of table cells are equally likely to be chosen by any element that maps to that component, so the previous analysis of cuckoo hash tables may be applied directly. Although we expect the failure probability for the original cuckoo filter to be at least as good as for the simplified version, we do not know how to extend our analysis to it.

We remark that, instead of randomly choosing a function h_2, and constructing a graph of cells and fingerprint-edges based on it, it would be possible to develop a generalized version of cuckoo filtering whose graph is any desired $(2^f - 1)$-regular graph on the cells of the cuckoo filter. If this graph is also $(2^f - 1)$-edge-colorable, this may be done by associating each fingerprint with an edge color, and making the two cells that can store the fingerprint of an element x be $h_1(x)$ and the neighbor of $h_1(x)$ along the edge with color $\phi(x)$. In the more general case, in which the graph is not $(2^f - 1)$-edge-colorable, this may be done by choosing a one-to-one correspondence between fingerprints and outgoing locations at each cell, and either storing $\phi(x)$ in $h_1(x)$ or storing $\varphi(x)$ in the neighbor of $h_1(x)$ associated with fingerprint $\phi(x)$, where $\varphi(x)$ is the fingerprint associated with the edge back to $h_1(x)$. In this way, for instance, it would be possible to make the graph connected even when $2^f \leq \log n$, something that is not possible for the original cuckoo filter.

4 Analysis

We have seen by the graphical analysis in the previous section that, in the simplified version of cuckoo filtering that we study here, the table of cells can be partitioned into connected components of the fingerprint-edge graph, each of which is a clique. A single connected component consists of 2^f cells. Each two cells within a connected component have locations that differ only within their least significant f bits. We call each of these connected components a *subtable*. Essentially, each subtable is itself a cuckoo filter, on a subset of the input data. We analyze two different failure modes of the global filter: either it can fail to uniformly distribute the set elements to its subtables, or the cuckoo filtering within a subtable can fail.

4.1 Even distribution into subtables

We assume a cuckoo filter representing n elements, with b fingerprints per cell and $N = (n/b)/(1 - \delta - \delta^2)$ cells where δ and b are related as $b \geq 1 + \frac{\ln(1/\delta)}{1-\ln 2}$, the same relation used by Kirsch et al. (Prop. 4.1) for blocked cuckoo hashing [12]. We also assume that the fingerprints

of the cuckoo filter have f bits each, so that the cuckoo filter may be partitioned into $N/2^f$ subtables of 2^f cells per subtable.

The subtable into which an element x is mapped is given by the most significant $-f+\log N$ bits of $h_1(x)$. Assuming that h_1 is a random hash function, the probability that each element falls into a particular subtable S is $2^f(1/N)$, and the events that elements fall into a subtable (for different elements) are mutually independent. Thus, the number of elements that are mapped into subtable S is the sum of n i.i.d. Bernoulli random variables with probability $p = 2^f(1/N)$ of being 1 and probability $1-p$ of being 0. The expected number μ of elements that are mapped into subtable S is $2^f(n/N) = 2^f b(1-\delta-\delta^2)$.

We will say that a subtable is *overfull* when the number of elements mapped to it exceeds $\mu(1-\delta)/(1-\delta-\delta^2) = \mu(1+\delta^2+o(\delta^2))$, giving it a load factor greater than $1-\delta$. By a standard form of the multiplicative Chernoff bound, for δ sufficiently smaller than 1, the probability that S is overfull is at most $\exp\bigl(-\delta^4\mu/3\bigr)$. To achieve probability $1/n^s$ of avoiding any overfull subtable, it suffices (by the union bound) to achieve probability $1/n^{s+1}$ that one table S is overfull. Plugging $\mu \geq 2^f b$ into the probability that S is overfull and solving for f gives that no table is overfull, with high probability $\geq 1-1/n^s$, whenever

$$f \geq \log\left(\frac{3(s+1)\ln n}{\delta^4 b}\right) = \log\left(\frac{\log n}{b}\right) + O(1).$$

where the simplification on the right hand side is based on the assumption that δ and s are constants. This $\Omega\bigl(\log((\log n)/b)\bigr)$ constraint on f will be insignificant in comparison with the $\Omega\bigl((\log n)/b\bigr)$ constraint coming from the failure probability within each subtable.

We summarize the results of this subsection as a lemma:

▶ **Lemma 4.1.** *Let n, N, b, f, and δ be as above. Suppose also that $f = \log\bigl((\log n)/b\bigr)+\Omega(1)$. Then the probability that a simplified cuckoo filter with these parameters has any overfull subtables is polynomially small, with an exponent that can be made arbitrarily large by using a larger constant factor in the Ω-notation of the bound for f.*

4.2 Failure probability within each subtable

Let S be a subtable that is not overfull; that is, at most $2^f b/(1+\delta)$ elements of the given n-element set are mapped into it, where again f is the fingerprint length, b is the number of fingerprints per cell, and δ is a constant that is yet to be determined. If an element x is mapped to subtable S, then the location within S to which it is mapped (the low-order bits of $h_1(x)$) are independent of the information causing it to be mapped to S (the high-order bits of $h_1(x)$), so these locations are uniformly distributed within S. Additionally, the fingerprint $\phi(x)$ is uniformly distributed among all of the valid fingerprints, and the set of locations given by the bitwise exclusive or of these fingerprints with $h_1(x)$ is exactly the set of all remaining locations within S. Therefore, we may analyze each subtable independently, as a data structure containing at most $2^f b/(1+\delta)$ fingerprints, each of which is mapped to a uniformly random pair of distinct locations within the subtable. That is, limiting our attention to a single subtable has eliminated the dependence between the pairs of locations used by each element of the cuckoo filter.

Such a data structure behaves exactly the same as a blocked cuckoo hash table with the same elements. Thus, we can apply the previously known analysis of a blocked cuckoo hash table directly. However we must keep in mind the fact that, because we are studying events that hold with high (inverse-polynomial) probability, the reduced size of a subtable (relative to an n-element cuckoo hash table) translates into weaker bounds on the failure probability.

Choose b and δ so that $b \geq 1 + \frac{\ln(1/\delta)}{1-\ln 2}$, the same constraints on b and δ made in the analysis of blocked cuckoo hash tables. With these choices, δ may be made arbitrarily small by choosing b large enough, and in particular the factor $(1+\delta)^2$ appearing in the analysis of how evenly the subtables are distributed may also be made arbitrarily close to one. Then by the previous analysis of blocked cuckoo hash tables, the failure probability per insertion, for a subtable containing $\Theta(b2^f)$ elements, is $O\big((b2^f)^{-b}\big)$, inversely proportional to the bth power of the number of elements.

We desire the data structure to succeed with high probability; that is, for an arbitrarily chosen constant s, we should be able to achieve failure probability $O(1/n^s)$. This will be true when $(b2^f)^b = \Omega(n^s)$ or, taking logarithms of both sides, when $b(f + \log b) > s \log n + O(1)$. The $b \log b$ term can be assumed to cancel the $O(1)$, giving $f > (s \log n)/b$ as a sufficient condition for high-probability success. This matches the $f = \Omega\big((s \log n)/b\big)$ necessary condition of Fan et al. [8] (based on a calculation of the probability that more than $2b$ elements hash to the same location/fingerprint pair), which also applies to this version of cuckoo filters.

We summarize the results of this subsection as a lemma:

▶ **Lemma 4.2.** *Let b be any sufficiently large constant, and let s also be constant. Let n, N, δ, and f be as defined above, and suppose that $f > (s \log n)/b$. Then the probability of failure for an insertion of an element into a simplified cuckoo filter with these parameters and without any overfull subtables is at most $1/n^s$.*

4.3 Overall failure probability

Combining our results on the two failure modes of cuckoo filters, we have the following result.

▶ **Theorem 4.3.** *Let n and b be given, and let the maximum load factor for high-probability-of-success blocked cuckoo hashing with block size b be $1-\delta$, where $b \geq 1 + \frac{\ln(1/\delta)}{1-\ln 2}$. Then creating a cuckoo filter for n elements with block size b and fingerprint size f succeeds with high probability for load factor $1-\delta-\delta^2$ when $f = \Omega\big((\log n)/b\big)$. More specifically, to achieve probability $O(1/n^s)$ of failure it is sufficient for b to be a sufficiently large constant and for f to obey the inequality $f > (s+1)(\log n)/b$.*

The $1 + o(1)$ factor by which the load factor $1-\delta$ of blocked cuckoo hashing differs from the load factor $1-\delta-\delta^2$ of this result is caused by the fact that, when n elements are filtered into subtables, some subtables will likely be larger than their expected size. The term δ^2 could be replaced here by any fixed power of δ. One way of avoiding this change in load factor altogether would be to redo the analysis of blocked cuckoo hashing based on the assumption that the input is chosen by including elements independently at random at a fixed rate, rather than that the set of input elements has a fixed size. With this modified input model, the same model with the same rate would automatically apply to each subtable, without need of Chernoff bounds. However, this refinement would make little difference to our overall results.

5 Cuckoo filter with a stash

The extra structure of the subtables in the simplified cuckoo filter that we analyze here also makes it easier to add a stash to this structure, to amplify its success probability. Specifically, we add a separate stash for each subtable of the filter. Each stash will store a collection of (location,fingerprint) pairs, for fingerprints that were not able to be stored within its subtable.

Because the stash is specific to the subtable, the location part of the stash need only store the low-order f bits of the location, specifying one of the two cells within the subtable that its fingerprint could have been stored in.

Recall that, in the simplified cuckoo filter, Each fingerprint $\phi(x)$ can be stored in two locations, $h_1(x)$ and $h_1(x) \oplus \phi(x)$. However, after storing the fingerprint in the filter, we no longer know which of its two locations is h_1. Therefore, in the pair that we store in the stash, we use the location with the smaller index, given by (the low-order f bits of) $\min(h_1(x), h_1(x) \oplus \phi(x))$.

To query whether an element x belongs to the set represented by a cuckoo filter with a stash, we test whether either of the two locations for x in the filter contains the fingerprint for x. If not, we determine the minimum of those two locations, and search for the resulting (location,fingerprint) pair in the stash for the subtable of x. To insert an element x into the filter, we attempt to insert it into the cuckoo filter as before, and if this fails we add it to the stash. And to delete an element x, we perform a query to determine which of the two locations for x contains its fingerprint, or whether the fingerprint is located in the stash, and remove one copy of this fingerprint from one of its locations.

The same analysis as for the cuckoo filter without a stash goes through in the same way as before, replacing the failure probability of a blocked cuckoo hash table (used to prove Theorem 4.2) with any improved failure probability (currently only conjectural) that can be proved for a blocked cuckoo hash table with a stash.

The failure probability can also be boosted, with theoretical guarantees rather than conjectures on the improvement, by increasing the block size b rather than adding a stash. Adding a stash to each subtable would increase the memory requirements of the data structure by only a lower-order term, but increasing the block size could actually decrease the memory requirements (assuming the fingerprint size f is kept constant) by allowing a load factor closer to 1 to be used. So, given that stashes are not an improvement over increased block sizes in the guarantees they provide, in the reliability that can be obtained with them, or in the storage space they use, why would one ever use a stash? The answer lies in another feature of their analysis, their effect on the false positive rate of the data structure.

Recall that, in a cuckoo filter with block size b and fingerprint size f, the false positive rate is at most $2b/(2^f - 1)$: each query examines at most $2b$ fingerprints, each of which has a $1/(2^f - 1)$ chance of colliding with a given query of an element that does not belong to the set. The probability is at most $2b/(2^f - 1)$ rather than exactly $2b/(2^f - 1)$ for two reasons: a block of cells may not be full, or it may have more than one copy of the same fingerprint. However, for typical choices of parameters, neither of these reasons gives a large effect on the false positive rate. Based on this calculation, increasing the block size from b to b' would also increase the false positive rate by the same b'/b factor. Naively, a stash of size σ would again increase the false positive rate to $(2b + \sigma)/(2^f - 1)$, since now there are potentially σ additional fingerprints that could collide with any given query. However, as we now prove, the stash does not actually change the false positive rate at all. The analysis below also gives a tighter analysis on the false positive rate for cuckoo hashes even without a stash, taking into account the possibility of under-full blocks.

▶ **Theorem 5.1.** *The false positive rate of a cuckoo filter with b blocks, n elements, N hash table cells, fingerprint size f, and load factor $1 - \delta = n/Nb$, regardless of whether it uses a stash or not, is at most $\frac{2n}{N(2^f-1)} = \frac{2b(1-\delta)}{2^f-1}$.*

Proof. There are n elements in the set represented by the filter, each of which independently selects a hash table location and fingerprint. A query collides with an element if and only if it has the same fingerprint and is mapped to one of the same two locations. Based on this

choice, any single element has a $2/N(2^f - 1)$ probability of colliding with the given query; the factor of 2 in the numerator comes from the fact that two different choices of hash table location give rise to the same pair of cells in which the fingerprint can be stored. The result follows by the union bound. ◀

Based on this result, and assuming that adding a stash to a blocked cuckoo hash table can be proved to improve its reliability, the same improvement to reliability can be obtained in a cuckoo filter with a stash, without sacrificing the failure rate or compensating for it by increasing the fingerprint size.

6 Bit-parallel querying

A query in a cuckoo filter involves testing whether a given fingerprint $\phi(x)$ is stored in one of two cells of the filter. However, a fingerprint may have many fewer bits than a word in the machine architecture on which the filter is implemented; therefore, it may be necessary to test whether the same fingerprint appears at each of several positions in a machine word. We describe here how to perform this task in constant time (independent of the number of positions to be tested) using only bitwise binary operations and arithmetic operations on binary numbers. This analysis shows that cuckoo filter queries can take constant time even for non-constant block sizes, as long as the fingerprint length multiplied by the block size is $O(\log n)$, machine words store $\Omega(\log n)$-bit words, and multiplication takes constant time.

Suppose that each fingerprint has f bits, and that b fingerprints are packed into a word w of bf bits. We perform the following steps:

- Let $F = \sum_{i=0}^{b-1} 2^{if}$, a word in which b fingerprints are packed, all equal to the number 1. This step can be done when the filter is created, so its calculation does not figure into the time complexity of the query algorithm.
- Compute $q = w \oplus (\phi(x) \oplus (2^f - 1))F$. The subexpression $\phi(x) \oplus (2^f - 1)$ creates a fingerprint complementary to $\phi(x)$, and the subexpression $(\phi(x) \oplus (2^f - 1))F$ packs b copies of this complementary fingerprint into a single word. Thus, the whole expression, which combines w with these complementary fingerprints by a bitwise exclusive or, gives a word packed with b fingerprints which are all-ones when they match $\phi(x)$, and which have at least one zero in their binary representations when they do not match.
- Compute $r = ((q + F) \oplus q \oplus F)$ & $2^f F$. Here the & operator represents bitwise Boolean and. The expression $q + F$ produces a carry above each matching fingerprint, and no carry above the fingerprints that do not match. The expression $q \oplus F$ has, in the positions of each of these carry bits, a binary value representing what would be in that position if no carry occurred. The exclusive or of these two subexpressions gives the carry bits that differ from their non-carry values. Masking with $2^f F$ keeps only the bits in the carry positions of this computation, setting the other bits to zero.
- If r is zero, there is no match. Otherwise, there is a match at the position given by shifting the least significant nonzero bit of r right by f positions. This least significant nonzero can be calculated as r & $\sim(r - 1)$, where $\sim$ is the bitwise complement operator.

Once a value with a nonzero bit at the position of the first match has been obtained, it is straightforward to perform other operations such as removing the fingerprint at that position; we omit the details. We summarize the results here as a theorem:

▶ **Theorem 6.1.** *Suppose that a cuckoo filter with block size b packs each block of b fingerprints into B machine words, in a model of computation in which bitwise Boolean operations and binary number arithmetic (including multiplication) take constant time per operation. Then each query in the filter may be performed in time $O(B)$.*

References

1. Martin Aumüller, Martin Dietzfelbinger, and Philipp Woelfel. Explicit and efficient hash families suffice for cuckoo hashing with a stash. *Algorithmica*, 70(3):428–456, 2014. doi:10.1007/s00453-013-9840-x.
2. Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970. doi:10.1145/362686.362692.
3. Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005. doi:10.1016/j.jalgor.2003.12.001.
4. Martin Dietzfelbinger and Ulf Schellbach. On risks of using cuckoo hashing with simple universal hash classes. In *Proc. 20th ACM–SIAM Symp. Discrete Algorithms (SODA'09)*, pages 795–804, 2009.
5. Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theoret. Comput. Sci.*, 380(1-2):47–68, 2007. doi:10.1016/j.tcs.2007.02.054.
6. David Eppstein and Michael T. Goodrich. Straggler identification in round-trip data streams via Newton's identities and invertible Bloom filters. *IEEE Trans. Knowledge and Data Engineering*, 23(2):297–306, 2011. arXiv:0704.3313, doi:10.1109/TKDE.2010.132.
7. David Eppstein, Michael T. Goodrich, Frank Uyeda, and George Varghese. What's the difference? Efficient set reconciliation without prior context. In *Proc. ACM SIGCOMM 2011*, pages 218–229, 2011. doi:10.1145/2018436.2018462.
8. Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. Cuckoo filter: Practically better than Bloom. In *Proc. 10th ACM Int. Conf. Emerging Networking Experiments and Technologies (CoNEXT'14)*, pages 75–88, 2014. doi:10.1145/2674005.2674994.
9. Li Fan, Pei Cao, Jussara Almeida, and Andrei Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Networking*, 8(3):281–293, 2000. doi:10.1109/90.851975.
10. M. Grissa, A.A. Yavuz, and B. Hamdaoui. Cuckoo filter-based location-privacy preservation in database-driven cognitive radio networks. In *Proc. World Symp. Computer Networks and Information Security (WSCNIS 2015)*, pages 1–7. IEEE, 2015. doi:10.1109/WSCNIS.2015.7368280.
11. Vikas Gupta and Frank Breitinger. How cuckoo filter can improve existing approximate matching techniques. In Joshua I. James and Frank Breitinger, editors, *Proc. 7th Int. Conf. Digital Forensics and Cyber Crime (ICDF2C 2015)*, volume 157 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 39–52. Springer, 2015. doi:10.1007/978-3-319-25512-5_4.
12. Adam Kirsch, Michael D. Mitzenmacher, and Udi Wieder. More robust hashing: cuckoo hashing with a stash. *SIAM J. Comput.*, 39(4):1543–1561, 2010. doi:10.1137/080728743.
13. Anna Pagh, Rasmus Pagh, and S. Srinivasa Rao. An optimal Bloom filter replacement. In *Proc. 16th ACM–SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 823–829. ACM, New York, 2005.
14. Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004. doi:10.1016/j.jalgor.2003.12.002.
15. Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation hashing. *J. ACM*, 59(3):A14, 2012. doi:10.1145/2220357.2220361.

Randomized Algorithms for Finding a Majority Element

Paweł Gawrychowski[1], Jukka Suomela[2], and Przemysław Uznański[3]

1 **Institute of Informatics, University of Warsaw, Warsaw, Poland**
2 **Helsinki Institute for Information Technology HIIT, Department of Computer Science, Aalto University, Aalto, Finland**
3 **Department of Computer Science, ETH Zürich, Zurich, Switzerland**

Abstract

Given n colored balls, we want to detect if more than $\lfloor n/2 \rfloor$ of them have the same color, and if so find one ball with such majority color. We are only allowed to choose two balls and compare their colors, and the goal is to minimize the total number of such operations. A well-known exercise is to show how to find such a ball with only $2n$ comparisons while using only a logarithmic number of bits for bookkeeping. The resulting algorithm is called the Boyer–Moore majority vote algorithm. It is known that any deterministic method needs $\lceil 3n/2 \rceil - 2$ comparisons in the worst case, and this is tight. However, it is not clear what is the required number of comparisons if we allow randomization. We construct a randomized algorithm which always correctly finds a ball of the majority color (or detects that there is none) using, with high probability, only $7n/6 + o(n)$ comparisons. We also prove that the expected number of comparisons used by any such randomized method is at least $1.019n$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases majority, randomized algorithms, lower bounds

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.9

1 Introduction

A classic exercise in undergraduate algorithms courses is to construct a linear-time constant-space algorithm for finding the majority in a sequence of n numbers $a_1, a_2, \ldots, a_n$, that is, a number x such that more than $\lfloor n/2 \rfloor$ numbers a_i are equal to x, or detect that there is no such x. The solution is to sweep the sequence from left to right while maintaining a candidate and a counter. Whenever the next number is the same as the candidate, we increase the counter; otherwise we decrease the counter and, if it drops down to zero, set the candidate to be the next number. It is not difficult to see that if the majority exists, then it is equal to the candidate after the whole sweep, therefore we only need to count how many times the candidate occurs in the sequence. This simple yet beautiful solution was first discovered by Boyer and Moore in 1980; see [4] for the history of the problem.

The only operation on the input numbers used by the Boyer–Moore algorithm is testing two numbers for equality, and furthermore at most $2n$ such checks are ever being made. This suggests that the natural way to think about the algorithm is that the input consists of n colored balls and the only possible operation is comparing the colors of any two balls. Now the obvious question is how many such comparisons are necessary and sufficient in the worst possible case. Fischer and Salzberg [11] proved that the answer is $\lceil 3n/2 \rceil - 2$. Their algorithm is a clever modification of the original Boyer–Moore algorithm that reuses the

results of some previously made comparisons during the verification phase. They also show that no better solution exists by an adversary-based argument. However, this argument assumes that the strategy is deterministic, so the next step is to allow randomization.

Surprisingly, not much seems to be known about randomized algorithms for computing the majority in the general case. For the special case of only two colors, Christofides [5] gives a randomized algorithm that uses $\frac{2}{3}(1-\frac{\epsilon}{3})n$ comparisons in expectation and returns the correct answer with probability $1-\epsilon$, and he also proves that this is essentially tight; this improves on a previous lower bound of $\Omega(n)$ by De Marco and Pelc [15]. Note that in the two-color case any deterministic algorithm needs precisely $n-B(n)$ comparisons, where $B(n)$ is the number of 1s in the binary expansion of n, and this is tight [17, 2, 19]. For a random input, with each ball declared to be red or blue uniformly at random, roughly $2n/3$ comparisons are sufficient and necessary in expectation to find the majority color [3]. However, to the best of our knowledge upper and lower bounds on the expected number of comparisons without any restrictions on the number of colors have not been studied before.

Related work include *oblivious* algorithms studied by Chung et al. [6], that is, algorithms in which subsequent comparisons do not depend on the previous answers, and finding majority with larger queries [14, 9, 18]. Another generalization is finding a ball of plurality color, that is, the color that occurs more often than any other [1, 12, 13].

We consider minimizing the number of comparisons mostly as an academic exercise, and believe that a problem with such a simple formulation deserves to be thoroughly studied. However, it is possible that a single comparison is so expensive that their number is the bottleneck. Such a line of thought motivated a large body of work studying the related questions of the smallest number of comparisons required to find the median element; see [7, 8, 16] and the references therein. Of course, the simplest Boyer–Moore algorithm has the advantage of using only two sequential scans over the input and a logarithmic number of bits, while our algorithm needs more space and random access to the input.

Given that the original motivation of Boyer and Moore was fault-tolerant computing, we find it natural to consider Las Vegas algorithms, that is, the number of comparisons depends on the random choices of the algorithm but the answer is always correct. This way the result will be correct even if the source of random bits is compromised; an adversary that is able to control the random number generator can only influence the running time.

Model. We identify balls with numbers $1, 2, \ldots, n$. We write $\mathsf{cmp}(i, j)$ for the result of comparing the colors of balls i and j (true for equality, false for inequality). We consider randomized algorithm that, after performing a number of such comparisons, either finds a ball of the majority color or detects that there is no such color. A majority color is a color with the property that more than $\lfloor n/2 \rfloor$ balls are of such color. The algorithm should always be correct, irrespectively of the random choices made during the execution. However, the colors of the balls are assumed to be fixed in advance, and therefore the number of comparisons is a random variable. We are interested in minimizing its expectation.

Contributions. We construct a randomized algorithm, which always correctly determines a ball of the majority color or detects that there is none, using $7n/6 + o(n)$ comparisons with high probability (in particular, in expectation). We also show that the expected number of comparisons used by any such algorithm must be at least $1.019n$. Therefore, randomization allows us to circumvent the lower bound of Fischer and Salzberg and construct a substantially better algorithm. Most probably our lower bound can be slightly strengthened, but achieving $7n/6$, which we conjecture to be the answer, seems to require a different approach.

2 Preliminaries

We denote the set of balls (items) by $M = \{1, 2, \ldots, n\}$. We write $\mathsf{color}(x)$ for the color of ball x, and $\mathsf{cmp}(x, y)$ returns true if the colors of balls x and y are identical.

An event occurs *with very high probability* (w.v.h.p.) if it happens with probability at least $1 - \exp(-\Omega(\log^2 n))$. Observe that the intersection of polynomially many very high probability events also happens with very high probability.

▶ **Lemma 1** (Symmetric Chernoff Bound). *The number of successes for n independent coin flips is w.v.h.p. at most $\frac{n}{2} + \mathcal{O}(\sqrt{n} \log n)$.*

▶ **Lemma 2** (Sampling). *Let $X \subseteq M$ such that $|X| = m$. Let m' denote the number of hits on elements from X if we sample uniformly at random $k \leq n$ elements from M without replacement. Then w.v.h.p. $|m'/k - m/n| = \mathcal{O}(k^{-1/2} \log n)$.*

Now we consider a process of pairing the items without replacement (choosing a random perfect matching on M; if n is odd then one item remains unpaired). For any $X \subseteq M$, let u_{XX} be a random variable counting the pairs with both elements belonging to X when choosing uniformly at random $\frac{n}{2}$ pairs of elements from M without replacement. Of course $\mathbb{E}[u_{XX}] = \frac{|X|(|X|-1)}{2(n-1)}$.

▶ **Lemma 3** (Concentration for Pairs). *For any $X \subseteq M$ w.v.h.p.*

$$\left| u_{XX} - \frac{|X|^2}{2n} \right| = \mathcal{O}\big(\sqrt{|X|} \log n\big).$$

▶ **Lemma 4** (Pairs in Partition). *Let $\mathcal{F} = \{X_1, \ldots, X_m\}$ be a partition of M. Then w.v.h.p.*

$$\left| \sum_{X \in \mathcal{F}} u_{XX} - \sum_{X \in \mathcal{F}} \frac{|X|^2}{2n} \right| = \mathcal{O}(n^{2/3} \log n).$$

▶ **Lemma 5.** *Let $X \subseteq M$ such that $|X| = m$. Let $k(m', m, n)$ denote the number of draws without replacement until we hit m' elements from X. Then w.v.h.p.*

$$k(m', m, n) \leq \frac{n}{m} m' + \mathcal{O}\Big(\frac{n}{\sqrt{m}} \cdot \log n\Big).$$

3 Algorithm

In this section we describe a randomized algorithm for finding majority. Recall that the algorithm is required to always either correctly determine a ball of the majority color or decide that there is no such color, and the majority color is a color of more than $\lfloor n/2 \rfloor$ balls. For simplicity we will assume for the time being that n is even, as the algorithm can be adjusted for odd n in a straightforward manner without any change to the asymptotic cost. Hence to prove that there is a majority color, it is sufficient to find $n/2 + 1$ balls with the same color. In such case our algorithm will actually calculate the multiplicity of the majority color. To prove that there is no majority color, it is sufficient to partition the input into $n/2$ pairs of balls with different colors.

The algorithm consists of three parts. Intuitively, by choosing a small random sample we can approximate the color frequencies and choose the right strategy: (i) There is one color with a large frequency. We use algorithm HEAVY. In essence, we have only one candidate for the majority, and we compute the frequency of the candidate in a naive manner. If the

frequency is too small, we need to form sufficiently many pairs of balls with different colors among the balls that are not of the candidate color. This can be done by virtually pairing the non-candidate color elements, and testing these pairs until we find enough of them that have distinct colors. Additionally, we show that one sweep through the pairs is enough. (ii) There are two colors with frequencies close to 0.5. Now we use algorithm BALANCED. In essence, we can now reduce the size of the input by a pairing process, and then find the majority recursively. If the recursion finds the majority, the necessary verification step is speeded up by reusing the results of the comparisons used to form the pairs. (iii) All frequencies are small. We use LIGHT which, as BALANCED, applies pairing and recursion. However, if the recursive call reports the majority, we construct enough pairs with different colors: whenever we find a pair of elements with both colors different than the majority color found by the recursive call, we pair them with elements of the majority color. Here we speeded up the process by reusing the results of the comparisons used to form the pairs as well.

We start with presenting the main procedure of the algorithm; see Algorithm 1. The parameters are chosen by setting $\alpha = \frac{1}{3}$, $\varepsilon = n^{-1/10}$ and $\beta = 0.45$. In fact we could chose any $\beta \in (\beta_1, \beta_2)$, where $\beta_1 = 1 - \frac{1}{\sqrt{3}} \approx 0.4226$ and $\beta_2 \approx 0.47580$ is a root to $p^3 - 19p^2 - 8p + 8 = 0$.

Algorithm 1: MAJORITY(M)

```
1  if |M| = 1 then return M[1] is the majority with multiplicity 1 in M
2  sample M' ⊆ M such that |M'| = n^α
3  let v_1, v_2, ..., v_k be the representatives of the colors in M'
4  let q_i|M'| be the frequency of color(v_i) in M', where q_1 ≥ q_2 ≥ ... ≥ q_k
5  if q_1, q_2 ∈ [1/2 − 4ε, 1/2 + 4ε] then
6      return BALANCED(M)
7  else if q_1 ≥ β and q_1^2 ≥ q_2^2 + ... + q_k^2 + 2ε then
8      return HEAVY(M, v_1)
9  else
10     return LIGHT(M)
```

Before we proceed to describe the subprocedures, we elaborate on the sampling performed in line 4. Intuitively, we would like to compute the frequencies of all colors in M. This would be too expensive, so we select a small sample M' and claim that the frequencies of all colors in M' are not too far from the frequencies of all colors in M. Formally, let $p_1, p_2, p_3, \ldots, p_\ell$ be the frequencies of all colors in M, that is there are $p_i \cdot n$ balls of color i in M and let q_i be the frequency of color i in the sample M'. By Lemma 2, w.v.h.p. $|p_i - q_i| = \mathcal{O}(n^{-\alpha/2} \log n) = o(\varepsilon)$. We argue that $\sum_i q_i^2$ is a good estimation of $\sum_i p_i^2$.

▶ **Lemma 6.** *Let p_i be the frequency of color i in M and q_i be its frequency in M', where $M' \subseteq M$ a random sample without replacement of size n^α. Then w.v.h.p.*

$$\left| \sum_i p_i^2 - \sum_i q_i^2 \right| = \mathcal{O}(n^{-\alpha/3} \log n) = o(\varepsilon).$$

Proof. Let $m = n^\alpha$. We analyze the following two sampling methods.

1. Partition the elements of M into $\frac{n}{2}$ disjoint pairs uniformly at random. Select $\frac{m}{2}$ of these pairs uniformly at random. Denote by A_1 and A_2 the pairs with both elements of the same colors in the first and the second pairing, respectively. By Lemma 4, w.v.h.p.

$\left||A_1| - \frac{n}{2}\sum_i p_i^2\right| = \mathcal{O}(n^{2/3}\log n)$. Observe that by Lemma 2 w.v.h.p. $\left||A_2| - \frac{m}{n}|A_1|\right| = \mathcal{O}(m^{1/2}\log n)$. Thus, by the triangle inequality, w.v.h.p.

$$\left|\frac{|A_2|}{m/2} - \sum_i p_i^2\right| = \mathcal{O}(n^{-1/3}\log n) + \mathcal{O}(m^{-1/2}\log n).$$

2. Partition the elements of M' into $\frac{m}{2}$ disjoint pairs uniformly at random, and denote by B all pairs with both elements of the same color. By Lemma 4, w.v.h.p. $\left||B| - \frac{m}{2}\sum_i q_i^2\right| = \mathcal{O}(m^{2/3}\log n)$, or equivalently $\left|\frac{|B|}{m/2} - \sum_i q_i^2\right| = \mathcal{O}(m^{-1/3}\log n)$.

Now, because A_2 and B have identical distributions, by the triangle inequality we have

$$\left|\sum_i p_i^2 - \sum_i q_i^2\right| = \mathcal{O}(n^{-1/3}\log n) + \mathcal{O}(m^{-1/2}\log n) + \mathcal{O}(m^{-1/3}\log n) = \mathcal{O}(m^{-1/3}\log n).\blacktriangleleft$$

Now we present the subprocedures; see Algorithms 2–4.

Algorithm 2: HEAVY(M, v)

```
cnt ← 0, X ← []
for i = 1 to |M| do
    if cmp(v, M[i]) then
        cnt ← cnt + 1
    else
        append M[i] to X
if cnt > |M|/2 then return color(v) is the majority with multiplicity k in M
k ← |M|/2 − cnt
randomly shuffle X
for i = 1 to |X|/2 do
    if ¬cmp(X[2i − 1], X[2i]) then k ← k − 1
    if k = 0 then return no majority in M
return Boyer–Moore(M)                          ▷ fallback, 2n comparisons
```

Algorithm 3: LIGHT(M)

```
randomly shuffle M
X ← [], Y ← []
for i = 1 to |M|/2 do
    if cmp(M[2i − 1], M[2i]) then
        append M[2i] to X
    else
        append M[2i − 1] and M[2i] to Y
run MAJORITY(X)
if there is no majority in X then return no majority in M
let color(v) be the majority with multiplicity k in X
cnt ← 2k − |X|
for i = 1 to |Y| do
    if ¬cmp(v, Y[2i − 1]) then
        if ¬cmp(v, Y[2i]) then
            cnt ← cnt − 1
    if cnt = 0 then return no majority in M
return color(v) is the majority with multiplicity (|M|/2 + cnt) in M
```

Algorithm 4: BALANCED(M)

```
randomly shuffle M
X ← [], Y ← []
for i = 1 to |M|/2 do
    if cmp(M[2i − 1], M[2i]) then
        append M[2i] to X
    else
        append M[2i − 1] and M[2i] to Y
run MAJORITY(X)
if there is no majority in X then return no majority in M
let color(v) be the majority with multiplicity k in X
cnt ← 2k
for i = 1 to |Y|/2 do
    if cmp(v, Y[2i − 1]) then
        cnt ← cnt + 1
    else if cmp(v, Y[2i]) then
        cnt ← cnt + 1
if cnt ≤ |M|/2 then
    return no majority in M
else
    return color(v) is the majority with multiplicity k in X
```

▶ **Lemma 7.** *Algorithm 1 always returns the correct answer.*

Proof. We analyze separately every subprocedure.

BALANCED(M). If the majority exists then removing two elements with different colors preserves it. Hence if the recursive call returns that there is no majority in X then indeed there is no majority in M, and otherwise color(v) is the only possible candidate for the majority in M. The remaining part of the subprocedure simply verifies it.

HEAVY(M, v). The subprocedure first checks if color(v) is the majority. Hence it is enough to analyze what happens if color(v) is not the majority. Then X contains all elements with other colors. We partition the elements in X into pairs and check which of these pairs consists of elements with different colors. If the number of elements in all the remaining pairs is smaller than the number of elements of color color(v), then clearly we can partition all elements in M into disjoint pairs of elements with different colors, hence indeed there is no majority. Otherwise, we revert to the simple $2n$ algorithm, which is always correct.

LIGHT(M). Again, if the majority exists then removing two elements with different color preserves it. Hence we can assume that color(v) is the only possible candidate for the majority. Then, Y consists of pairs of two elements with different colors. From the recursive call we also know what is the frequency of color(v) in $M \setminus Y$. We iterate through the elements of Y and check if their color is color(v). However, if the color of the first element in a pair is color(v), then the second element has a different color. So the subprocedure either correctly determines the frequency of the majority color(v), or find sufficiently many elements with different colors to conclude that color(v) is not the majority. ◀

▶ **Theorem 8.** *Algorithm 1 w.v.h.p. uses at most $\frac{7}{6}n + o(n)$ comparisons on an input of size n. The expected number of comparisons is also at most $\frac{7}{6}n + o(n)$.*

Proof. Let $T(n)$ be a random variable counting the comparisons on the given input of size n. We will inductively prove that $T(n) \leq \frac{7}{6}n + C \cdot n^{9/10}$ for a fixed constant C that is sufficiently large. In the analysis we will repeatedly invoke Lemmas 2, 3, 4, 5, 6 and Chernoff bound to bound different quantities. We will assume that each such the application succeeds. Since there will be a polynomial number of applications, each on a polynomial number of elements, this happens w.v.h.p. with respect to the size of the input. We also assume that n is large enough. Algorithm 1 uses at most $\mathcal{O}(n^{2\alpha}) = \mathcal{O}(n^{2/3})$ comparisons in the sampling stage. We bound the number of subsequent comparisons used by each subprocedure as follows.

BALANCED(M). We have that $p_1, p_2 = \frac{1}{2} \pm \mathcal{O}(\varepsilon)$. Thus also $\sum_i p_i^2 = \frac{1}{2} \pm \mathcal{O}(\varepsilon)$. By Lemma 4, $|X| = (\frac{n}{2} \sum_i p_i^2) \pm \mathcal{O}(n^{2/3} \log n)$, thus $|X| = (\frac{1}{4} \pm \mathcal{O}(\varepsilon))n$. Also $|Y| = n - 2|X| = (\frac{1}{2} \pm \mathcal{O}(\varepsilon))n$.

List Y consists of pairs of elements with different colors. Because at most $\mathcal{O}(\varepsilon n)$ of all elements are not of color 1 or 2, there are at most $\mathcal{O}(\varepsilon n)$ pairs not of the form $\{1, 2\}$. Since the relative order of elements $Y[2i-1]$ and $Y[2i]$ is random, for each pair $\{1, 2\}$ we pay 1 with probability 1/2 and pay 2 with probability 1/2, and for any other pair we pay always 2. Thus the total cost incurred by the loop in line 12 is (by Chernoff bound) at most

$$\mathcal{O}(\varepsilon n) \cdot 2 + \frac{3}{2}|Y|/2 + \mathcal{O}(\sqrt{|Y|} \log n) \leq \frac{3}{8}n \pm \mathcal{O}(\varepsilon n).$$

Thus the total cost is

$$T(n) \leq T\big((\tfrac{1}{4} + \varepsilon)n\big) + \tfrac{1}{2}n + \tfrac{3}{8}n + \mathcal{O}(\varepsilon n) \leq \tfrac{7}{6}n + \mathcal{O}(n^{9/10}) + C \cdot \big(\tfrac{1}{3}n\big)^{9/10}$$

and $\frac{7}{6}n + \mathcal{O}(n^{9/10}) + C \cdot (\frac{1}{3})^{9/10} \cdot n^{9/10} \leq \frac{7}{6}n + C \cdot n^{9/10}$ for a large enough C.

HEAVY(M, v). If $p_1 > \frac{1}{2}$, then we terminate in line 7 after n comparisons. Thus we can assume that $p_1 \in [0.45 - \varepsilon, \frac{1}{2}]$. Because by Lemmas 2 and 6 both p_1^2 and $\sum_i p_i^2$ are estimated within an absolute error of $o(\varepsilon)$, we have that $p_1^2 - \sum_{i \geq 2} p_i^2 \geq 2\varepsilon - 2o(\varepsilon) \geq \varepsilon$.

We argue that the loop in line 10 will eventually find sufficiently many pairs of elements with different colors, and thus return without falling back to the $2n$ algorithm. By definition, $|X| = (1 - p_1)n$ and initially $k = (1/2 - p_1)n$. By Lemma 4, after the random shuffle the number D of pairs of elements $(X[2i-1], X[2i])$ with different colors, can be bounded by

$$\begin{aligned} D &\geq \frac{|X|}{2} - \frac{\sum_{j \geq 2}(p_j n)^2}{2|X|} - \mathcal{O}\big(|X|^{2/3} \log |X|\big) \geq \\ &\geq \frac{1 - p_1}{2}n - \frac{p_1^2 - \varepsilon}{2(1 - p_1)}n - o(\varepsilon n) \geq \frac{1 - 2p_1}{2(1 - p_1)}n + \frac{\varepsilon}{2}n - o(\varepsilon n) \geq \big(\tfrac{1}{2} - p_1\big)n; \end{aligned}$$

thus indeed there are sufficiently many pairs. Hence, because the pairs are being considered in a random order, the total cost can be bounded using Lemma 5 by

$$\begin{aligned} T(n) &\leq n + \frac{|X|}{D}\left(\frac{1}{2} - p_1\right)n + \mathcal{O}\left(\frac{|X|}{\sqrt{D}} \log |X|\right) \leq \\ &\leq n + \frac{(1 - p_1)^2}{2}n + \mathcal{O}\left(n/\sqrt{\frac{\varepsilon}{3}n} \log n\right) \leq \\ &\leq (1 + 0.55^2/2)n + \mathcal{O}(\varepsilon n) + \mathcal{O}\left(\sqrt{\frac{n}{\varepsilon}} \log n\right) = 1.15125n + \mathcal{O}(n^{9/10}), \end{aligned}$$

where we used $D \geq \frac{\varepsilon}{2} - o(\epsilon n) \geq \frac{\varepsilon}{3}n$ for a large enough n.

LIGHT(M). We start by bounding $|X|$ and $|Y|$. By Lemma 4, $|X| = \frac{n}{2}\sum_i p_i^2 \pm \mathcal{O}(n^{2/3}\log n)$, and by Lemma 3 there are $\frac{n}{2}p_1^2 \pm \mathcal{O}(n^{1/2}\log n)$ elements from A_1 in X, thus there are $n(p_1 - p_1^2) \pm \mathcal{O}(n^{1/2}\log n)$ of elements from A_1 in Y (each paired with a non-A_1 element).

We know that either $p_1 \le 0.45 + \varepsilon$ or $p_1^2 - \sum_{i\ge 2}(p_i^2) \le \varepsilon$. If there is no majority in X, then $p_1 \le \frac{1}{2}$ and the total cost is bounded by

$$T(n) \le \frac{n}{2} + T(|X|) \le \frac{n}{2} + \frac{7}{6}|X| + C\cdot|X|^{9/10},$$

which, because $|X| \le \frac{n}{4} + \mathcal{O}(n^{2/3}\log n)$, is less than $\frac{19}{24}n + o(n)$. Hence we can assume that there is a majority in X. In such case, cnt is set to

$$c = \frac{n}{2}\left(p_1^2 - \sum_{i\ge 2} p_i^2\right) \pm \mathcal{O}(n^{2/3}\log n).$$

We denote by I the total number of iterations of the loop in line 12. By Lemma 5

$$I \le \frac{\frac{1}{2}|Y|}{\frac{1}{2}|Y| - |A_1 \cap Y|}\cdot c + \mathcal{O}(E),$$

where $E = |Y|/\sqrt{\frac{1}{2}|Y| - |A_1\cap Y|}$. Substituting $S = \sum_{i\ge 2} p_i^2$, by Lemma 4 we have

$$\begin{aligned} |Y| &= \big(1 - p_1^2 - S \pm \mathcal{O}(n^{-1/3}\log n)\big)n,\\ |Y| - 2|A_1\cap Y| &= \big((1-p_1)^2 - S \pm \mathcal{O}(n^{-1/3}\log n)\big)n,\\ c &= \frac{1}{2}\big(p_1^2 - S \pm \mathcal{O}(n^{-1/3}\log n)\big)n. \end{aligned}$$

Since $p_1 \le \frac{1}{2}$ and $p_2 \le \frac{1}{2} - 3\varepsilon$ (as for a larger p_2 the sampled q_2 would be sufficiently large for other subprocedure to be used), we have

$$(1-p_1)^2 - S - o(\varepsilon) \ge \left(\tfrac{1}{2}\right)^2 - \left(\tfrac{1}{2} - 3\varepsilon\right)^2 - (3\varepsilon)^2 - o(\varepsilon) = 3\varepsilon - 18\varepsilon^2 - o(\varepsilon) \ge 2\varepsilon.$$

Thus $E \le \sqrt{\frac{n}{2\varepsilon}}$. Now, since $|Y| = \Theta(n)$ we can bound I from above by

$$\begin{aligned} I &\le \frac{|Y|}{|Y| - 2|A_1\cap Y|}\cdot\frac{1}{2}(p_1^2 - S)n + \mathcal{O}(1/\varepsilon)\cdot\mathcal{O}(n^{2/3}\log n) + \mathcal{O}(E) \le\\ &\le \frac{1}{2}\frac{1 - p_1^2 - S + \mathcal{O}(n^{-1/3}\log n)}{(1-p_1)^2 - S - \mathcal{O}(n^{-1/3}\log n)}(p_1^2 - S)n + \mathcal{O}\left(\frac{n^{2/3}\log n}{\varepsilon}\right) + \mathcal{O}\left(\sqrt{\frac{n}{4\varepsilon}}\right) \le\\ &\le \frac{1}{2}\frac{1 - p_1^2 - S}{(1-p_1)^2 - S - \mathcal{O}(n^{-1/3}\log n)}(p_1^2 - S)n + \frac{\mathcal{O}(n^{2/3}\log n)}{2\varepsilon} + \mathcal{O}(n^{23/30}\log n), \end{aligned}$$

which, because $(1-p_1)^2 - S$ is sufficiently large, can be bounded by

$$\begin{aligned} I &\le \frac{1}{2}\frac{1 - p_1^2 - S}{(1-p_1)^2 - S}(p_1^2 - S)n\cdot\left(1 + \frac{\mathcal{O}(n^{-1/3}\log n)}{(1-p_1)^2 - S}\right) + \mathcal{O}(n^{23/30}\log n) \le\\ &\le \frac{1}{2}\frac{1 - p_1^2 - S}{(1-p_1)^2 - S}(p_1^2 - S)n\cdot\left(1 + \frac{\mathcal{O}(n^{-1/3}\log n)}{2\varepsilon}\right) + \mathcal{O}(n^{23/30}\log n) \le\\ &\le \frac{1}{2}(1 - p_1^2 - S)\frac{p_1^2 - S}{(1-p_1)^2 - S}n + \mathcal{O}(n^{26/30}\log n). \end{aligned}$$

For each of c iterations we pay 2, and for each of the remaining $I - c$ iterations we pay only $\frac{3}{2}$ in expectation (for each iteration independently). Thus, by Chernoff bound the total cost is

$$T(n) \leq \frac{1}{2}n + T(|X|) + \frac{3}{2}(I-c) + \mathcal{O}(\sqrt{I-c}\log(I-c)) + 2c \leq$$

$$= \frac{n}{2}\left(1 + \frac{19}{6}p_1^2 - \frac{5}{6}S + 3(p_1^2 - S)\frac{p_1 - p_1^2}{(1-p_1)^2 - S}\right) + \mathcal{O}(n^{9/10}).$$

We reason that, for a fixed p_1, the quantity

$$1 + \frac{19}{6}p_1^2 - \frac{5}{6}S + 3(p_1^2 - S)\frac{p_1(1-p_1)}{(1-p_1)^2 - S}$$

is a decreasing function of S, since $p_1^2 \leq (1-p_1)^2$. If $p_1^2 - S \leq \varepsilon$ then simplifying with either $p_1^2 - S \leq 0$ or, since $(1-p_1)^2 - S \geq 2\varepsilon$, with $0 \leq \frac{p_1^2 - S}{(1-p_1)^2 - S} \leq \frac{1}{2}$, we obtain that $T(n) \leq \frac{47}{48}n + o(n)$. Otherwise, $p_1 \leq 0.45 + \varepsilon$ and substituting $S = 0$ (since the cost is decreasing in S) we obtain $T(n) \leq 1.06915n + \mathcal{O}(n^{9/10})$.

Wrapping up. We see that in each subprocedure, the number of comparisons is bounded by $\frac{7}{6}n + C \cdot n^{9/10}$. Each subprocedure makes at most one recursive call, where the size of the input is reduced by at least a factor of 2. Thus the worst-case number of comparison is always bounded by $\mathcal{O}(n)$. Recall that the bound on the number of comparisons used by every recursive call holds w.v.h.p. with respected to the size of the input to the call. Eventually, the size of the input might become very small, and then w.v.h.p. with respect to the size of the input is no longer w.v.h.p. with respect to the original n. However, as soon as this size decreases to, say, $n^{0.1}$, the number of comparisons is $\mathcal{O}(n)$ irrespectively of the random choices made by the algorithm. Thus w.v.h.p. the number of comparisons is at most $\frac{7}{6}n + \mathcal{O}(n^{9/10})$, and the expected number of comparisons is also bounded by $\frac{7}{6}n + \mathcal{O}(n^{9/10})$. ◀

4 Lower bound

We consider Las Vegas algorithms. That is, the algorithm must always correctly determine whether a majority element exists. We will prove that the expected number of comparisons used by such an algorithm is at least $c \cdot n - o(n)$, for some constant $c > 1$. By Yao's principle, it is sufficient to construct a distribution on the inputs, such that the expected number of comparisons used by any deterministic algorithms run on an input chosen from the distribution is at least $c \cdot n - o(n)$. The distribution is that with probability $\frac{1}{n}$ every ball has a color chosen uniformly at random from a set of n colors. With probability $1 - \frac{1}{n}$ every ball is black or white, with both possibilities equally probable. We fix a correct deterministic algorithm $\mathcal{A}$ and analyze its behavior on an input chosen from the distribution. As a warm-up, we first prove that $\mathcal{A}$ needs $n - o(n)$ comparisons in expectation on such input.

4.1 A lower bound of $n - o(n)$

In every step $\mathcal{A}$ compares two balls, thus we can describe its current knowledge by defining an appropriate graph as follows. Every node corresponds to a ball. Two nodes are connected with a *negative edge* if the corresponding balls have been compared and found out to have different colors. Two nodes are connected with a *positive edge* if the corresponding balls

are known to have the same colors under the assumption that every ball is either black or white (either because they have been directly compared and found to have the same color, or because such knowledge has been indirectly inferred from the assumption). After every step of the algorithm the graph consists of a number of components $C_1, C_2, \ldots$. Every components is partitioned into two parts $C_i = A_i \cup B_i$, such that both A_i and B_i are connected components in the graph containing only positive edges and there is at least one (possibly more than one) negative edge between A_i and B_i. There are no other edges in the graph. Now we describe how the graph changes after $\mathcal{A}$ compares two balls $x \in C_i$ and $y \in C_j$ assuming that every ball is either black or white. If $i = j$ then the result of the comparison is already determined by the previous comparisons and the graph does not change. Otherwise, $i \neq j$ and assume by symmetry that $x \in A_i, y \in A_j$. The following two possibilities are equally probable:

1. $\mathsf{color}(x) = \mathsf{color}(y)$, then we merge both components into a new component $C = A \cup B$, where $A = A_i \cup A_j$ and $B = B_i \cup B_j$ by creating new positive edges (x, y) and (x', y') for some $x' \in B_i, y' \in B_j$ (if $B_i, B_j \neq \emptyset$).
2. $\mathsf{color}(x) \neq \mathsf{color}(y)$, then we merge both components into a new component $C = A \cup B$, where $A = A_i \cup B_j$ and $B = B_i \cup A_j$ by creating new positive edges (x, y') for some $y' \in B_j$ (if $B_j \neq \emptyset$) and (x', y) for some $x' \in B_i$ (if $B_i \neq \emptyset$). We also create a new negative edge (x, y). Here we crucially use the assumption that every ball is either black or white.

The graph exactly captures the knowledge of $\mathcal{A}$ about a binary input.

Any binary input contains a majority and $\mathcal{A}$ must report so. However, because with very small probability the input is arbitrary, this requires some work due to the following lemma.

▶ **Lemma 9.** *If $\mathcal{A}$ reports that a binary input contains a majority element, then the graph contains a component $C = A \cup B$ such that $|A| > \frac{n}{2}$ or $|B| > \frac{n}{2}$.*

Proof. Assume otherwise, that is, $\mathcal{A}$ reports that a binary input contains a majority element even though both parts of every component are of size less than $\frac{n}{2}$. Construct another input by choosing, for every component $C = A \cup B$, two fresh colors c_A and c_B and setting $\mathsf{color}(x) = c_A$ for every $x \in A$, $\mathsf{color}(y) = c_B$ for every $y \in B$. Every comparison performed by $\mathcal{A}$ is an edge of the graph, so its behavior on the new input is exactly the same as on the original binary input. Hence $\mathcal{A}$ reports that there is a majority element, while the frequency of every color in the new input is less than $\frac{n}{2}$, which is a contradiction. ◀

From now on we consider only binary inputs. If we can prove that the expected number of comparisons used by $\mathcal{A}$ on such input is $n - o(n)$, then the expected number of comparisons on an input chosen from our distribution is also $n - o(n)$. Because every comparison decreases the number of components by one, it is sufficient to argue that the expected size of some component when $\mathcal{A}$ reports that there is a majority is $n - o(n)$. We already know that there must exist a component $C = A \cup B$ such that (by symmetry) $|A| > n/2$. We will argue that $|B|$ must also be large. To this end, define *balance* of a component $C_i = A_i \cup B_i$ as $\mathsf{balance}(C_i) = (|A_i| - |B_i|)^2$, and the total balance as $\sum_i \mathsf{balance}(C_i)$. By considering the situation before and after a single comparison, we obtain the following.

▶ **Lemma 10.** *The expected total balance at termination of algorithm $\mathcal{A}$ is n.*

Total balance when $\mathcal{A}$ reports a majority is a random variable with expected value n. By Markov's inequality, with probability $1 - 1/n^{1/3}$ its value is at most $n^{4/3}$, which implies that for any component $C_i = A_i \cup B_i$, we have $\mathsf{balance}(C_i) \leq n^{4/3}$. If we apply this inequality to the component $C = A \cup B$ with $|A| > n/2$, we obtain $|B| \geq n/2 - n^{2/3}$. Hence with probability $1 - 1/n^{1/3}$ there is a component with at least $n - n^{2/3}$ nodes, which means that the algorithm must have performed at least $n - n^{2/3} - 1$ comparisons. Therefore the expected number of comparisons is at least $(1 - 1/n^{1/3})(n - n^{2/3} - 1) = n - o(n)$.

4.2 A stronger lower bound

To obtain a stronger lower bound, we extend the definition of the graph that captures the current knowledge of $\mathcal{A}$. Now a positive edge can be *verified* or *non-verified*. A verified positive edge (x, y) is created only after comparing two balls x and y such that $\mathsf{color}(x) = \mathsf{color}(y)$. All other positive edges are non-verified. The algorithm can also turn a non-verified positive edge (x, y) into a verified positive edge by comparing x and y. By the same reasoning as in Lemma 9 we obtain the following.

▶ **Lemma 11.** *If $\mathcal{A}$ reports that a binary input contains a majority element, then the graph consisting of all verified positive edges contains a connected component with at least $\frac{n}{2}$ nodes.*

Now the goal is to construct a large component in the graph that consists of all verified positive edges, so it makes sense for $\mathcal{A}$ to compare two balls from the same component. However, without loss of generality, such comparisons are executed after having identified a large component in the graph consisting of all positive edges. Then, $\mathcal{A}$ asks sufficiently many queries of the form (x, y), where (x, y) is a non-verified edge from the identified component. In other words, $\mathcal{A}$ first isolates a candidate for a majority, and then makes sure that all inferred equalities really hold, which is necessary because with very small probability the input is not binary. This allows us to bound the total number of comparisons from below as follows. We define that a *majority edge* is an edge between two nodes of the majority color.

▶ **Lemma 12.** *The expected number of comparisons used by $\mathcal{A}$ on a binary input is at least $n - o(n)$ plus the expected number of non-verified majority edges.*

Proof. Recall that if there exists a component $C = A \cup B$ with $|A| > n/2$ then with probability $1 - 1/n^{1/3}$ we also have $|B| \geq n/2 - n^{2/3}$. Set A consists of nodes of the majority color, although possibly not all nodes of the majority color are there. However, because B is large, there are at most $n^{2/3}$ nodes of the majority color outside of A. Also, because we consider binary inputs chosen uniformly at random, by Chernoff bound $|A| \leq n/2 + \mathcal{O}(\sqrt{n \log n})$ with probability $1 - 1/n$.

The expected number of comparisons used by $\mathcal{A}$ to construct a component $C = A \cup B$ such that $|A| > n/2$ is at least $n - n^{2/3} - 1$. Then, $\mathcal{A}$ needs to verify sufficiently many non-verified edges inside A to obtain a connected component of size $n/2$ in the graph that consists of verified positive edges. By construction, there are no cycles in the graph that consists of positive edges. Hence with probability $1 - 1/n^{1/3} - 1/n$ there will be no more than $n^{2/3} + \mathcal{O}(\sqrt{n \log n})$ non-verified positive edges between nodes outside of B when $\mathcal{A}$ reports a majority. Consequently, the additional verification cost is the expected number of non-verified majority edges minus $n^{2/3} + \mathcal{O}(\sqrt{n \log n}) = o(n)$. ◀

In the remaining part of this section we analyze the expected number of non-verified majority edges constructed during the execution of the algorithm. We show that this is at least $(c - 1)n - o(n)$ for some $c > 1$. Then, Lemma 12 implies the claimed lower bound.

A component $C = A \cup B$ is called *monochromatic* when $A = \emptyset$ or $B = \emptyset$ (by symmetry, we will assume the latter) and *dichromatic* otherwise. With probability $1 - 1/n^{1/3}$, when $\mathcal{A}$ reports a majority there is one large dichromatic component with at least $n - n^{2/3}$ nodes, and hence the total number of components is at most $n^{2/3} + 1$. It is convenient to interpret the execution of $\mathcal{A}$ as a process of eliminating components by merging two components into one. Each such merge might create a new non-verified edge. We define that the *cost* of such a non-verified edge is the probability that it is a majority edge. We want to argue that because all but $n^{2/3}$ components will be eventually eliminated, the total cost of all non-verified edges that we create is $(c - 1)n - o(n)$.

We analyze in more detail the merging process in terms of mono- and dichromatic components. Let $\mathsf{predict}_k$ be the random variable denoting the probability that, after k steps of $\mathcal{A}$, a node from the larger part of a component is of the majority color. It is rather difficult to calculate $\mathsf{predict}_k$ exactly, so we will use a crude upper bound instead. An important property of the upper bound will be that it is nondecreasing in k. When $\mathcal{A}$ compares two balls $x \in C_i$ and $y \in C_j$ with $i \neq j$ to obtain a new component $C = A \cup B$ there are three possible cases:

1. C_i and C_j are monochromatic. Then with probability $\frac{1}{2}$ the new component C is also monochromatic, and with probability $\frac{1}{2}$ it is dichromatic.
2. C_i is dichromatic and C_j is monochromatic. The new component is dichromatic. With probability $\frac{1}{2}$ we have a new non-verified edge, and with probability at least $\frac{1}{2}(1-\mathsf{predict}_k)$ we have a new non-verified majority edge.
3. C_i and C_j are dichromatic. Then with probability $\frac{1}{2}$ we create a new non-verified edge inside both A and B, and one of them is a majority edge.

We analyze the expected total cost of all non-verified edges when only one component remains. When $\mathcal{A}$ reports a majority up to $n^{2/3}$ components might remain, but this changes only the lower order terms of the bound.

▶ **Lemma 13.** *The expected total cost of all non-verified edges when only one component remains is at least* $\sum_{k=1}^{2n/3} \mathbb{E}\big[\min\big(\frac{1}{6}, \frac{1}{2}(1-\mathsf{predict}_k)\big)\big]$.

Proof. We start with n components and need to eliminate all but at most one of them. To each component we associate credit, $\frac{1}{2}$ to each dichromatic and $\frac{1}{6}$ to each monochromatic one. The algorithm can collect the credit from both of the components it merges, but it has to pay for credit of newly created one. Additionally algorithm has to pay for any non-verified majority edge created by merging.

In every step we have three possibilities:

1. Merge two monochromatic components into one. With probability $\frac{1}{2}$ the new component is dichromatic, and with probability $\frac{1}{2}$ the new component is monochromatic. Thus the expected amortized cost for this step is 0.
2. Merge a monochromatic components with a dichromatic component. Then the total number of monochromatic components decreases by 1 and we add with probability at least $\frac{1}{2}(1-\mathsf{predict}_k)$ a non-verified majority edge. The expected amortized cost for this step is $\frac{1}{2}(1-\mathsf{predict}_k) - \frac{1}{6}$.
3. Merge two dichromatic components while adding with probability $\frac{1}{2}$ a non-verified majority edge. The expected amortized cost for this step is 0.

In total algorithm has to pay for initial credits and for each step, making the total expected cost at least

$$\frac{n}{6} + \sum_{k=1}^{n-1} \mathbb{E}\big[\min\big(0, \tfrac{1}{2}(1-\mathsf{predict}_k) - \frac{1}{6}\big)\big] \geq \sum_{k=1}^{2/3n} \mathbb{E}\big[\min\big(\tfrac{1}{6}, \tfrac{1}{2}(1-\mathsf{predict}_k)\big)\big]. \qquad \blacktriangleleft$$

We note that by truncating the sum at $\frac{2}{3}n$ we do not lose any cost estimation, as for $k \geq \frac{2}{3}n$ our estimation for $\mathsf{predict}_k$ gives 1.

Now we focus deriving an upper bound for the expression obtained in Lemma 13. To bound $\mathsf{predict}_k$ we use an approach due to Christofides [5]. At any given step k we will look at all components with a nonzero balance. Specifically, we introduce two new random variables: M_k being the largest balance of a component, and N_k being the number of components with

a nonzero balance. Since at each step, N_k is decreased in expectation at most by $\frac{3}{2}$, we have $\mathbb{E}[N_k] \geq n - \frac{3}{2}(k-1)$, and w.v.h.p., by Chernoff bound $N_k \geq n - \frac{3}{2}k - \mathcal{O}(\sqrt{k}\log n)$.

Since by Lemma 10 the expected sum of balances is n, and each nonzero component contributes at least 1 to the sum, we have $\mathbb{E}[M_k] \leq n - \mathbb{E}[N_k - 1] = \frac{3}{2}k - \frac{1}{2}$.

Now to proceed, for a component $C_i = A_i \cup B_i$ we define $\delta_i = ||A_i| - |B_i||$, a positive value such that $\delta_i^2 = \mathsf{balance}(C_i)$. Thus, at any given step k, the algorithm observes the nonzero values $\delta_1, \delta_2, \ldots, \delta_{N_k}$. Without loss of generality we can narrow our focus on a component C_1. We are interested in bounding the probability

$$\Pr(A_1 \text{ in majority}) = \Pr(\delta_1 + \varepsilon_2\delta_2 \ldots + \varepsilon_{N_k}\delta_{N_k} \geq 0) = \tfrac{1}{2} + \tfrac{1}{2}\Pr(\varepsilon_2\delta_2 \ldots + \varepsilon_{N_k}\delta_{N_k} \in [-\delta_1, \delta_1]),$$

where $\epsilon_2, \epsilon_3, \ldots, \epsilon_{N_k} \in \{-1, 1\}$ are drawn independently and uniformly at random. By a result of Erdős [10], if $\delta_2, \ldots, \delta_{N_k} \geq 1$ then the above is maximized for $\delta_2 = \ldots = \delta_{N_k} = 1$.

We now approximate binomial distribution using the symmetric case of de Moivre–Laplace Theorem. Recall that

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\,\mathrm{d}t$$

is the *cumulative distribution function* of the *normal distribution*.

▶ **Theorem 14** (De Moivre–Laplace). *Let S_n be the number of successes in n independent coin flips. Then*

$$\Pr\left(\frac{n}{2} + x_1\sqrt{n} \leq S_n \leq \frac{n}{2} + x_2\sqrt{n}\right) \sim \Phi(2x_2) - \Phi(2x_1).$$

In our case we are interested in $N_k - 1$ coin flips and the number of successes in the range $[(N_k-1)/2 - \delta_1/2, (N_k-1)/2 + \delta_1/2]$. Thus probability that 1 is the majority can be bounded from above by

$$\Pr(A_1 \text{ is the majority}) \leq \frac{1}{2}\left(\Phi\left(\frac{\delta_1}{\sqrt{N_k - 1}}\right) - \Phi\left(-\frac{\delta_1}{\sqrt{N_k - 1}}\right)\right) + \frac{1}{2} = \Phi\left(\frac{\delta_1}{\sqrt{N_k - 1}}\right).$$

Because M_k is the largest balance of a component, $\delta_1, \delta_2, \ldots, \delta_{N_k}$ are bounded from above by $\sqrt{M_k}$. Additionally, w.v.h.p. $N_k \geq n - \frac{3}{2}k - \mathcal{O}(\sqrt{n}\log n)$, thus

$$\mathsf{predict}_k \leq \Phi\left(\sqrt{\frac{M_k}{n - \frac{3}{2}k - \mathcal{O}(\sqrt{n}\log n)}}\right).$$

Since $\Phi(\sqrt{x}/\text{const})$ is a concave function, we can apply expected value, and get

$$\mathbb{E}[\mathsf{predict}_k] \leq \Phi\left(\sqrt{\frac{\mathbb{E}[M_k]}{n - \frac{3}{2}k - \mathcal{O}(\sqrt{n}\log n)}}\right) \sim \Phi\left(\sqrt{\frac{\frac{3}{2}k}{n - \frac{3}{2}k}}\right).$$

Now we are ready to bound the sum from Lemma 13. Using the linearity of expectation and inequality $\min(\frac{1}{6}, \frac{1}{2}x) \geq \frac{1}{6}x$ for $x \in [0, 1]$ we obtain:

$$\mathbb{E}\left[\sum_{k=1}^{2n/3} \min\left(\frac{1}{6}, \frac{1}{2}(1 - \mathsf{predict}_k)\right)\right] = \sum_{k=1}^{2n/3} \mathbb{E}\left[\min\left(\frac{1}{6}, \frac{1}{2}(1 - \mathsf{predict}_k)\right)\right] \geq$$

$$\geq \sum_{k=1}^{2n/3} \frac{1}{6}(1 - \mathbb{E}[\mathsf{predict}_k]) \geq n \cdot \int_0^{2/3} \frac{1}{6}\left(1 - \Phi\left(\sqrt{\frac{\frac{3}{2}x}{1 - \frac{3}{2}x}}\right)\right)\mathrm{d}x - o(n).$$

Finally, we calculate

$$1 + \int_0^{2/3} \frac{1}{6} \left(1 - \Phi\left(\sqrt{\frac{\frac{3}{2}x}{1-\frac{3}{2}x}}\right)\right) \mathrm{d}x \approx 1.0191289.$$

▶ **Theorem 15.** *Any algorithm that reports majority exactly requires in expectation at least* $1.019n$ *comparisons.*

References

1 Martin Aigner, Gianluca De Marco, and Manuela Montangero. The plurality problem with three colors and more. *Theor. Comput. Sci.*, 337(1-3):319–330, 2005.
2 Laurent Alonso, Edward M. Reingold, and René Schott. Determining the majority. *Inf. Process. Lett.*, 47(5):253–255, 1993.
3 Laurent Alonso, Edward M. Reingold, and René Schott. The average-case complexity of determining the majority. *SIAM J. Comput.*, 26(1):1–14, 1997.
4 Robert S. Boyer and J. Strother Moore. MJRTY: A fast majority vote algorithm. In *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 105–118, 1991.
5 Demetres Christofides. On randomized algorithms for the majority problem. *Discrete Applied Mathematics*, 157(7):1481–1485, 2009.
6 Fan R. K. Chung, Ronald L. Graham, Jia Mao, and Andrew Chi-Chih Yao. Oblivious and adaptive strategies for the majority and plurality problems. *Algorithmica*, 48(2):147–157, 2007.
7 Dorit Dor and Uri Zwick. Selecting the median. *SIAM J. Comput.*, 28(5):1722–1758, 1999.
8 Dorit Dor and Uri Zwick. Median selection requires $(2+\epsilon)n$ comparisons. *SIAM J. Discrete Math.*, 14(3):312–325, 2001.
9 David Eppstein and Daniel S. Hirschberg. From discrepancy to majority. In *LATIN*, volume 9644 of *Lecture Notes in Computer Science*, pages 390–402. Springer, 2016.
10 P. Erdős. On a lemma of Littlewood and Offord. *Bull. Amer. Math. Soc.*, 51(12):898–902, 12 1945.
11 M. Fischer and S. Salzberg. Finding a majority among n votes: solution to problem 81-5. *Journal of Algorithms*, 1982.
12 Dániel Gerbner, Gyula O. H. Katona, Dömötör Pálvölgyi, and Balázs Patkós. Majority and plurality problems. *Discrete Applied Mathematics*, 161(6):813–818, 2013.
13 Daniel Král, Jirí Sgall, and Tomás Tichý. Randomized strategies for the plurality problem. *Discrete Applied Mathematics*, 156(17):3305–3311, 2008.
14 Gianluca De Marco and Evangelos Kranakis. Searching for majority with k-tuple queries. *Discrete Math., Alg. and Appl.*, 7(2), 2015.
15 Gianluca De Marco and Andrzej Pelc. Randomized algorithms for determining the majority on graphs. *Combinatorics, Probability & Computing*, 15(6):823–834, 2006.
16 Mike Paterson. Progress in selection. In *Algorithm Theory–SWAT'96*, pages 368–379. Springer, 1996.
17 Michael E. Saks and Michael Werman. On computing majority by comparisons. *Combinatorica*, 11(4):383–387, 1991.
18 Máté Vizer, Dániel Gerbner, Balázs Keszegh, Dömötör Pálvölgyi, Balázs Patkós, and Gábor Wiener. Finding a majority ball with majority answers. *Electronic Notes in Discrete Mathematics*, 49:345–351, 2015.
19 Gábor Wiener. Search for a majority element. *Journal of Statistical Planning and Inference*, 100(2):313–318, 2002.

A Framework for Dynamic Parameterized Dictionary Matching*

Arnab Ganguly[1], Wing-Kai Hon[2], and Rahul Shah[3]

1 **School of Electrical Engineering and Computer Science, Louisiana State University, Baton Rouge, USA**
agangu4@lsu.edu,rahul@csc.lsu.edu

2 **Department of Computer Science, National Tsing Hua University, Hsinchu City, Taiwan**
wkhon@cs.nthu.edu.tw

3 **School of Electrical Engineering and Computer Science, Louisiana State University, Baton Rouge, USA; and National Science Foundation, Arlington, USA**
rahul@nsf.gov

Abstract

Two equal-length strings S and S' are a parameterized-match (p-match) iff there exists a one-to-one function that renames the characters in S to those in S'. Let $\mathcal{P}$ be a collection of d patterns of total length n characters that are chosen from an alphabet Σ of cardinality σ. The task is to index $\mathcal{P}$ such that we can support the following operations:

- $\mathsf{search}(T)$: given a text T, report all occurrences $\langle j, P_i \rangle$ such that there exists a pattern $P_i \in \mathcal{P}$ that is a p-match with the substring $T[j, j + |P_i| - 1]$.
- $\mathsf{insert}(P_i)/\mathsf{delete}(P_i)$: modify the index when a pattern P_i is inserted/deleted.

We present a linear-space index that occupies $\mathcal{O}(n \log n)$ bits and supports (i) $\mathsf{search}(T)$ in worst-case $\mathcal{O}(|T| \log^2 n + occ)$ time, where occ is the number of occurrences reported, and (ii) $\mathsf{insert}(P_i)$ and $\mathsf{delete}(P_i)$ in amortized $\mathcal{O}(|P_i| \operatorname{polylog}(n))$ time. Then, we present a succinct index that occupies $(1 + o(1))n \log \sigma + \mathcal{O}(d \log n)$ bits and supports (i) $\mathsf{search}(T)$ in worst-case $\mathcal{O}(|T| \log^2 n + occ)$ time, and (ii) $\mathsf{insert}(P_i)$ and $\mathsf{delete}(P_i)$ in amortized $\mathcal{O}(|P_i| \operatorname{polylog}(n))$ time. We also present results related to the semi-dynamic variant of the problem, where deletion is not allowed.

1998 ACM Subject Classification F.2.2 Pattern Matching

Keywords and phrases Parameterized Dictionary Indexing, Generalized Suffix Tree, Succinct Data Structures, Sparsification

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.10

1 Introduction

Designing succinct data structures for the classical pattern matching problem of finding all occurrences of a pattern P in a fixed text T can be traced back to the seminal work of Grossi and Vitter [16], Ferragina and Manzini [13], and Sadakane [29]. This established an active research area of designing succinct data structures. The focus was now on either improving these initial breakthroughs (see [26] for a comprehensive survey), or designing succinct data

* The work of Arnab Ganguly was supported by National Science Foundation Grants CCF–1017623 and CCF–1218904. The work of Wing-Kai Hon was supported by National Science Council Grants 102-2221-E-007-068-MY3 and 105-2918-I-007-006.

structures for other variants [6, 8, 14, 25, 30]. *Dictionary Matching*, a typical example of these variants, is defined as follows. Let $\mathcal{P}$ be a collection of d patterns $\{P_1, P_2, \ldots, P_d\}$ of total length n characters which are chosen from a totally-ordered alphabet Σ of cardinality σ. Given a text T, also over Σ, the task is to report all positions j such that at least one of the patterns $P_i \in \mathcal{P}$ exactly matches an equal-length substring of T that starts at j. Typically, the patterns which occur at j are also reported. In the *Dictionary Indexing* problem, the patterns are provided upfront (and remain fixed) and the text comes as a query. The classical solution for this problem is the Aho-Corasick (AC) automaton [1] which occupies $\mathcal{O}(m \log m)$ bits of space, where $m \leq n+1$ is the number of states in the automaton, and finds all *occ* occurrences in optimal time $\mathcal{O}(|T| + occ)$. To the best of our knowledge, the first succinct index for this problem is by Hon et al. [17]. Later, Belazzougui [6] presented an $m \log \sigma + \mathcal{O}(m + d \log(n/d))$ bit index with optimal $\mathcal{O}(|T| + occ)$ query time.

Arguably, the most natural variant of the dictionary indexing problem is the *Dynamic Dictionary Indexing* problem in which we are allowed to insert a new pattern or delete an existing one. The challenge is to modify the index under such updates without having to rebuild it from scratch. Of course, the index should still be able to answer text queries in a reasonably efficient time. The first non-trivial solution was provided by Amir et al. [3]. As opposed to the AC-automaton, their technique comprised of the generalized suffix tree GST of all the patterns. Later, this was improved by Amir et al. [4] and again by Alstrup et al. [2]. Each of these indexes occupies $\mathcal{O}(n \log n)$ bits of space. The natural question to ask is "Does there exist a succinct index for dynamic dictionary matching?". Chan et al. [9] answered this by presenting an $\mathcal{O}(n\sigma)$-bit index which is (nearly) succinct only for $\sigma = \mathcal{O}(1)$. Moreover, their query time suffers from an $\mathcal{O}(\log^2 n)$ multiplicative slowdown (compared to the AC-automaton) due to the use of complicated dynamic versions of FM-Index [13] and Compressed Suffix Tree [16] as the underlying main ingredient. Updating the index for a pattern P_i required $\mathcal{O}(|P_i| \log^2 n)$ time. Hon et al. [18] improved this to a more space efficient $(1 + o(1))n \log \sigma + \mathcal{O}(d \log n)$ bit index with a faster $\mathcal{O}(|T| \log n + occ)$ query time and $\mathcal{O}(|P_i| \log \sigma + \log n)$ update time. Recently, Feigenblat et al. [12] improved the query time to $\mathcal{O}(|T|(\log \log n)^2 + occ)$ for $\sigma = \mathcal{O}(\text{polylog}(n))$.

Parameterized Pattern Matching has received significant attention (see [22] for a survey) since its inception by Baker [5]. The alphabet Σ is partitioned into two disjoint sets: Σ_s containing static-characters (s-characters) and Σ_p containing parameterized characters (p-characters). Two strings S and S', both over Σ, are a parameterized-match (p-match) iff $|S| = |S'|$, and there is a one-to-one function f such that $S[i] = f(S'[i])$. For any s-character $c \in \Sigma_s$, we have $f(c) = c$. Thus, for $\Sigma_s = \{A, B, C\}$ and $\Sigma_p = \{w, x, y, z\}$, the strings $AxBxCy$ and $AzBzCx$ are p-match, but $AxBxCy$ and $AzBwCx$ are not. We consider the *Parameterized Dictionary Matching* problem which was introduced by Idury and Schäffer [19]. This is similar to the standard dictionary problem, just that the alphabet Σ is partitioned into Σ_s and Σ_p, and we consider p-matches of a pattern to the text. Idury and Schäffer presented an AC-automaton like solution which occupies $\mathcal{O}(m \log m) = \mathcal{O}(n \log n)$ bits, and reports all *occ* occurrences in $\mathcal{O}(|T| \log \sigma + occ)$ time. Our main focus lies on the dynamic version of this problem. Specifically, we present the following results.

▶ **Theorem 1.** *By maintaining a linear-space index occupying $\mathcal{O}(n \log n)$ bits, we can answer:*

- $\mathsf{search}(T)$ *in worst-case $\mathcal{O}(|T| \log^2 n + occ)$ time.*
- $\mathsf{insert}(P_i)$ *in amortized $\mathcal{O}(|P_i| \log n)$ time.*
- $\mathsf{delete}(P_i)$ *in amortized $\mathcal{O}(|P_i| \log^2 n)$ time.*

▶ **Theorem 2.** *By maintaining a succinct-space index occupying* $(1+o(1))n\log\sigma+\mathcal{O}(d\log n)$ *bits, we can answer:*

- $\mathsf{search}(T)$ *in worst-case* $\mathcal{O}(|T|\log^2 n+occ)$ *time.*
- $\mathsf{insert}(P_i)$ *in amortized* $\mathcal{O}(|P_i|\log n)$ *time.*
- $\mathsf{delete}(P_i)$ *in amortized* $\mathcal{O}(|P_i|\log\sigma+\log d)$ *time.*

1.1 Roadmap

We show that if the patterns are appropriately encoded [5], then the problem can be solved using a generalized suffix tree GST of all the encoded patterns. Although the techniques are similar to that of Amir et al. [3] and Hon et al. [18], we need much more machinery to deal with parameterized patterns. This is because a crucial property, known as *suffix links*, of traditional suffix trees does not apply directly for parameterized strings. This makes navigating in the GST more tricky, and we have to augment the tree with additional data structures. Furthermore, it is difficult to maintain the analogous version of suffix links in the GST explicitly as they are more fragile to the deletion of patterns. Hence, we need an implicit representation. Also, following suffix links in the GST is trickier as the text and patterns have to be re-encoded. Moreover, maintaining the encoded patterns explicitly causes the space to increase to $n\log n$ bits as opposed to the $n\log\sigma$ bits occupied by the patterns.

The succinct solution is largely based on the sparsification technique [17, 18] for the (dynamic) dictionary matching problem. Broadly speaking, for a parameter Δ, the idea is to sample suffixes at an interval of Δ, and then maintain a GST for these sampled suffixes. Likewise, the text is also sampled. Now the sampled text starting from $i=1$ is matched, and all occurrences are reported. The occurrences reported in this run lie in the set $\{i, i+\Delta, i+2\Delta, \dots\}$. All occurrences are subsequently reported by repeating the process for $i=1,2,3,\dots,\Delta$. The sparsification technique, however, does not immediately extend to the case of parameterized matching. For one, handling and maintaining suffix links is trickier. Another issue is how to handle truncating of characters at the beginning of a currently matched text, which is essential for the approaches in [3, 17, 18].

In Section 2, we first present a linear space index and prove Theorem 1. This index forms the backbone of the succinct index (Theorem 2); the details are provided in Section 3. Section 4 discusses results on the semi-dynamic variant of the problem.

2 Linear Space Index

We assume that the alphabet Σ is disjoint from the set of integers. Any string S over Σ can be initially processed in $\mathcal{O}(|S|\log\sigma)$ time to ensure that this condition holds.

2.1 Parameterized Suffix Tree

Baker [5] introduced the following encoding scheme to enable matching of parameterized strings. Given a string S, obtain a string $\mathsf{prev}(S)$ by replacing the first occurrence of every p-character in S by 0, and any other occurrence by the difference in position from its previous occurrence. Thus, $\mathsf{prev}(AxByAxCz) = A0B0A4C0$, where $\{A,B,C\}\in\Sigma_s$ and $\{x,y,z\}\in\Sigma_p$. It is easy to see that $\mathsf{prev}(S)$ can be computed in $\mathcal{O}(|S|\log\sigma)$ time.[1] Baker showed that two strings S and S' are a p-match iff $\mathsf{prev}(S)=\mathsf{prev}(S')$. They introduced

[1] Read S from left to right, and use a balanced binary search tree (BST) to maintain the position of the latest occurrence of each p-character.

the *Parameterized Suffix Tree* (PST) of a string S, which is a compacted trie of the strings $\mathsf{prev}(S[i,|S|])$, $1 \leq i \leq |S|$. At each node u in PST, maintain $\mathsf{strDepth}(u)$ i.e., the length of the string formed by concatenating the edge labels from root to u. The label of an edge $e = (u, v)$ is derived dynamically as follows. We maintain two pointers from e to the start position sp and end position ep of the label in S. (Note that the encoding of the edge is not necessarily $\mathsf{prev}(S)[sp, ep]$.) Suppose we want to find the encoding of the jth character on e, where $\mathsf{strDepth}(u) = D$. Then we find the encoding $x = \mathsf{prev}(S)[sp + j - 1]$ using the pointers. If x is an s-character, then x is itself the desired encoding. Otherwise, if $x \geq D + j$ then the encoding is 0, else it is x. If $\mathsf{prev}(S)$ has been pre-computed and stored explicitly, then all operations require constant time per character. Suppose S_u is the string obtained by concatenating the labels (over S) of the edges from root to a node u. Then, the suffix link of u points to the location in the PST which is represented by $\mathsf{prev}(S_u[2, |S_u|])$. If $|S_u| \leq 1$, then the suffix link points at the root. Baker showed that unlike in suffix trees, a suffix link in PST can point to inside an edge.

Although Baker's encoding makes p-matching easier to handle, for our purposes, it suffers from a drawback. Specifically, $\mathsf{prev}(S)$ is a string over an alphabet of size $\Theta(n)$ in the worst case, whereas the original alphabet size σ may be much smaller in comparison. In order to alleviate this, our objective is to maintain S in $|S| \log \sigma$ bits so that we can still use the PST. In the above discussion, note that in order to find the prev-encoding of a p-character at the jth position, it suffices to find the last position (if any) in the interval $[sp - D, sp + j - 2]$ where the character $S[sp + j - 1]$ occurs. To facilitate this, instead of maintaining S explicitly, we build a Wavelet Tree [15] over it. Using this, we can easily find the desired encoding as follows (see Fact 3). Let $x = \mathsf{rank}(sp + j - 1, \mathsf{access}(sp + j - 1))$. If $x = 1$, the required encoding is 0. Otherwise, let $y = \mathsf{select}(x - 1, \mathsf{access}(sp + j - 1))$. If $y \geq sp - D$ then the encoding is $(sp + j - 1 - y)$ and is 0, otherwise.

▶ **Fact 3** ([15]). *Let S be a string of length m over an alphabet Σ of size σ. We can build a data structure in $\mathcal{O}(m \log \sigma)$ time that occupies $m \log \sigma + o(m \log \sigma)$ bits and supports the following operations in $\mathcal{O}(\log \sigma)$ time:*

- $\mathsf{access}(i) = S[i]$.
- $\mathsf{rank}(i, c) =$ *the number of occurrences of* $c \in \Sigma$ *in the substring* $S[1, i]$.
- $\mathsf{select}(j, c) =$ *the smallest position* i *such that* $\mathsf{rank}(i, c) = j$.

Suppose we are trying to find all p-matches of a string S' with S using the PST of S. Given a node v, in $\mathcal{O}(1)$ time, we can find the correct outgoing edge of v that matches the next (encoded) character of $\mathsf{prev}(S')$ by using a perfect hash function at each node. Specifically, the hash function maps the (encoded) first character of an edge to the edge itself. Every other p-character on an edge can be appropriately encoded as described above in $\mathcal{O}(\log \sigma)$ time. Therefore, the time to find all *occ* p-matches is $\mathcal{O}(|S'| \log \sigma + occ)$.

2.2 The Index

We assume that no two patterns P_i and P_j exist such that $\mathsf{prev}(P_i) = \mathsf{prev}(P_j)$. Recall that maintaining the patterns in their prev-encoded form requires $\Theta(n \log n)$ bits in the worst case. Although this will not affect our overall space (for the linear space index), we will use the following scheme, which would be carried forward to our succinct index. For every pattern $P_i \in \mathcal{P}$, we maintain a wavelet tree WT over P_i. Then we create a generalized parameterized suffix tree GST out of all the prev-encoded suffixes of $P_i\$_i$ and $P_i\#_i$, where $\$_i$ and $\#_i$ are two special s-characters neither of which belongs to Σ_s. Note that each leaf corresponds to the prev-encoded suffix of $P_i\$_i$ or $P_i\#_i$ for some pattern P_i. We maintain a link from

the leaf corresponding to the string $\mathsf{prev}(P_i[j, |P_i|])\$_i$ to the leaf corresponding to the string $\mathsf{prev}(P_i[j+1, |P_i|])\$_i$. Likewise, for the leaf corresponding to $\mathsf{prev}(P_i[j, |P_i|])\#_i$. This will help us in recognizing the suffix link of a node v implicitly.

For any node u, with slight abuse of notation, denote by $\mathsf{prev}(u)$ the string obtained by concatenating the encoded edge labels from root to u. As described in Section 2.1, (i) at each node u in the GST we maintain $\mathsf{strDepth}(u) = |\mathsf{prev}(u)|$ explicitly, and (ii) each edge is labeled by two pointers to the start and end positions of its label in a particular pattern. Using these and the WTs over the patterns, we can find the desired encoding of any character on an edge in $\mathcal{O}(\log \sigma)$ time (see Section 2.1). Furthermore, at each node we use the *Dynamic Perfect Hashing* technique of Dietzfelbinger et al. [11] such that given the next (encoded) character of the text, we can navigate to the appropriate edge (if any) in constant time. Moreover, we can update (both insert and delete) the hash table in amortized $\mathcal{O}(1)$ time. The total space needed to maintain the hash tables over all nodes is $\mathcal{O}(n \log n)$ bits.

Using the data structure of Sadakane and Navarro [27], we maintain a dynamic succinct representation of the GST (see Fact 4). The weight (for the purpose of wla queries) of a node u is $\mathsf{strDepth}(u) \leq n$. Clearly, the GST satisfies the min-heap property.

▶ **Fact 4** ([27]). *Given a dynamic tree with m weighted nodes, where a node's weight is greater than that of its parent. By encoding the tree topology in $2m + o(m)$ bits, we can support the following operations in $\mathcal{O}(\log m)$ time:*

- *Inserting or deleting a node.*
- *Lowest common ancestor* (LCA) *of two nodes.*
- *For any node, find its* (i) *pre-order rank,* (ii) *node-depth,* (iii) *parent,* (iv) *number of children,* (v) *ith leftmost child,* (vi) *number of sibling to its left,* (vii) *number of leaves in its subtree, and* (viii) *ith leftmost leaf in its subtree.*
- $\mathsf{levelAncestor}(v, D)$ *i.e., the node on the root to v path having node-depth D.*

Using this, in $\mathcal{O}(\log^2 m)$ time, we can find $\mathsf{wla}(u, W)$ i.e., the lowest ancestor (if any) of a node u that has weight at most W. This is facilitated by $\mathcal{O}(\log m)$ binary searches on the node-weights using levelAncestor *queries.*

For each pattern P_i, we locate the node u (which necessarily exists) such that $\mathsf{prev}(u) = \mathsf{prev}(P_i)$. We mark all such nodes with the corresponding pattern, and process the GST with dynamic nearest marked ancestor queries (see Fact 5). Consider a tree with m nodes, k of which are marked. Hon et al. [18] argued that by maintaining the order-maintenance data structure of Dietz and Sleator [10], the relative pre-order rank of two nodes can be compared in $\mathcal{O}(1)$ time. Furthermore, a node can be inserted into the data structure in $\mathcal{O}(1)$ time given either the predecessor or the successor (in pre-order) of the node; the node can also be deleted in $\mathcal{O}(1)$ time. Hon et al. used this to maintain the marked nodes in an interval tree. A marked node v is an ancestor of a node u iff the pre-order rank of u lies in the interval $[r_v, r_{v'}]$, where r_v and $r_{v'}$ are the pre-order ranks of v and of the last visited node in the subtree of v. The desired location where a new interval has to be inserted (or an existing one has to be deleted), can found in $\mathcal{O}(\log k)$ time using the interval tree. Likewise, the smallest interval which contains a node can be found in $\mathcal{O}(\log k)$ time; all subsequent intervals that encloses this smallest interval can be found in $\mathcal{O}(1)$ time per interval. The intervals in this tree are "elastic" in the sense that the pre-order ranks are compared in $\mathcal{O}(1)$ time "on the fly" using the order-maintenance data structure. Note that the pre-order rank of a node's successor/predecessor is found using Fact 4 in $\mathcal{O}(\log m)$ time.[2] Summarizing,

[2] The predecessor of each node is defined apart from the root. Given a non-root node u, its predecessor is

▶ **Fact 5** ([10, 18]). *Given a dynamic tree with m nodes and $k \leq m$ marked nodes. We can build an $\mathcal{O}(m \log m)$-bit data structure to support the following operations:*

- *Inserting or deleting a marked node in $\mathcal{O}(\log m)$ time.*
- *Report the K marked ancestors (if any) of a node in $\mathcal{O}(\log k + K)$ time.* ◀

2.3 Reporting Occurrences

A pattern P_i occurs at a position j in the text iff $\mathsf{prev}(P_i)$ is a prefix of $\mathsf{prev}(T[j, |T|])$. To find all patterns (if any) occurring at position j, we first find the deepest node v (called *locus*) such that $\mathsf{prev}(v)$ is a prefix of $\mathsf{prev}(T[j, |T|])$. Starting from v, we report all K marked ancestors of v using Fact 5 in $\mathcal{O}(\log d + K)$ time.

The task, therefore, is to find the locus of $\mathsf{prev}(T[j, |T|])$ for every $j \in [1, |T|]$ starting with $j = 1$. First we compute $\mathsf{prev}(T)$ in $\mathcal{O}(|T| \log \sigma)$ time. We use the WT and the edge pointers to traverse the GST starting from the root as follows. If we are at a node x, we use $\mathsf{prev}(T)[\mathsf{strDepth}(x) + 1]$ to select the correct edge in $\mathcal{O}(1)$ time. If we are inside an edge, then we use the next character of edge, say c, and verify it with the next character of $\mathsf{prev}(T)$. If c is static then it is easy. Otherwise, c needs to be encoded (as in Section 2.1) requiring $\mathcal{O}(\log \sigma)$ time. We continue this process until we hit a position $(k + 1)$ in the text such that the (encoded) character does not match. Let the corresponding edge be (u, v), where u is the locus of $\mathsf{prev}(T)$. Now, we need to find the locus of $\mathsf{prev}(T[j, |T|])$, where $j = 2$. We differ from the strategy of Amir et al. [3] in that we follow the suffix link of v instead of u.[3] (If $\mathsf{strDepth}(u) = k$, then follow the suffix link of u.) Recall that we do not explicitly maintain suffix links (other than in leaves). The following two cases are to be considered.

- $\boldsymbol{T[j-1] = T[1]}$ **is an s-character**: In this case, the suffix link necessarily points to a node w and $\mathsf{prev}\big(T[j, |T|]\big) = \mathsf{prev}\big(T[j-1, |T|]\big)[j, |T|]$. Our task is to locate the prefix of $\mathsf{prev}(w)$ which equals $\mathsf{prev}(T[j, k])$ (in this case, $j = 2$). Note that this prefix necessarily exists. We first locate a leaf ℓ in the subtree of v. Follow the pointer from ℓ to the leaf ℓ' depicting the starting position of the immediate next suffix as that of ℓ. We use the query $\mathsf{wla}(\ell', k - j + 1)$ to locate a node w'. If $\mathsf{strDepth}(w') = k - j + 1$, then we are done. Otherwise, we use the character $\mathsf{prev}(T[j, |T|])[\mathsf{strDepth}(w')]$ to select the proper edge. The desired location is given by $(k - j + 1 - \mathsf{strDepth}(w'))$ on this edge. Finally, we start matching from $T[k + 1]$ as defined previously until we hit a mismatch, resulting in the desired locus of $\mathsf{prev}(T[2, |T|])$.
- $\boldsymbol{T[j-1] = T[1]}$ **is a p-character**: The suffix link of v may point to the middle of an edge, say (x, y). Also, in this case as the encoding of T has to be modified. Specifically, $\mathsf{prev}(T[j, |T|])$ and $\mathsf{prev}(T[j-1, |T|])[j, |T|]$ may no longer be the same. Observe that for any j', if $\mathsf{prev}(T)[j']$ points to a location before j, then the desired encoding at j' is 0. Thus, we can easily update the encoding in $\mathcal{O}(1)$ time as characters are read. The correct position to start matching from $T[k + 1]$ can be found as described in the previous case by initially choosing a leaf in the subtree of v.

Summarizing, every time we locate the locus of $\mathsf{prev}(T[j, |T|])$, we truncate the character $T[j]$ by following the suffix link, obtain the encoding of $\mathsf{prev}(T[j + 1, |T|])$ if required, and

---

its parent v if u is the leftmost child of v; otherwise, the predecessor is the rightmost leaf in the subtree of its immediate left sibling. For the root node, its successor is its leftmost child.

[3] This is because if the first character of the current suffix (in this case, $T[1]$) is parameterized, then the suffix link from u can point to the middle of an edge (u', v'). Suppose, after reading the next characters we found a mismatch on the edge itself. Taking the suffix link from u' will push back us further and we may end up comparing too many characters.

then use the next characters of the text to find the locus of $\mathsf{prev}(T[j+1,|T|])$. By repeating the process, we will have located the locus of $\mathsf{prev}(T[j,|T|])$ for every $j \in [1,|T|]$.

The space occupied by the index is clearly $\mathcal{O}(n \log n)$ bits. Choosing the correct outgoing edge (if any) at any node takes $\mathcal{O}(1)$ time. Finding the leaf for an implicit suffix link operation takes $\mathcal{O}(\log n)$ time. Each weighted level ancestor query takes $\mathcal{O}(\log^2 n)$ time and WT query takes $\mathcal{O}(\log \sigma)$ time. Therefore, the time to find the loci is $\mathcal{O}(|T| \log^2 n)$, and the total time to report all occurrences is $\mathcal{O}(|T| \log^2 n + occ)$.

2.4 Handling Updates

We assume the pattern P_i that is to be inserted is not present in the dictionary. Likewise, for deletion, the pattern is present in the dictionary. Both can be easily verified in $\mathcal{O}(|P_i| \log \sigma)$ time by traversing the GST.

Insertion: To modify the GST, we use the algorithm of Kosaraju [21] which constructs the parameterized suffix tree PST of a string S in $\mathcal{O}(|S| \log \sigma)$ time. The algorithm, an adaptation of the McCreight's construction algorithm [24] for the traditional suffix tree, creates the PST by successively inserting the suffixes at positions $1, 2, \ldots, |S|$. Suffix links in the case of PST may point to the middle of an edge. These are termed as *bad* suffix links while the others (pointing to a node) are termed as *good* suffix links. Contrary to McCreight's algorithm, it no longer holds that every node other than the last entered leaf and its parent have good suffix links defined. For a node v, if $\mathsf{prev}(v)$ starts with an s-character then the suffix link of v is necessarily good. This allows insertion of suffixes starting with s-characters to remain the same as in case of McCreight's algorithm. Baker [5] showed that bad nodes (i.e., nodes with bad suffix links) have an outgoing edge labeled by a 0 and also form a chain in the PST. The number of bad nodes in this chain is at most $|\Sigma_p|$. Baker used this crucial observation to locate the desired bad suffix link to be followed for entering the next suffix, culminating in an $\mathcal{O}(|S|(|\Sigma_p| + \log \sigma))$ construction algorithm.

Kosaraju showed that by maintaining two pointers *low* and *high* to the lowest and highest nodes in the chain, the construction algorithm of Baker can be improved to $\mathcal{O}(|S| \log \sigma)$ when Σ_s is a constant-sized alphabet. Basically, the low and high pointers allow us to binary search on the chain of bad nodes to locate the proper position, rather than searching the entire chain. This improves the $|\Sigma_p|$ term to $\log |\Sigma_p|$. Kosaraju first created two separate suffix trees: (i) $\mathcal{T}_1$ for S with all p-characters replaced by 0 and (ii) $\mathcal{T}_2$ for S with all s-characters replaced by a single s-character. The first tree $\mathcal{T}_1$ can be constructed using Baker's algorithm and $\mathcal{T}_2$ using Kosaraju's algorithm for the constant-sized static alphabet. Using these trees, the final suffix tree is created. The trees are pre-processed with the data structure in [7] to support constant time LCA and $\mathsf{strDepth}$ queries for efficiently finding longest common prefix (LCP) information. For each suffix insertion, the number of such queries is $\mathcal{O}(\log |\Sigma_p|)$.

We show how to update the index for inserting a pattern P_i using the techniques above. The location to insert the first suffix i.e., $\mathsf{prev}(P_i)$ can be found by traversing the GST in $\mathcal{O}(|P_i| \log \sigma)$ time. Each suffix insertion in the GST will incur a cost of $\mathcal{O}(\log \sigma)$ for the $\mathcal{O}(\log |\Sigma_p|)$ number of LCA queries in $\mathcal{T}_1$ and $\mathcal{T}_2$, and $\mathcal{O}(\log n)$ time for inserting a constant number of nodes in the dynamic representation of the GST. Whenever a new node is to be inserted in the GST, we update the hash table in amortized $\mathcal{O}(1)$ time. The data structure of Fact 5 is modified once (insertion of a marked node corresponding to P_i in GST) and requires $\mathcal{O}(\log n)$ time. Finally, when the GST is constructed we will maintain the good suffix links (constructed by Kosaraju's algorithm) for each leaf corresponding to each suffix

of P_i. The WT for P_i can be constructed in $\mathcal{O}(|P_i| \log \sigma)$ time (see Fact 3). Thus, a pattern P_i can be inserted into the index in amortized $\mathcal{O}(|P_i| \log n)$ time.

Deletion: First, we find the locus u of $\mathsf{prev}(P_i)$ and unmark u. The time required is $\mathcal{O}(|P_i| \log \sigma + \log n)$. Then, we locate the loci of $\mathsf{prev}(P_i[j, |P_i|])$, $1 < j \leq |P_i|$. Let u be any such locus. Note that there are two edges out of u labeled by $\$_i$ and $\#_i$. Delete these edges and the corresponding children of u. There are two cases to be considered.

- ***u* is a leaf:** Remove u and its edge to its parent v. If v has more than one child, then modify the hash table at v. Otherwise, v is a node with a single child x. Let y be the parent of v. Add an edge from y to x with the label as the concatenated label of the edges from y to v and v to x (achieved by assigning the edge pointers appropriately). Modify the hash table at y. Remove the node v along with its edges.[4]
- ***u* is an internal node:** Modify u by treating it as node v in the above case.

Recall that the edge labels are maintained via two pointers to the start and end positions in a particular pattern. Upon pattern deletion, we may still have existing edges in the GST which have pointers to the deleted pattern P_i. (This happens as P_i may share a common prev-encoded prefix with many other patterns.) Relabeling of these edges is achieved as follows. Each edge can be found while locating the loci of each prev-encoded suffix of P_i. Consider such an edge (x, y). After deletion, we find a leaf in the subtree of y which is labeled with a pattern $P_{i'}$ and the starting position j' of the particular suffix. Then the pointers of the edge are modified easily in $\mathcal{O}(1)$ time using $P_{i'}$, j', $\mathsf{strDepth}(x)$, and $\mathsf{strDepth}(y)$.

Locating the loci requires $\mathcal{O}(|P_i| \log^2 n)$ time. For each locus, we perform a constant number of operations, each requiring amortized $\mathcal{O}(\log n)$ time (for modifying the data structure of Fact 4 and the hash table). Also, we relabel each edge correctly in $\mathcal{O}(\log n)$ time. The number of such edges is bounded by $\mathcal{O}(|P_i|)$. Finally, the WT corresponding to P_i can be easily deleted in $\mathcal{O}(1)$ time. Thus, the total time is bounded by $\mathcal{O}(|P_i| \log^2 n)$.

3 Succinct Index

We maintain a WT over each pattern. This occupies $n \log \sigma + o(n \log \sigma)$ bits (refer to Fact 3). We design our index by classifying the patterns into *long* and *short* based on a parameter Δ to be defined later. For short patterns (having length less than Δ), we create a compacted trie and use a rather brute-force approach. On the other hand, reporting the occurrences of long patterns (having length at least Δ) requires more sophisticated techniques. The set of occurrences of long patterns and short patterns are mutually disjoint, and are handled separately as indicated in the following lemmas.

▶ **Lemma 6.** *Let $\mathcal{P}$ be a dictionary consisting of d long patterns. By maintaining each pattern in a WT and a data structure occupying $\mathcal{O}(\frac{n}{\Delta} \log n)$ bits, we can report all occ_ℓ occurrences in $\mathcal{O}(|T|(\Delta \log \sigma + \log^2 n) + occ_\ell)$ time. Also, a long pattern P_i can be inserted in amortized $\mathcal{O}(\frac{|P_i|}{\Delta}(\Delta \log n + \log^2 n))$ time and deleted in amortized $\mathcal{O}(\frac{|P_i|}{\Delta}(\Delta \log \sigma + \log^2 n))$ time.*

▶ **Lemma 7.** *Let $\mathcal{P}$ be a dictionary consisting of d short patterns. By maintaining each pattern in a WT and a data structure occupying $\mathcal{O}(d \log n)$ bits, we can report all occ_s*

[4] Observe that there might still be a suffix link from a node v' pointing to the position corresponding to v on this new edge because truncating the first character of $\mathsf{prev}(v')$ may lead to merging of two outgoing edges of v'. Our motivation for implicit representation of suffix links is due to this property of PST.

occurrences in $\mathcal{O}(|T|(\Delta \log \sigma + \log d) + occ_s)$ *time. Also, a short pattern* P_i *can be inserted or deleted, both in amortized* $\mathcal{O}(|P_i| \log \sigma + \log d)$ *time.*

Theorem 2 is immediate by choosing $\Delta = \lceil \log n \log_\sigma n \rceil$, where $\epsilon > 0$ is an arbitrarily small constant. We proceed to prove the above two lemmas.

In what follows, we will assume the total length n of the patterns remains reasonably stable. This assumption is natural as we can use the following strategy of Overmars [28], or its subsequent improvement by Mäkinen and Navarro [23]. Roughly speaking, apart from maintaining the wavelet trees over the patterns, we will maintain three copies of the remaining component of the data structures in Lemmas 6 and 7. Specifically, apart from the data structures due to the choice of Δ above, we will keep two more copies, one for $\Delta = \Delta_{-1}$, and the other for $\Delta = \Delta_1$, where $\Delta_k = \lceil \log(2^k n) \log_\sigma (2^k n) \rceil$. Whenever the total length of the pattern doubles, we discard the structure for Δ_{-1}, and start building another structure by considering $\Delta = \Delta_2$. Likewise, when the total length halves, we discard the structure for Δ_1, and start building a structure by considering $\Delta = \Delta_{-2}$. Amortized per operation cost is $\mathcal{O}(1)$. Whenever, a pattern is inserted or deleted, we will modify all three copies simultaneously; a search query can be answered using any one of the copies. Clearly, the space-and-time bounds claimed in Lemmas 6 and 7 are not affected.

3.1 Long Patterns (Proof of Lemma 6)

For a string S and Δ, we use $\mathsf{head}(S)$ to denote the largest prefix of S whose length is a multiple of Δ and $\mathsf{tail}(S)$ is the remaining (possibly empty) suffix of S. We begin by obtaining $\mathsf{prev}(\mathsf{head}(P_i))$ for every $P_i \in \mathcal{P}$. We encode $\mathsf{tail}(P_i)$ from left to right using the same encoding that was used for $\mathsf{head}(P_i)$. More specifically, the desired encoding of the jth character in the tail is given by $\mathsf{prev}(P_i)[|\mathsf{head}(P_i)| + j]$. Then two equal-length strings S and S' are a p-match iff (i) $\mathsf{prev}(\mathsf{head}(S)) = \mathsf{prev}(\mathsf{head}(S'))$, and (ii) the encoded tails (as described here) of both S and S' are equal.

The Index: Note that in this case the number of patterns $d \leq n/\Delta$. We begin by sampling suffixes of each pattern head with sampling factor Δ. Specifically, for each pattern P_i, we obtain $\mathsf{prev}(P_i[k, |\mathsf{head}(P_i)|])$ for the suffixes starting at $k = 1, 1 + \Delta, 1 + 2\Delta, \ldots$. Starting from left, we group every Δ characters of these encoded suffixes. Let Σ' be an alphabet such that each character in Σ' corresponds to such a Δ-length substring. Replace the Δ-length substring by the corresponding character from Σ'. Create a generalized suffix tree $\mathcal{T}_{head}$ for all these suffixes of all the patterns. (If the pattern length is not a multiple of Δ, then we ignore its tail.) As in Section 2, we will append each condensed suffix with the special characters $\$_i$ and $\#_i$. Note that $\mathcal{T}_{head}$ has $\mathcal{O}(n/\Delta)$ nodes. Therefore, $\sum_u \delta(u) = \mathcal{O}(n/\Delta)$, where $\delta(u)$ is the number of outgoing edges of a node u. At each node u, we maintain $\mathsf{strDepth}(u)$, which is necessarily a multiple of Δ. Also, for each leaf ℓ, we maintain the pointers which will be used to find suffix links implicitly. The total space required is $\mathcal{O}((n/\Delta) \log n)$ bits.

Now, let us concentrate on how to navigate to a particular child of a node u. Consider all the outgoing edges of u. Create a compacted trie $\mathcal{T}_{head}(u)$ by treating the labels of these edges mapped to their corresponding Δ-length string. Note that each leaf in $\mathcal{T}_{head}(u)$ corresponds to a child of u in $\mathcal{T}_{head}$. Also, each edge in $\mathcal{T}_{head}(u)$ is labeled by a prev-encoded substring of a pattern P_i, and each outgoing edge of a node begins with a unique character from such a substring. As in the case of the linear space index, (i) at each edge of $\mathcal{T}_{head}(u)$ maintain the start and end pointers, and (ii) at each node maintain a dynamic perfect hashtable for navigating to the correct child based on the first (encoded) character of the edge. Since

the number of nodes in $\mathcal{T}_{head}(u)$ is at most $2\delta(u)$, the total space needed to maintain this information over all nodes in $\mathcal{T}_{head}$ is $\mathcal{O}(\sum_u \delta(u) \log n) = \mathcal{O}((n/\Delta) \log n)$ bits.

Now, we focus on the tail of each pattern. Consider a pattern P_i. First, we obtain the encoded tail of P_i (as described in the beginning of this section). Create two copies of the resultant tail, each of which is obtained by appending the s-characters $\$_i$ and $\#_i$. Locate the (distinct) node u in $\mathcal{T}_{head}$ such that $\mathsf{prev}(u)$ is same as $\mathsf{prev}(\mathsf{head}(P_i))$. Note that u is defined, and we call it the *head-node* of P_i. Consider all patterns which have the same head-node u. Create a compacted trie for the encoded tails of all those patterns, and let u be the root of that trie. We call this the *tail-trie* of u, and denote it by $\mathcal{T}_{tail}(u)$. The parent of each leaf in $\mathcal{T}_{tail}(u)$ corresponds to a pattern, say P_j, in the dictionary. We mark all such nodes in $\mathcal{T}_{tail}(u)$, and label them with the corresponding pattern index j. If there is a pattern P_j with an empty tail, then the corresponding tail-trie contains the head-node u, which is marked, and two leaves labeled by $\$_j$ and $\#_j$. The space occupied by each node for marking and labeling is $\mathcal{O}(\log n)$ bits. Each edge in $\mathcal{T}_{tail}(u)$ is labeled by a substring (of length less than Δ) of the encoded tail of a pattern. As in case of head tries, we maintain the two pointers on the edge to the corresponding pattern, and a perfect dynamic hash table to navigate to the correct child based on the first (encoded) character of the edge. This occupies $\mathcal{O}(\log n)$ bits for each node and edge. Since there are d patterns, the number of nodes and edges in all tail-tries combined is $\mathcal{O}(d)$. Since $d \leq n/\Delta$, the total space occupied for maintaining all tail-tries is $\mathcal{O}((n/\Delta) \log n)$ bits.

Denote the resultant trie by $\mathcal{T}_{long}$. We pre-process the head-trie with the data structure of Fact 4. Likewise, each tail-trie is pre-processed with the data structures of Facts 4 and 5. In summary, the total space occupied by $\mathcal{T}_{long}$ is $\mathcal{O}((n/\Delta) \log n)$ bits.

Reporting Occurrences: Starting from the position $j = 1$, we obtain $\mathsf{prev}(T[j, |T|])$ in $\mathcal{O}(|T| \log \sigma)$ time. Use it to traverse the trie $\mathcal{T}_{long}$ from the root. Each p-character labeling the edge of $\mathcal{T}_{long}$ can be properly encoded in $\mathcal{O}(\log \sigma)$ time as described in Section 2.1. Suppose, we have traversed up to node u in $\mathcal{T}_{head}$ and the character j' in $\mathsf{prev}(T)[j, |T|]$, where $j' - j + 1 = 0 \mod \Delta$. If $\mathcal{T}_{tail}(u)$ is not empty, then use the less than Δ characters of $\mathsf{prev}(T[j, |T|)$ starting from $j' + 1$ to traverse the tail trie, until we find a mismatch or reach a leaf. The time required is $\mathcal{O}(\Delta \log \sigma)$. Now, we use the marked ancestor data structure to report all occurrences starting at j corresponding to those patterns having head-node u. The time required is $\mathcal{O}(\log d + occ_{j,u})$ time. After this, by using the first Δ-characters of $\mathsf{prev}(T[j, |T|)$ starting from $j' + 1$, we have to select an edge (u, v) in $\mathcal{T}_{head}$. This is easily achieved in $\mathcal{O}(\Delta \log \sigma)$ time by using the navigation trie $\mathcal{T}_{head}(u)$ as follows. If we are at a node in $\mathcal{T}_{head}(u)$, then use the next character to find the correct edge using the hash table; otherwise, simply use the edge pointers to encode the next character of the edge, and match it with the next encoded character of T. In case we were no longer able to reach a leaf in $\mathcal{T}_{head}(u)$, then we have the following two scenarios. If no match was found with the first Δ characters starting from u, then we take the suffix link of u. Otherwise, we are necessarily on an edge to a leaf in $\mathcal{T}_{head}(u)$; in this case, take the suffix link of the node v in $\mathcal{T}_{head}$ corresponding to this leaf. In either case, we truncate Δ characters of $\mathsf{prev}(T)$ starting from j. As described in Section 2, the suffix link is simulated by the implicit suffix link i.e., by finding a leaf under u or v in the head-trie, and then using the leaf pointer and a wla query. Following this, the correct position to start matching is located in $\mathcal{O}(\Delta \log \sigma)$ time using the navigation trie of the node returned by the wla query. As before, locating a leaf requires $\mathcal{O}(\log n)$ time and a wla query takes $\mathcal{O}(\log^2 n)$ time. The number of times we have to select a proper edge, traverse a tail trie, or follow a suffix link, are all bounded by $\mathcal{O}(|T|/\Delta)$.

At the end of this process, for $j = 1$, we have reported occurrences of all patterns which start at a position of the form $j, j + \Delta, j + 2\Delta, \ldots$. The time required to find the loci and traversing the tail tries is $\mathcal{O}(|T| \log \sigma + \frac{|T|}{\Delta}(\Delta \log \sigma + \log^2 n))$. The time required to report the occurrences is $\mathcal{O}(\frac{|T|}{\Delta} \log d + occ_j)$. By repeating with $j = 2, 3, \ldots, \Delta$, all occ_ℓ occurrences of long patterns are reported in $\mathcal{O}(|T|(\Delta \log \sigma + \log^2 n) + occ_\ell)$ time.

Handling Updates: First we construct the head-trie when a pattern P_i is inserted. We begin by using Kosaraju's algorithm to construct a PST for P_i, and then find the locus of $\mathsf{prev}(P_i)$ in $\mathcal{T}_{head}$; this will take $\mathcal{O}(|P_i| \log \sigma)$ time. Now, we will create *actual* nodes in $\mathcal{T}_{head}$ only for those suffixes which start at a location of the form $k = 1, 1 + \Delta, 1 + 2\Delta, \ldots$. For other suffixes, we will create *dummy* nodes in $\mathcal{T}_{head}$ so as to perform the suffix link operations correctly. Specifically, suppose we have inserted an actual leaf ℓ_j for the suffix starting at $1 + j\Delta$. Subsequently, we will construct dummy leaves for the suffixes starting at $j' \in [2 + j\Delta, (j + 1)\Delta]$. Once, the actual leaf ℓ_{j+1} for the suffix starting $1 + (j + 1)\Delta$ is inserted, we will add the suffix link from ℓ_j to ℓ_{j+1}, and delete the intermediate dummy nodes. However, now we need to find the correct location of a (possibly new) node u in $\mathcal{T}_{head}$ such that $\mathsf{prev}(u)$ is the LCP of the suffixes corresponding to ℓ_j and ℓ_{j+1}, which is divisible by Δ. This can be found in $\mathcal{O}(\log^2 n)$ time using wla-queries on $\mathcal{T}_{head}$ by first finding the LCP using the PST of P_i. Each actual node insertion will take $\mathcal{O}(\Delta \log \sigma)$ amortized time for updating the structure of the navigation trie and the associated hash table, $\mathcal{O}(\log^2 n)$ time for wla-queries, and $\mathcal{O}(\log n)$ time for updating the data structure of Fact 4; the number of these operations is $\mathcal{O}(|P_i|/\Delta)$. We will make $\mathcal{O}(|P_i|)$ accesses for updating and querying the data structure of Fact 4 for inserting and deleting dummy nodes, each requiring $\mathcal{O}(\log n)$ time. Thus, the time needed to update $\mathcal{T}_{head}$ is $\mathcal{O}(|P_i| \log n + \frac{|P_i|}{\Delta} \log^2 n)$.

Modifying the tail-trie is much simpler. We traverse it with the encoded $\mathsf{tail}(P_i)$ starting from the head node of P_i until no more traversal is possible. Then, simply add the desired nodes and edges. Modify the data structures of Facts 4 and 5 accordingly (the latter for including a new marked node). Also, modify the hash table in $\mathcal{O}(1)$ amortized time per update. The time required is $\mathcal{O}(\Delta \log \sigma + \log d)$.

Since $|P_i| \geq \Delta$, inserting P_i needs amortized $\mathcal{O}(|P_i| \log n + \frac{|P_i|}{\Delta} \log^2 n)$ time.

In case of deletion, first we find the head-node of the pattern P_i. Then, use the encoded $\mathsf{tail}(P_i)$ to traverse the tail trie, unmark the node labeled by P_i, and delete its two children (leaves) labeled with $\$_i$ and $\#_i$. Also, the parent u of these leaves are deleted in case u is a leaf. The parent v of u is modified (if it has a single child) as in the case of linear index. If u is not a leaf, then it is treated in the same way if it has a single child, or else is left unmodified. To modify the edge pointers, find the lowest edge e that was traversed, but was not deleted. Then all the desired edges above e on the traversed path can be renamed by using any leaf corresponding to a pattern $P_{i'}$ under e. The hash table entries are deleted accordingly. The time required is $\mathcal{O}(|P_i| \log \sigma + \log d)$.

Deletion in the head trie is achieved by first locating the loci of all the condensed heads in time $\mathcal{O}(\frac{|P_i|}{\Delta}(\Delta \log \sigma + \log^2 n))$. Then, modify the edge labels in the navigation trie, and the adjoining hash table entries. Also, collapse nodes with a single child into an edge. The number of such operations is $\mathcal{O}(\frac{|P_i|}{\Delta})$, each requiring $\mathcal{O}(\Delta \log \sigma + \log n)$ time.

Since $|P_i| \geq \Delta$, deleting P_i needs amortized $\mathcal{O}(\frac{|P_i|}{\Delta}(\Delta \log \sigma + \log^2 n))$ time.

3.2 Short Patterns (Proof of Lemma 7)

Processing short patterns is similar to that for tail-tries. We create a compacted trie $\mathcal{T}_{short}$ for the strings $\mathsf{prev}(P_i)\$_i$ and $\mathsf{prev}(P_i)\#_i$. As in case of tail tries, we maintain the two pointers

for each edge, and also maintain the first (encoded) character of the edge in a dynamic perfect hashtable. Mark a node u if there is a pattern P_i such that $\mathsf{prev}(v) = \mathsf{prev}(P_i)$. Finally, we process the trie with the data structures of Facts 4 and 5. Since the number of patterns is at most d, the number of nodes in the trie is $\mathcal{O}(d)$, and the total space is $\mathcal{O}(d \log n)$ bits.

To find the occurrences of short patterns, first obtain $\mathsf{prev}(T)$ in $\mathcal{O}(|T| \log \sigma)$ time. Starting from $j = 1$, use $\mathsf{prev}(T)[j, \Delta - 1]$ to traverse the trie $\mathcal{T}_{short}$ until no more traversal is possible. The time required is $\mathcal{O}(\Delta \log \sigma)$. Now, starting the from the last encountered node, we report all occ_j occurrences starting at j in $\mathcal{O}(\log d + occ_j)$ time. We repeat the process for $j = 2, 3, \ldots, |T|$. The time required to report all occ_s occurrences is $\mathcal{O}(|T|(\Delta \log \sigma + \log d) + occ_s)$.

Insertion and deletion is similar as in the case of tail tries. Specifically, use $\mathsf{prev}(P_i)$ to traverse $\mathcal{T}_{short}$, and then insert/delete nodes accordingly. The hash table for navigation and the edge labels are also updated. Summarizing, both insertion and deletion needs amortized $\mathcal{O}(|P_i| \log \sigma + \log d)$ time.

4 Semi-Dynamic Dictionary

From the discussions in the previous section, closely observe that the $(\log^2 n)$-factor in the query complexity is due to the wla queries. To improve this, we present Fact 8.

▶ **Fact 8** ([20]). *Given a min-heap with m weighted nodes, with weights in $[1, m]$. We can build an $\mathcal{O}(m \log m)$-bit data structure in $\mathcal{O}(m)$ time to support the following operations.*

- insert *a weighted node maintaining the heap property in amortized $\mathcal{O}(\log \log m)$ time.*
- *report* $\mathsf{wla}(u, W)$ *in worst-case $\mathcal{O}(\log \log m)$ time.*

In conjunction with the techniques previously presented, for the semi-dynamic case, where only search and insert operations are supported, we obtain the following couple of corollaries to Theorems 1 and 2. The bound in Corollary 10 is attained by choosing $\Delta = \lceil \log^{\epsilon} n \log_{\sigma} n \rceil$ in Lemmas 6 and 7, where $\epsilon > 0$ is an arbitrarily small constant.

▶ **Corollary 9.** *By maintaining an $\mathcal{O}(n \log n)$-bit index, we can answer* $\mathsf{search}(T)$ *in $\mathcal{O}(|T| \log n + occ)$ time, and* $\mathsf{insert}(P_i)$ *in amortized $\mathcal{O}(|P_i| \log n)$ time.*

▶ **Corollary 10.** *By maintaining an $(1 + o(1))n \log \sigma + \mathcal{O}(d \log n)$-bit index, we can answer* $\mathsf{search}(T)$ *in $\mathcal{O}(|T| \log^{1+\epsilon} n + occ)$ time, and* $\mathsf{insert}(P_i)$ *in amortized $\mathcal{O}(|P_i| \log n)$ time.*

References

1. Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975. doi:10.1145/360825.360855.
2. Stephen Alstrup, Thore Husfeldt, and Theis Rauhe. Marked ancestor problems. In *39th Annual Symposium on Foundations of Computer Science, FOCS'98, November 8-11, 1998, Palo Alto, California, USA*, pages 534–544, 1998. doi:10.1109/SFCS.1998.743504.
3. Amihood Amir, Martin Farach, Zvi Galil, Raffaele Giancarlo, and Kunsoo Park. Dynamic dictionary matching. *J. Comput. Syst. Sci.*, 49(2):208–222, 1994. doi:10.1016/S0022-0000(05)80047-9.
4. Amihood Amir, Martin Farach, Ramana M. Idury, Johannes A. La Poutré, and Alejandro A. Schäffer. Improved dynamic dictionary matching. *Inf. Comput.*, 119(2):258–282, 1995. doi:10.1006/inco.1995.1090.
5. Brenda S. Baker. A theory of parameterized pattern matching: algorithms and applications. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 71–80, 1993. doi:10.1145/167088.167115.

6 Djamal Belazzougui. Succinct dictionary matching with no slowdown. In *Combinatorial Pattern Matching, 21st Annual Symposium, CPM 2010, New York, NY, USA, June 21-23, 2010. Proceedings*, pages 88–100, 2010. doi:10.1007/978-3-642-13509-5_9.

7 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, pages 88–94, 2000. doi:10.1007/10719839_9.

8 Sudip Biswas, Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. Forbidden extension queries. In *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, pages 320–335, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.320.

9 Ho-Leung Chan, Wing-Kai Hon, Tak Wah Lam, and Kunihiko Sadakane. Dynamic dictionary matching and compressed suffix trees. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 13–22, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070436.

10 Paul F. Dietz and Daniel Dominic Sleator. Two algorithms for maintaining order in a list. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 365–372, 1987. doi:10.1145/28395.28434.

11 Martin Dietzfelbinger, Anna R. Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, and Robert Endre Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM J. Comput.*, 23(4):738–761, 1994. doi:10.1137/S0097539791194094.

12 Guy Feigenblat, Ely Porat, and Ariel Shiftan. An improved query time for succinct dynamic dictionary matching. In *Combinatorial Pattern Matching – 25th Annual Symposium, CPM 2014, Moscow, Russia, June 16-18, 2014. Proceedings*, pages 120–129, 2014. doi:10.1007/978-3-319-07566-2_13.

13 Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 390–398, 2000. doi:10.1109/SFCS.2000.892127.

14 Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. Succinct non-overlapping indexing. In *Combinatorial Pattern Matching – 26th Annual Symposium, CPM 2015, Ischia Island, Italy, June 29 – July 1, 2015, Proceedings*, pages 185–195, 2015. doi:10.1007/978-3-319-19929-0_16.

15 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 841–850, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644250.

16 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching (extended abstract). In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 397–406, 2000. doi:10.1145/335305.335351.

17 Wing-Kai Hon, Tak Wah Lam, Rahul Shah, Siu-Lung Tam, and Jeffrey Scott Vitter. Compressed index for dictionary matching. In *2008 Data Compression Conference (DCC 2008), 25-27 March 2008, Snowbird, UT, USA*, pages 23–32, 2008. doi:10.1109/DCC.2008.62.

18 Wing-Kai Hon, Tak Wah Lam, Rahul Shah, Siu-Lung Tam, and Jeffrey Scott Vitter. Succinct index for dynamic dictionary matching. In *Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings*, pages 1034–1043, 2009. doi:10.1007/978-3-642-10631-6_104.

19 Ramana M. Idury and Alejandro A. Schäffer. Multiple matching of parameterized patterns. In *Combinatorial Pattern Matching, 5th Annual Symposium, CPM 94, Asilo-*

mar, California, USA, June 5-8, 1994, Proceedings, pages 226–239, 1994. doi:10.1007/3-540-58094-8_20.

20 Tsvi Kopelowitz and Moshe Lewenstein. Dynamic weighted ancestors. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 565–574, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283444.

21 S. Rao Kosaraju. Faster algorithms for the construction of parameterized suffix trees (preliminary version). In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, 23-25 October 1995*, pages 631–637, 1995. doi:10.1109/SFCS.1995.492664.

22 Moshe Lewenstein. Parameterized pattern matching. In *Encyclopedia of Algorithms*, 2015. doi:10.1007/978-3-642-27848-8_282-2.

23 Veli Mäkinen and Gonzalo Navarro. Dynamic entropy-compressed sequences and full-text indexes. *ACM Transactions on Algorithms*, 4(3), 2008. doi:10.1145/1367064.1367072.

24 Edward M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, 1976. doi:10.1145/321941.321946.

25 J. Ian Munro, Gonzalo Navarro, Jesper Sindahl Nielsen, Rahul Shah, and Sharma V. Thankachan. Top- k term-proximity in succinct space. In *Algorithms and Computation – 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014, Proceedings*, pages 169–180, 2014. doi:10.1007/978-3-319-13075-0_14.

26 Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1), 2007. doi:10.1145/1216370.1216372.

27 Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *ACM Trans. Algorithms*, 10(3):16:1–16:39, 2014. doi:10.1145/2601073.

28 Mark H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes in Computer Science*. Springer, 1983. doi:10.1007/BFb0014927.

29 Kunihiko Sadakane. Compressed text databases with efficient query algorithms based on the compressed suffix array. In *Algorithms and Computation, 11th International Conference, ISAAC 2000, Taipei, Taiwan, December 18-20, 2000, Proceedings*, pages 410–421, 2000. doi:10.1007/3-540-40996-3_35.

30 Dekel Tsur. Top-k document retrieval in optimal space. *Inf. Process. Lett.*, 113(12):440–443, 2013. doi:10.1016/j.ipl.2013.03.012.

Efficient Summing over Sliding Windows

Ran Ben Basat[1], Gil Einziger[2], Roy Friedman[3], and Yaron Kassner[4]

1 **Department of Computer Science, Technion, Haifa, Israel**
`sran@cs.technion.ac.il`
2 **Department of Computer Science, Technion, Haifa, Israel**
`gilga@cs.technion.ac.il`
3 **Department of Computer Science, Technion, Haifa, Israel**
`roy@cs.technion.ac.il`
4 **Department of Computer Science, Technion, Haifa, Israel**
`kassnery@cs.technion.ac.il`

Abstract

This paper considers the problem of maintaining statistic aggregates over the last W elements of a data stream. First, the problem of counting the number of 1's in the last W bits of a binary stream is considered. A lower bound of $\Omega(\frac{1}{\epsilon}+\log W)$ memory bits for $W\epsilon$-*additive approximations* is derived. This is followed by an algorithm whose memory consumption is $O(\frac{1}{\epsilon}+\log W)$ bits, indicating that the algorithm is optimal and that the bound is tight. Next, the more general problem of maintaining a *sum* of the last W integers, each in the range of $\{0,1,\ldots,R\}$, is addressed. The paper shows that approximating the sum within an *additive error* of $RW\epsilon$ can also be done using $\Theta(\frac{1}{\epsilon}+\log W)$ bits for $\epsilon=\Omega\left(\frac{1}{W}\right)$. For $\epsilon=o\left(\frac{1}{W}\right)$, we present a *succinct* algorithm which uses $\mathcal{B}\cdot(1+o(1))$ bits, where $\mathcal{B}=\Theta\left(W\log\left(\frac{1}{W\epsilon}\right)\right)$ is the derived lower bound. We show that all lower bounds generalize to randomized algorithms as well. All algorithms process new elements and answer queries in $O(1)$ worst-case time.

1998 ACM Subject Classification E.1 [Data Structures] Lists, stacks, and queues

Keywords and phrases Streaming, Statistics, Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.11

1 Introduction

Background

The ability to process and maintain statistics about large streams of data is useful in many domains, such as security, networking, sensor networks, economics, business intelligence, etc. Since the data may change considerably over time, there is often a need to keep the statistics only with respect to some window of the last W elements at any given point. A naive solution to this problem is to keep the W most recent elements, add an element to the statistic when it arrives, and subtract it when it leaves the window. Yet, when the window of interest is large, which is often the case when data arrive at high rate, the required memory overhead may become a performance bottleneck.

Though it may be tempting to think that RAM memory is cheap, a closer look indicates that there are still performance benefits in maintaining small data structures. For example, hardware devices such as network switches prefer to store important data in the faster and scarcely available SRAM than in DRAM. This is in order to keep up with the ever increasing line-speed of modern networks. Similarly, on a CPU, caches provide much faster performance

than DRAM memory. Thus, small data structures that fit inside a single cache line and can possibly be pinned there are likely to result in much faster performance than a solution that spans multiple lines that are less likely to be constantly maintained in the cache.

A well known method to conserve space is to approximate the statistics. Basic-Counting is one of the most basic textbook examples of such approximated stream processing problems [12]. In this problem, one is required to keep track of the number of 1's in a stream of binary bits. A $(1+\epsilon)$-*multiplicative approximation* algorithm for this problem using $O\left(\frac{1}{\epsilon}\log^2 W\epsilon\right)$ bits was shown in [12]. This solution works with amortized $O(1)$ time, but its worst case time complexity is $O(\log W)$.

A more practical related problem is Basic-Summing, in which the goal is to maintain the sum of the last W elements. When all elements are non-negative integers in the range $[R+1]=\{0,1,\ldots,R\}$, the work in [12] naturally extends to provide a $(1+\epsilon)$-multiplicative approximation of this problem using $O\left(\frac{1}{\epsilon}\cdot\left(\log^2 W+\log R\cdot(\log W+\log\log R)\right)\right)$ bits. The amortized time complexity becomes $O(\frac{\log R}{\log W})$ and the worst case is $O(\log W+\log R)$.

Our Contributions

In this paper, we explore the benefits of changing the approximation guarantee from *multiplicative* to *additive*. With a multiplicative approximation, the result returned can be different from the correct one by at most a multiplicative factor, e.g., 5%. On the other hand, in an additive approximation, the absolute error is bounded, e.g., a deviation of up to ± 5. When the expected number of ones in a stream is small, multiplicative approximation is more appealing, since its absolute error is small. However, in this case, an accurate (sparse) representation can be even more space efficient than the multiplicative approximation. On the other hand, when many ones are expected, additive approximation gives similar outcomes to multiplicative approximation. Furthermore, the potential space saving becomes significant in this case, motivating our exploration.

Our initial contribution is a formally proved memory lower bound of $\Omega(\frac{1}{\epsilon}+\log W)$ for $W\epsilon$-additive approximations for the Basic-Counting problem.

Our second contribution is a space optimal algorithm providing a $W\epsilon$-*additive approximation* for the Basic-Counting problem. It consumes $O(\frac{1}{\epsilon}+\log W)$ memory bits with a worst case time complexity of $O(1)$, matching the lower bound.

Next, we explore the more general Basic-Summing problem. Here, the results are split based on the value of ϵ. Specifically, our third contribution is an (asymptotically) space optimal algorithm providing an $RW\epsilon$-additive approximation for the Basic-Summing problem when $\epsilon^{-1}\leq 2W\left(1-\frac{1}{\log W}\right)$.[1] It uses $O(\frac{1}{\epsilon}+\log W)$ memory bits and has $O(1)$ worst case time complexity. For other values of ϵ, we show a lower bound of $\Omega(W\log\left(\frac{1}{W\epsilon}\right))$ and a corresponding algorithm requiring $O\left(W\log\left(\frac{1}{2W\epsilon}+1\right)\right)$ memory bits with $O(1)$ worst case time complexity. Furthermore, we show that this algorithm is succinct for $\epsilon=o(W^{-1})$, i.e. its space requirement is only (1+o(1)) times the lower bound.

To get a feel for the applicability of these results, consider for example an algorithmic trader that makes transactions based on a moving average of the gold price. He samples the spot price once every millisecond, and wishes to approximate the average price for the last hour, i.e., $W=3.6\cdot 10^6$ samples. The current price is around \$1200, and with a standard deviation of \$10, he safely assumes the price is bounded by $R\triangleq 1500$. The trader is willing to withstand an error of 0.1%, which is approximately \$1.2. Our algorithm provides a

[1] In this paper, the logarithms are of base 2 and the o(1) notation is for $W\to\infty$.

$WR\epsilon_A$ (the 'A' stands for *Additive*) additive-approximation using $\left(\frac{1}{2\epsilon_A} + 2\log W\right)(1 + o(1))$ memory bits, while the algorithm by Datar et al. [12] computes a multiplicative $(1 + \epsilon_M)$ (the 'M' stands for *Multiplicative*) approximation using $\left\lceil \frac{1}{2\epsilon_M} + 1 \right\rceil \lceil \log(2WR\epsilon_M + 1) + 1 \rceil$ buckets of size $\lceil \log W + \log(\log W + \log R) \rceil$ bits each. Using our algorithm, the trader sets $\epsilon_A = R^{-1} = \frac{1}{1500}$, which guarantees that as long as the price of gold stays above \$1000, the error remains lower than required. The multiplicative approximation algorithm requires setting $\epsilon_M = 0.1\%$, and uses $501 \cdot \lceil \log(1080001) + 1 \rceil = 12525$ buckets of size 27 bits each and about $41KB$ overall. In comparison, our algorithm with the parameters above requires only about 100 bytes.

Another useful application for our algorithm is counting within a *fixed* additive error. The straight-forward algorithm for solving Basic-Counting uses a W-bits array which stores the entire window, replacing the oldest recorded bit with a new one whenever such arrives. Assume a ± 5 error is allowed. Using the multiplicative-approximation algorithms, one has to set $\epsilon_M = \frac{5}{W}$, which requires more than W bits, worse than exact counting. In contrast, setting $\epsilon_A = \frac{5}{W}$ for our algorithm reduces the memory consumption of the exact solution by nearly 90%.

In summary, we show that additive approximations offer significant space reduction opportunities. They can be obtained with a constant worst case time complexity, which is important in real-time and time sensitive applications.

2 Related Work

In [12], Datar et al. first presented the problem of counting the number of 1's in a sliding window of size W over a binary stream, and its generalization to summing a window over a stream of integers in the range $\{0, 1, \ldots, R\}$. They have introduced a data structure called *exponential histogram* (*EH*). *EH* is a time-stamp based structure that partitions the stream into *buckets*, saving the time elapsed since the last 1 in the bucket was seen. Using *EH*, they have derived a space-optimal algorithm for approximating Basic-Summing within a multiplicative-factor of $(1+\epsilon)$, which uses $O\left(\frac{1}{\epsilon}\log^2 W + \log R \cdot (\log W + \log\log R)\right)$ memory bits. The structure allows estimating a class of aggregate functions such as counting, summing and computing the ℓ_1 and ℓ_2 norms of a sliding window in a stream containing integers. The exponential histogram technique was later expanded [3] to support computation of additional functions such as k-median and variance. Gibbons and Tirthapura [13] presented a different structure called *waves*, which improved the worst-case runtime of processing a new element to a constant, keeping space requirement comparable when $R = poly(W)$. Braverman and Ostrowsky [7] defined *smooth histogram*, a generalization of the exponential histogram, which allowed estimation of a wider class of aggregate functions and improved previous results for several functions such as l_p norms and frequency moments. Lee and Ting [15] presented an improved algorithm, requiring less space if a $(1 + \epsilon)$ approximation is guaranteed only when the ones consist of a significant fraction of the window. They also presented the λ counter [16] that counts bits over a sliding window as part of a frequent items algorithm. Our design is more space efficient as it requires $O(\frac{1}{\varepsilon} + log(n))$ bits instead of $O(\frac{1}{\varepsilon} \cdot log(n))$ bits.

In [8], Cohen and Strauss considered a generalization of the bit-counting problem on a sliding window for computing a weighted sum for some decay function, such that the more recent bits have higher weights. Cormode and Yi [9] solved bit counting in a distributed setting with optimal communication between nodes. Table 1 and Table 2 summarize previous works on the Basic-Counting and Basic-Summing problems and compare them to our own algorithms.

Basic-Counting	Approximation Guarantee	Memory Requirement	Amortized Addition Time	Worst-Case Addition Time	Maximal Additive Error
Datar et al. [12]	$(1+\epsilon)$-Multiplicative	$O\left(\frac{1}{\epsilon}\log^2 W\epsilon\right)$	$O(1)$	$O(\log W)$	$W\epsilon$
Gibbons and Tirthapura [13]	$(1+\epsilon)$-Multiplicative	$O\left(\frac{1}{\epsilon}\log^2 W\epsilon\right)$	$O(1)$	$O(1)$	$W\epsilon$
Lee and Ting [15]	$(1+\epsilon)$-Multiplicative, whenever there are at least θW 1-bits	$O\left(\frac{1}{\epsilon}\log^2\frac{1}{\theta} + \log W\theta\epsilon\right)$	$O(1)$	$O(1)$	$W\epsilon$
This Paper	$W\epsilon$-Additive	$O\left(\frac{1}{\epsilon}+\log W\epsilon\right)$	$O(1)$	$O(1)$	$W\epsilon$

Table 1 Comparison of Basic-Counting Algorithms.

Basic-Summing	Approximation Guarantee	Memory Requirement	Amortized Addition Time	Worst-Case Addition Time	Maximal Additive Error
Datar et al. [12]	$(1+\epsilon)$-Multiplicative	$O\left(\frac{1}{\epsilon}\left(\log^2 W + \log R\log W + \log R\log\log R\right)\right)$	$O\left(\frac{\log R}{\log W}\right)$	$O(\log W + \log R)$	$RW\epsilon$
Gibbons and Tirthapura [13]	$(1+\epsilon)$-Multiplicative	$O\left(\frac{1}{\epsilon}\left(\log W+\log R\right)^2\right)$	$O(1)$	$O(1)$	$RW\epsilon$
This Paper	$RW\epsilon$-Additive for $\epsilon \geq \frac{1}{2W}$	$O\left(\frac{1}{\epsilon}+\log W\right)$	$O(1)$	$O(1)$	$RW\epsilon$
	$RW\epsilon$-Additive for $\epsilon \leq \frac{1}{2W}$	$O\left(W\cdot\log\left(\frac{1}{W\epsilon}\right)\right)$			

Table 2 Comparison of Basic-Summing Algorithms.

Extensive studies were conducted on many other streaming problems over sliding windows such as Top-K [18, 20], Top-K tuples [22], Quantiles [2], heavy hitters [5, 6, 14], distinct items [24], duplicates [21], Longest Increasing Subsequences [7, 1], Bloom filters [17, 19], graph problems [10, 11] and more.

3 Basic-Counting Problem

▶ **Definition 1** (Approximation). Given a value V and a constant ϵ, we say that $\widehat{V}$ is an ϵ-*multiplicative approximation* of V if $|V-\widehat{V}| < \epsilon V$. We say that $\widehat{V}$ is an ϵ-*additive approximation* of V if $|V-\widehat{V}| < \epsilon$.

▶ **Definition 2** (Basic-Counting). Given a stream of bits and a parameter W, maintain the number of 1's in the last W bits of the stream. Denote this number by C^W.

3.1 Lower Bound

We now show lower bounds for the memory requirement for approximating Basic-Counting.

▶ **Lemma 3.** *For any ϵ and W, any deterministic algorithm that provides a $W\epsilon$-additive approximation for* Basic-Counting *requires at least* $\left\lfloor \frac{W}{\lfloor 2W\epsilon+1 \rfloor} \right\rfloor \geq \left\lfloor \frac{1}{2\epsilon+W^{-1}} \right\rfloor$ *bits.*

Proof. Denote $z \triangleq \left\lfloor \frac{W}{\lfloor 2W\epsilon+1 \rfloor} \right\rfloor$. We prove the lemma by showing 2^z arrangements that must lead to different configurations. Consider the language of all concatenations of z blocks of size $\lfloor 2W\epsilon+1 \rfloor$, such that each block consists of only ones or only zeros:

$$L_{W,\epsilon} = \{w_0 w_1 \cdots w_{z-1} \mid \forall j \in [z] : w_j = 0^{\lfloor 2W\epsilon+1 \rfloor} \vee w_j = 1^{\lfloor 2W\epsilon+1 \rfloor}\}$$

Assume, by way of contradiction, that two different words

$$s^1 = w_0^1 w_1^1 \cdots w_{z-1}^1, s^2 = w_0^2 w_1^2 \cdots w_{z-1}^2 \in L_{W,\epsilon}$$

lead the algorithm to the same configuration. Denote the index of the last block that differs between s^1 and s^2 by $t \triangleq max\{\tau \mid w_\tau^1 \neq w_\tau^2\}$. Next, consider the sequences $s^1 \cdot 0^{(t-1)\lfloor 2W\epsilon+1 \rfloor}$ and $s^2 \cdot 0^{(t-1)\lfloor 2W\epsilon+1 \rfloor}$. The algorithm must reach the same configuration after processing these sequences, even though the number of ones differs by $\lfloor 2W\epsilon+1 \rfloor > 2W\epsilon$. Therefore, the algorithm's error must be greater than $W\epsilon$ at least for one of the sequences, in contradiction to the assumption. We have shown 2^z words that lead to different configurations and therefore any deterministic algorithm that provides $\epsilon - additive$ approximation to Basic-Counting must have at least z bits of state. ◀

An immediate corollary of Lemma 3 is that any exact algorithm for Basic-Counting requires at least W bits, i.e., the naive solution is optimal. We next establish a second lower bound, which is useful for proving that our algorithm, presented below, is space optimal up to a constant factor.

▶ **Lemma 4.** *Fix some $\epsilon \leq \frac{1}{4}$. Any deterministic algorithm that provides a $W\epsilon$-additive approximation for the* Basic-Counting *problem requires at least $\lfloor \log W \rfloor$ bits.*

Proof. Assume that some algorithm A gives a $W\epsilon$-additive approximation using m memory bits. Consider A's run on the sequence $s = 0^W \cdot 1^{2^m}$. Since A is using m bits, it reaches some memory configuration c at least twice after processing the zeros in the sequence. Assume that A first reached c after seeing $0^W \cdot 1^y$ (where $y < 2^m$). This means that A must output some number $a_c \leq y + W\epsilon$ if queried. Now assume A returns to configuration c after reading z additional ones. This means A will return to c after every additional sequence of z ones. Therefore, for every integer q, after processing the sequence $0^W \cdot 1^{y+qz}$, A will reach configuration c. We can then pick a large q (such that $y + qz \geq W$), which means that the query answer for configuration c, a_c, has to be at least $W(1-\epsilon)$, as the window is now all-ones. We get $W(1-\epsilon) \leq a_c \leq y + W\epsilon$ and thus $2^m > y \geq W(1-2\epsilon)$. Putting everything together, we conclude that $m > \log(W(1-2\epsilon)) = \log W + \log(1-2\epsilon) \geq \log W - 1$, for $\epsilon \leq \frac{1}{4}$. Finally, since m is an integer, this implies $m \geq \lfloor \log W \rfloor$. ◀

▶ **Theorem 5.** *Let $\epsilon \leq \frac{1}{4}$. Any deterministic algorithm that provides a $W\epsilon$-additive approximation for the* Basic-Counting *problem requires at least* $\left\lfloor \max\left\{\log W, \frac{1}{2\epsilon+W^{-1}}\right\}\right\rfloor$ *bits.*

Proof. Immediate from lemmas 3 and 4. ◀

Finally, we extend our lower bounds to randomized algorithms.

▶ **Theorem 6.** *Let $\epsilon \leq \frac{1}{4}$. Any randomized Las Vegas algorithm that provides a $W\epsilon$-additive approximation for the* Basic-Counting *problem requires at least $\left\lfloor \max\left\{\log W, \frac{1}{2\epsilon+W^{-1}}\right\}\right\rfloor$ bits. Further, for any fixed $\delta \in (0, 1/2)$, any Monte Carlo algorithm that with probability at least $1-\delta$ approximates* Basic-Counting *within $W\epsilon$ error at any time instant, requires $\Omega(\frac{1}{\epsilon} + \log W)$ bits.*

Proof. We say that algorithm A is ϵ-correct on a input instance S if it is able to approximate the number of 1's in the last W bits, at every time instant while reading S, to within an additive error of $W\epsilon$.

We remind the reader that in our case, a Las Vegas (LV) algorithm for the Basic-Counting approximation problem is a randomized algorithm which is *always* ϵ-correct. In contrast, a Monte Carlo (MC) algorithm is a randomized procedure that is allowed to provide approximation with error larger than $W\epsilon$, with probability at most δ.

The Yao Minimax principle [23] implies that the amount of memory required for a deterministic algorithm to approximate a random input chosen according to a distribution $\mathbf{p}$ is a lower bound on the expected space consumption of a Las Vegas algorithm for the worst input. To prove a $\left\lfloor \frac{1}{2\epsilon+W^{-1}} \right\rfloor$ lower bound, we consider padding the language $L_{W,\epsilon}$ which is defined in Lemma 3. Specifically, we define $\mathbf{p}$ as the uniform distribution over all inputs in the language

$$L_{LV} = L_{W,\epsilon} \cdot \left\{0^W\right\}.$$

That is, the input consist of all bit sequences in $L_{W,\epsilon}$, followed by a sequence of W zeros. The trailing 0's are used to force the algorithm to reach distinct configurations after reading the first W input bits. As implied by the lemma, any deterministic algorithm which is always correct for a random instance requires at least $\left\lfloor \frac{1}{2\epsilon+W^{-1}} \right\rfloor$ bits, as it has to arrive to a distinct state for each input $s \in L_{W,\epsilon}$. The argument for a lower bound of $\lfloor \log W \rfloor$ bits is similar.

Next, we use the Minimax principle analogue for Monte Carlo algorithms [23], which states that for any input distribution $\mathbf{p}$ and $\delta \in [0, 1/2]$, any randomized algorithm that is always (for any input) ϵ-correct with probability at least $1-\delta$ uses in expectation at least half as much memory as the optimal deterministic algorithm that errs (i.e., is not ϵ-correct) with probability at most 2δ on a random instance drawn according to $\mathbf{p}$. Once again, we consider $\mathbf{p}$ to be the uniform distribution over

$$L_{MC} = L_{W,\epsilon} \cdot \left\{0^W\right\}.$$

Since the distribution is uniform, any deterministic algorithm, which is ϵ-correct with probability at least $1-2\delta$ on a random instance drawn according to $\mathbf{p}$, is actually ϵ-correct on $1-2\delta$ fraction of the inputs. Similar to the LV case, the argument in Lemma 3 implies that the algorithm must reach a distinct configuration after reading the first W bits of each of the $(1-2\delta)\cdot|L_{MC}|$ inputs it is ϵ-correct on. Consequently, the algorithm must use at least $\log\left((1-2\delta)\cdot|L_{MC}|\right)$ bits of memory. Applying the Minimax principle, the derived lower bound B_{MC} for any MC algorithm is:

$$B_{MC} \geq \frac{1}{2}\log\left((1-2\delta)\cdot|L_{MC}|\right) \geq \frac{1}{2}\left\lfloor \frac{1}{2\epsilon+W^{-1}} \right\rfloor + \frac{1}{2}\log(1-2\delta) = \Omega\left(\frac{1}{\epsilon}\right)$$

Once again, the case for a $\Omega(\log W)$ lower bound is based on Lemma 4 and follows from similar arguments. ◀

3.2 Upper Bound

We now present an algorithm for Basic-Counting that provides a $W\epsilon$-additive approximation $\widehat{C^W}$ for C^W over a binary stream with near-optimal memory. Denote $k \triangleq \frac{1}{2\epsilon}$. For simplicity, we assume that $\frac{W}{k}$ and k are integers. Intuitively, our algorithm partitions the stream into k *blocks* of size $\frac{W}{k}$, representing each using a single bit. A set bit corresponds to a count of $\frac{W}{k}$ in the input stream, while a clear bit corresponds to a count of 0. We then use an "optimistic" approach to reduce the error – the number of ones in the input stream not counted using the bit array is *propagated* to the next block; this means a block might be represented with 1, even if it contains only a single set bit. Surprisingly, we show that this approach allows us to keep the error bounded and that the errors do not accumulate. We keep a counter y for the number of 1s. At the end of a block, if y is larger than $\frac{W}{k}$, we mark the current block and subtract $\frac{W}{k}$ from y, propagating the remainder to the next block. Our algorithm answers queries by multiplying the number of marked blocks in the current window by $\frac{W}{k}$, making corrections to reduce the error. We maintain the following variables:
$\boldsymbol{y}$ – a counter for the number of ones.
$\boldsymbol{b}$ – a bit-array of size k.
$\boldsymbol{i}$ – the index of the "oldest" block in b.
$\boldsymbol{B}$ – the sum of all bits in b.
$\boldsymbol{m}$ – a counter for the current offset within the block.
Every arriving bit is handled as follows: We increment m, and if the bit is set we also increment y. At the end of a block, we check if y exceeds $\frac{W}{k}$. If so, we subtract $\frac{W}{k}$ from y and set the bit b_i. This way, the reduction in y is compensated for by the newly set bit in b. The previous value of b_i, holding information about 1s that just left the window, is forgotten.

To answer a query the algorithm returns the number of set bits in b multiplied by the block size $\frac{W}{k}$. We then add the value of y, which represents the number of ones not yet recorded in b, and subtract $m \cdot b_i$, as m bits of the oldest recorded block have already left the window. Finally, we remove any bias from the estimation by subtracting $\frac{W}{2k}$ (half a block).

In order to answer queries without iterating over b, we maintain another variable B, which keeps track of the number of ones in b. The entire pseudo-code is given in Algorithm 1.

▶ **Theorem 7.** *Algorithm 1 provides a $W\epsilon$-additive approximation of* Basic-Counting.

Proof. First, let us introduce some notations used in the proof. Assume that the index of the last bit is $W+m$, where x_W is the last bit of a block and $m < \frac{W}{k}$. b_i is considered after $W+m$ bits have been processed. We denote y_j the value of y *after* adding bit j.

The setting for the proof is given in Figure 1. We aim to approximate

$$C^W \triangleq \sum_{j=m+1}^{W+m} x_j. \tag{1}$$

Our algorithm uses the following approximation:

$$\widehat{C^W} = \frac{W}{k} \cdot B + y_{W+m} - \frac{W}{2k} - m \cdot b_i = \frac{W}{k} \cdot B + y_W + \sum_{j=W+1}^{W+m} x_j - \frac{W}{2k} - m \cdot b_i. \tag{2}$$

At times $1, 2, \ldots, W$, y is incremented once for every set bit in the input stream. At the end of block j, if y is reduced by $\frac{W}{k}$, then b_j is set and will not be cleared before time $W+m$. Therefore, $\frac{W}{k} \cdot B + y_W = y_0 + \sum_{j=1}^{W} x_j$. Substituting $\frac{W}{k} \cdot B + y_W$ in (2), we get

$$\widehat{C^W} = y_0 + \sum_{j=1}^{W} x_j + \sum_{j=W+1}^{W+m} x_j - \frac{W}{2k} - m \cdot b_i = y_0 + \sum_{j=1}^{m} x_j + \sum_{j=m+1}^{W+m} x_j - \frac{W}{2k} - m \cdot b_i.$$

Algorithm 1 Additive Approximation of Basic Counting

Initialization: $y = 0, b = 0, m = 0, i = 0$.
function ADD(Bit x)
  if $m = \frac{W}{k} - 1$ **then**
    $B = B - b_i$
    if $y + x \geq \frac{W}{k}$ **then**
      $b_i = 1$
      $y = y - \frac{W}{k} + x$
    else
      $b_i = 0$
      $y = y + x$
    $B = B + b_i$
    $m = 0$
    $i = i + 1 \mod k$
  else
    $y = y + x$
    $m = m + 1$
function QUERY
  return $\frac{W}{k} \cdot B + y - \frac{W}{2k} - m \cdot b_i$

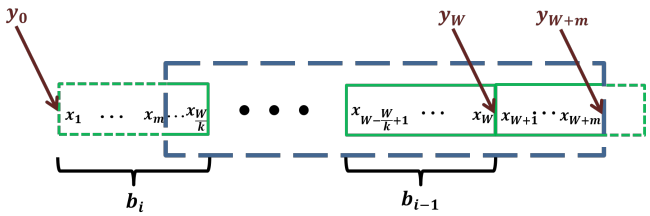


Figure 1 The setting for the proof of Theorem 7. b is cyclic – b_i represents the oldest block and b_{i-1} the newest completed block.

Plugging the definition of C^W, we get $\widehat{C^W} = y_0 + \sum_{j=1}^{m} x_j + C^W - \frac{W}{2k} - m \cdot b_i$. Therefore, the error is

$$\widehat{C^W} - C^W = y_0 + \sum_{j=1}^{m} x_j - m \cdot b_i - \frac{W}{2k}.$$

We consider two cases:

$b_i = 1$: This means that y had crossed the threshold by time $\frac{W}{k}$, i.e. $y_0 + \sum_{j=1}^{\frac{W}{k}} x_j \geq \frac{W}{k}$ and equivalently $y_0 + \sum_{j=1}^{m} x_j \geq \frac{W}{k} - \sum_{j=m+1}^{\frac{W}{k}} x_j$. Thus, on one side

$$\begin{aligned}\widehat{C^W} - C^W = y_0 + \sum_{j=1}^{m} x_j - m - \frac{W}{2k} &\geq \frac{W}{k} - \sum_{j=m+1}^{\frac{W}{k}} x_j - m - \frac{W}{2k} \\ &\geq \frac{W}{k} - \left(\sum_{j=m+1}^{\frac{W}{k}} 1\right) - m - \frac{W}{2k} \geq -\frac{W}{2k} = -W\epsilon.\end{aligned}$$

To bound the error from above we use the fact that the value of y at the end of a block never exceeds $\frac{W}{k}$. This can be shown by induction, as y is incremented at most $\frac{W}{k}$

times during one block, and then reduced by $\frac{W}{k}$ if it exceeds the block size. Therefore, $\widehat{C^W} - C^W = y_0 + \sum_{j=1}^{m} x_j - m - \frac{W}{2k} \leq y_0 - \frac{W}{2k} \leq \frac{W}{k} - \frac{W}{2k} = W\epsilon$.

$b_i = 0$: Similarly, this means that y was lower than the threshold at the end of block i, hence $y_0 + \sum_{j=1}^{\frac{W}{k}} x_j \leq \frac{W}{k} - 1$ or equivalently, $y_0 + \sum_{j=1}^{m} x_j \leq \frac{W}{k} - \sum_{j=m+1}^{\frac{W}{k}} x_j - 1$. Thus, our error is bounded from below by $\widehat{C^W} - C^W = y_0 + \sum_{j=1}^{m} x_j - \frac{W}{2k} \geq y_0 - \frac{W}{2k} \geq -\frac{W}{2k} = -W\epsilon$ and from above by

$$\widehat{C^W} - C^W = y_0 + \sum_{j=1}^{m} x_j - \frac{W}{2k} \leq \frac{W}{k} - \sum_{j=m+1}^{\frac{W}{k}} x_j - \frac{W}{2k} - 1 \leq \frac{W}{2k} - 1 \quad = W\epsilon - 1.$$

We have established that in all cases the absolute error is at most $W\epsilon$ as required. ◀

We next prove that the memory requirement of Algorithm 1 is nearly optimal.

▶ **Theorem 8.** *Algorithm 1 requires $\frac{1}{2\epsilon} + 2\log W + O(1)$ bits of memory.*

Proof. We represent y using $\lceil 2 + \log(W\epsilon) \rceil$ bits, m using $\lceil 1 + log(W\epsilon) \rceil$ bits and b using k bits. Additionally, i requires $\lceil \log k \rceil$ bits, and B another $\lceil \log(k+1) \rceil$ bits. Overall, the number of bits required is $k + \lceil 2 + \log(W\epsilon) \rceil + \lceil 1 + \log(W\epsilon) \rceil + \lceil \log k \rceil + \lceil \log(k+1) \rceil$ $\leq k + 2\log(W\epsilon) - 2\log(2\epsilon) + 8 = \frac{1}{2\epsilon} + 2\log(W) + 6 = \frac{1}{2\epsilon} + 2\log W + O(1)$. ◀

Theorem 5 shows that our algorithm uses at most twice as much memory as required by the lower bound (up to a constant number of bits) for every *constant* $\epsilon \leq \frac{1}{4}$. When ϵ is not constant, our memory requirement is at most 3 times the lower bound, as shown in the following lemma.

▶ **Corollary 9.** *For any $\epsilon \leq \frac{1}{4}$, the ratio between the memory consumption of Algorithm 1 and the lower bound for additive approximations for* Basic-Counting *is*

$$\frac{\frac{1}{2\epsilon} + 2\log W + O(1)}{\max\left\{\log W, \frac{1}{2\epsilon + W^{-1}}\right\}} = 3 + o(1).$$

Since the proof is very technical, and due to lack of space, it is left for the full version [4].

4 Basic-Summing Problem

We now consider an extension of Basic-Counting where elements are non-negative integers:

▶ **Definition 10** (Basic-Summing)**.** Given a stream of elements comprising of integers in the range $[R+1] = \{0, 1, \ldots, R\}$, maintain the sum S of the last W elements.

4.1 Lower Bound

We now show that approximating Basic-Summing to within an additive error of $RW\epsilon$ requires $\Omega(\frac{1}{\epsilon} + \log W)$ bits for $\epsilon \geq \frac{1}{2W}$ and $\Omega(W \log\left(\frac{1}{W\epsilon}\right))$ bits for $\epsilon \leq \frac{1}{2W}$.

▶ **Lemma 11.** *For any $\epsilon \leq \frac{1}{4}$, approximating* Basic-Summing *to within an additive error of $RW\epsilon$ requires $\left\lfloor \max\left\{\log W, \frac{1}{2\epsilon + W^{-1}}\right\} \right\rfloor$ memory bits.*

Proof. The proof of the lemma is very similar to the proof of Theorem 5 and is obtained by replacing every set bit with the integer R in Lemma 3 and Lemma 4. ◀

Next, we show a lower bound for smaller values of ϵ.

▶ **Lemma 12.** *For any ϵ, approximating* Basic-Summing *to within an additive error of $RW\epsilon$ requires at least $W \log \left\lfloor \frac{1}{4W\epsilon} + 1 \right\rfloor$ memory bits.*

Proof. Denote $x \triangleq \lfloor 2RW\epsilon + 1 \rfloor$ and $C \triangleq \left\{ n \cdot x \mid n \in \left\{ 0, 1, \ldots, \left\lfloor \frac{1}{2W\epsilon + R^{-1}} \right\rfloor \right\} \right\}$. Let L be the language of all W length strings over the number in C, i.e.,

$$L_{R,W,\epsilon} = \{\sigma_0\sigma_1 \cdots \sigma_{W-1} | \forall j \in [W] : \sigma_j \in C\}.$$

We show that every two distinct sequences in L must lead the algorithm into distinct configurations implying a lower bound of

$$\lceil \log |L| \rceil \geq W \log |C| = W \log \left\lfloor \frac{1}{2W\epsilon + R^{-1}} + 1 \right\rfloor \geq W \log \left\lfloor \frac{1}{4W\epsilon} + 1 \right\rfloor$$

bits, where the last inequality follows from the fact that any $\epsilon < \frac{1}{2RW}$ implies exact summing. Assume, by way of contradiction, that two different words

$$s^1 = \sigma_0^1\sigma_1^1 \cdots \sigma_{W-1}^1, s^2 = \sigma_0^2\sigma_1^2 \cdots \sigma_{W-1}^2 \in L$$

lead the algorithm to the same configuration. Denote the index of the last letter that differs between s^1 and s^2 by $t \triangleq max\{\tau \mid \sigma_\tau^1 \neq \sigma_\tau^2\}$. Next, consider the sequences $s^1 \cdot 0^{t-1}$ and $s^2 \cdot 0^{t-1}$. The algorithm must reach the same configuration after processing these sequences, even though the sum of the last W elements differ by at least $x = \lfloor 2RW\epsilon + 1 \rfloor > 2RW\epsilon$. Therefore, the algorithm's error must be greater than $RW\epsilon$ at least for one of the sequences, in contradiction to the assumption. ◀

▶ **Theorem 13.** *Approximating* Basic-Summing *to within an additive error of $RW\epsilon$ requires $\Omega(\frac{1}{\epsilon} + \log W)$ bits for $\frac{1}{2W} \leq \epsilon \leq \frac{1}{4}$ and $\Omega(W \log\left(\frac{1}{W\epsilon}\right))$ bits for $\epsilon \leq \frac{1}{2W}$.*

Proof. Lemma 11 shows that approximating Basic-Summing within $RW\epsilon$ requires

$$\max\left\{\log W, \frac{1}{2\epsilon + W^{-1}}\right\}$$

bits for $\frac{1}{2W} \leq \epsilon \leq \frac{1}{4}$. The same argument used in Lemma 9 shows that this implies $\Omega\left(\frac{1}{\epsilon} + \log W\right)$ bits lower bound for any $\epsilon \geq \frac{1}{2W}$. For $\epsilon < \frac{1}{2W}$ such that $\epsilon = \Theta(W^{-1})$, approximating Basic-Summing within $RW\epsilon$ implies a $\frac{R}{2}$-additive approximation and therefore the $\Omega\left(\frac{1}{\epsilon} + \log W\right)$ bound holds. For $\epsilon = o\left(\frac{1}{W}\right)$, we use Lemma 12, which implies a lower bound of $W \log \left\lfloor \frac{1}{4W\epsilon} + 1 \right\rfloor = \Omega\left(W \log\left(\frac{1}{W\epsilon}\right)\right)$ memory bits. ◀

An immediate corollary of Theorem 13 is that any exact algorithm for Basic-Summing requires at least $\Omega(W \log R)$ bits, i.e., the naive solution of maintaining a W-sized array of the elements in the window, encoding each using $\lceil \log(R+1) \rceil$ bits, is optimal (for exact Basic-Summing). Finally, we extend the results to randomized algorithms, where the proof is left for the full version [4] due to lack of space.

▶ **Theorem 14.** *For any fixed $\delta \in [0, 1/2)$, any randomized Monte Carlo algorithm that gives a $W\epsilon$ approximation to* Basic-Summing *with a probability of at least $1 - \delta$ requires $\Omega(\frac{1}{\epsilon} + \log W)$ bits for $\frac{1}{2W} \leq \epsilon \leq \frac{1}{4}$ and $\Omega(W \log\left(\frac{1}{W\epsilon}\right))$ bits for $\epsilon \leq \frac{1}{2W}$. Notice that the $\delta = 0$ case applies to Las Vegas algorithms.*

4.2 Upper Bound

We show that our Basic-Counting algorithm can be adapted to this problem with only a small memory overhead such that the algorithm's state size remains independent of R. We first present the extension of the algorithm for the $\epsilon^{-1} \leq 2W\left(1 - \frac{1}{\log W}\right)$ case. In Section 4.3 we complete the picture by giving an alternative algorithm for smaller values of ϵ. Intuitively, we "scale" the algorithm by dividing each added element by R and rounding the result. In order to keep the sum of elements not yet accounted for in b, y is now maintained as a fixed-point variable rather than an integer. Ideally, the fractional value of the remainder y should allow exact representation of $\{0, 1/R, \ldots, 1 - 1/R\}$, and therefore requires $\log R$ bits. When the range R is "large", or more precisely $R = \omega(\epsilon^{-1})$, we save space by storing the fractional value of y using less than $\log R$ bits, which inflicts a rounding error. That is, we keep y using $\left\lceil \log\left(2\frac{W}{k}\right)\right\rceil + \upsilon$ bits. Similarly to our Basic-Counting algorithm, $\left\lceil \log\left(2\frac{W}{k}\right)\right\rceil$ bits are used to store the integral part of y. The additional υ bits are used for the fractional value of y. The value of υ is determined later.

In order to keep the total error bounded, we compensate for the rounding error by using smaller block sizes, which are derived from the number of blocks k, determined in (3). Our algorithm keeps the following variables:

$\boldsymbol{b}$ – a bit-array of size k.

$\boldsymbol{y}$ – a counter for the sum of elements which is not yet accounted for in b.

$\boldsymbol{i}$ – the index of the "oldest" block in b.

$\boldsymbol{B}$ – the sum of all bits in b.

$\boldsymbol{m}$ – a counter for the current offset within the block.

Our Basic-Summing algorithm is presented in Algorithm 2. We use $\text{Round}_\upsilon(z)$ for some $z \in [0, 1]$ to denote rounding of z to the nearest value $\widetilde{z}$ such that $2^\upsilon \widetilde{z}$ is an integer.

Algorithm 2 Additive Approximation for Basic-Summing

```
1: Initialization: y = 0, b = 0, B = 0, i = 0, m = 0.
2: function ADD(ELEMENT x)
3:     x' = Round_υ(x/R)
4:     if m = W/k − 1 then
5:         B = B − b_i
6:         if y + x' ≥ W/k then
7:             b_i = 1
8:             y = y − W/k + x'
9:         else
10:            b_i = 0
11:            y = y + x'
12:        B = B + b_i
13:        m = 0
14:        i = i + 1 mod k
15:    else
16:        y = y + x'
17:        m = m + 1
18: function QUERY()
19:    return R · (W/k · B + y − W/2k − m · b_i)
```

Algorithm 3 Additive Approximation for Basic-Summing with Small Error

```
1: Initialization: y = 0, b = 0, B = 0, i = 0.
2: function ADD(element x)
3:     x' = Round_υ(x/R)
4:     B = B − b_i
5:     b_i = ⌊(y + x')/(W/k)⌋
6:     y = y + x' − b_i · W/k
7:     B = B + b_i
8:     i = i + 1 mod W
9: function QUERY()
10:    return R · (W/k · B + y − W/(2k))
```

▶ **Theorem 15.** *For any* $\epsilon^{-1} \leq 2W\left(1-\frac{1}{\log W}\right)$*, Algorithm 2 provides an* $RW\epsilon$*-additive approximation for* Basic-Summing.

Theorem 15 shows that for any $\epsilon^{-1} \leq 2W\left(1-\frac{1}{\log W}\right)$, by choosing $\upsilon \triangleq \left\lceil \log\left(\epsilon^{-1}\log W\right)\right\rceil$ and the number of blocks to be

$$k \triangleq \left\lceil \frac{1}{2\epsilon - 2^{-\upsilon}} \right\rceil, \tag{3}$$

our algorithm estimates S with an additive error of $RW\epsilon$. Due to lack of space, the proof of Theorem 15 can be found in the full version [4]. The following theorem analyzes the memory requirements of our algorithm.

▶ **Theorem 16.** *For any* $\epsilon^{-1} \leq 2W\left(1-\frac{1}{\log W}\right)$*, Algorithm 2 requires* $\left(2\log W + \frac{1}{2\epsilon}\right)(1+o(1))$ *memory bits.*

The proof is similar to the proof of Theorem 8 and therefore appears in the full version [4].

4.3 Summing with Small Error

Algorithm 2 only works for $\epsilon^{-1} \leq 2W\left(1-\frac{1}{\log W}\right)$ that satisfies $\frac{W}{k} \geq 1$; otherwise, k cannot represent the number of blocks, as blocks cannot be empty. To complete the picture, we present Algorithm 3 that works for smaller errors. Intuitively, we keep an array b of size W, such that every cell represents the number of integer multiples of $\frac{RW}{k}$ in an arriving item. Similarly to the above algorithms, we reduce the error by tracking the remainder in a variable y, propagating uncounted fractions to the following item. In this case as well, the optimistic approach reduces the error compared with keeping a W-sized array of rounded values for approximating the sum. Each cell in b needs to represent a value in $\left\{0,1,\ldots,\left\lfloor 1+\frac{k}{W}\right\rfloor\right\}$; the remainder y is now a fractional number, represented using υ bits. When a new item is added, we scale it, add the result to y, and update both b_i and the remainder.

▶ **Theorem 17.** *Algorithm 3 provides an* $RW\epsilon$*-additive approximation for* Basic-Summing.

The proof appears in the full version [4]. It considers the rounding error generated by representing x' using υ bits, and shows that the remainder propagation (Line 6) limits error accumulation.

▶ **Theorem 18.** *For any* $\epsilon^{-1} > 2W\left(1-\frac{1}{\log W}\right) = 2W(1-o(1))$*, Algorithm 3 requires* $W\log\left(\frac{1}{2W\epsilon}+1\right)\cdot(1+o(1)) \leq \frac{1}{2\epsilon}\cdot(1+o(1))$ *memory bits.*

The proof is similar to the proof of Theorem 8 and therefore appears in the full version [4].

We conclude the section by showing that our algorithm is succinct, requiring only $(1+o(1))$ times as much memory as the lower bound proved in Theorem 13.

▶ **Theorem 19.** *Let $\epsilon = o(W^{-1})$, and denote $\mathcal{B} \triangleq W \log \left\lfloor \frac{1}{4W\epsilon} + 1 \right\rfloor$. Algorithm 3 provides $RW\epsilon$ additive approximation to* Basic-Summing *using $\mathcal{B} \cdot (1 + o(1))$ memory bits.*

Proof. Theorem 18 shows that the number of bits our algorithm requires for $\epsilon = o(W^{-1})$ is $W \log \left(\frac{1}{2W\epsilon} + 1\right) \cdot (1 + o(1)) \leq \mathcal{B}(1 + \frac{2W}{\mathcal{B}}) \cdot (1 + o(1)) = \mathcal{B} \cdot (1 + o(1))$. ◀

5 Discussion

In this paper, we have investigated additive approximations for the Basic-Counting and Basic-Summing problems. For both cases, we have provided space efficient algorithms. Further, we have proved the first lower bound for additive approximations for the Basic-Counting problem, and showed that our algorithm achieves this bound, and is hence optimal. In the case of Basic-Summing, whenever $\epsilon^{-1} \leq 2W\left(1 - \frac{1}{\log W}\right)$, the same lower bound as in the Basic-Counting problems still holds and so our approximation algorithm for this domain is optimal up to a small factor. For other values of ϵ, we have shown an improved lower bound and a corresponding succinct approximation algorithm.

In the future, we would like to study lower and upper bounds for additive approximations for several related problems. These include, e.g., approximating the sliding window sum of weights for each item in a stream of (item, weight) tuples. Further, we intend to explore applying additive approximations in the case of multiple streams. Obviously, one can allocate a separate counter for each stream, thereby multiplying the space complexity by the number of concurrent streams. However, it was shown in [13] that for the case of multiplicative approximations, there is a more space efficient solution. We hope to show a similar result for additive approximations.

Acknowledgments. We thank Dror Rawitz for helpful comments. This work was partially funded by MOST grant #3-10886 and the Technion-HPI research school.

References

1 Michael H Albert, Alexander Golynski, Angèle M Hamel, Alejandro López-Ortiz, S Srinivasa Rao, and Mohammad Ali Safari. Longest increasing subsequences in sliding windows. *Theoretical Computer Science*, 321(2):405–414, 2004.

2 Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding windows. In *Proc. of the 23rd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, PODS 2004*. Association for Computing Machinery, Inc., June 2004.

3 Brian Babcock, Mayur Datar, Rajeev Motwani, and Liadan O'Callaghan. Maintaining variance and k-medians over data stream windows. In Frank Neven, Catriel Beeri, and Tova Milo, editors, *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, pages 234–243. ACM, 2003.

4 Ran Ben Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Efficient summing over sliding windows. *CoRR*, abs/1604.02450, 2016. URL: `http://arxiv.org/abs/1604.02450`.

5 Ran Ben Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Heavy hitters in streams and sliding windows. In *INFOCOM, 2016 Proceedings IEEE*, pages 307–315, April 2016.

6 Vladimir Braverman, Ran Gelles, and Rafail Ostrovsky. How to catch l2-heavy-hitters on sliding windows. *Theoretical Computer Science*, 554:82–94, 2014.

7 Vladimir Braverman and Rafail Ostrovsky. Smooth histograms for sliding windows. In *Foundations of Computer Science, 2007. FOCS'07. 48th Annual IEEE Symposium on*, pages 283–293. IEEE, 2007.

8 Edith Cohen and Martin J. Strauss. Maintaining time-decaying stream aggregates. *J. Algorithms*, 59(1):19–36, 2006.

9 Graham Cormode and Ke Yi. Tracking distributed aggregates over time-based sliding windows. In *Scientific and Statistical Database Management*, pages 416–430. Springer, 2012.

10 Michael Crouch and Daniel S. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, pages 96–104, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.96.

11 Michael S Crouch, Andrew McGregor, and Daniel Stubbs. Dynamic graphs in the sliding-window model. In *Algorithms–ESA 2013*, pages 337–348. Springer, 2013.

12 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002.

13 Phillip B. Gibbons and Srikanta Tirthapura. Distributed streams algorithms for sliding windows. In *SPAA*, pages 63–72, 2002.

14 Regant Y.S. Hung and H. F. Ting. Finding heavy hitters over the sliding window of a weighted data stream. In E. Laber, C. Bornstein, L. Nogueira, and L. Faria, editors, *LATIN 2008: Theoretical Informatics*, volume 4957 of *LNCS*, pages 699–710. Springer, 2008. doi:10.1007/978-3-540-78773-0_60.

15 Lap-Kei Lee and H. F. Ting. Maintaining significant stream statistics over sliding windows. In *Proceedings of the Seventeenth Annual Symposium on Discrete Algorithms, SODA*, pages 724–732. ACM Press, 2006.

16 Lap-Kei Lee and HF Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *Proc. of the SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 290–297. ACM, 2006.

17 Yang Liu, Wenji Chen, and Yong Guan. Near-optimal approximate membership query over time-decaying windows. In *INFOCOM, Proceedings IEEE*, pages 1447–1455, April 2013.

18 Kyriakos Mouratidis, Spiridon Bakiras, and Dimitris Papadias. Continuous monitoring of top-k queries over sliding windows. In *Proc. of the International Conference on Management of Data*, SIGMOD, pages 635–646, New York, NY, USA, 2006. ACM.

19 Moni Naor and Eylon Yogev. Sliding bloom filters. In Leizhen Cai, Siu-Wing Cheng, and Tak-Wah Lam, editors, *Algorithms and Computation*, volume 8283 of *Lecture Notes in Computer Science*, pages 513–523. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-45030-3_48.

20 Krešimir Pripužić, Ivana Podnar Žarko, and Karl Aberer. Time- and space-efficient sliding window top-k query processing. *ACM Trans. Database Syst.*, 40(1):1:1–1:44, March 2015.

21 Hong Shen and Yu Zhang. Improved approximate detection of duplicates for data streams over sliding windows. *Journal of Computer Science and Technology*, 23(6):973–987, 2008.

22 Zhitao Shen, M.A. Cheema, Xuemin Lin, Wenjie Zhang, and Haixun Wang. Efficiently monitoring top-k pairs over sliding windows. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 798–809, April 2012.

23 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symp. on Foundations of Computer Science*, pages 222–227. IEEE, 1977.

24 Wenjie Zhang, Ying Zhang, Muhammad Aamir Cheema, and Xuemin Lin. Counting distinct objects over sliding windows. In *Proceedings of the Twenty-First Australasian Conference on Database Technologies – Volume 104*, ADC'10, pages 75–84, Darlinghurst, Australia, Australia, 2010. Australian Computer Society, Inc.

Lower Bounds for Approximation Schemes for Closest String*

Marek Cygan[1], Daniel Lokshtanov[2], Marcin Pilipczuk[3], Michał Pilipczuk[4], and Saket Saurabh[5]

1 **Institute of Informatics, University of Warsaw, Warsaw, Poland**
cygan@mimuw.edu.pl
2 **Department of Informatics, University of Bergen, Bergen, Norway**
daniello@ii.uib.no
3 **Institute of Informatics, University of Warsaw, Warsaw, Poland**
malcin@mimuw.edu.pl
4 **Institute of Informatics, University of Warsaw, Warsaw, Poland**
michal.pilipczuk@mimuw.edu.pl
5 **Department of Informatics, University of Bergen, Bergen, Norway; and Institute of Mathematical Sciences, Chennai, India**
saket.saurabh@ii.uib.no, saket@imsc.res.in

Abstract

In the Closest String problem one is given a family $\mathcal{S}$ of equal-length strings over some fixed alphabet, and the task is to find a string y that minimizes the maximum Hamming distance between y and a string from $\mathcal{S}$. While polynomial-time approximation schemes (PTASes) for this problem are known for a long time [Li et al.; J. ACM'02], no *efficient* polynomial-time approximation scheme (EPTAS) has been proposed so far. In this paper, we prove that the existence of an EPTAS for Closest String is in fact unlikely, as it would imply that FPT = W[1], a highly unexpected collapse in the hierarchy of parameterized complexity classes. Our proof also shows that the existence of a PTAS for Closest String with running time $f(\varepsilon)\cdot n^{o(1/\varepsilon)}$, for any computable function f, would contradict the Exponential Time Hypothesis.

1998 ACM Subject Classification F.2.2 Nonnumerical algorithms and problems

Keywords and phrases closest string, PTAS, efficient PTAS

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.12

1 Introduction

Closest String and Closest Substring are two computational problems motivated by questions in molecular biology connected to identifying functionally similar regions of DNA or RNA sequences, as well as by applications in coding theory. In Closest String we are given a family $\mathcal{S}$ of strings over some fixed alphabet Σ, each of length L. The task is to find one string $y \in \Sigma^L$ for which $\max_{x\in\mathcal{S}} \mathcal{H}(x, y)$ is minimum possible, where $\mathcal{H}(x, y)$ is

* M. Cygan and Ma. Pilipczuk have been supported by Polish National Science Centre grant DEC-2012/05/D/ST6/03214. Mi. Pilipczuk has been supported by Polish National Science Centre grant DEC-2013/11/D/ST6/03073 and by the Foundation for Polish Science via the START stipend programme. During the work on these results, Mi. Pilipczuk held a post-doc position at Warsaw Center of Mathematics and Computer Science. D. Lokshtanov is supported by the BeHard grant under the recruitment programme of the of Bergen Research Foundation. S. Saurabh is supported by PARAPPROX, ERC starting grant no. 306992.

the *Hamming distance* between x and y, that is, the number of positions on which x and y have different letters. We will consider both the optimization variant of the problem where the said distance is to be minimized, and the decision variant where an upper bound d is given on the input, and the algorithm needs to decide whether there exists a string y with $\max_{x\in\mathcal{S}} \mathcal{H}(x,y) \leq d$. Closest Substring is a more general problem where the strings from the input family $\mathcal{S}$ all have length $m \geq L$, and we look for a string $y \in \Sigma^L$ that minimizes $\max_{x\in\mathcal{S}} \min_{x' \text{ substring of } x} \mathcal{H}(x',y)$. In other words, we look for y that can be fit as close as possible to a substring of length L of each of the input strings from $\mathcal{S}$.

Both Closest String and Closest Substring, as well as numerous variations on these problems, have been studied extensively from the point of view of approximation algorithms. Most importantly for us, for both of these problems there are classic results providing *polynomial-time approximation schemes* (*PTASes*): for every $\varepsilon > 0$, it is possible to approximate in polynomial time the optimum distance within a multiplicative factor of $(1+\varepsilon)$. The first PTASes for these problems were given by Li et al. [9], and they had running time bounded by $n^{\mathcal{O}(1/\varepsilon^4)}$. This was later improved by Andoni et al. [1] to $n^{\mathcal{O}(\frac{\log 1/\varepsilon}{\varepsilon^2})}$, and then by Ma and Sun [11] to $n^{\mathcal{O}(1/\varepsilon^2)}$, which constitutes the current frontier of knowledge. We refer to the works [3, 8, 7, 9, 11, 12] for a broad introduction to biological applications of Closest String, Closest Substring, and related problems, as well as pointers to relevant literature.

One of the immediate questions stemming from the works of Li et al. [9], Andoni et al. [1], and Ma and Sun [11], is whether either for Closest String or Closest Substring one can also give an *efficient polynomial-time approximation scheme* (*EPTAS*), i.e., an approximation scheme that for every $\varepsilon > 0$ gives a $(1+\varepsilon)$-approximation algorithm with running time $f(\varepsilon)\cdot n^{\mathcal{O}(1)}$, for some computable function f. In other words, the degree of the polynomial should be independent of ε, whereas the exponential blow-up (inevitable due to NP-completeness) should happen only in the multiplicative constant standing in front of the running time. EPTASes are desirable from the point of view of applications, since they provide approximation algorithms that can be useful in practice already for relatively small values of ε, whereas running times of general PTASes are usually prohibitive.

For the more general Closest Substring problem, this question was answered negatively by Marx [12] using the techniques from parameterized complexity. More precisely, Marx considered various parameterizations of Closest Substring, and showed that when parameterized by d and $|\mathcal{S}|$, the problem remains W[1]-hard even for the binary alphabet. This means that the existence of a fixed-parameter algorithm with running time $f(d,|\mathcal{S}|)\cdot n^{\mathcal{O}(1)}$, where n is the total size of the input, would imply that FPT = W[1], a highly unexpected collapse in the parameterized complexity. This result shows that, under FPT $\neq$ W[1], also an EPTAS for Closest Substring can be excluded. Indeed, if such an EPTAS existed, then by setting any $\varepsilon < \frac{1}{d}$ one could in time $f(d)\cdot n^{\mathcal{O}(1)}$ distinguish instances with optimum distance value d from the ones with optimum distance value $d+1$, thus solving the decision variant in fixed-parameter tractable (FPT) time. Using more precise results about the parameterized hardness of the Clique problem, Marx [12] showed that, under the assumption of *Exponential Time Hypothesis* (*ETH*), which states that 3-SAT cannot be solved in time $\mathcal{O}(2^{\delta n})$ for some $\delta > 0$, one even cannot expect PTASes for Closest Substring with running time $f(\varepsilon)\cdot n^{o(\log(1/\varepsilon))}$ for any computable function f. We refer to a survey of Marx [13] for more examples of links between parameterized complexity and the design of approximation schemes.

The methodology used by Marx [12], which is the classic connection between parameterized complexity and EPTASes that dates back to the work of Bazgan [2] and of Cesati and

Trevisan [4], completely breaks down when applied to Closest String. This is because this problem actually does admit an FPT algorithm when parameterized by d. An algorithm with running time $d^d \cdot n^{\mathcal{O}(1)}$ was proposed by Gramm et al. [7]. Later, Ma and Sun [11] gave an algorithm with running time $2^{\mathcal{O}(d)} \cdot |\Sigma|^d \cdot n^{\mathcal{O}(1)}$, which is more efficient for constant-size alphabets. Both the algorithms of Gramm et al. and of Ma and Sun are known to be essentially optimal under ETH [10], and nowadays they constitute textbook examples of advanced branching techniques in parameterized complexity [5]. Therefore, in order to settle the question about the existence of an EPTAS for Closest String, one should look for a substantial refinement of the currently known techniques.

An approach for overcoming this issue was recently used by Boucher et al. [3], who attribute the original idea to Marx [13]. Boucher et al. considered a problem called Consensus Patterns, which is a variation of Closest Substring where the goal function is the total sum of Hamming distances between the center string and best-fitting substrings of the input strings, instead of the maximum among these distances. The problem admits a PTAS due to Li et al. [8], and was shown by Marx [12] to be fixed-parameter tractable when parameterized by the target distance d. Despite the latter result, Boucher et al. [3] managed to prove that the existence of an EPTAS for Consensus Patterns would imply that FPT = W[1]. The main idea is to provide a reduction from a W[1]-hard problem, such as Clique, where the output target distance d is not bounded by a function of the input parameter k (indeed, the existence of such a reduction would prove that FPT = W[1]), but the multiplicative gap between the optimum distances yielded for yes- and no-instances is $1 + \frac{1}{g(k)}$, for some computable function g. Even though the output parameter is unbounded in terms of k, an EPTAS for the problem could be still used to distinguish between output instances obtained from yes- and no-instances of Clique in FPT time, thus proving that FPT = W[1].

Our contribution

In this paper we provide a negative answer to the question about the existence of an EPTAS for Closest String by proving the following theorem.

▶ **Theorem 1.1.** *The following assertions hold:*

- *Unless* FPT = W[1]*, there is no EPTAS for* Closest String *over binary alphabet.*
- *Unless ETH fails, there is no PTAS for* Closest String *over binary alphabet with running time* $f(\varepsilon) \cdot n^{o(1/\varepsilon)}$*, for any computable function* f*.*

Thus, one should not expect an EPTAS for Closest String, whereas for PTASes there is still a room for improvement between the running time of $n^{\mathcal{O}(1/\varepsilon^2)}$ given by Ma and Sun [11] and the lower bound of Theorem 1.1. It is worth noting that our $f(\varepsilon) \cdot n^{o(1/\varepsilon)}$ time lower bound for $(1+\varepsilon)$-approximating Closest String also holds for the more general Closest Substring problem. This yields a significantly stronger lower bound than the previous $f(\varepsilon) \cdot n^{o(\log(1/\varepsilon))}$ lower bound of Marx [12].

Our proof of Theorem 1.1 follows the methodology proposed Marx [13] and used by Boucher et al. [3] for Consensus Patterns. The following theorem, which is the main technical contribution of this work, states formally the properties of our reduction.

▶ **Theorem 1.2.** *There is an integer* c *and an algorithm that, given an instance* (G, k) *of* Clique*, works in time* $2^k \cdot n^{\mathcal{O}(1)}$ *and outputs an instance* $(\mathcal{S}, L, d)$ *of* Closest String *over alphabet* $\{0,1\}$ *with the following properties:*

- *If* G *contains a clique on* k *vertices, then there is a string* $w \in \{0,1\}^L$ *such that* $\mathcal{H}(w, x) \leq d$ *for each* $x \in \mathcal{S}$*.*

- *If G does not contain a clique on k vertices, then for each string $w \in \{0,1\}^L$ there is $x \in \mathcal{S}$ such that $\mathcal{H}(w,x) > (1+\frac{1}{ck}) \cdot d$.*

The statement of Theorem 1.2 is similar to the core of the hardness proof of Boucher et al. [3]. However, our reduction is completely different from the reduction of Boucher et al., because the causes of the computational hardness of Closest String and Consensus Patterns are quite orthogonal to each other. In Consensus Patterns the difficulty lies in *picking* the right substrings of the input strings. Once these substrings are known the center string is easily computed in polynomial time, since we are minimizing the sum of the Hamming distances. In Closest String there are no substrings to pick, we just have to find a center string for the given input strings. This is a computationally hard task because we are minimizing the maximum of the Hamming distances to the center, rather than the sum.

Theorem 1.1 follows immediately by combining Theorem 1.2 with the known parameterized hardness results for Clique, gathered in the following theorem, and setting $\varepsilon = \frac{1}{ck}$.

▶ **Theorem 1.3** (cf. Theorem 13.25 and Corollary 14.23 of [5]). *The following assertions hold:*

- *Unless* FPT = W[1], Clique *cannot be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$ for any computable function f.*
- *Unless ETH fails,* Clique *cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f.*

The main idea of the proof of Theorem 1.2 is to encode the n vertices of the given graph G as an "almost orthogonal" family $\mathcal{T}$ of strings from $\{0,1\}^\ell$, for some $\ell = \mathcal{O}(\log n)$. Strings from $\mathcal{T}$ are used as identifiers of vertices of G, and the fact that they are almost orthogonal means that the identifiers of two distinct vertices of G differ on approximately $\ell/2$ positions. On the other hand $\ell = \mathcal{O}(\log n)$, so the whole space of strings into which $V(G)$ is embedded has size polynomial in n. Using these properties, the reduction promised in Theorem 1.2 is designed by a careful construction.

Notation

By $\log p$ we denote the base-2 logarithm of p. For a positive integer n, we denote $[n] = \{1,2,\ldots,n\}$. The length of a string x is denoted by $|x|$. For an alphabet Σ and two equal-length strings x, y over Σ, the *Hamming distance* between x and y, denoted $\mathcal{H}(x,y)$, is the number of positions on which x and y have different letters. If $\Sigma = \{0,1\}$ is the binary alphabet, then the *Hamming weight* of a string x over Σ, denoted $\mathcal{H}(x)$, is the number of 1s in it. The *complement* of a string x over a binary alphabet, denoted $\overline{x}$, is obtained from x by replacing all 0s with 1s and vice versa. Note that if $|x| = |y| = n$, then $\mathcal{H}(x,y) = n - \mathcal{H}(\overline{x},y) = n - \mathcal{H}(x,\overline{y}) = \mathcal{H}(\overline{x},\overline{y})$.

2 Selection gadget

For the rest of this paper, we fix the following constants: $\rho = 1/100$, $\alpha = 1/10$, $\beta = 1/20$. Instead of giving a set of constraints for ρ, α, and β which are satisfied by a range or assignments we decided to use this particular numerical examples to make the proof easier to follow.

In the proof we will set $\ell = C \cdot \lceil \log n \rceil$ for some large integer C divisible by 100; this will ensure that $\rho\ell$, $\alpha\ell$, and $\beta\ell$ are all integers. First, we prove that among binary strings of logarithmic length one can find a linearly-sized family of "almost orthogonal" strings of

balanced Hamming weight, which will be later used in the reduction to represent vertices of a Clique instance. The proof is by a simple greedy argument.

▶ **Lemma 2.1.** *There exist positive integers C and N, where C is divisible by 100, with the following property. Let $n > N$ be any integer, and let us denote $\ell = C \cdot \lceil \log n \rceil$. Then there exists a set $\mathcal{T} \subseteq \{0,1\}^\ell$ with the following properties:*
1. $|\mathcal{T}| = n$,
2. $\mathcal{H}(x) = \ell/2$ *for each* $x \in \mathcal{T}$, *and*
3. $(1/2 - \rho)\ell < \mathcal{H}(x,y) < (1/2+\rho)\ell$ *for each distinct* $x, y \in \mathcal{T}$.

Moreover, given n, $\mathcal{T}$ can be constructed in time polynomial in n.

Proof. Let $H_2(\cdot)$ denote the binary entropy, i.e., $H_2(p) = -p\log p - (1-p)\log(1-p)$ for $p \in (0,1)$. Since ℓ is some positive integer divisible by 100. Then it is well known that

$$\sum_{i=0}^{k} \binom{\ell}{i} \leq 2^{\ell \cdot H_2(k/\ell)} \tag{1}$$

for all integers k with $0 < k \leq \ell/2$; cf. [6, Lemma 16.19]. Let us denote

$$A = \sum_{i=0}^{(1/2-\rho)\ell} \binom{\ell}{i} + \sum_{i=(1/2+\rho)\ell}^{\ell} \binom{\ell}{i}.$$

Then from (1) it follows that

$$A \leq 2 \cdot 2^{\sigma\ell},$$

where $\sigma = H_2(1/2-\rho) < 1$.

Suppose now that $\ell = C \cdot \lceil \log n \rceil$ for some positive integers C and $n > 1$, where C is divisible by 100. Then

$$\begin{aligned} n(\ell+1)\cdot A &\leq 2n \cdot (C\lceil \log n \rceil + 1) \cdot 2^{\sigma \cdot C \lceil \log n \rceil} \\ &\leq 2 \cdot (2C+1) \cdot 2^{\sigma C} \cdot n \log n \cdot 2^{\sigma \cdot C \log n} \\ &\leq (4C+2) \cdot 2^{\sigma C} \cdot n^{\sigma C + 2}. \end{aligned}$$

Since $\sigma < 1$, we can choose C to be an integer divisible by 100 so that $\sigma C + 2 < C$. Then, we can choose N large enough so that

$$(4C+2) \cdot 2^{\sigma C} \cdot n^{\sigma C + 2} \leq n^C$$

for all integers $n > N$. Hence,

$$nA \leq \frac{n^C}{\ell + 1}. \tag{2}$$

We now verify that this choice of C, N satisfies the required properties.

Consider the following greedy procedure performed on $\{0,1\}^\ell$. Start with $\mathcal{T} = \emptyset$ and all strings of $\{0,1\}^\ell$ marked as unused. In consecutive rounds perform the following:
1. Pick any $x \in \{0,1\}^\ell$ with $\mathcal{H}(x) = \ell/2$ that was not yet marked as used, and add x to $\mathcal{T}$.
2. Mark every $y \in \{0,1\}^\ell$ with $\mathcal{H}(x,y) \leq (1/2-\rho)\ell$ or $\mathcal{H}(x,y) \geq (1/2+\rho)\ell$ as used.

It is clear that at each step of the procedure, the constructed family $\mathcal{T}$ satisfies properties (2) and (3) from the lemma statement. Hence, it suffices to prove that the procedure can be performed for at least n rounds.

Note that the number of strings marked as used at each round is at most A. On the other hand, if $\mathcal{D}$ is the set of strings from $\{0,1\}^\ell$ that have Hamming weight exactly $\ell/2$, then

$$|\mathcal{D}| \geq \frac{|\{0,1\}^\ell|}{\ell+1} = \frac{2^{C\lceil \log n\rceil}}{\ell+1} \geq \frac{n^C}{\ell+1}.$$

From (2) we infer that $|\mathcal{D}| \geq nA$. This means that the algorithm will be able to find an unmarked $x \in \mathcal{D}$ for at least n rounds, and hence to construct the family $\mathcal{T}$ with $|\mathcal{T}| = n$. It is easy to implement the algorithm in polynomial time using the fact that the size of $\{0,1\}^\ell$ is polynomial in n. ◀

From now on, we adopt the constants C, N given by Lemma 2.1 to the notation. Let us also fix $n > N$; then let $\ell = C \cdot \lceil \log n \rceil$ and $\mathcal{T}$ be the set of strings given by Lemma 2.1, which we shall call *selection strings*. We define the set of *forbidden strings* $\mathcal{F} = \mathcal{F}(\mathcal{T})$ as follows:

$$\mathcal{F} = \{y\colon y \in \{0,1\}^\ell \text{ and } \mathcal{H}(x,y) \leq (1-\alpha)\ell \text{ for all } x \in \mathcal{T}\}.$$

In other words, $\mathcal{F}$ comprises all the strings that are not almost diametrically opposite to some string from $\mathcal{T}$. The following lemma asserts the properties of $\mathcal{T}$ and $\mathcal{F}$ that we shall need later on.

▶ **Lemma 2.2.** *Suppose $u \in \{0,1\}^\ell$. Then the following assertions hold:*
1. *If $u \in \mathcal{T}$, then $\mathcal{H}(u,y) \leq (1-\alpha)\ell$ for each $y \in \mathcal{F}$.*
2. *If $\mathcal{H}(x,u) \geq \beta\ell$ for all $x \in \mathcal{T}$, then there exists $y \in \mathcal{F}$ such that $\mathcal{H}(u,y) \geq (1-\beta)\ell$.*

Proof. Property (1) follows directly from the definition of $\mathcal{F}$, so we proceed to the proof of (2).

Suppose $\mathcal{H}(u,x) \geq \beta\ell$ for all $x \in \mathcal{T}$. If $\overline{u} \in \mathcal{F}$, then we could take $y = \overline{u}$, so suppose that $\overline{u} \notin \mathcal{F}$. This means that there exists $x_0 \in \mathcal{T}$, for which $\mathcal{H}(x_0,\overline{u}) > (1-\alpha)\ell$; equivalently, $\mathcal{H}(\overline{x_0},\overline{u}) < \alpha\ell$. On the other hand, we have that $\mathcal{H}(x_0,u) \geq \beta\ell$, so also $\mathcal{H}(\overline{x_0},\overline{u}) \geq \beta\ell$. Construct y from $\overline{u}$ by taking any set of positions X of size $\beta\ell$ on which $\overline{u}$ and $\overline{x_0}$ have the same letters, and flipping the letters on these positions (replacing 0s with 1s and vice versa). Such a set of positions always exists because $\alpha + \beta < 1$. Then we have that $\mathcal{H}(\overline{x_0},y) = \mathcal{H}(\overline{x_0},\overline{u}) + \beta\ell$, which implies that

$$\alpha\ell = \beta\ell + \beta\ell \leq \mathcal{H}(\overline{x_0},y) < (\alpha+\beta)\ell.$$

We claim that $y \in \mathcal{F}$; suppose otherwise. Since $\mathcal{H}(\overline{x_0},y) \geq \alpha\ell$, then also $\mathcal{H}(x_0,y) \leq (1-\alpha)\ell$. As $y \notin \mathcal{F}$, there must exist some $x_1 \in \mathcal{T}$, $x_0 \neq x_1$, such that $\mathcal{H}(x_1,y) > (1-\alpha)\ell$; equivalently $\mathcal{H}(\overline{x_1},y) < \alpha\ell$. Hence, from the triangle inequality we infer that

$$\mathcal{H}(x_0,x_1) = \mathcal{H}(\overline{x_0},\overline{x_1}) \leq \mathcal{H}(\overline{x_0},y) + \mathcal{H}(y,\overline{x_1}) < (2\alpha+\beta)\ell.$$

This is a contradiction with the assumption that $\mathcal{H}(x_0,x_1) \geq (1/2-\rho)\ell$, which is implied by $x_0, x_1 \in \mathcal{T}$. Indeed, we have that $2\alpha+\beta = \frac{1}{4} < \frac{49}{100} = 1/2-\rho$.

Hence $y \in \mathcal{F}$. By definition we have that $\mathcal{H}(\overline{u},y) = \beta\ell$, which implies that $\mathcal{H}(u,y) = (1-\beta)\ell$. Thus, y satisfies the required properties. ◀

3 Main construction

In this section we provide the proof of Theorem 1.2. Let (G,k) be the input instance of Clique, let $n = |V(G)|$, and without loss of generality assume $k \leq n$. Let C, N be the

constants given by Lemma 2.1. We can assume that $n > N$, because otherwise the instance (G, k) can be solved in constant time. Let $\ell = C\lceil \log n \rceil$. We run the polynomial-time algorithm given by Lemma 2.1 that computes the set $\mathcal{T} \subseteq \{0,1\}^\ell$ of selection strings. Let $\mathcal{F} = \mathcal{F}(\mathcal{T})$ be the set of forbidden strings, as defined in Section 2. Note that $\mathcal{F}$ can be computed in polynomial time directly from the definition, due to $|\{0,1\}^\ell| = n^{\mathcal{O}(1)}$.

We now present the construction of the output instance $(\mathcal{S}, L, d)$ of Closest String. Set $L = k\ell + \gamma\ell$, where $\gamma = \rho + \alpha = \frac{11}{100}$, and partition the set $[L]$ of positions in strings of length L into $k+1$ *blocks*:

- k blocks B_i for $i \in [k]$ of length ℓ each, where $B_i = \{(i-1)\ell + 1, (i-1)\ell + 2, \ldots, i\ell\}$;
- special *balancing block* Γ of length $\gamma\ell$, where $\Gamma = \{k\ell + 1, k\ell + 2, \ldots, L\}$.

For $w \in \{0,1\}^L$ and a contiguous subset of positions X, by $w[X]$ we denote the substring of w formed by positions from X.

Let us first discuss the intuition. The choice the solution string makes on consecutive blocks B_i will encode a selection of a k-tuple of vertices in G. Vertices of G will be mapped one-to-one to strings from $\mathcal{T}$. The family of constraint strings $\mathcal{S}$ will consist of two subfamilies $\mathcal{S}_{\mathrm{sel}}$ and $\mathcal{S}_{\mathrm{adj}}$ with the following roles:

- Strings from $\mathcal{S}_{\mathrm{sel}}$ ensure that on each block B_i, the solution picks a substring that is close to some element of $\mathcal{T}$. The selection of this element encodes the choice of the ith vertex from the k-tuple.
- Strings from $\mathcal{S}_{\mathrm{adj}}$ verify that vertices of the chosen k-tuple are pairwise different and adjacent, and hence they form a clique.

A small technical caveat is that for strings from $\mathcal{S}_{\mathrm{sel}}$ and from $\mathcal{S}_{\mathrm{adj}}$, the intended Hamming distance from the solution string will be slightly different. The role of the balancing block Γ is to equalize this distance by a simple additional construction.

We proceed to the formal description. Since $|V(G)| = |\mathcal{T}|$, let $\iota \colon V(G) \to \mathcal{T}$ be an arbitrary bijection.

The family $\mathcal{S}_{\mathrm{sel}}$ consists of strings $a(i, y, \phi, z)$, for all $i \in [k]$, $y \in \mathcal{F}$, ϕ being a function from $[k] \setminus \{i\}$ to $\{0,1\}$, and z being a binary string of length $\gamma\ell$. String $a(i, y, \phi, z)$ is constructed as follows:

- On block B_i put the string y.
- For each $j \in [k] \setminus \{i\}$, on block B_j put a string consisting of ℓ zeroes if $\phi(j) = 0$, and a string consisting of ℓ ones if $\phi(j) = 1$.
- On balancing block Γ put the string z.

Thus, $|\mathcal{S}_{\mathrm{sel}}| = k \cdot |\mathcal{F}| \cdot 2^{k-1} \cdot 2^{\gamma\ell} \leq 2^k \cdot n^{\mathcal{O}(1)}$; here and in some later estimates we use that $k \leq n$. Also, $\mathcal{S}_{\mathrm{sel}}$ can be constructed in time $2^k \cdot n^{\mathcal{O}(1)}$ directly from the definition.

The family $\mathcal{S}_{\mathrm{adj}}$ consists of strings $b(i, j, (u, v), \psi)$, for all $i, j \in [k]$ with $i < j$, (u, v) being an ordered pair of vertices of G that are either equal or non-adjacent, and ψ being a function from $[k] \setminus \{i, j\}$ to $\{0,1\}$. String $b(i, j, (u, v), \psi)$ is constructed as follows:

- On block B_i put the string $\overline{\iota(u)}$.
- On block B_j put the string $\overline{\iota(v)}$.
- On block B_q, for $q \in [k] \setminus \{i, j\}$, put a string consisting of ℓ zeroes if $\psi(q) = 0$, and a string consisting of ℓ ones if $\psi(q) = 1$.
- On balancing block Γ put a string consisting of $\gamma\ell$ zeroes.

Thus, $|\mathcal{S}_{\mathrm{adj}}| \leq \binom{k}{2} \cdot n^2 \cdot 2^{k-2} \leq 2^k \cdot n^{\mathcal{O}(1)}$. Again, $\mathcal{S}_{\mathrm{adj}}$ can be constructed in time $2^k \cdot n^{\mathcal{O}(1)}$ directly from the definition.

Set $\mathcal{S} = \mathcal{S}_{\mathrm{sel}} \cup \mathcal{S}_{\mathrm{adj}}$ and $d = (k/2 + 1/2 + \rho) \cdot \ell$. This concludes the construction. Its correctness will be verified in two lemmas that mirror the properties listed in Theorem 1.2.

▶ **Lemma 3.1.** *If G contains a clique on k vertices, then there exists a string $w \in \{0,1\}^L$ such that $\mathcal{H}(w,x) \leq d$ for each $x \in \mathcal{S}$.*

Proof. Let $\{c_1, c_2, \ldots, c_k\}$ be a k-clique in G. Construct w by putting $\iota(c_i)$ on block B_i, for each $i \in [k]$, and zeroes on all the positions of the balancing block Γ.

First, take any string $a = a(i, y, \phi, z) \in \mathcal{S}_{\mathrm{sel}}$. Since $\iota(c_i) \in \mathcal{T}$ and $y \in \mathcal{F}$, by Lemma 2.2(1) we infer that $\mathcal{H}(w[B_i], a[B_i]) = \mathcal{H}(\iota(c_i), y) \leq (1-\alpha)\ell$. For each $j \in [k] \setminus \{i\}$, since $\mathcal{H}(\iota(c_j)) = \ell/2$ due to $\iota(c_j) \in \mathcal{T}$, we have that $\mathcal{H}(w[B_j], a[B_j]) = \mathcal{H}(\iota(c_j), a[B_j]) = \ell/2$, regardless of the value of $\phi(j)$. Finally, obviously $\mathcal{H}(w[\Gamma], a[\Gamma]) \leq |\Gamma| = \gamma\ell$. Hence

$$\mathcal{H}(w,a) \leq (1-\alpha)\ell + (k-1)\ell/2 + \gamma\ell = d.$$

Second, take any string $b = b(i, j, (u, v), \psi) \in \mathcal{S}_{\mathrm{adj}}$. Since c_i and c_j are different and adjacent, whereas u and v are either equal or non-adjacent, we have $(c_i, c_j) \neq (u, v)$. Without loss of generality suppose that $c_i \neq u$; the second case will be symmetric. Then $\mathcal{H}(w[B_i], b[B_i]) = \mathcal{H}(\iota(c_i), \overline{\iota(u)}) \leq (1/2+\rho)\ell$, due to property (3) of Lemma 2.1. Obviously, $\mathcal{H}(w[B_j], b[B_j]) \leq |B_j| \leq \ell$. Finally, for every $q \in [k] \setminus \{i, j\}$ we have that $\mathcal{H}(\iota(c_q)) = \ell/2$, and hence $\mathcal{H}(w[B_q], b[B_q]) = \mathcal{H}(\iota(c_q), b[B_q]) = \ell/2$, regardless of the value of $\psi(q)$. Strings w and b match on positions of Γ, so $\mathcal{H}(w[\Gamma], b[\Gamma]) = 0$. Summarizing,

$$\mathcal{H}(w,b) \leq (1/2+\rho)\ell + \ell + (k-2)\ell/2 = d. \qquad \blacktriangleleft$$

▶ **Lemma 3.2.** *If there is a string $w \in \{0,1\}^L$ such that $\mathcal{H}(w,x) < d + \beta\ell$ for each $x \in \mathcal{S}$, then G contains a clique on k vertices.*

Proof. We first prove that on each block B_i, w is close to selecting an element of $\mathcal{T}$.

▶ **Claim 3.3.** *For each $i \in [k]$ there exists a unique $x_i \in \mathcal{T}$ such that $\mathcal{H}(w[B_i], x_i) < \beta\ell$.*

Proof. Uniqueness follows directly from property (3) of Lemma 2.1 and the triangle inequality, so it suffices to prove existence.

Let $u = w[B_i]$. For the sake of contradiction, suppose $\mathcal{H}(u,x) \geq \beta\ell$ for each $x \in \mathcal{T}$. From Lemma 2.2(2) we infer that there exists $y \in \mathcal{F}$ such that $\mathcal{H}(u,y) \geq (1-\beta)\ell$. Let us take $\phi\colon [k]\setminus\{i\} \to \{0,1\}$ defined as follows: $\phi(j) = 0$ if in w the majority of positions of B_j contain a one, and $\phi(j) = 1$ otherwise. Also, define $z = \overline{w[\Gamma]}$. Consider string $a = a(i, y, \phi, z) \in \mathcal{S}_{\mathrm{sel}}$. Then, it follows that

- $\mathcal{H}(w[B_i], a[B_i]) = \mathcal{H}(u,y) \geq (1-\beta)\ell$;
- $\mathcal{H}(w[B_j], a[B_j]) \geq \ell/2$ for each $j \in [k] \setminus \{i\}$;
- $\mathcal{H}(w[\Gamma], a[\Gamma]) = \mathcal{H}(w[\Gamma], \overline{w[\Gamma]}) = |\Gamma| = \gamma\ell$.

Consequently,

$$\mathcal{H}(w,a) \geq (1-\beta)\ell + (k-1)\ell/2 + \gamma\ell = d + \beta\ell.$$

This is a contradiction with the assumption that $\mathcal{H}(w,x) < d + \beta\ell$ for each $x \in \mathcal{S}$. $\blacktriangleleft$

For each $i \in [k]$, let $c_i = \iota^{-1}(x_i)$.

▶ **Claim 3.4.** *For all $i, j \in [k]$ with $i < j$, vertices c_i and c_j are different and adjacent.*

Proof. For the sake of contradiction, suppose c_i and c_j are either equal or non-adjacent. Define $\psi\colon [k] \setminus \{i,j\} \to \{0,1\}$ as follows: $\psi(q) = 0$ if in w the majority of positions of B_q contain a one, and $\psi(q) = 1$ otherwise. Then, for (c_i, c_j) we have constructed string $b = b(i, j, (c_i, c_j), \psi) \in \mathcal{S}_{\mathrm{adj}}$. Observe now that

- $\mathcal{H}(w[B_i], b[B_i]) = \mathcal{H}(w[B_i], \overline{x_i}) > (1-\beta)\ell$, since $\mathcal{H}(w[B_i], x_i) < \beta\ell$;
- Similarly, $\mathcal{H}(w[B_j], b[B_j]) > (1-\beta)\ell$;
- $\mathcal{H}(w[B_q], b[B_q]) \geq \ell/2$ for each $q \in [k] \setminus \{i, j\}$;
- $\mathcal{H}(w[\Gamma], b[\Gamma]) \geq 0$.

Consequently,

$$\mathcal{H}(w, b) \geq 2(1-\beta)\ell + (k-2)\ell/2 = (k/2 + 1 - 2\beta)\ell > d + \beta\ell.$$

This is a contradiction with the assumption that $\mathcal{H}(w, x) < d + \beta\ell$ for each $x \in \mathcal{S}$. ◀

Claim 3.4 asserts that, indeed, $\{c_1, c_2, \ldots, c_k\}$ is a k-clique in G. ◀

Lemmas 3.1 and 3.2 conclude the proof of Theorem 1.2, where c can be taken to be any constant larger than $\frac{d}{\beta\ell \cdot k} \leq \frac{2}{\beta} = 40$.

4 Conclusions

In this paper we have proved that Closest String does not have an EPTAS under the assumption of FPT $\neq$ W[1]. Moreover, under the stronger assumption of the Exponential Time Hypothesis, one can also exclude PTASes with running time $f(\varepsilon) \cdot n^{o(1/\varepsilon)}$, for any computable function f. However, the fastest currently known approximation scheme for Closest String has running time $n^{\mathcal{O}(1/\varepsilon^2)}$ [11]. This leaves a significant gap between the known upper and lower bounds. Despite efforts, we were unable to close this gap, and hence we leave it as an open problem.

References

1. Alexandr Andoni, Piotr Indyk, and Mihai Pătraşcu. On the optimality of the dimensionality reduction method. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 449–458. IEEE Computer Society, 2006.
2. Cristina Bazgan. *Schémas d'approximation et complexité paramétrée*. PhD thesis, Université Paris Sud, 1995. In French.
3. Christina Boucher, Christine Lo, and Daniel Lokshtanov. Consensus Patterns (probably) has no EPTAS. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 239–250. Springer, 2015. Full version available at `http://www.ii.uib.no/~daniello/papers/ConsensusPatterns.pdf`.
4. Marco Cesati and Luca Trevisan. On the efficiency of polynomial time approximation schemes. *Inf. Process. Lett.*, 64(4):165–171, 1997.
5. Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. URL: `http://dx.doi.org/10.1007/978-3-319-21275-3`, `doi:10.1007/978-3-319-21275-3`.
6. Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.
7. Jens Gramm, Rolf Niedermeier, and Peter Rossmanith. Fixed-parameter algorithms for Closest String and related problems. *Algorithmica*, 37(1):25–42, 2003.
8. Ming Li, Bin Ma, and Lusheng Wang. Finding similar regions in many sequences. *J. Comput. Syst. Sci.*, 65(1):73–96, 2002.

9 Ming Li, Bin Ma, and Lusheng Wang. On the Closest String and Substring problems. *J. ACM*, 49(2):157–171, 2002.

10 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 760–776. SIAM, 2011.

11 Bin Ma and Xiaoming Sun. More efficient algorithms for Closest String and Substring problems. *SIAM J. Comput.*, 39(4):1432–1443, 2009.

12 Dániel Marx. Closest substring problems with small distances. *SIAM J. Comput.*, 38(4):1382–1410, 2008.

13 Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008.

Coloring Graphs Having Few Colorings Over Path Decompositions*

Andreas Björklund

Department of Computer Science, Lund University, Lund, Sweden
andreas.bjorklund@yahoo.se

Abstract

Lokshtanov, Marx, and Saurabh SODA 2011 proved that there is no $(k-\epsilon)^{\mathrm{pw}(G)}\,\mathrm{poly}(n)$ time algorithm for deciding if an n-vertex graph G with pathwidth $\mathrm{pw}(G)$ admits a proper vertex coloring with k colors unless the Strong Exponential Time Hypothesis (SETH) is false, for any constant $\epsilon > 0$. We show here that nevertheless, when $k > \lfloor \Delta/2 \rfloor + 1$, where Δ is the maximum degree in the graph G, there is a better algorithm, at least when there are few colorings. We present a Monte Carlo algorithm that given a graph G along with a path decomposition of G with pathwidth $\mathrm{pw}(G)$ runs in $(\lfloor \Delta/2 \rfloor + 1)^{\mathrm{pw}(G)}\,\mathrm{poly}(n)s$ time, that distinguishes between k-colorable graphs having at most s proper k-colorings and non-k-colorable graphs. We also show how to obtain a k-coloring in the same asymptotic running time. Our algorithm avoids violating SETH for one since high degree vertices still cost too much and the mentioned hardness construction uses a lot of them.

We exploit a new variation of the famous Alon–Tarsi theorem that has an algorithmic advantage over the original form. The original theorem shows a graph has an orientation with outdegree less than k at every vertex, with a different number of odd and even Eulerian subgraphs only if the graph is k-colorable, but there is no known way of efficiently finding such an orientation. Our new form shows that if we instead count another difference of even and odd subgraphs meeting modular degree constraints at every vertex picked uniformly at random, we have a fair chance of getting a non-zero value if the graph has few k-colorings. Yet every non-k-colorable graph gives a zero difference, so a random set of constraints stands a good chance of being useful for separating the two cases.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Graph vertex coloring, path decomposition, Alon–Tarsi theorem

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.13

1 Introduction

One of the classical NP-hard problems on graphs is *proper* vertex k-coloring [14]: can you color the vertices from a palette of k colors such that each pair of vertices connected by an edge are colored differently? The problem has numerous applications in both theory and practice, for instance to model resource allocation.

A well-known algorithmic technique to attack such a challenging task for many graphs is to use dynamic programming over a graph decomposition. In this paper we consider one of the most common ones, namely the path decomposition introduced in the seminal work on graph minors by Robertson and Seymour [17]. Lokshtanov, Marx, and Saurabh [15] proved

* This research was supported in part by the Swedish Research Council grant VR 2012-4730 Exact Exponential–Time Algorithms.

that there cannot exist a $(k-\epsilon)^{\mathrm{pw}(G)}\,\mathrm{poly}(n)$ time algorithm for deciding if an n-vertex graph G with pathwidth $\mathrm{pw}(G)$ admits a proper vertex k-coloring for any $\epsilon > 0$ unless the Strong Exponential Time Hypothesis is false. Actually, they state their result in terms of the more general concept treewidth, but the result holds as well for pathwidth as pointed out in their Theorem 6.1 [1]. The Strong Exponential Time Hypothesis (SETH) [12] says that $s_\infty = 1$, where s_k is the infimum of all real values r for which there exists a $O(2^{rn})$ time algorithm that solves any n-variate k-SAT given in conjunctive normal form. In recent years many problems have been proven having known algorithms that are optimal under SETH, see e.g. [1, 3, 8, 15].

Indeed, for k-coloring nothing better than the natural $k^{\mathrm{pw}(G)}\,\mathrm{poly}(n)$ time algorithm that explicitly keeps track of all ways to color the presently active vertices is known for general graphs. However, the hardness construction from the result mentioned above uses many vertices of high degree and it would be interesting to understand to what extent this is necessary to enforce such strong lower bounds. To be precise, we believe that the conditional lower bound is strong evidence that there are no general k-coloring algorithm running significantly faster than the natural algorithm. Still, given that graph coloring is such an important topic that there are whole books devoted to the subject, e.g. [13], even small algorithmic improvements where none were found for decades may be worth noting.

To this end we consider coloring bounded degree graphs with many colors. If the maximum degree in the graph is Δ, it is trivial to find a $(\Delta+1)$-coloring by just coloring the vertices greedily in an arbitrary order. By Brook's theorem [6], one can also decide if there is a Δ-coloring in polynomial time. Reed [16] goes even further and shows that for large enough Δ, whenever there is no Δ-clique, there is a $(\Delta-1)$-coloring. In general though, this is already a difficult coloring problem as it is NP-hard to 3-color a graph of maximum degree 4 (follows from taking the line graph of the construction in [11]).

We present in this paper an algorithm that is faster than the natural one when the number of colors $k \geq \lfloor \Delta/2 \rfloor + 1$. However, we also need that the number of k-colorings isn't too large as our algorithm gets *slower* the more solutions there are. This counterintuitive behavior is symptomatic for the type of algorithm we use: we indirectly compute a fixed linear combination of all solutions and see if the result is non-zero. As the number of solutions increases, the number of ways that the solutions can annihilate each other also grows. Another example of such an algorithm (for directed Hamiltonian cycles) was recently given in [5].

Still, already the class of uniquely k-colorable graphs is a rich and interesting one [18]. One might suspect that it could be easier to find unique solutions. However, there are parsimonious reductions from Satisfiability to 3-coloring [4] (up to permutations of the colors), and we know that unique Satisfiability isn't easier than the general case [7], so it cannot be too much easier. In particular, we still should expect it to take exponential time.

[10] uses ideas related to the present work to algebraically classify uniquely colorable graphs. The proposed means to solve for them though involve computations of Gröbner bases, which are known to be very slow in the worst case, and the paper does not discuss worst case computational efficiency. Our contribution here is to find a variation of the Alon-Tarsi theorem [2], reusing the idea from [10] to look at the graph polynomial in points of powers of a primitive k:th root of unity, to get an efficient algorithm for solving a promise few k-coloring problem in bounded degree graphs. Our main theorem says:

[1] The Theorem says $(3-\epsilon)^{\mathrm{pw}(G)}$ but it is a misprint, it should be $(q-\epsilon)^{\mathrm{pw}(G)}$ in their notation.

▶ **Theorem 1.** *For every undirected graph G, and path decomposition of G of pathwidth* $\mathrm{pw}(G)$*, there is a* $(\lfloor \Delta/2 \rfloor + 1)^{\mathrm{pw}(G)} \mathrm{poly}(n)s$ *time Monte Carlo algorithm that outputs Yes with constant non-zero probability if G is k-colorable but has at most s proper k-colorings, and always outputs No if G is non-k-colorable.*

This means that for $k > \lfloor \Delta/2 \rfloor + 1$ and small s we improve exponentially over the natural $k^{\mathrm{pw}(G)} \mathrm{poly}(n)$ time algorithm. Still it does not violate the Strong Exponential Time Hypothesis lower bound from [15], since that construction uses $\Omega(n/\log k)$ vertices of degree much larger than $2k$.

By a simple self-reduction argument, we can also obtain a witness coloring for the k-colorable graphs:

▶ **Corollary 2.** *For every k-colorable graph G with s proper k-colorings, and a path decomposition of G of pathwidth* $\mathrm{pw}(G)$*, we can find a k-coloring in* $(\lfloor \Delta/2 \rfloor + 1)^{\mathrm{pw}(G)} \mathrm{poly}(n)s$ *time, with constant non-zero probability.*

1.1 The Alon–Tarsi theorem

Let $G = (V, E)$ be an undirected graph, and let k be a positive integer. An *orientation* of G is a directed graph $D = (V, A)$ in which each edge $uv \in E$ is given an orientation, i.e. either uv or vu is in A but not both. Denote by $\delta^+(A, v)$ and $\delta^-(A, v)$ the out- and indegree of the vertex v, respectively. A *Eulerian subgraph* of D is a subset $A' \subseteq A$ such that $\delta^+(A', v) = \delta^-(A', v)$ for all $v \in V$. Note that the notion of Eulerian here is somewhat non-standard as it does not require the subgraph to be connected. The subgraph is *even* if $|A'|$ is even and *odd* otherwise. The theorem of Alon and Tarsi says

▶ **Theorem 3** ([2])**.** *If there is an orientation $D = (V, A)$ of a graph G such that $\delta^+(A, v) < k$ for all vertices $v \in V$ for some integer k, and the number of even and odd Eulerian subgraphs of D differ, then G is k-colorable.*

The theorem gives no promise in the other direction though, but Hefetz [9] proved that if G is uniquely k-colorable with a minimal number of edges then there also exist orientations meeting the criteria of the theorem. However, there are as far as we know no known ways of efficiently finding such an orientation for a general uniquely k-colorable graph and hence any successful algorithm for k-coloring based on computing the difference of the number of even and odd Eulerian subgraphs for a fixed orientation seems aloof. Still, this is what our algorithm does, albeit after relaxing Eulerian subgraphs to something broader.

1.2 Our Approach

We denote by $[k]$ the set $\{0, 1, \cdots, k-1\}$. Let $\delta(A', v) = \delta^+(A', v) - \delta^-(A', v)$, i.e. be equal to the number of arcs in A' outgoing from v minus the arcs incoming to v. For a vector $w \in [k]^n$, which we also think of as a function $V \to [k]$, a *w-mod-k subgraph* is a subgraph with arc set $A' \subseteq A$ such that for all vertices $v \in V$, $\delta(A', v) \equiv w(v) (\bmod\ k)$.

Our algorithm is centered around a quantity $\kappa_{k,w}(A)$ that we define as the difference of the number of even w-mod-k subgraphs and the number of odd ones.

Our main technical lemma that may be of independent combinatorial interest says:

▶ **Lemma 4.** *Let an n-vertex graph G and positive integer k be given. If G has s proper k-colorings, then for any fixed orientation A of the edges and a vector $w \in [k]^n$ chosen uniformly at random, it holds that:*

1. $\mathrm{P}(\kappa_{k,w}(A) \neq 0) = 0$ *if* $s = 0$,
2. $\mathrm{P}(\kappa_{k,w}(A) \neq 0) \geq s^{-1}$ *if* $s > 0$.

Note that in this broad form the poor dependency on s is best possible: the empty n-vertex graph has $s = k^n$ proper k-colorings but $\kappa_{k,w}(\emptyset) \neq 0$ only for $w = 0$. Also note that for a uniquely k-colorable graph $s = k!$, so this is the smallest non-zero s we can get.

We will prove the Lemma in Section 2. It immediately suggests an algorithm for separating k-colorable graphs with few colorings from non-k-colorable graphs that we will use to prove Theorem 1. The proof is in Section 3. The algorithm is:

Decide-k-Colorable

1. Pick any orientation A of the edges.
2. Repeat $p(n,k)s$ times
3. Pick a vector $w \in [k]^n$ uniformly at random.
4. Compute $\kappa_{k,w}(A)$.
5. Output yes if $\kappa_{k,w}(A) \neq 0$ for any w, otherwise output no.

The function $p(n,k)$ depends on wether we just want to decide k-colorability or we also want to use the algorithm as a subroutine to find a k-coloring. The values of the function are set in Sec. 3.2 and 3.3, respectively. In step 4 we compute $\kappa_{k,w}(A)$ over a path decomposition. The key insight that makes this a faster algorithm than the natural one, is that we need to keep a much smaller state space when we count the w-mod-k subgraphs than if we were to keep track of all colors explicitly. For a fixed orientation and decomposition, the edges are considered in a certain predetermined order during the execution of a path decomposition dynamic programming. Hence we only need to store states that we know will stand the chance to result in a w-mod-k subgraph. To exemplify, say we are to count w-mod-5 subgraphs and a certain vertex v has $w(v) = 1$ and three incoming arcs and three outgoing arcs, and they are considered in order $++-+--$, where $+$ indicates an outgoing arc and $-$ an incoming. Now, when we have processed the first four of these, we only need to remember the partial solutions $A' \subseteq A$ that has $\delta(A', v) \in \{1, 2, 3\}$. It is possible to form partial solutions with $\delta(A', v) \in \{-1, 0\}$ as well, but the remaining two incoming arcs could never compensate for this imbalance to end up in a $\delta(A'', v) \equiv 1 (\mathrm{mod}\, 5)$ final state for some $A'' \supseteq A'$.

2 The Proof of Lemma 4

Let ω be a primitive k:th root of unity over the complex numbers. Consider an undirected graph $G = (V, E)$ on n vertices. For any directed graph $D = (V, B)$ where $uv \in E$ implies at least one of uv and vu to be in B, and $uv \in B$ implies $uv \in E$, we define a graph function on any vertex coloring $c : V \to [k]$ as

$$f_B(c) = \prod_{uv \in B} (1 - \omega^{c(u)-c(v)}).$$

Note that $f_B(c) \neq 0$ if and only if c is a proper k-coloring of G. Our previously defined difference $\kappa_{k,w}(B)$ of w-mod-k subgraphs are related to f_B through

▶ **Lemma 5.**

$$\kappa_{k,w}(B) = \frac{1}{k^n} \sum_{c \in [k]^n} \left(\prod_{v \in V} \omega^{-w(v)c(v)} \right) f_B(c). \tag{1}$$

Proof. We first observe that the right side of Eq. 1 can be rewritten as a summation over subgraphs by expanding the inner product, i.e.

$$\frac{1}{k^n}\sum_{c:V\to[k]}\prod_{v\in V}\omega^{-w(v)c(v)}\prod_{uv\in B}(1-\omega^{c(u)-c(v)}) = \frac{1}{k^n}\sum_{B'\subseteq B}(-1)^{|B'|}\sum_{c:V\to[k]}\prod_{v\in V}\omega^{(\delta(B',v)-w(v))c(v)}.$$

Consider a fixed subgraph $B' \subseteq B$. It contributes the term

$$\frac{1}{k^n}(-1)^{|B'|}\sum_{c:V\to[k]}\prod_{v\in V}\omega^{(\delta(B',v)-w(v))c(v)}.$$

First observe that if there is a u such that $k \not| (\delta(B',u)-w(u))$, then we can factor out u from the expression to get

$$\frac{1}{k^n}(-1)^{|B'|}\sum_{c_u\in[k]}\omega^{(\delta(B',u)-w(u))c_u}\sum_{c:V\setminus\{u\}\to[k]}\prod_{v\in V\setminus\{u\}}\omega^{(\delta(B',v)-w(v))c(v)}.$$

Let $l = \delta(B',u)-w(u)$ and note that $(\sum_{c_u\in[k]}\omega^{lc_u})(1-\omega^l) = (1-\omega^{lk}) = 0$. Since ω is primitive and $k \not| l$ we have that $(1-\omega^l)$ is non-zero. We conclude that $\sum_{c_u\in[k]}\omega^{lc_u} = 0$ and that such B' contributes zero to the right hand expression of Eq. 1.

Second when $k|(\delta(B',v)-w(v))$ for all $v \in V$, then the term $\prod_{v\in V}\omega^{(\delta(B',v)-w(v))c(v)}$ equals 1 regardless of c since ω is a k:th root of unity. Hence such subgraphs B' contributes the value $(-1)^{|B'|}$ after the division by the factor k^n to the right hand expression of Eq. 1. We are left with $\sum_{w\text{-mod-}k\text{ subgraph }B'\subseteq B}(-1)^{|B'|}$ as claimed. ◀

In particular it follows from the above lemma that if G is non-k-colorable, $\kappa_{k,w}(B) = 0$ for every $w \in [k]^n$ because $f_B(c) \equiv 0$ in this case. Hence with $B = A$ we have proved item a in Lemma 4.

To prove item b of the Lemma, we will associate three directed graphs on n vertices with G. First, let A be the fixed orientation of the edges in E in the formulation of the Lemma. Second, let $\overline{A}$ be the reversal of A, i.e. for each arc $uv \in A$, $vu \in \overline{A}$. Finally, let $C = A \cup \overline{A}$.

▶ **Lemma 6.**

$$\kappa_{k,0}(C) = \sum_{w\in[k]^n}\kappa_{k,w}(A)^2.$$

Proof. We first note that

$$\kappa_{k,0}(C) = \sum_{w\in[k]^n}\kappa_{k,w}(A)\kappa_{k,-w}(\overline{A}).$$

This is true because any even 0-mod-k graph in C is either composed of an even w-mod-k subgraph in A and an even $(-w)$-mod-k subgraph in $\overline{A}$ for some w, or is composed by two odd ones. Similarly, an odd 0-mod-k subgraph in C is composed by an even-odd or odd-even pair in A and $\overline{A}$ respectively for some w. Summing over all w we count all subgraphs. Next we note that $\kappa_{k,w}(A) = \kappa_{k,-w}(\overline{A})$ because $\delta(A,v) \equiv -\delta(\overline{A},v)(\text{ mod } k)$ for all $v \in V$. ◀

We will next use Lemma 5 to bound $|\kappa_{k,0}(C)|$ and $|\kappa_{k,w}(A)|$. We have

▶ **Lemma 7.**

$$\kappa_{k,0}(C) = \frac{1}{k^n}\sum_{c\in[k]^n}|f_A(c)|^2.$$

Proof. From Lemma 5 we have

$$\kappa_{k,0}(C) = \frac{1}{k^n} \sum_{c \in [k]^n} f_C(c).$$

Since $f_C(c) = f_A(c) f_{\overline{A}}(c)$ and $f_{\overline{A}}(c) = \overline{f_A(c)}$, we have that $f_C(c) = |f_A(c)|^2$ and the Lemma follows. ◀

▶ **Lemma 8.**

$$|\kappa_{k,w}(A)| \leq \frac{1}{k^n} \sum_{c \in [k]^n} |f_A(c)|.$$

Proof. From Lemma 5 we have

$$|\kappa_{k,w}(A)| = \frac{1}{k^n} \left| \sum_{c \in [k]^n} \left(\prod_{v \in V} \omega^{-w(v)c(v)} \right) f_A(c) \right|.$$

Since

$$\left| \sum_{c \in [k]^n} \left(\prod_{v \in V} \omega^{-w(v)c(v)} \right) f_A(c) \right| \leq \sum_{c \in [k]^n} \left(\prod_{v \in V} |\omega^{-w(v)c(v)}| \right) |f_A(c)|,$$

and $|\omega^{-w(v)c(v)}| = 1$ for every $w(v), c(v)$, the Lemma follows. ◀

Combining Lemmas 6, 7, and 8, we get

$$\frac{1}{k^n} \sum_{c \in S} |f_A(c)|^2 \leq \sum_{w \in T} \left(\frac{1}{k^n} \sum_{c \in S} |f_A(c)| \right)^2.$$

Here $S \subseteq [k]^n$ is the set of proper k-colorings and $T \subseteq [k]^n$ is the set of good w's, i.e. $w \in T$ if and only if $\kappa_{k,w}(A) \neq 0$. By assuming that $s > 0$ we can rewrite the above inequality as

$$|T| \geq \frac{\frac{1}{k^n} \sum_{c \in S} |f_A(c)|^2}{\frac{1}{k^{2n}} (\sum_{c \in S} |f_A(c)|)^2} \geq \frac{k^n}{|S|},$$

where the last inequality follows from Jensen's inequality, $\psi(\sum_{i=1}^t x_i/t) \leq \sum_{i=1}^t \psi(x_i)/t$ for a convex function ψ. Dividing $|T|$ by k^n gives the claimed probability bound in item b of Lemma 4. This concludes the proof of our main Lemma.

3 Details of the Algorithm

We will prove Theorem 1. We will first describe an algorithm that computes $\kappa_{k,w}(A)$ efficiently over a path decomposition and argue its correctness. Then we will prove Corollary 2 by showing how one with polynomial overhead can obtain a witness k-coloring.

3.1 The Path Decomposition Algorithm

Given a directed graph $H = (V, A)$ a *path decomposition* of H is a path graph $P = (U, F)$ where the vertices $U = \{u_1, \ldots, u_p\}$ represent subsets of V called *bags*, and the edges F simply connect u_i with u_{i+1} for every $i < p$. Every vertex $v \in V$ is associated with an

interval I_v on $\{1,\ldots,p\}$ such that $v \in u_i$ iff $i \in I_v$. Furthermore, for each arc $ab \in A$, there exists an i such that $\{a,b\} \subseteq u_i$. We set $r(ab) = i$ for the smallest such i. The *pathwidth*, denoted $\mathrm{pw}(H)$, is the minimum over all path decompositions of G of $\max_i(|u_i|-1)$. In a *nice* path decomposition, either a new vertex is added to u_i to form u_{i+1}, in which case we call u_{i+1} an *introduce* bag, or a vertex is removed, in which case we call u_{i+1} a *forget* bag. We can assume w.l.o.g. that we have a nice path decomposition since it is straightforward to make any path decomposition nice by simply extending the path with enough bags.

We will see how one can compute $\kappa_{k,w}(A)$ for a fixed orientation A over a path decomposition in step 4 in the algorithm **Decide-*k*-Colorable**. We impose an ordering of the m arcs, such that arc a_i precedes a_{i+1} if $r(a_i) < r(a_{i+1})$ or $r(a_i) = r(a_{i+1})$ and a_i is lexicographically before a_{i+1}. We will loop over the arcs in the above order, virtually moving over the bags from u_1 to u_p monotonically as necessary. For arc a_i, we let $D_{i,v}$ for every vertex $v \in u_{r(a_i)}$ denote every possible modular degree difference $\delta(A',v) \bmod k$ a vertex v can have in a subgraph $A' \subseteq \{a_j : j \leq i\}$ such that there still are enough arcs in $\{a_j : j > i\}$ to form a subgraph $A'', A \supseteq A'' \supseteq A'$ with $k|(\delta(A'',v) - w(v))$. In particular, every w-mod-k subgraph A^* must have $\delta(A^* \cap \{a_j : j \leq i\}, v) \in D_{i,v}$ for every $v \in u_{r(a_i)}$ and i.

▶ **Lemma 9.** *For all i and $v \in u_{r(a_i)}$,*

$$|D_{i,v}| \leq \lfloor \Delta/2 \rfloor + 1.$$

Proof. Let $D^{\text{before}}_{i,v}$ be the set of possible modular difference degrees $\delta(A',v) \bmod k$ for any $A' \subseteq \{a_j : j \leq i\}$, and let $D^{\text{after}}_{i,v}$ be the set of possible negated difference degrees $(w(v) - \delta(A'',v)) \bmod k$ for any $A'' \subseteq \{a_j : j > i\}$. Observe that $D_{i,v} = D^{\text{before}}_{i,v} \cap D^{\text{after}}_{i,v}$. If the number of arcs incident to v in $\{a_j : j \leq i\}$ is d^{before}_v, and the ones in $\{a_j : j > i\}$ is d^{after}_v, we have that $|D^{\text{before}}_{i,v}| \leq d^{\text{before}} + 1$ and $|D^{\text{after}}_{i,v}| \leq d^{\text{after}} + 1$. Since $\Delta \geq d^{\text{before}}_v + d^{\text{after}}_v$, the bound follows. ◀

Our algorithm tabulates for each possible modular degree difference in $D_{i,v}$ for each $v \in u_{r(a_i)}$, the difference of the number of even and odd subgraphs in $\{a_j : j \leq i\}$ matching the degree constraints on those vertices, while having modular degree difference equal to $w(v)$ on every vertex v that are forgotten by the algorithm, i.e. vertices v that were abandoned in a forget bag u_j for $j < r(a_i)$. That is, the complete state is described by a function $s_i : D_{i,v_1} \times \cdots \times D_{i,v_l} \to \mathbf{Z}$, where $\{v_1,\ldots,v_l\} = u_{r(a_i)}$, and where s_i for a specific difference degree vector holds the above difference of the number of even and odd subgraphs. It is easy to compute s_{i+1} from s_i, since each point in s_{i+1} depends on at most two points of s_i (either we use the arc a_{i+1} in our partial w-mod-k subgraph, or we do not). To compute the new value we just subtract the two old in reversed order, i.e. with an abuse of notation $s_{i+1}(d) = s_i(d) - s_i(d - a_{i+1})$. We initialize s_0 to all-zero except for $s_0(0) = 1$ since the empty subgraph is an even 0-mod-k subgraph. We store s_i in an array sorted after the lexicographically order on the (Cartesian product) keys which allows for quick access when we construct s_{i+1}. We only need to precompute $D_{i,v}$ for all i and v which is easily done in polynomial time, in order to see what modular degree differences to consider for s_{i+1} and what is stored in the previous function table s_i. Once we have computed s_m, we can read off $\kappa_{k,w}(A)$ from $s_m(w)$.

3.2 Runtime and Correctness Analysis

We finish the proof of Theorem 1. It follows from the bound in Lemma 9 that step 4 of the algorithm takes $(\lfloor \Delta/2 \rfloor + 1)^{\mathrm{pw}(G)}\,\mathrm{poly}(n)$ time. It is executed $p(n,k)s$ times so that we in

expectation sample $p(n,k)$ good w's for which $\kappa_{k,w}(A) \neq 0$ for k-colorable graphs having at most s proper k-colorings, as seen from item b in Lemma 4. From Markov's inequality, the probability of false negatives is at most $\frac{1}{p(n,k)}$. Thus already $p(n,k) = 2$ will do. From item a in Lemma 4 the probability of false positives is zero.

3.3 Coloring a Graph

We proceed with the proof of Corollary 2. The idea for recovering a witness k-coloring is to use self-reduction, i.e. to use algorithm **Decide-k-Colorable** many times for several graphs obtained by modifying G so as to gradually learn more and more of the vertices' colors. We will learn a coloring one color a time. That is, we will find a maximal subset of the vertices that can be colored in one color so that the remaining graph can be colored by the remaining $k-1$ colors.

For every subset $S \subseteq V$ of the vertices that form an independent set in G, we let G_S be the graph obtained by collapsing all vertices in S into a single supervertex v_S that retain all edges that goes to S in the original graph. Note that the number of colorings cannot increase by the contraction, and the degree is only increased for the newly formed vertex v_S, the rest of the graph is intact. Also note that the path width increases by at most one, since removing v_S from the graph leaves a subgraph of G. We use algorithm **Decide-k-Colorable** on G_S as a subroutine and note that the runtime is at most a factor k larger than it was on G due to the supervertex v_S that now may be in all bags and we need to keep track of all modular degree differences for this vertex. Our algorithm for extracting a color class is:

Find-Maximal-Color-Class

1. Let $S = v_1$.
2. For every vertex $u \in V \setminus v_1$,
3. If $S \cup u$ is an independent set and **Decide-k-Colorable($G_{S\cup\{u\}}$)** returns Yes,
4. Let $S = S \cup \{u\}$.
5. Return S.

With $p(n,k) = 2nk$ we get the true verdict on all queried graphs with probability at least $1 - \frac{1}{2k}$. Once we have found a maximal color class S, we can continue coloring the induced subgraph $G[V \setminus S]$ which is $(k-1)$-colorable by extracting a second color class and so on. We note that neither the maximum degree, pathwidth, or number of proper colorings can increase in an induced subgraph. In particular, a path decomposition for $G[V \setminus S]$ of pathwidth at most $\mathrm{pw}(G)$ can be readily obtained from the given path decomposition for G by just omitting the removed vertices and edges. By the union bound, with probability at least $\frac{1}{2}$ we correctly extract all k color classes.

4 Improvements and Limitations

The striking dependence on s in Theorem 1 is at least in part due to the poor uniform sampling employed. For many w's $\kappa_{k,w}(A)$ is zero for a trivial reason, namely that there are no w-mod-k subgraphs at all. A better idea would be to try to sample uniformly over all w's that has at least one w-mod-k-subgraph. We don't know how to do that, but it is probably still better to sample non-uniformly over this subset of good w's by uniformly picking a subgraph of G, and letting w be given by the degree differences of that subgraph. However, we can give an example of graphs where even a uniform sampling over the attainable w's will require running time that grows with s. Our construction is very simple, we just consider the

graph consisting of $n/3$ disjoint triangles. Let A be an orientation such that each vertex has one incoming and one outgoing arc. Then, the number of attainable w's is $7^{n/3}$ since every non-empty subset of the three arcs in a triangle gives a unique modular degree difference. The number of w's that give a non-zero $\kappa_{3,w}(A)$ is just $6^{n/3}$ which can be seen by inspecting each of the 7 attainable w's for a triangle, and noting that $\kappa_{k,w}(A' \cup A") = \kappa_{k,w'}(A')\kappa_{k,w"}(A")$ for vertex-disjoint arc subsets A' and $A"$. The number of 3-colorings s is also $6^{n/3}$, so with probability $s^{-0.086}$ we pick a w that has $\kappa_{3,w}(A) \neq 0$. While being a great improvement over s^{-1}, it still demonstrates a severe limitation of the technique presented in this paper when there are many solutions.

Acknowledgments. I thank several anonymous reviewers for comments on the paper.

References

1. A. Abboud and V. Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the IEEE FOCS*, pages 434–443, 2014.
2. N. Alon and M. Tarsi. Colorings and orientations of graphs. *Combinatorica*, 12:125–134, 1992.
3. A. Backurs and P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the ACM STOC*, pages 51–58, 2015.
4. R. Barbanchon. On unique graph 3-colorability and parsimonious reductions in the plane. *Theoretical Computer Science*, 319:455–482, 2004.
5. A. Björklund, H. Dell, and T. Husfeldt. The parity of set systems under random restrictions with applications to exponential time problems. In *Proceedings of ICALP*, pages 231–242, 2015.
6. R. L. Brooks. On colouring the nodes of a network. *Proc. Cambridge Philosophical Society, Math. Phys. Sci*, 37:194–197, 1941.
7. C. Calabro, R. Impagliazzo, V. Kabanets, and R. Paturi. The complexity of unique k-sat: An isolation lemma for k-cnfs. *Journal of Computer and System Sciences*, 74:386–393, 2008.
8. M. Cygan, S. Kratsch, and J. Nederlof. Fast hamiltonicity checking via bases of perfect matchings. In *Proceedings of the ACM STOC*, pages 301–310, 2013.
9. D. Hefetz. On two generalizations of the alon–tarsi polynomial method. *Journal of Combinatorial Theory Series B*, 101:403–414, 2011.
10. C. Hillar and T. Windfeldt. An algebraic characterization of uniquely vertex colorable graphs. *Journal of Combinatorial Theory Series B*, 98:400–414, 2008.
11. I. Holyer. The np-completeness of edge-coloring. *SIAM J. Comput.*, 10:718–720, 1981.
12. R. Impagliazzo and R. Paturi. The complexity of k-sat. In *Proceedings of the IEEE Computational Complexity Conference*, pages 237–240, 1999.
13. T. B. Jensen and B. Toft. *Graph Coloring Problems*. Wiley-Interscience, 1st edition, 1994.
14. R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.
15. D. Lokshtanov, D. Marx, and S. Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. In *Proceedings of ACM-SIAM SODA*, pages 777–789, 2011.
16. B. Reed. A strengthening of brook's theorem. *Journal of Combinatorial Theory Series B*, 76:136–149, 1999.
17. N. Robertson and P. Seymour. Graph minors. i. excluding a forest. *Journal of Combinatorial Theory Series B*, 35:39–61, 1983.
18. S. Xu. The size of uniquely colorable graphs. *Journal of Combinatorial Theory Series B*, 50:319–320, 1990.

Parameterized Algorithms for Recognizing Monopolar and 2-Subcolorable Graphs*

Iyad Kanj[1], Christian Komusiewicz[2], Manuel Sorge[3], and Erik Jan van Leeuwen[4]

1 **School of Computing, DePaul University, Chicago, USA**
ikanj@cs.depaul.edu
2 **Institut für Informatik, Friedrich-Schiller-Universität, Jena, Germany**
christian.komusiewicz@uni-jena.de
3 **Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Berlin, Germany**
manuel.sorge@tu-berlin.de
4 **Max-Planck-Institut für Informatik, Saarbrücken, Germany**
erikjan@mpi-inf.mpg.de

Abstract

We consider the recognition problem for two graph classes that generalize split and unipolar graphs, respectively. First, we consider the recognizability of graphs that admit a *monopolar partition*: a partition of the vertex set into sets A, B such that $G[A]$ is a disjoint union of cliques and $G[B]$ an independent set. If in such a partition $G[A]$ is a single clique, then G is a split graph. We show that in $\mathcal{O}(2^k \cdot k^3 \cdot (|V(G)| + |E(G)|))$ time we can decide whether G admits a monopolar partition (A, B) where $G[A]$ has at most k cliques. This generalizes the linear-time algorithm for recognizing split graphs corresponding to the case when $k = 1$.

Second, we consider the recognizability of graphs that admit a *2-subcoloring*: a partition of the vertex set into sets A, B such that each of $G[A]$ and $G[B]$ is a disjoint union of cliques. If in such a partition $G[A]$ is a single clique, then G is a unipolar graph. We show that in $\mathcal{O}(k^{2k+2} \cdot (|V(G)|^2 + |V(G)| \cdot |E(G)|))$ time we can decide whether G admits a 2-subcoloring (A, B) where $G[A]$ has at most k cliques. This generalizes the polynomial-time algorithm for recognizing unipolar graphs corresponding to the case when $k = 1$.

We also show that in $\mathcal{O}^*(4^k)$ time we can decide whether G admits a 2-subcoloring (A, B) where $G[A]$ and $G[B]$ have at most k cliques in total.

To obtain the first two results above, we formalize a technique, which we dub inductive recognition, that can be viewed as an adaptation of iterative compression to recognition problems. We believe that the formalization of this technique will prove useful in general for designing parameterized algorithms for recognition problems. Finally, we show that, unless the Exponential Time Hypothesis fails, no subexponential-time algorithms for the above recognition problems exist, and that, unless P=NP, no generic fixed-parameter algorithm exists for the recognizability of graphs whose vertex set can be bipartitioned such that one part is a disjoint union of k cliques.

1998 ACM Subject Classification F.2.0 Analysis of Algorithms and Problem Complexity, G.2.2 Graph Theory

Keywords and phrases vertex-partition problems, monopolar graphs, subcolorings, split graphs, unipolar graphs, fixed-parameter algorithms

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.14

* Iyad Kanj, Christian Komusiewicz, and Manuel Sorge gratefully acknowledge the support by the DFG, projects DAPA, NI 369/12 and MAGZ, KO 3669/4-1.

1 Introduction

A (Π_A, Π_B)-*graph*, for graph properties Π_A, Π_B, is a graph $G = (V, E)$ for which V admits a partition into two sets A, B such that $G[A]$ satisfies Π_A and $G[B]$ satisfies Π_B. There is an abundance of (Π_A, Π_B)-graph classes, and important ones include, in addition to bipartite graphs (*i.e.*, 2-colorable graphs), well-known graph classes such as *split graphs* (which admit a bipartition into a clique and an independent set), and *unipolar graphs* (which admit a bipartition into a clique and a cluster graph). Here a *cluster graph* is a disjoint union of cliques.

The problem of recognizing whether a given graph belongs to a particular class of (Π_A, Π_B)-graphs is called (Π_A, Π_B)-Recognition, and is known as a *vertex-partition* problem. In general, most recognition problems for (Π_A, Π_B)-graphs are NP-hard [11], but bipartite, split, and unipolar graphs can all be recognized in polynomial time [17, 13, 16, 10, 20]. With the aim of generalizing these polynomial-time algorithms, we study the complexity of recognizing two classes of (Π_A, Π_B)-graphs that generalize split and unipolar graphs.

First, we consider *monopolar graphs*; these are graphs in which the vertex set admits a bipartition into a cluster graph and an independent set, and thus generalize split graphs. Monopolar graphs have applications in the analysis of protein-interaction networks [3]. The recognition problem of monopolar graphs can be formulated as follows:

Monopolar Recognition
Input: A graph $G = (V, E)$.
Question: Does G have a *monopolar partition* (A, B), that is, can V be partitioned into sets A and B such that $G[A]$ is a cluster graph and $G[B]$ is an independent set?

Second, we study *2-subcolorable graphs*; these are graphs in which the vertex set admits a bipartition into two cluster graphs [2], and thus generalize unipolar graphs. The recognition problem of 2-subcolorable graphs can be formulated as follows:

2-Subcoloring
Input: A graph $G = (V, E)$.
Question: Does G have a *2-subcoloring* (A, B), that is, can V be partitioned into sets A and B such that each of $G[A]$ and $G[B]$ is a cluster graph?

Monopolar Recognition and 2-Subcoloring are both NP-hard [1, 11]. This is a stark contrast to the variants of both problems where $G[A]$ is required to consist of a single cluster, which correspond to the recognition of split graphs and unipolar graphs, respectively, and admit polynomial-time algorithms [13, 16, 10, 20]. This has left the complexity of Monopolar Recognition and 2-Subcoloring parameterized by the number of clusters in $G[A]$ as intriguing open questions.

Our Results. We show that both Monopolar Recognition and 2-Subcoloring are fixed-parameter tractable parameterized by the number of clusters in $G[A]$. More formally, let $G = (V, E)$ be a graph and k a nonnegative integer. We prove the following:

▶ **Theorem 1.1.** *In $\mathcal{O}(2^k \cdot k^3 \cdot (|V| + |E|))$ time, we can decide whether G admits a monopolar partition (A, B) such that $G[A]$ is a cluster graph with at most k clusters.*

Observe that the algorithm runs in linear time for any fixed k. In particular, the algorithm recognizes split graphs (the case $k = 1$) in linear time, matching the running time of the existing algorithm for this problem [13].

▶ **Theorem 1.2.** *In $\mathcal{O}(k^{2k+2} \cdot (|V|^2 + |V| \cdot |E|))$ time, we can decide whether G admits a 2-subcoloring (A, B) such that $G[A]$ is a cluster graph with at most k clusters.*

For both problems, one faces various technical difficulties when designing recognition algorithms due to the following: First, we parameterize by the number of clusters in $G[A]$ and not by the number of vertices. Second, the number of clusters or vertices of $G[B]$ can be arbitrary. In particular, 2-Subcoloring does not seem to yield to standard approaches in parameterized algorithms. To overcome these difficulties, we propose a technique, which we dub *inductive recognition*, that is an adaptation of iterative compression to recognition problems. We believe that the applications of this technique extend beyond the problems under consideration in this paper, and will have general use for designing parameterized algorithms for recognition problems.

Inductive recognition on a graph $G = (V, E)$ works as follows. Start with an empty graph G_0 that trivially belongs to the graph class. In iteration i, we recognize whether the subgraph G_i of G induced by the first i vertices of V still belongs to the graph class, given that G_{i-1} belongs to the graph class. This technique is very similar to the well-known iterative compression technique [18]. The crucial difference, however, is that in iterative compression we can always add the i-th vertex v_i to the solution from the previous iteration to obtain a new solution (which we compress if it is too large). However, in our problems, we cannot simply add v_i to one part and witness that G_i is still a member of the graph class with possibly too many clusters. For example, if we consider v_i with respect to a monopolar partition (A, B) of G_{i-1}, then potentially v_i could neighbor a vertex in B and vertices of two clusters in $G[A]$. Therefore, we cannot add v_i to A or B to obtain another monopolar partition, even if G_i is monopolar, and hence, we also cannot perform a "compression step". Instead, we must "repair" the solution by rearranging vertices. This idea is formalized in the inductive recognition framework in Section 3.

In the case of 2-Subcoloring, we also consider the weaker parameter of the total number of clusters in $G[A]$ and $G[B]$. This parameterization makes the problem amenable to a branching strategy that branches on the placement of the endpoints of suitably-chosen edges and nonedges of the graph. This way, we create partial 2-subcolorings (A', B') where each vertex in $V \setminus (A' \cup B')$ is adjacent to the vertices of exactly two partial clusters, one in each of $G[A']$ and $G[B']$. Then we show that whether such a partial 2-subcoloring extends to an actual 2-subcoloring of G can be tested in polynomial time via a reduction to 2-CNF-Sat.

▶ **Theorem 1.3** ($\star$[1])**.** *In $\mathcal{O}^*(4^k)$ time, we can decide whether G admits a 2-subcoloring (A, B) such that $G[A]$ and $G[B]$ are cluster graphs with at most k clusters in total.*

Finally, we consider the parameter consisting of the total number of vertices in $G[A]$. We observe that a straightforward branching strategy yields a generic fixed-parameter algorithm for many (Π_A, Π_B)-Recognition problems.

▶ **Proposition 1.4** ($\star$)**.** *Let Π_A and Π_B be two graph properties such that membership of Π_A can be decided in polynomial time and Π_B can be characterized by a finite set of forbidden induced subgraphs. Then we can decide in $\mathcal{O}^*(2^{\mathcal{O}(k)})$ time whether V can be partitioned into sets A and B such that $G[A] \in \Pi_A$, $G[B] \in \Pi_B$, and $|A| \leq k$.*

We observe that several possible improvements or generalizations of our results are unlikely. First, we notice that subexponential-time fixed-parameter algorithms for both Monopolar Recognition and 2-Subcoloring are unlikely.

[1] The proofs of the results marked with a "$\star$" are omitted due to the lack of space.

▶ **Proposition 1.5** (⋆). MONOPOLAR RECOGNITION *parameterized by the number k of clusters in $G[A]$ and* 2-SUBCOLORING *parameterized by the total number k of clusters in $G[A]$ and $G[B]$ cannot be solved in $\mathcal{O}^*(2^{o(k)})$ time, unless the Exponential Time Hypothesis fails.*

Second, observe that Theorems 1.1 and 1.2 give fixed-parameter algorithms for two (Π_A, Π_B)-RECOGNITION problems, in both of which Π_A defines the set of all cluster graphs, parameterized by the number of clusters in $G[A]$. Hence, one might hope for a generic fixed-parameter algorithm for such problems, irrespective of Π_B. However, polar graphs stand in our way. A graph $G = (V, E)$ has a *polar partition* if V can be partitioned into sets A and B such that $G[A]$ is a cluster graph and $G[B]$ is the complement of a cluster graph [21].

▶ **Proposition 1.6** (⋆). *It is NP-hard to decide whether G has a polar partition (A, B) such that $G[A]$ is a cluster graph with one cluster or $G[B]$ is a co-cluster graph with one co-cluster.*

Related Work. To the best of the authors' knowledge, the parameterized complexity of MONOPOLAR RECOGNITION and 2-SUBCOLORING has not been studied before. The known algorithms for both problems are not parameterized, and assume that the input graph belongs to a structured graph class; see [4, 5, 9, 15] and [2, 12, 19], respectively. Recently, Kolay and Panolan [14] considered the problem of deleting k vertices or edges to obtain an (r, ℓ)-graph. For integers r, ℓ, a graph $G = (V, E)$ is an *(r, ℓ)-graph* if V can be partitioned into r independent sets and ℓ cliques. For example, $(2, 0)$-graphs are precisely bipartite graphs and $(1, 1)$-graphs are precisely split graphs. However, observe that $(1, \cdot)$-graphs are *not* monopolar graphs, because monopolar graphs do not allow edges between the cliques (as $G[A]$ is a cluster graph), whereas such edges are allowed in $(1, \cdot)$-graphs. These differences lead to substantially different algorithmic techniques. For example, since Kolay and Panolan consider the deletion problem, they can use iterative compression in their work. Moreover, they consider $r, \ell < 3$, which makes even $n^{f(r,\ell)}$-time algorithms polynomial. Neither of these ideas works for the problems in this paper. We were, however, inspired by their Observation 2, which we adapt to our setting to help bound the running time of our algorithms.

2 Preliminaries

For the relevant notions from parameterized algorithms, we refer to the literature [8, 6]. We follow standard graph-theoretic notation [7]. Let G be a graph. By $V(G)$ and $E(G)$ we denote the vertex-set and the edge-set of G, respectively. For $X \subseteq V(G)$, $G[X]$ denotes the subgraph of G induced by X. For a vertex $v \in G$, $N(v)$ and $N[v]$ denote the open neighborhood and the closed neighborhood of v, respectively. For $X \subseteq V(G)$, we define $N(X) := (\bigcup_{v \in X} N(v)) \setminus X$ and $N[X] := \bigcup_{v \in X} N[v]$. We say that a vertex v is *adjacent to a subset $X \subseteq V(G)$ of vertices* if v is adjacent to at least one vertex in X. A P_3 is a path on 3 vertices. We repeatedly use the following well-known characterization of cluster graphs:

▶ **Fact 2.1.** *A graph is a cluster graph if and only if it contains no P_3 as an induced subgraph.*

The asymptotic notation $\mathcal{O}^*()$ suppresses a polynomial factor in the input length. For $\ell \in \mathbb{N}$, by $[\ell]$ we denote the set $\{1, \ldots, \ell\}$.

3 Foundations for Inductive Recognition

We describe the foundations of the general technique that we use to recognize monopolar and 2-subcolorable graphs. The technique works in a similar way to the iterative compression

technique by Reed *et al.* [18]. Let $\mathcal{G}$ be an arbitrary hereditary graph class (*i.e.* if $G \in \mathcal{G}$, then $G' \in \mathcal{G}$ for every induced subgraph G' of G). We call an algorithm $\mathcal{A}$ an *inductive recognizer* for $\mathcal{G}$ if given a graph $G = (V, E)$, a vertex $v \in V$ such that $G - v \in \mathcal{G}$, and a certificate for $G - v \in \mathcal{G}$, algorithm $\mathcal{A}$ correctly decides whether $G \in \mathcal{G}$ and gives a membership certificate if $G \in \mathcal{G}$.

▶ **Theorem 3.1** (⋆). *Given an inductive recognizer $\mathcal{A}$ for $\mathcal{G}$, we can recognize whether a given graph $G = (V, E)$ is a member of $\mathcal{G}$ in time $\mathcal{O}(|V| + |E|) + \sum_{i=1}^{|V|} T(i)$, where $T(i) > 0$ is the worst-case running time of $\mathcal{A}$ on a graph of size at most i.*

For the purpose of this paper, we consider *parameterized inductive recognizers*. In addition to G and v, these recognizers take a nonnegative integer k as input. The above general theorem can then be instantiated as follows.

▶ **Corollary 3.2** (⋆). *Let k be a nonnegative integer, and let Π_A and Π_B be two graph properties. Let $\mathcal{G}_k$ be a hereditary class of (Π_A, Π_B)-graphs with an arbitrary additional property that may depend on k.*

- *Given a parameterized inductive recognizer $\mathcal{A}$ for $\mathcal{G}_k$, we can recognize whether a given graph $G = (V, E)$ is a member of $\mathcal{G}_k$ in time $\mathcal{O}(|V| + |E|) + \sum_{i=1}^{|V|} T(i, k)$, where $T(i, k)$ is the worst-case running time of $\mathcal{A}$ with parameter k on a graph of size at most i.*
- *Given a parameterized inductive recognizer $\mathcal{A}$ for $\mathcal{G}_k$ that runs in time $f(k) \cdot \Delta$, where Δ is the maximum degree of the input graph and f is an arbitrary computable function, we can recognize whether a given graph $G = (V, E)$ is a member of $\mathcal{G}_k$ in time $f(k) \cdot (|V| + |E|)$.*

4 An FPT algorithm for Monopolar Recognition

In this section, we give an FPT algorithm for Monopolar Recognition parameterized by the number of clusters. Throughout, given a graph $G = (V, E)$ and a nonnegative integer k, we say that a monopolar partition (A, B) of G is *valid* if $G[A]$ is a cluster graph with at most k clusters. Using Corollary 3.2, it suffices to give a parameterized inductive recognizer for graphs with a valid monopolar partition. That is, we need to solve the following problem in time $f(k) \cdot \Delta$, where f is some computable function and Δ the maximum degree of G:

Inductive Monopolar Recognition
Input: A graph $G = (V, E)$, a vertex $v \in V$, and a valid monopolar partition (A', B') of $G' = G - v$.
Question: Does G have a valid monopolar partition (A, B)?

Fix an instance of Inductive Monopolar Recognition with a graph $G = (V, E)$, a vertex $v \in V$, and a valid monopolar partition (A', B') of $G' = G - v$.

To find a valid monopolar partition (A, B) of G, we try the two possibilities of placing v in A or placing v in B. More precisely, in one case, we start a search from the bipartition $(A' \cup \{v\}, B')$, and in the other case, we start a search from the bipartition $(A', B' \cup \{v\})$. Neither of these two partitions is necessarily a valid monopolar partition of G. The search strategy is to try to "repair" a candidate partition by moving few vertices from one part of the partition to the other part. During this process, if a vertex is moved from one part to the other, then it will never be moved back. To formalize this approach, we introduce the notion of constraints.

▶ **Definition 4.1.** A *constraint* $\mathcal{C} = (A^{\mathcal{C}}_*, A^{\mathcal{C}}_P, B^{\mathcal{C}}_*, B^{\mathcal{C}}_P)$ is a four-partition of V such that $A^{\mathcal{C}}_* \subseteq A'$ and $B^{\mathcal{C}}_* \subseteq B'$. The vertices in $A^{\mathcal{C}}_P$ and $B^{\mathcal{C}}_P$ are called *permanent* vertices of the constraint. A constraint $\mathcal{C} = (A^{\mathcal{C}}_*, A^{\mathcal{C}}_P, B^{\mathcal{C}}_*, B^{\mathcal{C}}_P)$ is *fulfilled* by a vertex bipartition (A, B) of G if (A, B) is a valid monopolar partition of G such that:

1. $A_P^{\mathcal{C}} \subseteq A$; and
2. $B_P^{\mathcal{C}} \subseteq B$.

The permanent vertices in $A_P^{\mathcal{C}}$ and $B_P^{\mathcal{C}}$ in the above definition will correspond to those vertices that were moved during the search from one part to the other part. Note that:

▶ **Fact 4.2.** *Each valid monopolar partition (A, B) of G fulfills either $(A', \{v\}, B', \emptyset)$ or $(A', \emptyset, B', \{v\})$.*

We call the two constraints in Fact 4.2 the *initial constraints* of the search. We solve Inductive Monopolar Recognition by giving a search-tree algorithm that determines for each of the two initial constraints whether there is a partition fulfilling it. The root of the search tree is a dummy node that has two children, associated with the two initial constraints. Each non-root node in the search tree is associated with a constraint $\mathcal{C}$, and the algorithm searches for a solution that fulfills $\mathcal{C}$. To this end, the algorithm applies reduction and branching rules that find vertices that in *every* valid monopolar partition (A, B) fulfilling $\mathcal{C}$ are in $A_*^{\mathcal{C}} \cap B$ or $B_*^{\mathcal{C}} \cap A$; that is, these vertices must "switch sides".

Formally, a *reduction rule* that is applied to a constraint $\mathcal{C}$ associated with a node α in the search tree associates α with a new constraint $\mathcal{C}'$ or rejects $\mathcal{C}$; the reduction rule is *correct* either if $\mathcal{C}$ has a fulfilling partition if and only if $\mathcal{C}'$ does, or if the rule rejects $\mathcal{C}$, then no valid monopolar partition of G fulfills $\mathcal{C}$. A *branching rule* applied to a constraint $\mathcal{C}$ associated with a node α in the search tree produces more than one child node of α, each associated with a constraint; the branching rule is *correct* if $\mathcal{C}$ has a fulfilling partition if and only if at least one of the child nodes of α is associated with a constraint $\mathcal{C}'$ that has a fulfilling partition.

The algorithm first performs the reduction rules exhaustively, in order, and then performs the branching rules, in order. That is, Reduction Rule i may only be applied if Reduction Rule i' for all $i' < i$ cannot be applied. In particular, after Reduction Rule i is applied, we start over and apply Reduction Rule 1, etc. The same principle applies to the branching rules; moreover, branching rules are only applied if no reduction rule can be applied.

Let $\mathcal{C} = (A_*^{\mathcal{C}}, A_P^{\mathcal{C}}, B_*^{\mathcal{C}}, B_P^{\mathcal{C}})$ be a constraint. We now describe the reduction rules. Bear in mind that cluster graphs contain no P_3 as an induced subgraph (Fact 2.1). The first reduction rule identifies obvious cases in which a constraint cannot be fulfilled.

▶ **Reduction Rule 4.3** ($\star$). *If $G[A_P^{\mathcal{C}}]$ is not a cluster graph with at most k clusters, or if $G[B_P^{\mathcal{C}}]$ is not an independent set, then reject the current constraint.*

The second reduction rule finds vertices that must be moved from $B_*^{\mathcal{C}}$ to $A_P^{\mathcal{C}}$.

▶ **Reduction Rule 4.4** ($\star$). *If there is a vertex $u \in B_*^{\mathcal{C}}$ that has a neighbor in $B_P^{\mathcal{C}}$, then set $A_P^{\mathcal{C}} \leftarrow A_P^{\mathcal{C}} \cup \{u\}$ and $B_*^{\mathcal{C}} \leftarrow B_*^{\mathcal{C}} \setminus \{u\}$; that is, replace $\mathcal{C}$ with the constraint $(A_*^{\mathcal{C}}, A_P^{\mathcal{C}} \cup \{u\}, B_*^{\mathcal{C}} \setminus \{u\}, B_P^{\mathcal{C}})$.*

The third reduction rule finds vertices that must be moved from $A_*^{\mathcal{C}}$ to $B_P^{\mathcal{C}}$.

▶ **Reduction Rule 4.5** ($\star$). *If there is a vertex $u \in A_*^{\mathcal{C}}$ and two vertices $w, x \in A_P^{\mathcal{C}}$ such that $G[\{u, w, x\}]$ is a P_3, set $A_*^{\mathcal{C}} \leftarrow A_*^{\mathcal{C}} \setminus \{u\}$ and $B_P^{\mathcal{C}} \leftarrow B_P^{\mathcal{C}} \cup \{u\}$.*

The first branching rule identifies pairs of vertices from $A_*^{\mathcal{C}}$ such that at least one of them must be moved to $B_P^{\mathcal{C}}$ because they form a P_3 with a vertex in $A_P^{\mathcal{C}}$.

▶ **Branching Rule 4.6** ($\star$). *If there are two vertices $u, w \in A_*^{\mathcal{C}}$ and a vertex $x \in A_P^{\mathcal{C}}$ such that $G[\{u, w, x\}]$ is a P_3, then branch into two branches, one associated with the constraint $(A_*^{\mathcal{C}} \setminus \{u\}, A_P^{\mathcal{C}}, B_*^{\mathcal{C}}, B_P^{\mathcal{C}} \cup \{u\})$ and one with constraint $(A_*^{\mathcal{C}} \setminus \{w\}, A_P^{\mathcal{C}}, B_*^{\mathcal{C}}, B_P^{\mathcal{C}} \cup \{w\})$.*

It is important to observe that if none of the previous rules applies, then $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$ is a monopolar partition (we prove this rigorously in Lemma 4.8). However, $G[A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P]$ may consist of too many clusters for this to be a *valid* monopolar partition. To check whether it is possible to reduce the number of clusters in $G[A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P]$, we apply a second branching rule that deals with singleton clusters in $G[A']$.

▶ **Branching Rule 4.7 ($\star$).** *If there is a vertex $u \in A^{\mathcal{C}}_*$ such that $\{u\}$ is a cluster in $G[A']$, then branch into two branches: the first is associated with the constraint $(A^{\mathcal{C}}_* \setminus \{u\}, A^{\mathcal{C}}_P \cup \{u\}, B^{\mathcal{C}}_*, B^{\mathcal{C}}_P)$, and the second is associated with the constraint $(A^{\mathcal{C}}_* \setminus \{u\}, A^{\mathcal{C}}_P, B^{\mathcal{C}}_*, B^{\mathcal{C}}_P \cup \{u\})$.*

If no more rules apply to a constraint $\mathcal{C}$, then we can determine whether $\mathcal{C}$ can be fulfilled:

▶ **Lemma 4.8.** *Let $\mathcal{C} = (A^{\mathcal{C}}_*, A^{\mathcal{C}}_P, B^{\mathcal{C}}_*, B^{\mathcal{C}}_P)$ be a constraint to which Reduction Rules 4.3, 4.4, and 4.5, and Branching Rules 4.6 and 4.7 do not apply. Then $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$ is a monopolar partition. Moreover, there is a valid monopolar partition (A, B) fulfilling $\mathcal{C}$ if and only if $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$ is valid.*

Proof. First, we show that $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$ is a monopolar partition. There are no induced P_3's in $G[A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P]$, because Reduction Rules 4.3 and 4.5 and Branching Rule 4.6 do not apply, and because there are no induced P_3's in G containing three vertices from $A^{\mathcal{C}}_* \subseteq A'$. Similarly, there are no edges in $G[B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P]$, because Reduction Rules 4.3 and 4.4 do not apply, and because there are no edges in $G[B']$ and $B^{\mathcal{C}}_* \subseteq B'$.

To show the second statement in the lemma, observe that, if $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$ is valid, then $\mathcal{C}$ is fulfilled by $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$. It remains to show that, if $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$ is not valid, then each monopolar partition (A, B) of G fulfilling $\mathcal{C}$ is not valid. For the sake of contradiction, assume that this is not the case and let (A, B) be a valid monopolar partition fulfilling $\mathcal{C}$. Since $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$ is a monopolar partition of G that is not valid, there are more than k clusters in $G[A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P]$. Thus, there is a cluster Q in $G[A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P]$ such that $Q \subseteq B$. Note that $|Q| = 1$, because $G[B]$ is an independent set and $Q \subseteq B$. Because (A, B) fulfills $\mathcal{C}$, $Q \cap A^{\mathcal{C}}_P = \emptyset$ and thus $Q \subseteq A^{\mathcal{C}}_*$. Hence, Q is a subset of a cluster Q' of $G[A']$, as $Q \subseteq A^{\mathcal{C}}_* \subseteq A'$. However, $|Q'| \geq 2$, because Branching Rule 4.7 does not apply even though $Q \subseteq A^{\mathcal{C}}_*$. Hence, any rule that moved the vertices of $Q' \setminus Q$ was not Branching Rule 4.7. Then the description of the other rules implies that $Q' \setminus Q \subseteq B^{\mathcal{C}}_P$. Note that $B^{\mathcal{C}}_P \subseteq B$, because (A, B) fulfills $\mathcal{C}$. Hence, $Q' \subseteq B$ and thus $G[B]$ is not an independent set. Therefore, (A, B) is not a monopolar partition, a contradiction to our choice of (A, B). ◀

The following lemmas will be used to upper bound the depth of the search tree, and the number of applications of each rule along each root-leaf path in this tree. Herein a *leaf* of the search tree is a node associated either with a constraint that Reduction Rule 4.3 rejects, or with a constraint to which no rule applies.

▶ **Lemma 4.9.** *Along any root-leaf path in the search tree of the algorithm, Reduction Rule 4.4 is applied at most $k + 1$ times.*

Proof. Let $\mathcal{C} = (A^{\mathcal{C}}_*, A^{\mathcal{C}}_P, B^{\mathcal{C}}_*, B^{\mathcal{C}}_P)$ be a constraint obtained from an initial constraint via $k+1$ applications of Reduction Rule 4.4 and an arbitrary number of applications of Reduction Rules 4.3 and 4.5, and Branching Rules 4.6 and 4.7. Each application of Reduction Rule 4.4 adds a vertex of B' to $A^{\mathcal{C}}_P$. Since $G[B']$ is an independent set, any monopolar partition (A, B) with $A^{\mathcal{C}}_P \subseteq A$ has at least $k + 1$ clusters in $G[A]$ and, therefore, is not valid. Reduction Rule 4.3 will then be applied before any further application of Reduction Rule 4.4, and the constraint $\mathcal{C}$ will be rejected. ◀

▶ **Lemma 4.10.** *Along any root-leaf path in the search tree of the algorithm, Reduction Rule 4.5 and Branching Rules 4.6 and 4.7 are applied at most $k+1$ times in total.*

Proof. Let $\mathcal{C} = (A^{\mathcal{C}}_*, A^{\mathcal{C}}_P, B^{\mathcal{C}}_*, B^{\mathcal{C}}_P)$ be a constraint obtained from an initial constraint via $k+1$ applications of Reduction Rule 4.5 and Branching Rules 4.6 and 4.7, and an arbitrary number of applications of the other rules. Let k_s denote the number of singleton clusters in $G[A']$. Observe that each application of Reduction Rule 4.5 or Branching Rules 4.6 and 4.7 makes a vertex of $A^{\mathcal{C}}_* \subseteq A'$ permanent by placing it in $A^{\mathcal{C}}_P$ or $B^{\mathcal{C}}_P$. By the description of all rules, a vertex will never be made permanent twice. Hence, out of the $k+1$ applications of Reduction Rule 4.5 and Branching Rules 4.6 and 4.7, at most k_s make the vertex from a singleton cluster of $G[A']$ permanent. Observe that Branching Rule 4.7 cannot make a vertex from a non-singleton cluster in $G[A']$ permanent. Thus, Reduction Rule 4.5 and Branching Rule 4.6 make at least $k-k_s+1$ vertices in the $k-k_s$ non-singleton clusters of $G[A']$ permanent. Since $k-k_s+1 \geq 1$, this also implies that a non-singleton cluster exists. By the pigeonhole principle, out of the $k-k_s+1$ vertices that are made permanent by Reduction Rule 4.5 and Branching Rule 4.6, two are from the same non-singleton cluster in $G[A']$. Since both Reduction Rule 4.5 and Branching Rule 4.6 only move vertices from $A^{\mathcal{C}}_*$ to $B^{\mathcal{C}}_P$, it follows that $B^{\mathcal{C}}_P$ contains two adjacent vertices. Then the constraint $\mathcal{C}$ will be rejected by Reduction Rule 4.3, which is applied before any further rule is applied. ◀

▶ **Theorem 4.11.** Inductive Monopolar Recognition *can be solved in $\mathcal{O}(2^k \cdot k^3 \cdot \Delta)$ time, where Δ is the maximum degree of G.*

Proof. We call a leaf of the search tree associated with a constraint to which no rule applies an *exhausted leaf*. By Lemma 4.8 and the correctness of the rules, G has a valid monopolar partition if and only if for at least one exhausted leaf node, the partition $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$, induced by the constraint $\mathcal{C}$ associated with that node, is a valid monopolar partition. Hence, if the search tree has an exhausted leaf for which the partition $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$, induced by the constraint $\mathcal{C}$ associated with that node, is a valid monopolar partition, the algorithm answers "yes"; otherwise, it answers "no". Therefore, the described search-tree algorithm correctly decides an instance of Inductive Monopolar Recognition.

To upper bound the running time, let $\mathcal{T}$ denote the search tree of the algorithm. By Lemma 4.10, Branching Rules 4.6 and 4.7 are applied at most $k+1$ times in total along any root-leaf path in $\mathcal{T}$. It follows that the depth of $\mathcal{T}$ is at most $k+2$. As each of the branching rules is a two-way branch, $\mathcal{T}$ is a binary tree, and thus the number of leaves in $\mathcal{T}$ is $\mathcal{O}(2^k)$.

The running time along any root-leaf path in $\mathcal{T}$ is dominated by the overall time taken along the path to test the applicability of the reduction and branching rules, and to apply them. By Lemma 4.9 and Lemma 4.10, along any root-leaf path in $\mathcal{T}$ the total number of applications of Reduction Rules 4.4 and 4.5 and Branching Rules 4.6 and 4.7 is $\mathcal{O}(k)$. Reduction Rule 4.3 is applied once before the application of each of the aforementioned rules. It follows that the total number of applications of all rules along any root-leaf path in $\mathcal{T}$ is $\mathcal{O}(k)$. Moreover, $\mathcal{T}$ has $\mathcal{O}(2^k)$ leaves as argued before. Therefore, we test for the applicability of the rules and apply them, or use the check of Lemma 4.8, at most $\mathcal{O}(2^k \cdot k)$ times. We next upper bound the time to test the applicability of the rules and to apply them by $\mathcal{O}(k^2 \cdot \Delta)$.

Let $\mathcal{C} = (A^{\mathcal{C}}_*, A^{\mathcal{C}}_P, B^{\mathcal{C}}_*, B^{\mathcal{C}}_P)$ be a constraint associated with a node in $\mathcal{T}$. Observe that each cluster in $G[A^{\mathcal{C}}_*]$ has size $\mathcal{O}(\Delta)$. Since $G[A^{\mathcal{C}}_*]$ has at most k clusters, this implies that $|A^{\mathcal{C}}_*| \leq k \cdot \Delta$. Thus, in $\mathcal{O}(k \cdot \Delta)$ time, we can compute a list of all clusters in $G[A^{\mathcal{C}}_*]$ and the size of each cluster. The same holds for $G[A']$. Observe that we can always check in $\mathcal{O}(1)$ time, for a given vertex v, whether v is contained in A', $A^{\mathcal{C}}_*$, $A^{\mathcal{C}}_P$, $B^{\mathcal{C}}_*$, or $B^{\mathcal{C}}_P$ and, in case v is contained in A' or $A^{\mathcal{C}}_*$, we can find the index and the size of the cluster that contains v.

Moreover, by Lemma 4.9 and 4.10, we can assume that $|A_P^{\mathcal{C}}| = \mathcal{O}(k)$, and by Lemma 4.10, we can assume that $|B_P^{\mathcal{C}}| = \mathcal{O}(k)$.

To test the applicability of Reduction Rules 4.3 and 4.4, we check whether $G[A_P^{\mathcal{C}}]$ is a cluster graph with at most k clusters, whether $G[B_P^{\mathcal{C}}]$ is an independent set, and whether there is an edge with one endpoint in $B_P^{\mathcal{C}}$ and the other endpoint in $B_*^{\mathcal{C}}$. This can be done in $\mathcal{O}(k \cdot \Delta)$ time since $|A_P^{\mathcal{C}}| = \mathcal{O}(k)$ and $|B_P^{\mathcal{C}}| = \mathcal{O}(k)$.

To test the applicability of Reduction Rule 4.5, we consider each pair v, w of vertices in $A_P^{\mathcal{C}}$. If v and w are adjacent, then in $\mathcal{O}(\Delta)$ time we can check whether there is a vertex $u \in A_*^{\mathcal{C}}$ such that u is adjacent to exactly one of v and w. If v and w are not adjacent, then in $\mathcal{O}(\Delta)$ time we can check whether they have a common neighbor in $A_*^{\mathcal{C}}$. If neither condition applies to any pair v, w, then Reduction Rule 4.5 does not apply. Overall, this test takes $\mathcal{O}(k^2 \cdot \Delta)$ time.

To test the applicability of Branching Rule 4.6, we can check for each vertex v of the at most k vertices of $A_P^{\mathcal{C}}$ in $\mathcal{O}(\Delta)$ time whether v has neighbors in two different clusters of $A_*^{\mathcal{C}}$, or whether there are two vertices u, w in the same cluster of $A_*^{\mathcal{C}}$ such that v is adjacent to u but not adjacent to w. If one of the two cases applies to some vertex $v \in A_P^{\mathcal{C}}$, then Branching Rule 4.6 applies to v. Otherwise, there is no P_3 containing exactly one vertex from $A_P^{\mathcal{C}}$ and exactly two vertices from $A_*^{\mathcal{C}}$, and Branching Rule 4.6 does not apply. Hence, the applicability of Branching Rule 4.6 can be tested in $\mathcal{O}(k \cdot \Delta)$ time.

To test the applicability of Branching Rule 4.7, we can check in $\mathcal{O}(k)$ time, whether $G[A_*^{\mathcal{C}}]$ contains a singleton cluster that is also a singleton cluster of $G[A']$.

All rules can trivially be applied in $\mathcal{O}(1)$ time if they were found to be applicable. Hence, the running time to test and apply any of the rules is $\mathcal{O}(k^2 \cdot \Delta)$.

Finally, if none of the rules applies, then we can check in $\mathcal{O}(k \cdot \Delta)$ time whether the number of clusters in $G[A_*^{\mathcal{C}} \cup A_P^{\mathcal{C}}]$ is at most k. Hence, the algorithm runs in $\mathcal{O}(2^k \cdot k^3 \cdot \Delta)$ time in total. ◀

Given the above theorem, Corollary 3.2 immediately implies Theorem 1.1.

5 An FPT algorithm for 2-Subcoloring

In this section, we give an FPT algorithm for 2-Subcoloring parameterized by the smallest number of clusters in the two parts. Although the general approach is similar to the approach used for Monopolar Recognition, in that it relies on the inductive recognition technique and the notion of constraints, the algorithm is substantially more complex. In particular, the notion of constraints and the reduction and branching rules are more involved, mainly due to the much more complicated structure of 2-subcolorable graphs.

Throughout, given a graph G and a nonnegative integer k, we call a 2-subcoloring (A, B) of G *valid* if $G[A]$ has at most k cliques. Using the inductive recognition approach, we need a parameterized inductive recognizer for the following problem:

Inductive 2-Subcoloring
Input: A graph $G = (V, E)$, a vertex $v \in V$, and a valid 2-subcoloring (A', B') of $G' = G - v$.
Question: Does G have a valid 2-subcoloring (A, B)?

Fix an instance of Inductive 2-Subcoloring with a graph $G = (V, E)$, a vertex $v \in V$, and a valid 2-subcoloring (A', B') of $G' = G - v$. Let $n = |V|$. We again apply a search-tree algorithm that starts with initial partitions $(A_*^{\mathcal{C}}, B_*^{\mathcal{C}})$ of V, derived from (A', B'), that are

not necessarily 2-subcolorings of G. Then, we try to "repair" those partitions by moving vertices between $A^{\mathcal{C}}_*$ and $B^{\mathcal{C}}_*$ to form a valid 2-subcoloring (A, B) of G. As before, each node in the search tree is associated with one constraint.

▶ **Definition 5.1.** A *constraint* $\mathcal{C} = (A^{\mathcal{C}}_1, \ldots, A^{\mathcal{C}}_k, B^{\mathcal{C}}_1, \ldots, B^{\mathcal{C}}_n, A^{\mathcal{C}}_P, B^{\mathcal{C}}_P)$ consists of a partition $(A^{\mathcal{C}}_1, \ldots, A^{\mathcal{C}}_k, B^{\mathcal{C}}_1, \ldots, B^{\mathcal{C}}_n)$ of V and two vertex sets $A^{\mathcal{C}}_P \subseteq A^{\mathcal{C}}_*$ and $B^{\mathcal{C}}_P \subseteq B^{\mathcal{C}}_*$, where $A^{\mathcal{C}}_* = \bigcup_{i=1}^{k} A^{\mathcal{C}}_i$ and $B^{\mathcal{C}}_* = \bigcup_{i=1}^{n} B^{\mathcal{C}}_i$, such that for any $i \neq j$:
- u and w are not adjacent for any $u \in A^{\mathcal{C}}_i \setminus A^{\mathcal{C}}_P$ and $w \in A^{\mathcal{C}}_j \setminus A^{\mathcal{C}}_P$, and
- u and w are not adjacent for any $u \in B^{\mathcal{C}}_i \setminus B^{\mathcal{C}}_P$ and $w \in B^{\mathcal{C}}_j \setminus B^{\mathcal{C}}_P$.

We explicitly allow (some of) the sets of the partition $(A^{\mathcal{C}}_1, \ldots, A^{\mathcal{C}}_k, B^{\mathcal{C}}_1, \ldots, B^{\mathcal{C}}_n)$ of V to be empty. The vertices in $A^{\mathcal{C}}_P$ and $B^{\mathcal{C}}_P$ are called *permanent* vertices of the constraint.

The permanent vertices in $A^{\mathcal{C}}_P$ and $B^{\mathcal{C}}_P$ in the definition will correspond precisely to those vertices that have switched sides during the algorithm. We refer to the sets $A^{\mathcal{C}}_1, \ldots, A^{\mathcal{C}}_k$ and $B^{\mathcal{C}}_1, \ldots, B^{\mathcal{C}}_n$ as *groups*; during the algorithm, $G[A^{\mathcal{C}}_*]$ and $G[B^{\mathcal{C}}_*]$ are not necessarily cluster graphs and, thus, we avoid using the term clusters.

We now define the notion of a valid 2-subcoloring fulfilling a constraint. Intuitively speaking, a constraint $\mathcal{C}$ is fulfilled by a bipartition (A, B) if (A, B) respects the assignment of the permanent vertices stipulated by $\mathcal{C}$, and if all vertices that do not switch sides stay in the bipartition (A, B) in the same groups they belong to in $\mathcal{C}$. This notion is formalized as follows.

▶ **Definition 5.2.** A constraint $\mathcal{C} = (A^{\mathcal{C}}_1, \ldots, A^{\mathcal{C}}_k, B^{\mathcal{C}}_1, \ldots, B^{\mathcal{C}}_n, A^{\mathcal{C}}_P, B^{\mathcal{C}}_P)$ is *fulfilled* by a bipartition (A, B) of V if $G[A]$ is a cluster graph with k clusters $A_1, \ldots, A_k$ and $G[B]$ is a cluster graph with n clusters $B_1, \ldots, B_n$ (some of the clusters may be empty) such that:
1. for each $i \in [k]$, $A_i \cap A^{\mathcal{C}}_* \subseteq A^{\mathcal{C}}_i$;
2. for each $i \in [n]$, $B_i \cap B^{\mathcal{C}}_* \subseteq B^{\mathcal{C}}_i$;
3. $A^{\mathcal{C}}_P \subseteq A$; and
4. $B^{\mathcal{C}}_P \subseteq B$.

We now need a set of initial constraints to jumpstart the search-tree algorithm.

▶ **Lemma 5.3** (⋆). *Let $A'_1, \ldots, A'_k$ denote the clusters of $G'[A']$ and let $B'_1, \ldots, B'_n$ denote the clusters of $G'[B']$. Herein, if there are less than k clusters in $G[A']$ or less than n clusters in $G[B']$, we add an appropriate number of empty sets. By relabeling, we may assume that only $B'_1, \ldots, B'_i$ contain neighbors of v, and $B'_{i+1} = \emptyset$. Each valid 2-subcoloring (A, B) of G fulfills either:*
- *$(A'_1, \ldots, A'_j \cup \{v\}, \ldots, A'_k, B'_1, \ldots, B'_n, \{v\}, \emptyset)$ for some $j \in [k]$, or*
- *$(A'_1, \ldots, A'_k, B'_1, \ldots, B'_j \cup \{v\}, \ldots, B'_n, \emptyset, \{v\})$ for some $j \in [i+1]$.*

Now that we have identified the initial constraints, we turn to the search-tree algorithm and its reduction and branching rules. A crucial ingredient to the rules and the analysis of the running time is the following lemma. A consequence of the lemma is that if the number of initial constraints is too large, then most of them should be rejected immediately.

▶ **Lemma 5.4** (⋆). *Let $\mathcal{C} = (A^{\mathcal{C}}_1, \ldots, A^{\mathcal{C}}_k, B^{\mathcal{C}}_1, \ldots, B^{\mathcal{C}}_n, A^{\mathcal{C}}_P, B^{\mathcal{C}}_P)$ be a constraint and let (A, B) be any valid 2-subcoloring of G fulfilling $\mathcal{C}$. If $u \in V$ has neighbors in more than $k+1$ groups among $B^{\mathcal{C}}_1, \ldots, B^{\mathcal{C}}_n$, then $u \in A$.*

Lemma 5.4 implies that if v has neighbors in more than $k+1$ clusters of A', then we should immediately reject the initial constraints generated by Lemma 5.3 that place v in $B^{\mathcal{C}}_P$. Hence, we obtain the following corollary of Lemmas 5.3 and 5.4.

▶ **Corollary 5.5** (⋆). *Lemma 5.3 generates at most* $2k+2$ *constraints that are not immediately rejected.*

As before, each non-root node of the search tree is associated with a constraint. The root of the search tree is a dummy node with children associated with the constraints generated by Lemma 5.3 that are not immediately rejected due to Lemma 5.4. We now give two reduction rules, which are applied exhaustively to each search-tree node, in the order they are presented.

Let $\mathcal{C} = (A_1^{\mathcal{C}}, \ldots, A_k^{\mathcal{C}}, B_1^{\mathcal{C}}, \ldots, B_n^{\mathcal{C}}, A_P^{\mathcal{C}}, B_P^{\mathcal{C}})$ be a constraint. The first reduction rule identifies some obvious cases in which the constraint cannot be fulfilled.

▶ **Reduction Rule 5.6** (⋆). *If* $G[A_P^{\mathcal{C}}]$ *or* $G[B_P^{\mathcal{C}}]$ *is not a cluster graph, or if there are* $i \neq j$ *such that there is an edge between* $A_i^{\mathcal{C}} \cap A_P^{\mathcal{C}}$ *and* $A_j^{\mathcal{C}} \cap A_P^{\mathcal{C}}$ *or an edge between* $B_i^{\mathcal{C}} \cap B_P^{\mathcal{C}}$ *and* $B_j^{\mathcal{C}} \cap B_P^{\mathcal{C}}$*, then reject* $\mathcal{C}$*.*

The second reduction rule is the natural consequence of Lemma 5.4.

▶ **Reduction Rule 5.7** (⋆). *If there is a vertex* $u \in A_i^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ *that has neighbors in more than* $k+1$ *groups of* $B_*^{\mathcal{C}}$*, then set* $A_P^{\mathcal{C}} \leftarrow A_P^{\mathcal{C}} \cup \{u\}$*.*

The algorithm contains a single branching rule. This rule, called switch(u), uses branching to fix a vertex u in one of the clusters in one of the parts of the 2-subcoloring. The vertices to which switch() must be applied are identified by *switching rules*. We say that a switching rule that calls for applying switch(u) is *correct* if for all valid 2-subcolorings (A, B) of G fulfilling $\mathcal{C}$, we have $u \in A_*^{\mathcal{C}} \cap B$ or $u \in B_*^{\mathcal{C}} \cap A$. We first describe the switching rules, and then describe switch(u). Recall from Fact 2.1 that cluster graphs do not contain induced P_3's.

The first switching rule identifies vertices that are not adjacent to some permanent vertices of their group.

▶ **Switching Rule 5.8.** *If there is a vertex* u *such that* $u \in A_i^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ *and* u *is not adjacent to some vertex in* $A_i^{\mathcal{C}} \cap A_P^{\mathcal{C}}$*, or* $u \in B_i^{\mathcal{C}} \setminus B_P^{\mathcal{C}}$ *and* u *is not adjacent to some vertex in* $B_i^{\mathcal{C}} \cap B_P^{\mathcal{C}}$*, then call* switch$(u)$*.*

The second switching rule finds vertices that have permanent neighbors in another group.

▶ **Switching Rule 5.9.** *If there is a vertex* u *such that* $u \in A_i^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ *and* u *has a neighbor in* $A_P^{\mathcal{C}} \setminus A_i^{\mathcal{C}}$*, or* $u \in B_i^{\mathcal{C}} \setminus B_P^{\mathcal{C}}$ *and* u *has a neighbor in* $B_P^{\mathcal{C}} \setminus B_i^{\mathcal{C}}$*, then call* switch$(u)$*.*

Now, we describe switch(u), which is a combination of a reduction rule and a branching rule. There are two main scenarios that we distinguish. If u has permanent neighbors in the other part, then there is only one choice for assigning u to a group. Otherwise, we branch into all (up to symmetry when a group is empty) possibilities to place u into a group. It is important to note that the switching rules never apply switch(u) to a permanent vertex.

▶ **Branching Rule 5.10** (switch(u))**.**

- *If* $u \in A_i^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ *and* u *has a permanent neighbor in some* $B_j^{\mathcal{C}}$*, then set* $A_i^{\mathcal{C}} \leftarrow A_i^{\mathcal{C}} \setminus \{u\}$*,* $B_j^{\mathcal{C}} \leftarrow B_j^{\mathcal{C}} \cup \{u\}$*,* $B_P^{\mathcal{C}} \leftarrow B_P^{\mathcal{C}} \cup \{u\}$*.*
- *If* $u \in A_i^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ *and* u *has only nonpermanent neighbors in* $B_*^{\mathcal{C}}$*, then, for each* $B_j^{\mathcal{C}}$ *such that* $N(u) \cap B_j^{\mathcal{C}} \neq \emptyset$ *and* $B_j^{\mathcal{C}} \cap B_P^{\mathcal{C}} = \emptyset$*, and for one* $B_j^{\mathcal{C}}$ *such that* $B_j^{\mathcal{C}} = \emptyset$ *(chosen arbitrarily), branch into a branch associated with the constraint* $(A_1^{\mathcal{C}}, \ldots, A_i^{\mathcal{C}} \setminus \{u\}, \ldots, A_k^{\mathcal{C}}, B_1^{\mathcal{C}}, \ldots, B_j^{\mathcal{C}} \cup \{u\}, \ldots, B_n^{\mathcal{C}}, A_P^{\mathcal{C}}, B_P^{\mathcal{C}} \cup \{u\})$*.*

- *If* $u \in B_i^{\mathcal{C}} \setminus B_P^{\mathcal{C}}$ *and* u *has a permanent neighbor in some* $A_j^{\mathcal{C}}$*, then set* $B_i^{\mathcal{C}} \leftarrow B_i^{\mathcal{C}} \setminus \{u\}$, $A_j^{\mathcal{C}} \leftarrow A_j^{\mathcal{C}} \cup \{u\}$, $A_P^{\mathcal{C}} \leftarrow A_P^{\mathcal{C}} \cup \{u\}$.
- *If* $u \in B_i^{\mathcal{C}} \setminus B_P^{\mathcal{C}}$ *and* u *has only nonpermanent neighbors in* $A_*^{\mathcal{C}}$*, then for each* $A_j^{\mathcal{C}}$ *with* $A_j^{\mathcal{C}} \cap A_P^{\mathcal{C}} = \emptyset$*, branch into a branch associated with the constraint* $(A_1^{\mathcal{C}}, \ldots, A_j^{\mathcal{C}} \cup \{u\}, \ldots, A_k^{\mathcal{C}}, B_1^{\mathcal{C}}, \ldots, B_i^{\mathcal{C}} \setminus \{u\}, \ldots, B_n^{\mathcal{C}}, A_P^{\mathcal{C}} \cup \{u\}, B_P^{\mathcal{C}})$*; if no such* $A_j^{\mathcal{C}}$ *exists, reject* $\mathcal{C}$*.*

If none of the previous rules applies, then the constraint directly gives a solution:

▶ **Lemma 5.11.** *Let* $\mathcal{C} = (A_1^{\mathcal{C}}, \ldots, B_n^{\mathcal{C}}, A_P^{\mathcal{C}}, B_P^{\mathcal{C}})$ *be a constraint such that none of the rules applies. Then* $(A_*^{\mathcal{C}}, B_*^{\mathcal{C}})$ *is a valid 2-subcoloring.*

Proof. We need to show that $G[A_*^{\mathcal{C}}]$ and $G[B_*^{\mathcal{C}}]$ are cluster graphs and that $G[A_*^{\mathcal{C}}]$ has at most k clusters. First, we claim that $G[A_i^{\mathcal{C}}]$ is a clique for every $i = 1, \ldots, k$. Every vertex in $A_i^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ is adjacent to every vertex in $A_i^{\mathcal{C}} \cap A_P^{\mathcal{C}}$; otherwise, Switching Rule 5.8 applies. Any two vertices in $A_i^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ are also adjacent, because they are in the same cluster of A'. It remains to show that $G[A_i^{\mathcal{C}} \cap A_P^{\mathcal{C}}]$ is a clique. By the description of switch(u), if a vertex x is placed into $A_i^{\mathcal{C}}$ and $A_i^{\mathcal{C}} \cap A_P^{\mathcal{C}} \neq \emptyset$, then x is adjacent to a vertex of $A_i^{\mathcal{C}} \cap A_P^{\mathcal{C}}$. Hence, $G[A_i^{\mathcal{C}} \cap A_P^{\mathcal{C}}]$ is connected. Since Reduction Rule 5.6 does not apply, $G[A_i^{\mathcal{C}} \cap A_P^{\mathcal{C}}]$ does not contain an induced P_3 and, thus, it is a clique. Hence, $G[A_i^{\mathcal{C}}]$ is a clique, as claimed.

Second, we claim that there are no edges between $A_i^{\mathcal{C}}$ and $A_j^{\mathcal{C}}$, where $i \neq j$. Suppose for the sake of a contradiction that e is such an edge. Since Reduction Rule 5.6 does not apply, e is incident with at least one nonpermanent vertex. Since Switching Rule 5.9 does not apply, e is in fact incident with two nonpermanent vertices. Then e cannot exist by the definition of a constraint. The claim follows.

The combination of the above claims shows that $G[A_*^{\mathcal{C}}]$ is a cluster graph with the clusters $A_i^{\mathcal{C}}$ (some of which may be empty) and, thus, has at most k clusters. Similar arguments show that $G[B_*^{\mathcal{C}}]$ is a cluster graph: in the above argument, we used only Reduction Rule 5.6 and Switching Rules 5.8 and 5.9, which apply to vertices in $A_*^{\mathcal{C}}$ and $B_*^{\mathcal{C}}$ symmetrically. ◀

▶ **Theorem 5.12.** Inductive 2-Subcoloring *can be solved in* $\mathcal{O}(k^{2k+2} \cdot (|V| + |E|))$ *time.*

Proof. Given the valid 2-subcoloring (A', B') of G', we use Lemma 5.3 to generate a set of initial constraints, and reject those which cannot be fulfilled due to Lemma 5.4. By Corollary 5.5, at most $2k+2$ initial constraints remain, which are associated with the children of the (dummy) root node. For each node of the search tree, we first exhaustively apply the reduction rules on the associated constraint. Afterwards, if there exists a vertex u to which a switching rule applies, then we apply switch(u). If switch(u) does not branch but instead reduces to a new constraint, then we apply the reduction rules exhaustively again, etc.

A *leaf* of the search tree is a node associated either with a constraint that is rejected, or with a constraint to which no rule applies. The latter is called an *exhausted leaf*. If the search tree has an exhausted leaf, then the algorithm answers "yes"; otherwise, it answers "no". By the correctness of the reduction, branching, and switching rules, and by Lemma 5.11, graph G has a valid 2-subcoloring if and only if the search tree has at least one exhausted leaf node. Therefore, the described search-tree algorithm correctly decides an instance of Inductive 2-Subcoloring.

We now bound the running time of the algorithm. Observe that each described reduction rule and the branching rule switch() either rejects the constraint or makes a vertex permanent. Hence, along each root-leaf path, $\mathcal{O}(n)$ rules are applied. Each rule can trivially be tested for applicability and applied in polynomial time. Hence, it remains to bound the number of leaves of the search tree.

As mentioned, at the root of the search tree, we create at most $\mathcal{O}(n)$ constraints, out of which at most $2k+2$ constraints do not correspond to leaf nodes by Lemma 5.3, Corollary 5.5

and Reduction Rule 5.7. The only branches are created by a call to switch(u) for a vertex u that has only non-permanent neighbors in the other part of the bipartition $(A_*^{\mathcal{C}}, B_*^{\mathcal{C}})$. Observe that if such a vertex $u \in B_*^{\mathcal{C}} \setminus B_P^{\mathcal{C}}$, then in each constraint $\mathcal{C}'$ constructed by switch(u) the number of groups in $A_*^{\mathcal{C}'}$ that have at least one permanent vertex increases by one compared to $\mathcal{C}$. Since each constraint has k groups in $A_*^{\mathcal{C}}$, this branch can be applied at most k times along each root-leaf path in the search tree.

Similarly, if $u \in A_*^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$, then in each constraint $\mathcal{C}'$ constructed by switch(u) the number of groups in $B_*^{\mathcal{C}}$ that have at least one permanent vertex increases by one compared to $\mathcal{C}$. We claim that, if $B_*^{\mathcal{C}}$ has k groups with a permanent vertex, then u has a neighbor in $B_P^{\mathcal{C}}$. First, each permanent vertex in $B_*^{\mathcal{C}}$ is part of A' by the description of the rules. Moreover, the permanent vertices of the k groups in $B_*^{\mathcal{C}}$ with a permanent vertex stem from k different clusters in $G[A']$, because switch() places a vertex of $A_*^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ that has neighbors in $B_P^{\mathcal{C}}$ in the same group as its neighbors in $B_P^{\mathcal{C}}$. This implies that one of the clusters in $G[A']$ that the permanent vertices stem from contains u. Hence, u is adjacent to a vertex in $B_P^{\mathcal{C}}$, as claimed. The claim implies that if $B_*^{\mathcal{C}}$ has k groups with a permanent vertex, then switch(u) applied to a vertex $u \in A_*^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ does not branch. Hence, also the branch of switch(u) in which $u \in A_*^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ is performed at most k times along each root-leaf path in the search tree.

In summary, the branchings of switch(u) in which $u \in B_*^{\mathcal{C}} \setminus B_P^{\mathcal{C}}$ branch into at most k cases, and the branchings in which $u \in A_*^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ branch into at most $k+2$ cases, since Reduction Rule 5.7 does not apply. Observe that k of the initial constraints have already one group in $A_*^{\mathcal{C}}$ with a permanent vertex, and the other $k+1$ initial constraints have one group in B with a permanent vertex. Thus, if the initial constraint $\mathcal{C}$ places v in $A_P^{\mathcal{C}}$, then the overall number of constraints from $\mathcal{C}$ by branching is at most $k^{k-1} \cdot (k+2)^k$. If the initial constraint $\mathcal{C}$ places v in $B_P^{\mathcal{C}}$, then the overall number of constraints created from $\mathcal{C}$ by branching is at most $k^k \cdot (k+2)^{k-1}$. Altogether, the number of constraints created by branching is thus

$$(2k+1) \cdot k^k \cdot (k+2)^k = (2k+1) \cdot k^k \cdot k^k \cdot [(1+1/(k/2))^{k/2}]^2 = \mathcal{O}(k^{2k+1})$$

after noting that $[(1+1/(k/2))^{k/2}]^2 = \mathcal{O}(1)$. This provides the claimed bound on the number of leaves of the search tree. By performing an analysis similar to that in the proof of Theorem 4.11, we can show that the time spent along each root-leaf path of the search tree is $\mathcal{O}(k \cdot (|V| + |E|))$, which yields an overall running time of $\mathcal{O}(k^{2k+2} \cdot (|V| + |E|))$ for Inductive 2-Subcoloring. ◀

Given the above theorem, Corollary 3.2 immediately implies Theorem 1.2.

References

1. Demetrios Achlioptas. The complexity of G-free colourability. *Discrete Mathematics*, 165–166(0):21–30, 1997.
2. Hajo Broersma, Fedor V. Fomin, Jaroslav Nešetřil, and Gerhard J. Woeginger. More about subcolorings. *Computing*, 69(3):187–203, 2002.
3. Sharon Bruckner, Falk Hüffner, and Christian Komusiewicz. A graph modification approach for finding core-periphery structures in protein interaction networks. *Algorithms for Molecular Biology*, 10:16, 2015.
4. Ross Churchley and Jing Huang. List monopolar partitions of claw-free graphs. *Discrete Mathematics*, 312(17):2545–2549, 2012.
5. Ross Churchley and Jing Huang. Solving partition problems with colour-bipartitions. *Graphs and Combinatorics*, 30(2):353–364, 2014.
6. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

7 Reinhard Diestel. *Graph Theory, 4th Edition*. Springer, 2012.

8 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, Berlin, Heidelberg, 2013.

9 Tınaz Ekim, Pavol Hell, Juraj Stacho, and Dominique de Werra. Polarity of chordal graphs. *Discrete Applied Mathematics*, 156(13):2469–2479, 2008.

10 E. M. Eschen and X. Wang. Algorithms for unipolar and generalized split graphs. *Discrete Applied Mathematics*, 162:195–201, 2014.

11 Alastair Farrugia. Vertex-partitioning into fixed additive induced-hereditary properties is NP-hard. *The Electronic Journal of Combinatorics*, 11(1):R46, 2004.

12 Jirí Fiala, Klaus Jansen, Van Bang Le, and Eike Seidel. Graph subcolorings: Complexity and algorithms. *SIAM Journal on Discrete Mathematics*, 16(4):635–650, 2003.

13 Peter L. Hammer and Bruno Simeone. The splittance of a graph. *Combinatorica*, 1(3):275–284, 1981.

14 Sudeshna Kolay and Fahad Panolan. Parameterized Algorithms for Deletion to (r, ℓ)-Graphs. In *Proceedings of the 35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 45 of *LIPIcs*, pages 420–433. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.

15 Van Bang Le and Ragnar Nevries. Complexity and algorithms for recognizing polar and monopolar graphs. *Theoretical Computer Science*, 528:1–11, 2014.

16 Colin McDiarmid and Nikola Yolov. Recognition of unipolar and generalised split graphs. *Algorithms*, 8(1):46–59, 2015.

17 C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

18 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.

19 Juraj Stacho. On 2-subcolourings of chordal graphs. In *Proceedings of the 8th Latin American Theoretical Informatics Symposium*, volume 4957 of *LNCS*, pages 544–554. Springer, 2008.

20 Regina I. Tyshkevich and Arkady A. Chernyak. Algorithms for the canonical decomposition of a graph and recognizing polarity. *Izvestia Akad. Nauk BSSR, ser. Fiz. Mat. Nauk*, 6:16–23, 1985. In Russian.

21 Regina I. Tyshkevich and Arkady A. Chernyak. Decomposition of graphs. *Cybernetics and Systems Analysis*, 21(2):231–242, 1985.

On Routing Disjoint Paths in Bounded Treewidth Graphs

Alina Ene[1], Matthias Mnich*[2], Marcin Pilipczuk†[3], and Andrej Risteski[4]

1 **Department of Computer Science and DIMAP, University of Warwick, Warwick, United Kingdom**
A.Ene@dcs.warwick.ac.uk
2 **University of Bonn, Bonn, Germany**
mmnich@uni-bonn.de
3 **Institute of Informatics, University of Warsaw, Warsaw, Poland**
malcin@mimuw.edu.pl
4 **Department of Computer Science, Princeton University, Princeton, USA**
risteski@princeton.edu

Abstract

We study the problem of routing on disjoint paths in bounded treewidth graphs with both edge and node capacities. The input consists of a capacitated graph G and a collection of k source-destination pairs $\mathcal{M} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$. The goal is to maximize the number of pairs that can be routed subject to the capacities in the graph. A routing of a subset $\mathcal{M}'$ of the pairs is a collection $\mathcal{P}$ of paths such that, for each pair $(s_i, t_i) \in \mathcal{M}'$, there is a path in $\mathcal{P}$ connecting s_i to t_i. In the Maximum Edge Disjoint Paths (MaxEDP) problem, the graph G has capacities $\mathrm{cap}(e)$ on the edges and a routing $\mathcal{P}$ is *feasible* if each edge e is in at most $\mathrm{cap}(e)$ of the paths of $\mathcal{P}$. The Maximum Node Disjoint Paths (MaxNDP) problem is the node-capacitated counterpart of MaxEDP.

In this paper we obtain an $\mathcal{O}(r^3)$ approximation for MaxEDP on graphs of treewidth at most r and a matching approximation for MaxNDP on graphs of pathwidth at most r. Our results build on and significantly improve the work by Chekuri et al. [ICALP 2013] who obtained an $\mathcal{O}(r \cdot 3^r)$ approximation for MaxEDP.

1998 ACM Subject Classification G.2.2 Graph algorithms

Keywords and phrases Algorithms and data structures, disjoint paths, treewidth

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.15

1 Introduction

In this paper, we study disjoint paths routing problems on bounded treewidth graphs. In this setting, we are given an undirected capacitated graph G and a collection of source-destination pairs $\mathcal{M} = \{(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)\}$. The goal is to select a maximum-sized subset $\mathcal{M}' \subseteq \mathcal{M}$ of the pairs that can be *routed* subject to the capacities in the graph. More precisely, a routing of $\mathcal{M}'$ is a collection $\mathcal{P}$ of paths such that, for each pair $(s_i, t_i) \in \mathcal{M}'$, there is a path in $\mathcal{P}$ connecting s_i to t_i. In the Maximum Edge Disjoint Paths (MaxEDP)

* Research partially supported by ERC Starting Grant 306465 (BeyondWorstCase).
† Research done while at University of Warwick, partially supported by DIMAP and by Warwick-QMUL Alliance in Advances in Discrete Mathematics and its Applications.

problem, the graph G has capacities $\text{cap}(e)$ on the edges and a routing $\mathcal{P}$ is *feasible* if each edge e is in at most $\text{cap}(e)$ of the paths of $\mathcal{P}$. The Maximum Node Disjoint Paths (MaxNDP) problem is the node-capacitated counterpart of MaxEDP.

Disjoint paths problems are fundamental problems with a long history and significant connections to optimization and structural graph theory. The decision versions of MaxEDP and MaxNDP ask whether all of the pairs can be routed subject to the capacities. Karp [18] showed that, when the number of pairs is part of the input, the decision problem is NP-complete. In undirected graphs, MaxEDP and MaxNDP are solvable in polynomial time when the number of pairs is constant; this is a deep result of Robertson and Seymour [22] that builds on several fundamental structural results from their graph minors project.

In this paper, we consider the optimization problems MaxEDP and MaxNDP when the number of pairs are part of the input. These problems are NP-hard and the main focus in this paper is on approximation algorithms for these problems in bounded treewidth graphs. Although they may appear to be quite specialized at first, MaxEDP and MaxNDP on capacitated graphs of small treewidth capture a surprisingly rich class of problems; in fact, as shown by Garg, Vazirani, and Yannakakis [16], these problems are quite interesting and general even on trees.

MaxEDP and MaxNDP have received considerable attention, leading to several breakthroughs both in terms of approximation algorithms and hardness results. MaxEDP is APX-hard even in edge-capacitated trees [16], whereas the decision problem is trivial on trees; thus some of the hardness of the problem stems from having to select a subset of the pairs to route. Moreover, by subdividing the edges, one can easily show that MaxNDP generalizes MaxEDP in capacitated graphs. However, node capacities pose several additional technical challenges and extending the results for MaxEDP to MaxNDP is far from immediate even in restricted graph classes and our understanding of MaxNDP is more limited.

In general graphs, the best approximation for MaxEDP and MaxNDP is an $\mathcal{O}(\sqrt{n})$ approximation [4, 19], where n is the number of nodes, whereas the best hardness for undirected graphs is only $\Omega((\log n)^{1/2-\epsilon})$ [2]. Bridging this gap is a fundamental open problem that seems quite challenging at the moment. There have been several breakthrough results on a relaxed version of these problems where congestion is allowed[1]. This line of work has culminated with a $\text{polylog}(n)$ approximation with congestion 2 for MaxEDP [14] and congestion 51 for MaxNDP [6]. In addition to the routing results, this work has led to several significant insights into the structure of graphs with large treewidth and to several surprising applications [5].

Most of the results for routing on disjoint paths use a natural multi-commodity flow relaxation as a starting point. A well-known integrality gap instance due to Garg et al. [16] shows that this relaxation has an integrality gap of $\Omega(\sqrt{n})$, and this is the main obstacle for improving the $\mathcal{O}(\sqrt{n})$ approximation in general graphs. The integrality gap example is an instance on an $n \times n$ grid that exploits a topological obstruction in the plane that prevents a large integral routing (see Fig. 2). Since an $n \times n$ grid has treewidth $\Theta(\sqrt{n})$, it suggests the following natural and tantalizing conjecture that was asked by Chekuri et al. [9].

▶ **Conjecture 1** ([9]). *The integrality gap of the standard multi-commodity flow relaxation for* MaxEDP/MaxNDP *is* $\Theta(r)$ *with congestion* 1*, where* r *is the treewidth of the graph.*

Recently, Chekuri et al. [10] showed that MaxEDP admits an $\mathcal{O}(r \cdot 3^r)$ approximation on graphs of treewidth at most r. This is the first approximation for the problem that is

[1] A collection of paths has an *edge* (resp. *node*) *congestion* of c if each edge (resp. node) is in at most $c \cdot \text{cap}(e)$ (resp. $c \cdot \text{cap}(v)$) paths.

independent of n and k, and the first step towards resolving the conjecture. One of the main questions left open by the work of Chekuri et al. [10]—that was explicitly asked by them—is whether the exponential dependency on the treewidth is necessary. In this paper, we address this question and we make a significant progress towards resolving Conjecture 1.

▶ **Theorem 2.** *The integrality gap of the multi-commodity flow relaxation is $\mathcal{O}(r^3)$ for* MaxEDP *in edge-capacitated undirected graphs of treewidth at most r. Moreover, there is a polynomial time algorithm that, given a tree decomposition of G of width at most r and a fractional solution to the relaxation of value* OPT*, constructs an integral routing of size $\Omega(\mathrm{OPT}/r^3)$.*

As mentioned above, MaxNDP in node-capacitated graphs is more general than MaxEDP and it poses several additional technical challenges. In this paper, we give an $\mathcal{O}(r^3)$ approximation for MaxNDP on graphs of pathwidth at most r with arbitrary node capacities. This is the first approximation guarantee for MaxNDP that is independent of n and it improves the $\mathcal{O}(r \log r \log n)$ approximation of Chekuri et al. [9].

▶ **Theorem 3.** *The integrality gap of the multi-commodity flow relaxation is $\mathcal{O}(r^3)$ for* MaxNDP *in node-capacitated undirected graphs of pathwidth at most r. Moreover, there is a polynomial time algorithm that, given a path decomposition of G of width at most r and a fractional solution to the relaxation of value* OPT*, constructs an integral routing of size $\Omega(\mathrm{OPT}/r^3)$.*

The study of routing problems in bounded treewidth graphs is motivated not only by the goal of understanding the integrality gap of the multi-commodity flow relaxation but also by the broader goal of giving a more refined understanding of the approximability of routing problems in undirected graphs. Andrews et al. [2] have shown that MaxEDP and MaxNDP in general graphs cannot be approximated within a factor better than $(\log n)^{\Omega(1/c)}$ even if we allow a constant congestion $c \geq 1$. Thus in order to obtain constant factor approximations one needs to use additional structure. However, this seems challenging with our current techniques and there are only a handful of results in this direction.

One of the main obstacles for obtaining constant factor approximations for disjoint paths problems is that most approaches rely on a powerful pre-processing step that reduces an arbitrary instance of MaxEDP/MaxNDP to a much more structured instance in which the terminals[2] are *well-linked*. This reduction is achieved using the well-linked decomposition technique of Chekuri, Khanna, and Shepherd [7], which necessarily leads to an $\Omega(\log n)$ loss even in very special classes of graphs such as bounded treewidth graphs. Chekuri, Khanna, and Shepherd [8] showed that the well-linked decomposition framework can be bypassed in planar graphs, leading to an $\mathcal{O}(1)$ approximation for MaxEDP with congestion 4 (the congestion was later improved by Séguin-Charbonneau and Shepherd [24] from 4 to 2). This result suggests that it may be possible to obtain constant factor approximations with constant congestion for much more general classes of graphs. In particular, Chekuri et al. [10] conjecture that this is the case for the class of all minor-free graphs.

▶ **Conjecture 4** ([10])**.** *Let $\mathcal{G}$ be any proper minor-closed family of graphs. Then the integrality gap of the multi-commodity flow relaxation for* MaxEDP *is at most a constant $c_{\mathcal{G}}$ when congestion 2 is allowed.*

[2] The vertices participating in the pairs $\mathcal{M}$ are called *terminals*.

A natural approach is to attack Conjecture 4 using the structure theorem for minor-free graphs given by Robertson and Seymour [21, 23] that asserts that every such graph admits a tree decomposition where the size of every adhesion (the intersection of neighboring bags) is bounded, and after turning the adhesions into cliques, every bag induces a structurally simpler graph: one of bounded genus, with potentially a bounded number of apices and vortices. Thus in some sense, in order to resolve Conjecture 4, one needs to understand the base graph class (bounded genus graphs with apices and vortices) and how to tackle bounded width tree decompositions.

The recent work of Chekuri et al. [10] has made a significant progress toward resolving Conjecture 4 by providing a toolbox for the latter issue, and the only ingredient that is still missing is an algorithm for planar and bounded genus graphs with a constant number of vortices (in the disjoint paths setting, apices are very easy to handle). However, one of the main drawbacks of their approach is that it leads to approximation guarantees that are *exponential* in the treewidth. Our work strengthens the approach of Chekuri et al. [10] and it gives a much more graceful *polynomial* dependence in the approximation ratio.

▶ **Theorem 5.** *Let $\mathcal{G}$ be a minor-closed class of graphs such that the integrality gap of the multi-commodity flow relaxation is α with congestion β. Let $\mathcal{G}_\ell$ be the class of graphs that admit a tree decomposition where, after turning all adhesions into cliques, each bag induces a graph from $\mathcal{G}$, and each adhesion has size at most ℓ. Then the integrality gap of the relaxation for the class $\mathcal{G}_\ell$ is $\mathcal{O}(\ell^3) \cdot \alpha$ with congestion $\beta + 3$.*

We also revisit the well-linked decomposition framework of Chekuri et al. [7] and we ask whether the $\Omega(\log n)$ loss is necessary for very structured graph classes. For bounded treewidth graphs, we give a well-linked decomposition framework that reduces an arbitrary instance of MaxEDP to node-disjoint instances of MaxEDP that are *well-linked*. The loss in the approximation for our decomposition is only $\mathcal{O}(r^3)$, which improves the guarantee of $\mathcal{O}(\log r \log n)$ from Chekuri et al. [9] when r is much smaller than n.

It is straightforward to obtain the improved well-linked decomposition from our algorithm for MaxEDP. Nevertheless, we believe it is beneficial to have such a well-linked decomposition, given that well-linked decompositions are one of the technical tools at the heart of the recent algorithms for routing on disjoint paths, integral concurrent flows [3], and flow and cut sparsifiers [13]. In particular, we hope that such a well-linked decomposition will have applications to finding flow and cut sparsifiers with Steiner nodes for bounded treewidth graphs. A sparsifier for a graph G with k source-sink pairs is a significantly smaller graph H containing the terminals (and potentially other vertices, called Steiner nodes) that approximately preserves multi-commodity flows or cuts between the terminals. Such sparsifiers have been extensively studied and several results are known both in general graphs and in bounded treewidth graphs (see Andoni et al. [1] and references therein).

A different question one could ask for problems in bounded treewidth graphs is whether additional computational power beyond polynomial-time running time can help with MaxEDP or MaxNDP. It is a standard exercise to design an $n^{\mathcal{O}(r)}$-time dynamic programming algorithm (i.e., polynomial for every constant r) for MaxNDP in uncapacitated graphs of treewidth r, while the aforementioned results on hardness of MaxEDP in capacitated trees [16] rule out similar results for capacitated variants. Between the world of having r as part of the input, and having r as a fixed constant, lies the world of *parameterized complexity*, that asks for algorithms (called *fixed-parameter algorithms*) with running time $f(r) \cdot n^c$, where f is any computable function, and c is a constant independent of the parameter. It is natural to ask whether allowing such running time can lead to better approximation algorithms. As a

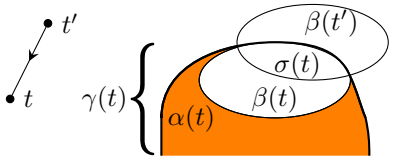


Figure 1 Notations used for a node t with parent t' in a tree decomposition $(\mathcal{T}, \beta)$. The shaded part defines $\alpha(t)$.

first step towards resolving this question, we show a hardness for MaxNDP parameterized by *treedepth*, a much more restrictive graph parameter than treewidth (cf. [20]).

▶ **Theorem 6.** MaxNDP *parameterized by the treedepth of the input graph is* $W[1]$*-hard, even with unit capacities.*

Consequently, the existence of an *exact* fixed-parameter algorithm is highly unlikely. We remark that our motivation for the choice of treedepth as parameter stems from the observation that a number of algorithms using the Sherali-Adams hierarchy to approximate a somewhat related problem of Nonuniform Sparsest Cut in bounded treewidth graphs [12, 17] in fact implicitly uses a rounding scheme based on treedepth rather than treewidth.

Due to space constraints, we defer the proof of Thereom 6 to the full version of this paper.

2 Preliminaries

Tree and path decompositions. In this paper all tree decompositions are rooted; that is, a tree decomposition of a graph G is a pair $(\mathcal{T}, \beta)$ where $\mathcal{T}$ is a rooted tree and $\beta : V(\mathcal{T}) \to 2^{V(G)}$ is a mapping such that (i) for every $e \in E(G)$ there is a node $t \in V(\mathcal{T})$ with $e \subseteq \beta(t)$, and (ii) for every $v \in V(G)$ the set $\{t \mid v \in \beta(t)\}$ is non-empty and connected in $\mathcal{T}$.

For a node $t \in V(\mathcal{T})$, we call the set $\beta(t)$ the *bag* at node t, while for an edge $st \in E(\mathcal{T})$, the set $\beta(t) \cap \beta(s)$ is called an *adhesion*. For a non-root node $t \in V(\mathcal{T})$, by $\mathrm{parent}(t)$ we denote the parent of t, and by $\sigma(t) := \beta(t) \cap \beta(\mathrm{parent}(t))$ the adhesion on the edge to the parent of t, called henceforth *the parent adhesion*; for the root node $t_0 \in V(\mathcal{T})$ we put $\sigma(t_0) = \emptyset$. For two nodes $s, t \in V(\mathcal{T})$, we denote by $s \preceq t$ if s is a descendant of t, and put $\gamma(t) := \bigcup_{s \preceq t} \beta(s)$, $\alpha(t) := \gamma(t) \setminus \sigma(t)$, and $G(t) := G[\gamma(t)] \setminus E(G[\sigma(t)])$.

A *torso* at node t is a graph obtained from $G[\beta(t)]$ by turning every adhesion for an edge incident to t into a clique.

We say that (A, B) is a *separation* in G if $A \cup B = V(G)$ and there does not exist an edge of G with an endpoint in $A \setminus B$ and the other endpoint in $B \setminus A$. We use the following well-known property of a tree decomposition.

▶ **Lemma 7** ([15, Lemma 12.3.1])**.** *Let $(\mathcal{T}, \beta)$ be a tree decomposition of a graph G. Then for each $t \in V(\mathcal{T})$ the pair $(\gamma(t), V(G) \setminus \alpha(t))$ is a separation of G, and $\gamma(t) \cap (V(G) \setminus \alpha(t)) = \sigma(t)$.*

A *path decomposition* is a tree decomposition where $\mathcal{T}$ is a path, rooted at one of its endpoints.

The width of a tree or path decomposition $(\mathcal{T}, \beta)$ is defined as $\max_t |\beta(t)| - 1$. To ease the notation, we will always consider decompositions of width *less* than r, for some integer r, so that every bag is of size at most r.

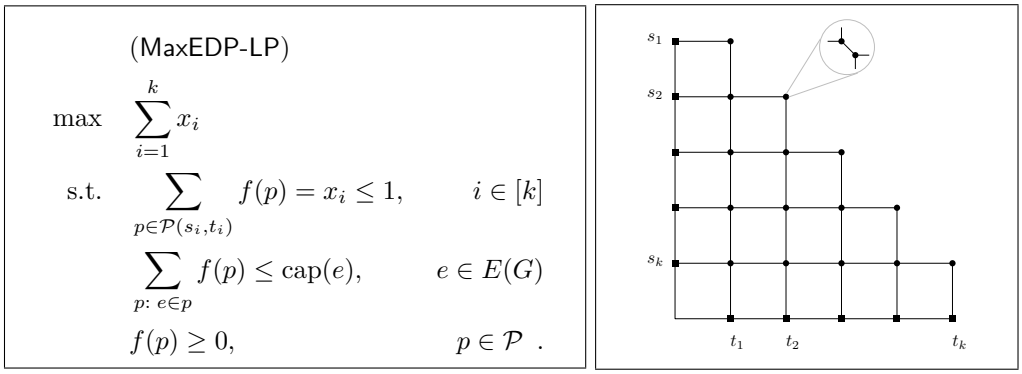


Figure 2 The multi-commodity flow relaxation for MaxEDP. The instance on the right is the $\Omega(\sqrt{n})$ integrality gap example for MaxEDP with unit edge capacities [16]. Any integral routing routes at most one pair whereas there is a multi-commodity flow that sends 1/2 units of flow for each pair (s_i, t_i) along the canonical path from s_i to t_i in the grid.

Problem definitions. The input to MaxEDP is an undirected graph G with edge capacities $\mathrm{cap}(e) \in \mathbb{Z}_+$ and a collection $\mathcal{M} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ of vertex pairs. A *routing* for a subset $\mathcal{M}' \subseteq \mathcal{M}$ is a collection $\mathcal{P}$ of paths in G such that, for each pair $(s_i, t_i) \in \mathcal{M}'$, $\mathcal{P}$ contains a path connecting s_i to t_i. The routing is *feasible* if every edge e is in at most $\mathrm{cap}(e)$ paths. In the Maximum Edge Disjoint Paths problem (MaxEDP), the goal is to maximize the number of pairs that can be feasibly routed. The Maximum Node Disjoint Paths problem (MaxNDP) is the node-capacitated variant of MaxEDP in which each node v has a capacity $\mathrm{cap}(v)$ and in a feasible routing each node appears in at most $\mathrm{cap}(v)$ paths.

We refer to the vertices participating in the pairs $\mathcal{M}$ as *terminals*. It is convenient to assume that $\mathcal{M}$ form a matching on the terminals; this can be ensured by making several copies of a terminal and attaching them as leaves.

Multicommodity flow relaxation. We use the following standard multicommodity flow relaxation for MaxEDP (there is an analogous relaxation for MaxNDP). We use $\mathcal{P}(u, v)$ to denote the set of all paths in G from u to v, for each pair (u, v) of nodes. Since the pairs $\mathcal{M}$ form a matching, the sets $\mathcal{P}(s_i, t_i)$ are pairwise disjoint. Let $\mathcal{P} = \bigcup_{i=1}^{k} \mathcal{P}(s_i, t_i)$. The LP has a variable $f(p)$ for each path $p \in \mathcal{P}$ representing the amount of flow on p. For each pair $(s_i, t_i) \in \mathcal{M}$, the LP has a variable x_i denoting the total amount of flow routed for the pair (in the corresponding IP, x_i denotes whether the pair is routed or not). The LP imposes the constraint that there is a flow from s_i to t_i of value x_i. Additionally, the LP has capacity constraints that ensure that the total amount of flow on paths using a given edge (resp. node for MaxNDP) is at the capacity of the edge (resp. node).

It is well-known that the relaxation MaxEDP-LP can be solved in polynomial time, since there is an efficient separation oracle for the dual (alternatively, one can write a compact relaxation). Let $(f, \mathbf{x})$ denote a feasible solution to MaxEDP-LP for an instance $(G, \mathcal{M})$ of MaxEDP. For each vertex v, let $x(v)$ denote the *marginal value* of v in the multi-commodity flow f; thus, $x(v)$ is the amount of flow routed for each terminal v.

3 Algorithm for MaxEDP in Bounded Treewidth Graphs

We give a polynomial time algorithm for MaxEDP that achieves an $\mathcal{O}(r^3)$ approximation for graphs with treewidth less than r. Our algorithm builds on the work of Chekuri et al. [10],

and it improves their approximation guarantee from $\mathcal{O}(r \cdot 3^r)$ to $\mathcal{O}(r^3)$. We use the following routing argument as a building block.

▶ **Proposition 8** ([11, Proposition 3.4]). *Let $(G, \mathcal{M})$ be an instance of* MaxEDP *and let $(f, \mathbf{x})$ be a feasible fractional solution for the instance. If there is a set $S \subseteq V(G)$, a value $\alpha \geq 1$ and a flow g that for each $v \in V(G)$ routes routes at least $x(v)/\alpha$ units of flow to some vertex in S, then there is an integral routing of at least $\binom{|f|}{36\alpha|S|}$ pairs.*

We will later apply Proposition 8 by letting S be a subset of a bag in a tree decomposition of G.

Our starting point is a tree decomposition $(\mathcal{T}, \beta)$ for G of width less than r and a fractional solution $(f, \mathbf{x})$ to the multicommodity flow relaxation for MaxEDP given in Section 2, that is, the flow f routes $\mathbf{x}(v)$ units of flow for each vertex $v \in V$. We let $|f|$ denote the total amount of flow routed by f, i.e., $|f| = \binom{1}{2} \sum_{v \in V} \mathbf{x}(v)$.

The following definitions play a key role in our algorithm.

▶ **Definition 9** (Safe node). A node $t \in V(\mathcal{T})$ is *safe* with respect to $(f, \mathbf{x})$ if there is a second multicommodity flow g in $G(t)$ that satisfies the edge capacities of $G(t)$ and, for each vertex $z \in \gamma(t)$, g routes at least $\binom{1}{4r} \cdot \mathbf{x}(z)$ units of flow from z to the adhesion $\sigma(t)$. The node t is *unsafe* if it is not safe.

▶ **Definition 10** (Good node). A node $t \in V(\mathcal{T})$ is *good* with respect to $(f, \mathbf{x})$ if every flow path in the support of f that has an endpoint in $\gamma(t)$ also intersects $\sigma(t)$; in other words, no flow path is completely contained in $G[\alpha(t)]$. A node is *bad* if it is not good.

▶ Remark. If a node t is good then it is also safe, as shown by the following multicommodity flow g in $G(t)$. For each path p in the support of f that originates in $\gamma(t)$, let p' be the smallest prefix of p that ends at a vertex of $\sigma(t)$ (since p intersects $\sigma(t)$, there is such a prefix); we set $g(p') = f(p)$. The resulting flow g is a feasible multicommodity flow in $G(t)$ that routes $\mathbf{x}(z)$ units of flow from z to $\sigma(t)$ for each vertex $z \in \gamma(t)$. Therefore, t is safe.

Our approach is an inductive argument based on the maximum size of a parent adhesion that is bad or unsafe. More precisely, we prove the following:

▶ **Theorem 11.** *Let $(G, \mathcal{M})$ be an instance of* MaxEDP *and let $(f, \mathbf{x})$ be a fractional solution for $(G, \mathcal{M})$, where f is a feasible multicommodity flow in G for $\mathcal{M}$ with marginal values $\mathbf{x}$. Let $(\mathcal{T}, \beta)$ be a tree decomposition for G of width less than r. Let ℓ_1 be the maximum size of a parent adhesion of an unsafe node, and let ℓ_2 be the maximum size of a parent adhesion of a bad node. There is a polynomial time algorithm that constructs an integral routing of size at least $\binom{1}{144r^3} \cdot \left(1 - \binom{1}{r}\right)^{\ell_1+\ell_2} \cdot |f|$.*

Proof. We start with a bit of preprocessing. If $|f| = 0$, then we return an empty routing. Otherwise, the root node of $\mathcal{T}$ is always unsafe and bad, and the integers ℓ_1 and ℓ_2 are well-defined. By considering every connected component of G independently (with inherited tree decomposition from $(\mathcal{T}, \beta)$), we assume that G is connected; note that in this step all safe or good adhesions remain safe or good for every connected component. Furthermore, we delete from $(\mathcal{T}, \beta)$ all nodes with empty bags; note that the connectivity of G ensures that the nodes with non-empty bags induce a connected subtree of $\mathcal{T}$. In this step, the root of $\mathcal{T}$ may have moved to a different node (the topmost node with non-empty bag), but the parent-children relation in the tree remains unchanged.

Once G is connected and no bag is empty, the only empty parent adhesion is the one for the root node. We prove Theorem 11 by induction on $\ell_1 + \ell_2 + |V(G)|$.

Base case. In the base case, we assume that $\ell_1 = \ell_2 = 0$. Since every parent adhesion of a non-root node is non-empty, that implies that the only bad node is the root t_0, that is, every flow path in f passes through $\beta(t_0)$, which is of size at most r. By applying Proposition 8 with $S = \beta(t_0)$ and $\alpha = 1$, we construct an integral routing of size at least $\frac{1}{36r}|f| \geq \frac{1}{144r^3}|f|$.

In the inductive step, we consider two cases, depending on if $0 \leq \ell_1 < \ell_2$ or $0 < \ell_1 = \ell_2$.

Inductive step when $0 \leq \ell_1 < \ell_2$. Let $\{t_1, t_2, \ldots, t_p\}$ be the topmost bad nodes of $\mathcal{T}$ with parent adhesions of size ℓ_2, that is, it is a minimal set of such bad nodes such that for every bad node t with parent adhesion of size ℓ_2, there exists an $i \in \{1, \ldots, p\}$ with $t \preceq t_i$. For $i = 1, \ldots, p$, let f_i^{inside} be the subflow of f consisting of all paths that are completely contained in $G[\alpha(t_i)]$. Furthermore, since $\ell_1 < \ell_2$, the node t_i is safe; let g_i be the corresponding flow, i.e., a flow that routes $\frac{1}{4r}x(v)$ from every $v \in \gamma(t_i)$ to $\sigma(t_i)$ in $G(t_i)$. By applying Proposition 8, there is an integral routing $\mathcal{P}_i$ in $G(t_i)$ that routes at least $\binom{1}{144r^2}|f_i^{\text{inside}}|$ pairs. Since the subgraphs $\{G(t_i) \colon 1 \leq i \leq p\}$ are edge-disjoint, we get an integral routing $\mathcal{P} := \bigcup_i \mathcal{P}_i$ of size at least $\binom{1}{144r^2}\sum_{i=1}^{p}|f_i^{\text{inside}}|$.

If $\sum_{i=1}^{p}|f_i^{\text{inside}}| > \frac{1}{r}|f|$, then we can return the routing $\mathcal{P}$ as the desired solution. Otherwise, we drop the flows f_i^{inside}, that is, consider a flow $f' := f - \sum_{i=1}^{p} f_i^{\text{inside}}$. Clearly, $|f'| \geq (1-\frac{1}{r})|f|$. Furthermore, by definition of f_i^{inside}, every node t_i is good with respect to f'. Since deleting a flow path cannot turn a good node into a bad one nor a safe node into an unsafe one, and all descendants of a good node are also good, we infer that every unsafe node with respect to f' has parent adhesion of size at most ℓ_1, while every bad node with respect to f' has parent adhesion of size *less* than ℓ_2. Consequently, by the induction hypothesis we obtain an integral routing of size at least $\frac{1}{144r^3}\left(1-\frac{1}{r}\right)^{\ell_1+\ell_2-1}|f'| \geq \frac{1}{144r^3}\left(1-\frac{1}{r}\right)^{\ell_1+\ell_2}|f|$.

Inductive step when $0 < \ell_1 = \ell_2$. In this case, we pick a node t° to be the lowest node of $\mathcal{T}$ that is unsafe and has parent adhesion of size ℓ_1. By the definition of an unsafe node and Menger's theorem, there exists a set $U \subseteq \alpha(t^\circ)$ such that $\mathrm{cap}(\delta(U)) < \frac{1}{4r}\mathbf{x}(U)$. With a bit more care, we can extract a set U with one more property:

▶ **Lemma 12.** *In polynomial time we can find a set $U \subseteq \alpha(t^\circ)$ for which* (i) $\mathrm{cap}(\delta(U)) < \frac{1}{4r}\mathbf{x}(U)$*, and* (ii) *for every non-root node t, if $\sigma(t) \subseteq U$, then $\gamma(t) \subseteq U$.*

Proof. Consider an auxiliary graph G', obtained from $G[\gamma(t^\circ)]$ by adding a super-source s^*, linked for every $v \in \gamma(t^\circ)$ by an arc (s^*, v) of capacity $\frac{1}{4r}\mathbf{x}(v)$, and a super-sink t^*, linked for every $v \in \sigma(t^\circ)$ by an arc (v, t^*) of infinite capacity. Let U be such a set that $\delta(U \cup \{s^*\})$ is a minimum s^*-t^* cut in this graph. Clearly, since U is unsafe, $\mathrm{cap}(\delta_{G'}(U \cup \{s^*\})) < \frac{1}{4r}\mathbf{x}(\gamma(t^\circ)) = \mathrm{cap}(\delta_{G'}(s^*))$, so $U \neq \emptyset$. Also, $U \subseteq \alpha(t^\circ)$, as each node in $\sigma(t^\circ)$ is connected to t^* with an infinite-capacity arc.

We claim that U satisfies the desired properties. The first property is immediate:

$$\mathrm{cap}(\delta_G(U)) = \mathrm{cap}(\delta_{G'}(U \cup \{s^*\})) - \frac{1}{4r}\mathbf{x}(\gamma(t^\circ) \setminus U) < \frac{1}{4r}\left(\mathbf{x}(\gamma(t^\circ)) - \mathbf{x}(\gamma(t^\circ) \setminus U)\right) = \frac{1}{4r}\mathbf{x}(U).$$

For the second property, pick a non root node t with $\sigma(t) \subseteq U$. Since $\sigma(t) \subseteq U \subseteq \alpha(t^\circ)$, we have $t \preceq t^\circ$, $t \neq t^\circ$, and $\gamma(t) \subseteq \alpha(t^\circ)$. Let $U' := U \cup \gamma(t)$. By Lemma 7, $\delta_G(U') \subseteq \delta_G(U)$, and hence $\delta_{G'}(U' \cup \{s^*\}) \subseteq \delta_{G'}(U \cup \{s^*\})$. However, since $\delta_{G'}(U \cup \{s^*\})$ is a minimum cut, we have actually $\delta_G(U') = \delta_G(U)$. Since G is connected, this implies that $U = U'$, and thus $\gamma(t) \subseteq U$. As the choice of t was arbitrary, U satisfies the second property. ◀

Using the cut U, we split the graph G and the flow f into two pieces as follows. Let $G_1 = G[U]$ and $G_2 = G - U$. Let f_i be the restriction of f to G_i, i.e., the flow consisting of

only flow paths that are contained in G_i. Let $\mathbf{x}_i$ be the marginals of f_i and let $\mathcal{M}_i$ be the subset of $\mathcal{M}$ consisting of all pairs (s,t) such that $\{s,t\} \subseteq V(G_i)$; note that $x_i(s) = x_i(t)$ for each pair $(s,t) \in \mathcal{M}_i$ and thus $(f_i, \mathbf{x}_i)$ is a fractional routing for the instance $(G_i, \mathcal{M}_i)$. Let $(\mathcal{T}, \beta_1)$ and $(\mathcal{T}, \beta_2)$ be the restriction of $(\mathcal{T}, \beta)$ to the vertices of G_1 and G_2, respectively; we define mappings σ_i, γ_i, and α_i naturally. In what follows, we consider separately two instances $\mathcal{I}_i := \langle (G_i, \mathcal{M}_i), (f_i, \mathbf{x}_i), (\mathcal{T}, \beta_i) \rangle$ for $i = 1, 2$.

An important observation is the following:

▶ **Lemma 13.** *Every node $t \in V(\mathcal{T})$ that is good in the original instance (i.e., as a node of $\mathcal{T}$, with respect to $(f, \mathbf{x})$) is also good in $\mathcal{I}_i$ with respect to $(f_i, \mathbf{x}_i)$.*

Proof. Note that every flow path in f_i is also present in f, and therefore intersects the parent adhesion of f if t is a good node in the original instance. ◀

Consequently, every node $t \in V(\mathcal{T})$ with $|\sigma(t)| > \ell_2$ is good in the instance $\mathcal{I}_i$, and the maximum size of a parent adhesion of a bad node in instance $\mathcal{I}_i$ is at most ℓ_2. Hence, both $\mathcal{I}_1$ and $\mathcal{I}_2$ satisfy the assumptions of Theorem 11 with not larger values of ℓ_1 and ℓ_2. Furthermore, note that $|V(G_i)| < |V(G)|$ for $i = 1, 2$.

For $\mathcal{I}_2$, the above reasoning allows us to simply just apply inductive step, obtaining an integral routing $\mathcal{P}_2$ of size at least

$$|\mathcal{P}_2| \geq \binom{1}{144r^3} \left(1 - \binom{1}{r}\right)^{\ell_1+\ell_2} \cdot |f_2| \ . \tag{1}$$

For $\mathcal{I}_1$, we are going to obtain a larger routing via an inductive step with better bounds.

▶ **Lemma 14.** *The size of the largest parent adhesion of an unsafe node in $\mathcal{I}_1$ is* less *than ℓ_1.*

Proof. Assume the contrary, let $t \in V(\mathcal{T})$ be an unsafe adhesion with $|\sigma_1(t)| \geq \ell_1$. If $|\sigma(t)| > \ell_1$, then t is good in the original instance, and by Lemma 13 it remains good in $\mathcal{I}_1$. Consequently, $|\sigma(t)| = |\sigma_1(t)| = \ell_1$; in particular, $\sigma(t) = \sigma_1(t) \subseteq U$.

By Lemma 12(ii) we have $\gamma(t) \subseteq U$. Consequently, t is safe in the original instance if and only if it is safe in $\mathcal{I}_1$. Since $t \preceq t^\circ$, $t \neq t^\circ$, but $|\sigma(t)| = \ell_2$, by the choice of t° it holds that t is safe in the original instance, a contradiction. ◀

Lemma 14 allows us to apply the inductive step to $\mathcal{I}_1$ and obtain an integral routing $\mathcal{P}_1$ of size at least

$$|\mathcal{P}_1| \geq \binom{1}{144r^3} \left(1 - \binom{1}{r}\right)^{\ell_1-1+\ell_2} \cdot |f_1| \ . \tag{2}$$

Let us now estimate the amount of flow lost by the separation into $\mathcal{I}_1$ and $\mathcal{I}_2$, i.e., $g = f - f_1 - f_2$. As every flow path in g passes through $\delta(U)$, we have $|g| \leq \mathrm{cap}(\delta(U)) < \frac{1}{4r}\mathbf{x}(U)$. Since $|f_1| + |g| \geq \frac{1}{2}\mathbf{x}(U)$ (no flow path in f_2 originates in U), we have that $|g| \leq \frac{1}{4r} \cdot 2 \cdot (|f_1| + |g|)$. Hence,

$$|g| \leq \frac{1}{2r} \cdot \left(1 - \frac{1}{2r}\right)^{-1} |f_1| \leq \frac{1}{r}|f_1| \ . \tag{3}$$

By putting up together (1), (2), and (3), we obtain that

$$\begin{aligned} |\mathcal{P}_1| + |\mathcal{P}_2| &\geq \frac{1}{144r^3}\left(1 - \frac{1}{r}\right)^{\ell_1+\ell_2}\left(|f_2| + \left(1 - \frac{1}{r}\right)^{-1}|f_1|\right) \\ &\geq \frac{1}{144r^3}\left(1 - \frac{1}{r}\right)^{\ell_1+\ell_2}(|f_2| + |f_1| + |g|) = \frac{1}{144r^3}\left(1 - \frac{1}{r}\right)^{\ell_1+\ell_2}|f| \ . \end{aligned}$$

This concludes the proof of Theorem 11. Since $\ell_1, \ell_2 \leq r$, while $(1 - \frac{1}{r})^{2r} = \Omega(1)$, Theorem 11 immediately implies the promised $\mathcal{O}(r^3)$-approximation algorithm. ◀

▶ Remark. We conclude with observing that the improved approximation ratio of $\mathcal{O}(r^3)$ directly translates to the more general setting of k-sums of graph from some minor closed family $\mathcal{G}$, as discussed by Chekuri et al. [10]. That is, if we are able to α-approximate MaxEDP with congestion β in graphs from $\mathcal{G}$, we can have $\mathcal{O}(\alpha r^5)$-approximation algorithm with congestion $(\beta + 3)$ in graphs admitting a tree decomposition of maximum adhesion size at most r, and the torso of every bag being from the class $\mathcal{G}$.

To see this, observe that the only place when our algorithm uses that the *bags* are of bounded size (as opposed to *adhesions*) is the base case, where all flow paths pass through the bag $\beta(t_0)$ of the root node t_0. However, in this case we can proceed exactly as Chekuri et al. [10]: using the flow paths, move the terminals to $\beta(t_0)$, replace connected components of $G - \beta(t_0)$ with their $(r^2, 2)$-sparsifiers, and apply the algorithm for the class $\mathcal{G}$. In addition to the $\mathcal{O}(r^3)$ approximation factor of our algorithm, the application of the algorithm for $\mathcal{G}$ incurs an approximation ratio of α and congestion of β, the use of sparsifiers adds a factor of r^2 to the approximation ratio and an additive constant $+1$ to the congestion, while the terminal move adds an additional amount of 2 to the final congestion.

4 Algorithm for MaxNDP in Bounded Pathwidth Graphs

In this section we develop an $\mathcal{O}(r^3)$-approximation algorithm for MaxNDP in graphs of *pathwidth* less than r. We follow the outline of the MaxEDP algorithm from the previous section, with few essential changes.

Most importantly, we can no longer use Proposition 8, as it refers to edge disjoint paths, and the proof of its main ingredient by Chekuri et al. [4] relies on a clustering technique that stops to work for node disjoint paths. We fix this by providing in Sect. 4.1 a node-disjoint variant of Proposition 8, using the more involved clustering approach of Chekuri et al. [7].

Then, in Sect. 4.2 we revisit step-by-step the arguments for MaxEDP, pointing out remaining differences. We remark that the use of pathwidth instead of treewidth is only essential in the inductive step for the case $\ell_1 < \ell_2$: if we follow the argument for MaxEDP for bounded-treewidth graphs, the graphs $G(t_i)$ may not be node disjoint (but they are edge disjoint), breaking the argument. Note that for bounded pathwidth graphs, there is only one such graph considered, and the issue is nonexistent.

4.1 Routing to a small adhesion in a node-disjoint setting

▶ **Proposition 15.** *Let $(G, \mathcal{M})$ be an instance of* MaxNDP *and let $(f, \mathbf{x})$ be a feasible fractional solution for the instance. Suppose that there is also a second (feasible, i.e., respecting node capacities) flow that routes at least $x(v)/\alpha$ units of flow for each v to some set $S \subseteq V$, where $\alpha \geq 1$. Then there is an integral routing of $\Omega(|f|/(\alpha|S|))$ pairs.*

Proof. Without loss of generality, we may assume that the terminals of $\mathcal{M}$ are pairwise distinct and of degree and capacity one: we can always move a terminal from a vertex t to a newly-created degree-1 capacity-1 neighbour of t.

Let g be the second flow mentioned in the statement. In what follows, we modify and simplify the flows f and g in a number of steps. We denote by $f_1, f_2, \ldots$ and $g_1, g_2, \ldots$ flows after subsequent modification steps; for the flow f_i, by $\mathbf{x}_i$ we denote its marginals.

Symmetrizing the flow g. In the first step, we construct flows f_1 and g_1 with the following property: for every terminal pair $(s,t) \in \mathcal{M}$, for every $v \in S$, g_1 sends the same amount of flow from s to v as from t to v. To obtain this goal, we first take the flow $g/3$, and then for every $(s,t) \in \mathcal{M}$ redirect the flow originating at s to first go along the commodity for the pair (s,t) in flow $f/(3\alpha)$ to the vertex t, and then go to S in exactly the same manner as the flow originating at t does. It is easy to see that g_1 consists of three feasible flows scaled down by at least $1/3$, thus it is feasible. Finally, we set $f_1 := f/3$, so that g_1 again sends an amount of $\mathbf{x}_1(v)/\alpha$ flow from every vertex v to S. Note that $|f_1| = |f|/3$.

Restricting to single vertex of S. To construct flows f_2 and g_2, pick a vertex $u \in S$ that receives the most flow in g_1. Take g_2 to be the flow g_1, restricted only to flow paths ending in u. Then, restrict f_1 to obtain f_2 as follows: for every terminal pair $(s,t) \in \mathcal{M}$, reduce the amount of flow from s to t to α times the total amount of flow sent from s to u by g_2; note that, by the previous step, it is also equal α times the total amount of flow sent from t to u by g_2. By this step, we maintain the invariant that g_2 sends $\mathbf{x}_2(v)/\alpha$ flow from every $v \in V(G)$, and we have $|f_2| \geq |f_1|/|S| \geq |f|/(3|S|)$.

Rounding to a half-integral flow. In the next step, we essentially repeat the integral rounding procedure by Chekuri et al. [4, Section 3.2]. We use the following operation as a basic step in the rounding.

▶ **Lemma 16** ([4, Theorem 2.1]). *Let G be a directed graph with edge capacities. Given a flow h in G that goes from set $X \subseteq V(G)$ to a single vertex $u \in V(G)$, such that for every $v \in X$ the amount of flow originating in v is $\mathbf{z}(v)$, and a vertex $v_0 \in X$ such that $\mathbf{z}(v_0)$ is not an integer, one can in polynomial time compute a flow h' in G, sending $\mathbf{z}'(v)$ amount of flow from every $v \in X$ to u, such that $|h'| \geq |h|$, $\mathbf{z}'(v) = \mathbf{z}(v)$ for every $v \in X$ where $\mathbf{z}(v)$ is an integer, and $\mathbf{z}'(v_0) = \lceil \mathbf{z}(v_0) \rceil$.*

Since a standard reduction reduces flows in undirected node-capacitated graphs to directed edge-capacitated ones[3], Lemma 16 applies also to undirected graphs with node capacities.

Split g_2 into two flows h_s and h_t: for every terminal pair $(s,t) \in \mathcal{M}$, we put the flow originating in s into h_s, and the flow originating in t into h_t. We perform a sequence of modifications to the flows h_s and h_t, maintaining the invariant that the same amount of flow originates in s in h_s as in t in h_t for every $(s,t) \in \mathcal{M}$. Along the process, both h_s and h_t are feasible flows, but $h_s + h_t$ may not be.

In a single step, we pick a terminal pair $(s,t) \in \mathcal{M}$ such that the amount of flow in h_s originating in s is not integral (and stop if no such pair exists). We apply Lemma 16 separately to s in h_s and to t in h_t, obtaining flows h'_s and h'_t. Finally, if for some terminal pair (s',t'), the amount of flow originating in s' in h'_s and in t' in h'_t differ, we restrict one of the flows so that both route the same amount of flow (being the minimum of the flows routed by h'_s from s' and by h'_t from t').

Since the rounding algorithm of Lemma 16 never modifies a source that already has an integral flow, this procedure stops after at most $|\mathcal{M}|$ steps. Furthermore, if in one step the flow from s has been increased from z to $\lceil z \rceil$, the total loss of flow to other pairs is $2(\lceil z \rceil - z)$.

[3] Replace every edge with two infinite-capacity arcs in both directions, and then split every vertex into two vertices, connected by an edge of capacity equal to the capacity of the vertex, with all in-edges connected to the first copy, and all out-edges connected to the second copy.

Therefore, if h_s° and h_t° are the final integral flows, we have $|h_s^\circ| + |h_t^\circ| \geq (|h_s| + |h_t|)/2 = |g_2|/2 = |f_2|/\alpha \geq |f|/(3\alpha|S|)$. We define $g_3 := (h_s^\circ + h_t^\circ)/2$; note that g_3 is a feasible flow.

Clustering a node-flow-linked set. Note that for every $(s,t) \in \mathcal{M}$, the flow g_3 routes either 0 or 1/2 flow from both s and t to u. Let $\mathcal{M}'$ be the set of pairs with flow value 1/2, and let X' be the set of terminals in $\mathcal{M}'$. Note that $|\mathcal{M}'| = |g_3|/2 \geq |f|/(6\alpha|S|)$.

Using the flow g_3, we now find a multicommodity flow that for every $(a,b) \in X' \times X'$ routes $\frac{1}{4|X'|}$ amount of flow from a to b. First, we use a flow $\frac{1}{2}g_3$ to send, for every $a \in X'$, a tuple of $|X'|$ portions of $\frac{1}{4|X'|}$ flow each from a to u. Second, we use a reversed flow $\frac{1}{2}g_3$ to send, for every $b \in X'$, a tuple of $|X'|$ portions of $\frac{1}{4|X'|}$ flow each from u to b. For every $(a,b) \in X' \times X'$, we combine one portion sent from a to u with one portion sent from u to b to obtain the commodity from a to b. We obtain the desired multicommodity flow, and we infer that X' is $\frac{1}{4}$-node-flow-linked. This allows us to apply the following clustering result:

▶ **Lemma 17** ([7, Lemma 2.7]). *If X is α-node-flow-linked in a graph G with unit node capacities, then for any $h \geq 2$ there exists a forest F in G of maximum degree $\mathcal{O}(\frac{1}{\alpha}\log h)$ such that every tree in F spans at least h nodes from X.*

Since we can assume that no capacity in G exceeds $|\mathcal{M}|$, we can replace every vertex v of capacity $\mathrm{cap}(v)$ with its $\mathrm{cap}(v)$ copies. To such unweighted graph G' we apply Lemma 17 for X', $\alpha = 1/4$ and $h = 3$, obtaining a forest F'; recall that the terminals X' are of capacity 1, thus they are kept unmodified in G'. By standard argument we split the forest F' into node-disjoint trees $T_1', T_2', \ldots, T_p'$, such that every tree T_i' contains at least three, and at most $d = \mathcal{O}(1)$ terminals of X'. By projecting the trees T_i' back onto G, we obtain a sequence of trees $T_1, T_2, \ldots, T_p$, such that every vertex $v \in V(G)$ is present in at most $\mathrm{cap}(v)$ trees T_i. Furthermore, since terminals are of capacity one, every terminal belongs to at most one tree, and every tree T_i contains at least three and at most d terminals.

In a greedy fashion, we chose a set $\mathcal{M}'' \subseteq \mathcal{M}'$ of size at least $|\mathcal{M}'|/d^2$, such that for every tree T_i, at most one terminal pair of $\mathcal{M}''$ has at least one terminal in T_i. A pair $(s,t) \in \mathcal{M}''$ is *local* if both s and t lie in the same tree T_i, and *distant* otherwise. If at least half of the pairs of $\mathcal{M}''$ are local, we can route them along trees T_i, obtaining a desired routing of size at least $|\mathcal{M}''|/2 \geq |\mathcal{M}'|/(2d^2) = \Omega(|f|/(\alpha|S|))$ and terminate the algorithm. Otherwise, we obtain a flow g_4 as follows: for every terminal t in a distant pair in $\mathcal{M}''$, we take the tree T_i it lies on, route 3/5 amount of flow along T_i equidistributed to three arbitrarily chosen terminals t^1, t^2, t^3 on T_i from $\mathcal{M}'$ (i.e., every terminal t^j receives 1/5 amount of flow), and then route the flow along the flow $\frac{2}{5}g_3$ to u. Since every tree T_i routes 3/5 amount of flow, and g_3 is a feasible flow, the flow g_4 is a feasible flow that routes 3/5 amount of flow from every terminal of $\mathcal{M}''$ to u. Furthermore, since at least half terminal pairs in $\mathcal{M}''$ is distant, we have $|g_4| \geq \frac{1}{2} \cdot 2|\mathcal{M}''| = \Omega(|f|/(\alpha|S|))$.

Final rounding of the flow. Let X'' be the set of all terminals of $\mathcal{M}''$. Since the flow g_4 routes *more than* 1/2 amount of flow for every terminal in X'', we can conclude with simple rounding the flow g_4 in the same manner as it is done by Chekuri et al. [4, Section 3]. Construct an auxiliary graph G' by adding a super-source s^* of infinite capacity, adjacent to all terminals of $\mathcal{M}''$. Extend g_4 in the natural manner, by routing every flow path first from s^* to an appropriate terminal. The extended flow g_4 is now a single source single sink flow from s^* to u in a graph with integer capacities, thus there exists an integral flow g_5 of no smaller size: $|g_5| \geq |g_4| = \frac{3}{5}|X''| = \frac{6}{5}|\mathcal{M}''|$. Hence, for at least 1/5 of the pairs $(s,t) \in \mathcal{M}''$, the flow g_5 routes a single unit of flow both from s and from t to u. By

combining these paths into a single path from s to t, we obtain an integral routing of size at least $\frac{1}{5}|\mathcal{M}''| = \Omega(|f|/(\alpha|S|))$. This finishes the proof of Proposition 15. ◀

4.2 Details of the algorithm

Equipped with Proposition 15, we can now proceed to the description of the approximation algorithm. Assume we are given an MaxNDP instance $(G, \mathcal{M})$ and a path decomposition $(\mathcal{T}, \beta)$ of G of width less than r; recall that $\mathcal{T}$ rooted in one of its endpoints. Let $(f, \mathbf{x})$ be a fractional solution to the multicommodity flow relaxation for MaxNDP, as in Sect. 2.

The definitions of safe and good node, as well as the induction scheme, are analogous.

▶ **Definition 18** (Safe node)**.** A node $t \in V(\mathcal{T})$ is *safe* with respect to $(f, \mathbf{x})$ if there is a second multicommodity flow g in $G(t)$ that satisfies the node capacities of $G(t)$ and, for each vertex $z \in \gamma(t)$, g routes at least $\binom{1}{4r} \cdot \mathbf{x}(z)$ units of flow from z to the adhesion $\sigma(t)$. The node t is *unsafe* if it is not safe.

▶ **Definition 19** (Good node)**.** A node $t \in V(\mathcal{T})$ is *good* with respect to $(f, \mathbf{x})$ if every flow path in the support of f that has an endpoint in $\gamma(t)$ also intersects $\sigma(t)$; in other words, no flow path is completely contained in $G[\alpha(t)]$. A node is *bad* if it is not good.

▶ **Theorem 20.** *Let $(G, \mathcal{M})$ be an instance of* MaxNDP *and let $(f, \mathbf{x})$ be a fractional solution for the instance, where f is a feasible multicommodity flow in G for the pairs $\mathcal{M}$ with marginals $\mathbf{x}$. Let $(\mathcal{T}, \beta)$ be a path decomposition for G of width less than r. Let ℓ_1 be the maximum size of a parent adhesion of an unsafe node, and let ℓ_2 be the maximum size of a parent adhesion of a bad node. There is a constant c and a polynomial time algorithm that constructs an integral routing of size at least $\binom{1}{cr^3} \cdot \left(1 - \binom{1}{r}\right)^{\ell_1+\ell_2} \cdot |f|$.*

Proof. As in the case of MaxEDP, we can assume that the considered graph G is connected and that no bag is empty, and thus the only empty adhesion is the parent adhesion of the root.

Base case. In the base case $\ell_1 = \ell_2 = 0$ nothing changes compared to MaxEDP: all flow paths pass through the root bag, and Proposition 15 allows to route integrally $\Omega(|f|/r)$ paths.

Inductive step when $0 \leq \ell_1 < \ell_2$. Since we are considering now a path decomposition (as opposed to tree decomposition in the previous section), there exists a single topmost bad node t° with parent adhesion of size ℓ_2. Let f^{inside} be the subflow of f consisting of all flow paths completely contained in $G[\alpha(t^\circ)]$. Since $\ell_1 < \ell_2$, the node t° is safe, and the flow witnessing it together with Proposition 15 allows to integrally route $\Omega(|f^{\text{inside}}|/r^2)$ terminal pairs. If $|f^{\text{inside}}| > |f|/r$, then we are done. Otherwise, we drop the flow f^{inside} from f, making t° and all its descendants good (thus decreasing ℓ_2 in the constructed instance), while losing only $1/r$ fraction of the flow f, and pass the instance to an inductive step.

Inductive step when $0 < \ell_1 = \ell_2$. Here again we take t° to be the lowest node of $\mathcal{T}$ that is unsafe and has parent adhesion of size ℓ_1. By the definition of an unsafe node and Menger's theorem, there exists a set $U \subseteq \alpha(t^\circ)$ such that $\mathrm{cap}(N(U)) < \frac{1}{4r}\mathbf{x}(U)$. Using the same argument as in the proof of Lemma 12, we can ensure property 12, that is that if U contains an adhesion $\sigma(t)$, it contains as well the entire set $\gamma(t)$.

As in the case of MaxEDP, we split into instances $\mathcal{I}_1$ and $\mathcal{I}_2$ by taking $G_1 = G[U]$ and $G_2 = G - N[U]$, with inherited tree decompositions from $(\mathcal{T}, \beta)$. Since all nodes with parent

adhesions of size larger than $\ell_1 = \ell_2$ are good, there are also good in instances $\mathcal{I}_i$ (i.e., Lemma 13 holds here as well) and we can again apply the inductive step to every connected component of the instance $\mathcal{I}_2$ with the same values of ℓ_1 and ℓ_2, obtaining a routing $\mathcal{P}_2$ of size as in (1) (with 144 replaced by a constant c).

We analyse the instance $\mathcal{I}_1$, without breaking it first into connected components. That is, we argue that in $\mathcal{I}_1$ the value of ℓ_1 dropped, that is, all nodes t satisfying $|\sigma(t)| = |\sigma_1(t)| = \ell_1$ are safe; note that they will remain safe once we consider every connected component separatedly. However, this fact follows from property 12 of the set U (Lemma 12): if for some node t we have $|\sigma(t)| = |\sigma_1(t)|$, it follows that $\sigma(t) \subseteq U$ hence $\gamma(t) \subseteq U$ and the notion of safeness for t is the same in $\mathcal{I}_1$ and in the original instance. However, $\sigma(t) \subseteq U \subseteq \alpha(t^\circ)$ implies $t \preceq t^\circ$ and $t \neq t^\circ$, hence t is safe in the original instance.

Consequently, an application of inductive step for every connected component of $\mathcal{I}_1$ uses strictly smaller value of ℓ_1, and we obtain an integral routing $\mathcal{P}_1$ in $\mathcal{I}_1$ of size as in (2) (again with 144 replaced by a constant c). The remainder of the analysis from the previous section does not change, concluding the proof of Theorem 20. ◀

References

1 Alexandr Andoni, Anupam Gupta, and Robert Krauthgamer. Towards $(1+\varepsilon)$-approximate flow sparsifiers. In *Proc. SODA 2014*, pages 279–293, 2014.

2 Matthew Andrews, Julia Chuzhoy, Venkatesan Guruswami, Sanjeev Khanna, Kunal Talwar, and Lisa Zhang. Inapproximability of edge-disjoint paths and low congestion routing on undirected graphs. *Combinatorica*, 30(5):485–520, 2010.

3 Parinya Chalermsook, Julia Chuzhoy, Alina Ene, and Shi Li. Approximation algorithms and hardness of integral concurrent flow. In *Proc. STOC 2012*, pages 689–708, 2012.

4 C. Chekuri, S. Khanna, and F.B. Shepherd. An $O(\sqrt{n})$ approximation and integrality gap for disjoint paths and unsplittable flow. *Theory Comput.*, 2(7):137–146, 2006.

5 Chandra Chekuri and Julia Chuzhoy. Large-treewidth graph decompositions and applications. In *Proc. STOC 2013*, pages 291–300, 2013.

6 Chandra Chekuri and Alina Ene. Poly-logarithmic approximation for maximum node disjoint paths with constant congestion. In *Proc. SODA 2013*, pages 326–341, 2013.

7 Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. Multicommodity flow, well-linked terminals, and routing problems. In *Proc. STOC 2005*, pages 183–192, 2005.

8 Chandra Chekuri, Sanjeev Khanna, and F Bruce Shepherd. Edge-disjoint paths in planar graphs with constant congestion. *SIAM J. Comput.*, 39(1):281–301, 2009.

9 Chandra Chekuri, Sanjeev Khanna, and F Bruce Shepherd. A note on multiflows and treewidth. *Algorithmica*, 54(3):400–412, 2009.

10 Chandra Chekuri, Guyslain Naves, and F. Bruce Shepherd. Maximum edge-disjoint paths in k-sums of graphs. In *Proc. ICALP 2013*, volume 7965 of *LNCS*, pages 328–339, 2013.

11 Chandra Chekuri, Guyslain Naves, and F. Bruce Shepherd. Maximum edge-disjoint paths in k-sums of graphs. Technical report, ArXiv, 2013. URL: `http://arxiv.org/abs/1303.4897`.

12 Eden Chlamtac, Robert Krauthgamer, and Prasad Raghavendra. Approximating sparsest cut in graphs of bounded treewidth. In *Proc. APPROX-RANDOM 2010*, volume 6302 of *LNCS*, pages 124–137, 2010.

13 Julia Chuzhoy. On vertex sparsifiers with Steiner nodes. In *Proc. STOC 2012*, pages 673–688, 2012.

14 Julia Chuzhoy and Shi Li. A polylogarithmic approximation algorithm for edge-disjoint paths with congestion 2. In *Proc. FOCS 2012*, pages 233–242, 2012.

15 Reinhard Diestel. *Graph theory*, volume 173. Springer, third edition, 2005.

16 Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

17 Anupam Gupta, Kunal Talwar, and David Witmer. Sparsest cut on bounded treewidth graphs: algorithms and hardness results. In *Proc. STOC 2013*, pages 281–290, 2013.

18 R.M. Karp. On the computational complexity of combinatorial problems. *Networks*, 5:45–68, 1975.

19 S.G. Kolliopoulos and C. Stein. Approximating disjoint-path problems using packing integer programs. *Math. Prog.*, 99(1):63–87, 2004.

20 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity – Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.

21 Neil Robertson and Paul D Seymour. Graph minors. V. Excluding a planar graph. *J. Combinatorial Theory, Ser. B*, 41(1):92–114, 1986.

22 Neil Robertson and Paul D Seymour. Graph minors. XIII. The disjoint paths problem. *J. Combinatorial Theory, Ser. B*, 63(1):65–110, 1995.

23 Neil Robertson and Paul D Seymour. Graph minors. XVI. Excluding a non-planar graph. *J. Combinatorial Theory, Ser. B*, 89(1):43–76, 2003.

24 Löc Séguin-Charbonneau and F Bruce Shepherd. Maximum edge-disjoint paths in planar graphs with congestion 2. In *Proc. FOCS 2011*, pages 200–209, 2011.

Colouring Diamond-free Graphs*

Konrad K. Dabrowski[1], François Dross[2], and Daniël Paulusma[3]

1 **School of Engineering and Computing Sciences, Durham University, Durham, United Kingdom**
konrad.dabrowski@durham.ac.uk

2 **Université de Montpellier - Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, Montpellier, France**
francois.dross@ens-lyon.fr

3 **School of Engineering and Computing Sciences, Durham University, Durham, United Kingdom**
daniel.paulusma@durham.ac.uk

Abstract

The Colouring problem is that of deciding, given a graph G and an integer k, whether G admits a (proper) k-colouring. For all graphs H up to five vertices, we classify the computational complexity of Colouring for (diamond, H)-free graphs. Our proof is based on combining known results together with proving that the clique-width is bounded for (diamond, P_1+2P_2)-free graphs. Our technique for handling this case is to reduce the graph under consideration to a k-partite graph that has a very specific decomposition. As a by-product of this general technique we are also able to prove boundedness of clique-width for four other new classes of (H_1, H_2)-free graphs. As such, our work also continues a recent systematic study into the (un)boundedness of clique-width of (H_1, H_2)-free graphs, and our five new classes of bounded clique-width reduce the number of open cases from 13 to 8.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases colouring, clique-width, diamond-free, graph class, hereditary graph class

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.16

1 Introduction

The Colouring problem is that of testing whether a given graph can be coloured with at most k colours for some given integer k, such that any two adjacent vertices receive different colours. The complexity of Colouring is fully understood for general graphs: it is NP-complete even if $k = 3$ [35]. Therefore it is natural to study its complexity when the input is restricted. A classic result in this area is due to Grötschel, Lovász, and Schrijver [26], who proved that Colouring is polynomial-time solvable for perfect graphs.

As surveyed in [14, 20, 25, 42], Colouring has been well studied for hereditary graph classes, that is, classes that can be defined by a family $\mathcal{H}$ of forbidden induced subgraphs. For a family $\mathcal{H}$ consisting of one single forbidden induced subgraph H, the complexity of Colouring is completely classified: the problem is polynomial-time solvable if H is an induced subgraph of P_4 or $P_1 + P_3$ and NP-complete otherwise [34]. Hence, many papers (e.g. [13, 18, 29, 34, 36, 39, 40, 44]) have considered the complexity of Colouring for bigenic hereditary graph classes, that is, graph classes defined by families $\mathcal{H}$ consisting of

* First and last author supported by EPSRC (EP/K025090/1).

two forbidden graphs H_1 and H_2; such classes of graphs are also called (H_1, H_2)-free. This classification is far from complete (see [25] for the state of art). In fact there are still an infinite number of open cases, including cases where both H_1 and H_2 are small. For instance, Lozin and Malyshev [36] determined the computational complexity of Colouring for (H_1, H_2)-free graphs for all graphs H_1 and H_2 up to four vertices except when $\mathcal{H} \in \{(K_{1,3}, 4P_1), (K_{1,3}, 2P_1 + P_2), (C_4, 4P_1)\}$ (we refer to Section 2 for notation and terminology).

The *diamond* is the graph $\overline{2P_1 + P_2}$, that is, the graph obtained from the clique on four vertices by removing an edge. Diamond-free graphs are well studied in the literature. For instance, Tucker [45] gave an $O(kn^2)$ time algorithm for Colouring for perfect diamond-free graphs. It is also known that that Colouring is polynomial-time solvable for diamond-free graphs that contain no even induced cycles [32] as well as for diamond-free graphs that contain no induced cycle of length at least 5 [8]. Diamond-free graphs also played an important role in proving that the class P_6-free graphs contains 24 minimal obstructions for 4-Colouring [15].

1.1 Our Main Result

In this paper we focus on Colouring for (diamond, H)-free graphs where H is a graph on at most five vertices. It is known that Colouring is NP-complete for (diamond, H)-free graphs when H contains a cycle or a claw [34] and polynomial-time solvable for $H = sP_1 + P_2$ ($s \geq 0$) [18], $H = 2P_1 + P_3$ [5], $H = P_1 + P_4$ [11], $H = P_2 + P_3$ [19] and $H = P_5$ [1]. Hence, the only graph H on five vertices that remains is $H = P_1 + 2P_2$, for which we prove polynomial-time solvability in this paper. This leads to the following result.

▶ **Theorem 1.** *Let H be a graph on at most five vertices. Then* Colouring *is polynomial-time solvable for (diamond, H)-free graphs if H is a linear forest and* NP*-complete otherwise.*

To solve the case $H = P_1 + 2P_2$, one could try to reduce to a subclass of diamond-free graphs, for which Colouring is polynomial-time solvable, such as the aforementioned results of [8, 32, 45]. This would require us to deal with the presence of small cycles up to C_7, which may not be straightforward. Instead we aim to identify tractability from an underlying property: we show that the class of (diamond, $P_1 + 2P_2$)-free graphs has bounded clique-width. This approach has several advantages and will lead to a number of additional results, as we will discuss in the remainder of Section 1.

Clique-width is a graph decomposition that can be constructed via vertex labels and four specific graph operations, which ensure that vertices labelled alike will always keep the same label and thus behave identically. The clique-width of a graph G is the minimum number of different labels needed to construct G using these four operations (we refer to Section 2 for a precise definition). A graph class $\mathcal{G}$ has *bounded* clique-width if there exists a constant c such that every graph from $\mathcal{G}$ has clique-width at most c.

Clique-width is a well-studied graph parameter (see, for instance, the surveys [27, 31]). An important reason for the popularity of clique-width is that a number of classes of NP-complete problems, such as those that are definable in Monadic Second Order Logic using quantifiers on vertices but not on edges, become polynomial-time solvable on any graph class $\mathcal{G}$ of bounded clique-width (this follows combining results from [16, 23, 33, 43] with a result from [41]). The Colouring problem is one of the best-known NP-complete problems that is solvable in polynomial time on graph classes of bounded clique-width [33]; another well-known example of such a problem is Hamilton Path [23].

1.2 Methodology

The key technique for proving that (diamond, $P_1 + 2P_2$)-free graphs have bounded clique-width is the use of a certain graph decomposition of k-partite graphs. We obtain this decomposition by generalizing the so-called canonical decomposition of bipartite graphs, which decomposes a bipartite graph into two smaller bipartite graphs such that edges between these two smaller bipartite graphs behave in a very restricted way. Fouquet, Giakoumakis and Vanherpe [24] introduced this decomposition and characterized exactly those bipartite graphs that can recursively be canonically decomposed into graphs isomorphic to K_1. Such bipartite graphs are said to be totally decomposable by canonical decomposition. We say that k-partite graphs are totally k-decomposable if they can be, according to our generalized definition, recursively k-decomposed into graphs isomorphic to K_1. We show that totally k-decomposable graphs have clique-width at most $2k$.

Our goal is to transform (diamond, $P_1 + 2P_2$)-free graphs into graphs in some class for which we already know that the clique-width is bounded. Besides the class of totally k-decomposable graphs, we will also reduce to other known graph classes of bounded clique-width, such as the class of (diamond, $P_2 + P_3$)-free graphs [19] and certain classes of H-free bipartite graphs [21]. Of course, our transformations must not change the clique-width by "too much". We ensure this by using certain graph operations that are known to preserve (un)boundedness of clique-width [31, 37].

1.3 Consequences for Clique-Width

There are numerous papers (as listed in, for instance, [22, 27, 31]) that determine the (un)boundedness of the clique-width or variants of it (see e.g. [4, 28]) of special graph classes. Due to the complex nature of clique-width, proofs of these results are often long and technical, and there are still many open cases. In particular gaps exist in a number of dichotomies on the (un)boundedness of clique-width for graph classes defined by one or more forbidden induced subgraphs. As such our paper also continues a line of research [5, 6, 19, 21, 22] in which we focus on these gaps in a *systematic* way. It is known [22] that the class of H-free graphs has bounded clique-width if and only if H is an induced subgraph of P_4. Over the years many partial results [2, 7, 9, 10, 11, 12, 20, 38] on the (un)boundedness of clique-width appeared for classes of (H_1, H_2)-free graphs, but until recently [22] it was not even known whether the number of missing cases was bounded. Combining these older results with recent progress [5, 18, 19, 22] reduced the number of open cases to 13 (up to an equivalence relation) [22].

As a by-product of our methodology, we are able not only to settle the case $(H_1, H_2) =$ (diamond, $P_1 + 2P_2$), but in fact we solve **five** of the remaining 13 open cases by proving that the class of (H_1, H_2)-free graphs has bounded clique-width if

1–4: $H_1 = K_3$ and $H_2 \in \{P_1 + 2P_2, P_1 + P_2 + P_3, P_1 + P_5, S_{1,2,2}\}$ or

5: $H_1 =$ diamond and $H_2 = P_1 + 2P_2$.

The above graphs are displayed in Figure 1. Note that the $(K_3, P_1 + 2P_2)$ case is properly contained in all four of the other cases. These four other newly solved cases are pairwise incomparable.

Updating the classification (see [22]) with our five new results gives the following theorem. Here, $\mathcal{S}$ is the class of graphs, each connected component of which is either a subdivided claw or a path, and we write $H \subseteq_i G$ if H is an induced subgraph of G; see Section 2 for notation that we have not formally defined yet.

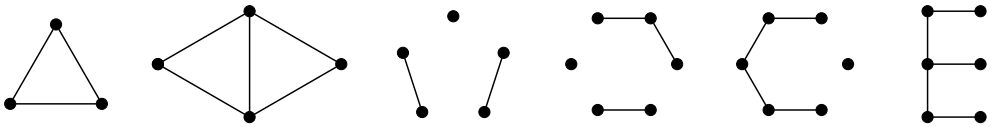


Figure 1 The forbidden graphs considered in this paper.

▶ **Theorem 2.** *Let $\mathcal{G}$ be a class of graphs defined by two forbidden induced subgraphs. Then:*

1. *$\mathcal{G}$ has bounded clique-width if it is equivalent*[1] *to a class of (H_1, H_2)-free graphs such that one of the following holds:*
   - **(a)** *H_1 or $H_2 \subseteq_i P_4$;*
   - **(b)** *$H_1 = sP_1$ and $H_2 = K_t$ for some s, t;*
   - **(c)** *$H_1 \subseteq_i P_1 + P_3$ and $\overline{H_2} \subseteq_i K_{1,3} + 3P_1$, $K_{1,3} + P_2$, $P_1 + P_2 + P_3$, $P_1 + P_5$, $P_1 + S_{1,1,2}$, P_6, $S_{1,2,2}$ or $S_{1,1,3}$;*
   - **(d)** *$H_1 \subseteq_i 2P_1 + P_2$ and $\overline{H_2} \subseteq_i P_1 + 2P_2$, $2P_1 + P_3$, $3P_1 + P_2$ or $P_2 + P_3$;*
   - **(e)** *$H_1 \subseteq_i P_1 + P_4$ and $\overline{H_2} \subseteq_i P_1 + P_4$ or P_5;*
   - **(f)** *$H_1 \subseteq_i 4P_1$ and $\overline{H_2} \subseteq_i 2P_1 + P_3$;*
   - **(g)** *$H_1, \overline{H_2} \subseteq_i K_{1,3}$.*
8. *$\mathcal{G}$ has unbounded clique-width if it is equivalent to a class of (H_1, H_2)-free graphs such that one of the following holds:*
   - **(a)** *$H_1 \notin \mathcal{S}$ and $H_2 \notin \mathcal{S}$;*
   - **(b)** *$\overline{H_1} \notin \mathcal{S}$ and $\overline{H_2} \notin \mathcal{S}$;*
   - **(c)** *$H_1 \supseteq_i K_{1,3}$ or $2P_2$ and $\overline{H_2} \supseteq_i 4P_1$ or $2P_2$;*
   - **(d)** *$H_1 \supseteq_i 2P_1 + P_2$ and $\overline{H_2} \supseteq_i K_{1,3}$, $5P_1$, $P_2 + P_4$ or P_6;*
   - **(e)** *$H_1 \supseteq_i 3P_1$ and $\overline{H_2} \supseteq_i 2P_1 + 2P_2$, $2P_1 + P_4$, $4P_1 + P_2$, $3P_2$ or $2P_3$;*
   - **(f)** *$H_1 \supseteq_i 4P_1$ and $\overline{H_2} \supseteq_i P_1 + P_4$ or $3P_1 + P_2$.*

1.4 Future Work

Naturally we would like to extend Theorem 1 and solve the following open problem.

▶ **Open Problem 1.** *What is the computational complexity of* Colouring *for (diamond, H)-free graphs when H is a graph on at least six vertices?*

Solving Open Problem 1 is highly non-trivial. It is known that 4-Colouring is NP-complete for (C_3, P_{22})-free graphs [30]. Hence, the polynomial-time results in Theorem 1 cannot be extended to all linear forests. The first open case to consider would be $H = P_6$, for which only partial results are known. Indeed, the Colouring problem is polynomial-time solvable for (C_3, P_6)-free graphs [9], but its complexity is unknown for (C_3, P_7)-free graphs (on a side note, a recent result for the latter graph class is that 3-Colouring is polynomial-time solvable [3]).

[1] Given four graphs H_1, H_2, H_3, H_4, the class of (H_1, H_2)-free graphs and the class of (H_3, H_4)-free graphs are *equivalent* if the unordered pair H_3, H_4 can be obtained from the unordered pair H_1, H_2 by some combination of the operations (i) complementing both graphs in the pair and (ii) if one of the graphs in the pair is K_3, replacing it with $\overline{P_1 + P_3}$ or vice versa. If two classes are equivalent, then one of them has bounded clique-width if and only if the other one does (see [22]).

We observe that boundedness of the clique-width of (diamond, $P_1 + 2P_2$)-free graphs implies boundedness of the clique-width of $(2P_1 + P_2, \overline{P_1 + 2P_2})$-free graphs (recall that the diamond is the complement of the graph $2P_1 + P_2$). Hence our results imply that Colouring can also be solved in polynomial time for graphs in this class. In fact, Colouring has been studied extensively for (H_1, H_2)-free graphs, and we refer to the survey of Golovach et al. [25] for a summary of known results. After incorporating the consequences of our new results, there are 13 classes of (H_1, H_2)-free graphs for which Colouring could potentially still be solved in polynomial time by showing that their clique-width is bounded (see also [25]):

▶ **Open Problem 2.** *Is* Colouring *polynomial-time solvable for* (H_1, H_2)*-free graphs when:*

1. $\overline{H_1} \in \{3P_1, P_1 + P_3\}$ *and* $H_2 \in \{P_1 + S_{1,1,3}, S_{1,2,3}\}$;
2. $H_1 = 2P_1 + P_2$ *and* $\overline{H_2} \in \{P_1 + P_2 + P_3, P_1 + P_5\}$;
3. $H_1 =$ *diamond and* $H_2 \in \{P_1 + P_2 + P_3, P_1 + P_5\}$;
4. $H_1 = P_1 + P_4$ *and* $\overline{H_2} \in \{P_1 + 2P_2, P_2 + P_3\}$;
5. $\overline{H_1} = P_1 + P_4$ *and* $H_2 \in \{P_1 + 2P_2, P_2 + P_3\}$;
6. $H_1 = \overline{H_2} = 2P_1 + P_3$.

As mentioned in Section 1.3, after updating the list of remaining open cases for clique-width from [22], we find that eight non-equivalent open cases remain for clique-width. These are the following cases.

▶ **Open Problem 3.** *Does the class of* (H_1, H_2)*-free graphs have bounded or unbounded clique-width when:*

1. $H_1 = 3P_1$ *and* $\overline{H_2} \in \{P_1 + S_{1,1,3}, P_2 + P_4, S_{1,2,3}\}$;
2. $H_1 = 2P_1 + P_2$ *and* $\overline{H_2} \in \{P_1 + P_2 + P_3, P_1 + P_5\}$;
3. $H_1 = P_1 + P_4$ *and* $\overline{H_2} \in \{P_1 + 2P_2, P_2 + P_3\}$ *or*
4. $H_1 = \overline{H_2} = 2P_1 + P_3$.

Bonomo, Grippo, Milanič and Safe [4] determined all pairs of connected graphs H_1, H_2 for which the class of (H_1, H_2)-free graphs has power-bounded clique-width. In order to compare their result with our results for clique-width, we only need to solve the open case $(H_1, H_2) = (K_3, S_{1,2,3})$, which is equivalent to the (open) case $(H_1, H_2) = (3P_1, \overline{S_{1,2,3}})$ mentioned in Open Problem 3, as our new result for the case $(H_1, H_2) = (K_3, S_{1,2,2})$ has reduced the number of open cases (H_1, H_2) with H_1, H_2 both connected from two to one.

2 Preliminaries

Below we define further graph terminology used throughout our paper. The *disjoint union* $(V(G) \cup V(H), E(G) \cup E(H))$ of two vertex-disjoint graphs G and H is denoted by $G + H$ and the disjoint union of r copies of a graph G is denoted by rG. The *complement* of a graph G, denoted by $\overline{G}$, has vertex set $V(\overline{G}) = V(G)$ and an edge between two distinct vertices if and only if these vertices are not adjacent in G. For a subset $S \subseteq V(G)$, we let $G[S]$ denote the subgraph of G *induced* by S, which has vertex set S and edge set $\{uv \mid u, v \in S, uv \in E(G)\}$. If $S = \{s_1, \ldots, s_r\}$ then, to simplify notation, we may also write $G[s_1, \ldots, s_r]$ instead of $G[\{s_1, \ldots, s_r\}]$. We use $G \setminus S$ to denote the graph obtained from G by deleting every vertex in S, i.e. $G \setminus S = G[V(G) \setminus S]$. Let H be another graph. We write $H \subseteq_i G$ to indicate that H is an induced subgraph of G.

The graphs $C_r, K_r, K_{1,r-1}$ and P_r denote the cycle, complete graph, star and path on r vertices, respectively. The graph $K_{1,3}$ is also called the *claw*. The graph $S_{h,i,j}$, for

$1 \leq h \leq i \leq j$, denotes the *subdivided claw*, that is, the tree that has only one vertex x of degree 3 and exactly three leaves, which are of distance h, i and j from x, respectively. Observe that $S_{1,1,1} = K_{1,3}$. The graph $S_{1,2,2}$ is also known as the E, since it can be drawn like a capital letter E (see Figure 1). Recall that the graph $\overline{P_1 + 2P_2}$ is known as the *diamond*. The graphs K_3 and $\overline{P_1 + 2P_2}$ are also known as the *triangle* and the 5-vertex *wheel*, respectively. For a set of graphs $\{H_1, \ldots, H_p\}$, a graph G is $(H_1, \ldots, H_p)$*-free* if it has no induced subgraph isomorphic to a graph in $\{H_1, \ldots, H_p\}$; if $p = 1$, we may write H_1-free instead of (H_1)-free.

For a graph $G = (V, E)$, the set $N(u) = \{v \in V \mid uv \in E\}$ denotes the neighbourhood of $u \in V$. A graph is k*-partite* if its vertex set can be partitioned into k independent sets (some of which may be empty). A graph is *bipartite* if it is 2-partite. The *bipartite complement* of a bipartite graph G with bipartition (X, Y) is the graph obtained from G by replacing every edge from a vertex in X to a vertex in Y by a non-edge and vice versa. The *biclique* $K_{r,s}$ is the bipartite graph with sets in the partition of size r and s respectively, such that every vertex in one set is adjacent to every vertex in the other set.

Let X be a set of vertices in a graph $G = (V, E)$. A vertex $y \in V \setminus X$ is *complete* to X if it is adjacent to every vertex of X and *anti-complete* to X if it is non-adjacent to every vertex of X. Similarly, a set of vertices $Y \subseteq V \setminus X$ is *complete* (resp. *anti-complete*) to X if every vertex in Y is complete (resp. anti-complete) to X. A vertex y or a set Y is *trivial* to X if it is either complete or anti-complete to X. Note that if Y contains both vertices complete to X and vertices not complete to X, we may have a situation in which every vertex in Y is trivial to X, but Y itself is not trivial to X.

Clique-Width. The *clique-width* of a graph G, denoted $\mathrm{cw}(G)$, is the minimum number of labels needed to construct G by using the following four operations:

1. creating a new graph consisting of a single vertex v with label i;
2. taking the disjoint union of two labelled graphs G_1 and G_2;
3. joining each vertex with label i to each vertex with label j $(i \neq j)$;
4. renaming label i to j.

An algebraic term that represents such a construction of G and uses at most k labels is said to be a k*-expression* of G (i.e. the clique-width of G is the minimum k for which G has a k-expression). Recall that a class of graphs $\mathcal{G}$ has bounded clique-width if there is a constant c such that the clique-width of every graph in $\mathcal{G}$ is at most c; otherwise the clique-width of $\mathcal{G}$ is *unbounded*.

Let G be a graph. We define the following operations. For an induced subgraph $G' \subseteq_i G$, the *subgraph complementation* operation (acting on G with respect to G') replaces every edge present in G' by a non-edge, and vice versa. Similarly, for two disjoint vertex subsets S and T in G, the *bipartite complementation* operation with respect to S and T acts on G by replacing every edge with one end-vertex in S and the other one in T by a non-edge and vice versa.

We now state some useful facts about how the above operations (and some other ones) influence the clique-width of a graph. We will use these facts throughout the paper. Let $k \geq 0$ be a constant and let γ be some graph operation. We say that a graph class $\mathcal{G}'$ is (k, γ)*-obtained* from a graph class $\mathcal{G}$ if the following two conditions hold:

1. every graph in $\mathcal{G}'$ is obtained from a graph in $\mathcal{G}$ by performing γ at most k times, and
2. for every $G \in \mathcal{G}$ there exists at least one graph in $\mathcal{G}'$ obtained from G by performing γ at most k times.

We say that γ *preserves* boundedness of clique-width if for any finite constant k and any graph class $\mathcal{G}$, any graph class $\mathcal{G}'$ that is (k,γ)-obtained from $\mathcal{G}$ has bounded clique-width if and only if $\mathcal{G}$ has bounded clique-width.

Fact 1: Vertex deletion preserves boundedness of clique-width [37].

Fact 2: Subgraph complementation preserves boundedness of clique-width [31].

Fact 3: Bipartite complementation preserves boundedness of clique-width [31].

Two vertices are *false twins* if they have the same neighbourhood. (note that such vertices must be non-adjacent). The following lemma follows immediately from the definition of clique-width.

▶ **Lemma 3.** *If a vertex x in a graph G has a false twin then* $\mathrm{cw}(G) = \mathrm{cw}(G \setminus \{x\})$.

We will also make use of the following two results.

▶ **Lemma 4** ([19]). *The class of (diamond, $P_2 + P_3$)-free graphs has bounded clique-width.*

▶ **Lemma 5** ([21]). *Let H be a graph. The class of H-free bipartite graphs has bounded clique-width if and only if $H = sP_1$ for some $s \geq 1$; $H \subseteq_i K_{1,3} + 3P_1$; $H \subseteq_i K_{1,3} + P_2$; $H \subseteq_i P_1 + S_{1,1,3}$; or $H \subseteq_i S_{1,2,3}$.*

3 Totally k-Decomposable Graphs

In this section we describe our key technique, which is based on the following notion introduced by Fouquet, Giakoumakis and Vanherpe [24]. A bipartite graph G is *totally decomposable by canonical decomposition* if it can be recursively decomposed into graphs isomorphic to K_1 by decomposition of a bipartite graph G with bipartition (V_1, V_2) into two non-empty graphs $G[V_1' \cup V_2']$ and $G[V_1'' \cup V_2'']$ where V_i' and V_i'' form a partition of V_i for $i \in \{1,2\}$ such that each of $G[V_1' \cup V_2'']$ and $G[V_1'' \cup V_2']$ is either an independent set or a biclique.

For our purposes we need to generalize the above notion to k-partite graphs. Let G be a k-partite graph with a fixed vertex k-partition $(V_1, V_2, \ldots, V_k)$. We say that a *k-decomposition of G* with respect to this partition consists of two non-empty graphs, each with their own partition: $G[V_1' \cup V_2' \cup \cdots \cup V_k']$ with partition $(V_1', V_2', \ldots, V_k')$ and $G[V_1'' \cup V_2'' \cup \cdots \cup V_k'']$ with partition $(V_1'', V_2'', \ldots, V_k'')$, such that the following two conditions hold:

1. for every $i \in \{1, \ldots, k\}$, V_i' and V_i'' form a partition of V_i, and
2. for every $i, j \in \{1, \ldots, k\}$ with $i \neq j$, the set V_i' is either complete or anti-complete to V_j'' in G (note that V_i is an independent set for every $i \in \{1, \ldots, k\}$, so V_i' will automatically be anti-complete to V_i'').

We say that G is *totally k-decomposable* if it can be recursively k-decomposed into graphs isomorphic to K_1. Note that every connected bipartite graph has a unique bipartition (up to isomorphism). If a graph is totally decomposable by canonical decomposition then this can recursively be done component-wise. Thus the definition of total canonical decomposability is indeed the same as total 2-decomposability. Fouquet, Giakoumakis and Vanherpe proved the following characterization, which we will need for our proofs (see Figure 2 for pictures of P_7 and $S_{1,2,3}$).

▶ **Lemma 6** ([24]). *A bipartite graph is totally decomposable by canonical decomposition if and only if it is $(P_7, S_{1,2,3})$-free.*

It seems difficult to generalize Lemma 6 to give a full characterization for totally k-decomposable graphs for $k \geq 3$. However, the following lemma is sufficient for our purposes.

P_7 $S_{1,2,3}$

Figure 2 The forbidden graphs from Lemma 6.

▶ **Lemma 7.** *A 3-partite graph is totally 3-decomposable with respect to a 3-partition (V_1, V_2, V_3) if the following two conditions are both satisfied:*

- *$G[V_1 \cup V_2], G[V_1 \cup V_3]$ and $G[V_2 \cup V_3]$ are all $(P_7, S_{1,2,3})$-free, and*
- *for every $v_1 \in V_1$, every $v_2 \in V_2$ and every $v_3 \in V_3$, the graph $G[v_1, v_2, v_3]$ is isomorphic neither to K_3 nor to $3P_1$.*

Proof. Let G be such a graph. Note that any induced subgraph H of G also satisfies the hypotheses of the lemma, with partition $(V(H) \cap V_1, V(H) \cap V_2, V(H) \cap V_3)$. It is therefore sufficient to show that G has a 3-decomposition.

If V_i is empty for some $i \in \{1, 2, 3\}$ then G is a $(P_7, S_{1,2,3})$-free bipartite graph and is therefore totally 2-decomposable with respect to the given partition by Lemma 6. We may therefore assume that every set V_i is non-empty.

Now $G[V_1, V_2]$ is a bipartite $(P_7, S_{1,2,3})$-free graph, so by Lemma 6, $G[V_1 \cup V_2]$ is totally 2-decomposable. Since V_1 and V_2 are both non-empty, it follows that V_1 can be partitioned into two sets V_1' and V_1'' and V_2 can be partitioned into two sets V_2' and V_2'' such that V_1' is either complete or anti-complete to V_2'' and V_2' is either complete or anti-complete to V_1''. Furthermore, we may assume $V_1' \cup V_2' \neq \emptyset$ and $V_1'' \cup V_2'' \neq \emptyset$.

Since $V_1, V_2, V_1' \cup V_2'$ and $V_1'' \cup V_2''$ are non-empty, we may assume without loss of generality that V_1' and V_2'' are non-empty. Assume that these sets are maximal i.e. no vertex of V_1'' (respectively V_2') can be moved to V_1' (respectively V_2''). Note that V_1'' or V_2' may be empty.

We will prove that we can partition V_3 into sets V_3' and V_3'', such that for all $i, j \in \{1, 2, 3\}$ with $j \neq i$, V_i' is complete or anti-complete to V_j''. Note that we already know that V_1' (respectively V_2') is complete or anti-complete to V_2'' (respectively V_1'').

First suppose that V_1' is complete to V_2''. If a vertex of V_3 has a neighbour in both V_1' and V_2'' then these three vertices would form a forbidden K_3, so every vertex in V_3 is anti-complete to V_1' or V_2''. Let V_3' be the set of vertices in V_3 that are anti-complete to V_2'' and let $V_3'' = V_3 \setminus V_3'$. Note that every vertex of V_3'' must be anti-complete to V_1'. Suppose, for contradiction, that $z \in V_3'$ has a non-neighbour $v \in V_1''$. Since V_1' is maximal, v must have a non-neighbour $w \in V_2''$. This means that $G[v, w, z]$ is a $3P_1$. This contradiction means that V_1'' is complete to V_3'. Similarly, V_2' is complete to V_3''. Therefore $G[V_1' \cup V_2' \cup V_3']$ and $G[V_1'' \cup V_2'' \cup V_3'']$ form the required 3-decomposition of G.

Similarly, if V_1' is anti-complete to V_2'' then V_3 can be partitioned into sets V_3' and V_3'' that are complete to V_2'' and V_1', respectively. By analogous arguments, we find that V_1'' is anti-complete to V_3' and V_2' is anti-complete to V_3''. We then proceed as in the previous case. This completes the proof. ◀

We also need the following lemma.

▶ **Lemma 8.** *Let G be a k-partite graph with vertex partition $(V_1, \ldots, V_k)$. If G is totally k-decomposable with respect to this partition then the clique-width of G is at most $2k$. Moreover, there is a $2k$-expression for G that assigns, for $i \in \{1, \ldots, k\}$, label i to every vertex of V_i.*

Proof. We prove the lemma by induction. Clearly, if G contains only one vertex then the lemma holds. Suppose that the lemma is true for all such graphs on at most n vertices.

Let G be a totally k-decomposable graph on $n+1$ vertices with vertex partition $(V_1,\ldots,V_k)$. Since G has a k-decomposition, we can partition every set V_i into two sets V_i' and V_i'' such that each set V_i' is either complete or anti-complete to each set V_j'' for $i,j\in\{1,\ldots,k\}$. By the induction hypothesis, we can find a $2k$-expression that constructs the non-empty graph $G[V_1'\cup V_2'\cup\cdots\cup V_k']$ such that the vertices in each set V_i' have label i for $i\in\{1,\ldots,k\}$. Similarly, we can find a $2k$-expression that constructs the non-empty graph $G[V_1''\cup V_2''\cup\cdots\cup V_k'']$ such that the vertices in each set V_j'' have label $k+j$ for $j\in\{1,\ldots,k\}$. We take the disjoint union of these two constructions. Next, for $i,j\in\{1,\ldots,k\}$, we join the vertices label i to the vertices label $k+j$ if V_i' is complete to V_j'' in G. Finally, for $i\in\{1,\ldots,k\}$, we relabel the vertices with label $k+i$ to have label i. By induction, this completes the proof of the lemma. ◀

4 Bounding the Clique-Width

To prove our results on clique-width we need two more lemmas. The first lemma (we omit the proof due to space restrictions[2]) implies that the four triangle-free cases in our new results hold when the graph under consideration is C_5-free. In the second lemma we state a number of sufficient conditions for a graph class to be of bounded clique-width when C_5 is no longer a forbidden induced subgraph. While we will not use these lemmas directly in the proof of the diamond-free case, that result also relies on these two lemmas, as it depends on the (K_3, P_1+2P_2)-free case.

▶ **Lemma 9.** *The class of $(K_3,C_5,S_{1,2,3})$-free graphs has bounded clique-width.*

▶ **Lemma 10.** *A $(K_3,S_{1,2,3})$-free graph has bounded clique-width if its vertices can be partitioned into ten independent sets $V_1,\ldots,V_5,W_1,\ldots,W_5$ such that the following conditions hold (we interpret subscripts modulo 5):*

1. *for all i, V_i is anti-complete to $V_{i-2}\cup V_{i+2}\cup W_{i-1}\cup W_{i+1}$;*
2. *for all i, W_i is complete to $W_{i-1}\cup W_{i+1}$;*
3. *for all i, each vertex of V_i is either trivial to V_{i+1} or trivial to V_{i-1};*
4. *for all i, every vertex in V_i is trivial to W_i;*
5. *for all i, W_i is trivial to W_{i-2} and to W_{i+2};*
6. *for all i,j, the graphs induced by $V_i\cup V_j$ and $V_i\cup W_j$ are P_7-free;*
7. *for all i, there are no three vertices $v\in V_i$, $w\in V_{i+1}$ and $x\in W_{i+3}$ such that v,w and x are pairwise non-adjacent.*

Proof. Let G be a $(K_3,S_{1,2,3})$-free graph with such a partition that satisfies Conditions 1–7 of the lemma. Note that for all i, every vertex $v\in V_i$ is trivial to $V_{i+2},V_{i-2},W_{i-1},W_{i+1},W_i$ and either trivial to V_{i+1} or trivial to V_{i-1}. Therefore a vertex $v\in V_i$ can only be non-trivial to W_{i-2},W_{i+2} and at most one of V_{i-1} and V_{i+1}. Likewise, every vertex $w\in W_i$ is trivial to $W_{i-1},W_{i+1},W_{i-2},W_{i+2},V_{i-1}$ and V_{i+1}. Therefore, a vertex $w\in W_i$ can only be non-trivial to V_i,V_{i-2} and V_{i+2} (and every vertex in V_i is trivial to W_i).

For $i\in\{1,\ldots,5\}$, let W_i' be the set of elements of W_i that are non-trivial to both V_{i-2} and V_{i+2}, let V_i' be the set of elements of V_i that are non-trivial to both V_{i+1} and W_{i-2} and let V_i'' be the set of elements of V_i that are non-trivial to both V_{i-1} and W_{i+2}. Note that $V_i'\cap V_i''=\emptyset$ by Condition 3.

[2] Omitted proofs can be found in the arXiv preprint of this paper [17].

We say that an edge is *irrelevant* if one of its end-points is in a set V_i, V_i', V_i'', W_i or W_i', and its other end-point is complete to this set, otherwise we say that the edge is *relevant*. We will now show that for $i \in \{1, \ldots, 5\}$, the graph $G[V_i' \cup V_{i+1}'' \cup W_{i-2}']$ can be separated from the rest of G by using a bounded number of bipartite complementations. To do this, we first prove the following claim.

Claim 1. *If $u \in V_i' \cup V_{i+1}'' \cup W_{i-2}'$ and $v \notin V_i' \cup V_{i+1}'' \cup W_{i-2}'$ are adjacent then uv is an irrelevant edge.*

We split the proof of Claim 1 into the following cases.

Case 1: $u \in V_i'$.

Since u is in V_i, v must be in $V_{i-1} \cup V_{i+1} \cup W_{i-2} \cup W_{i+2}$, otherwise uv would be irrelevant by Condition 1 or 4. We consider the possible cases for v.

Case 1a: $v \in V_{i-1}$.

Since u is in V_i', it is non-trivial to V_{i+1}, so by Condition 3, u is trivial to V_{i-1}. Therefore uv is irrelevant.

Case 1b: $v \in V_{i+1}$.

Suppose, for contradiction, that v is complete to W_{i-2}. Let $w \in W_{i-2}$ be a neighbour of u (such a vertex w exists, since u is non-trivial to W_{i-2}). Then $G[u, v, w]$ is a K_3, a contradiction, so v cannot be complete to W_{i-2}. Now suppose, for contradiction that v is anti-complete to W_{i-2}. We may assume that v has a non-neighbour $u' \in V_i'$, otherwise v would be trivial to V_i', in which case uv would be irrelevant. Since $u' \in V_i'$, u' is non-trivial to W_{i-2}, so it must have a non-neighbour $w \in W_{i-2}$. Then, since v is anti-complete to W_{i-2}, it follows that $G[u, v, w]$ is a $3P_1$, contradicting Condition 7. We may therefore assume that v is non-trivial to W_{i-2}. We know that $v \notin V_{i+1}''$. Therefore v must be trivial to V_i, so uv is irrelevant.

Case 1c: $v \in W_{i-2}$.

Reasoning as in the previous case, we find that v cannot be complete or anti-complete to V_{i+1}. Hence, as $v \notin W_{i-2}'$, v must be trivial to V_i, so uv is irrelevant.

Case 1d: $v \in W_{i+2}$.

Since u is non-trivial to W_{i-2} (by definition of V_i'), there is a vertex $w \in W_{i-2}$ that is adjacent to u. By Condition 2, w is adjacent to v. Therefore $G[u, v, w]$ is a K_3. This contradiction implies that $v \notin W_{i+2}$. This completes Case 1.

Now assume that $u \notin V_i'$. Then, by symmetry, $u \notin V_{i+1}''$. This means that the following case holds.

Case 2: $u \in W_{i-2}'$.

We argue similarly to Case 1b. We may assume that v is non-trivial to W_{i-2}', otherwise uv would be irrelevant. By Conditions 1, 2 and 5, it follows that $v \in V_i \cup V_{i+1}$. Without loss of generality assume that $v \in V_i$. Since $v \notin V_i'$ and v is non-trivial to W_{i-2}, it follows that v is trivial to V_{i+1}. If v is complete to V_{i+1} then since u is non-trivial to V_{i+1}, there must be a vertex $w \in V_{i+1}$ adjacent to u, in which case $G[u, v, w]$ is a K_3, a contradiction. Therefore v must be anti-complete to V_{i+1}. Since v is non-trivial to W_{i-2}', there must be a vertex $u' \in W_{i-2}'$ that is non-adjacent to v. Since $u' \in W_{i-2}'$, u' must have a non-neighbour $w \in V_{i+1}$. Then $G[u', v, w]$ is a $3P_1$, contradicting Condition 7. This completes Case 2.

We conclude that, if $u \in V_i' \cup V_{i+1}'' \cup W_{i-2}'$ and $v \notin V_i' \cup V_{i+1}'' \cup W_{i-2}'$ are adjacent, then uv is an irrelevant edge. Hence we have proven Claim 1.

By Claim 1 we find that if $u \in V_i' \cup V_{i+1}'' \cup W_{i-2}'$ and $v \notin V_i' \cup V_{i+1}'' \cup W_{i-2}'$ are adjacent then u or v is complete to some set V_j, V_j', V_j'', W_j or W_j' that contains v or u, respectively. Applying a bounded number of bipartite complements (which we may do by Fact 3), we can separate $G[V_i' \cup V_{i+1}'' \cup W_{i-2}']$ from the rest of G. By Conditions 6 and 7 and the fact that G is $(K_3, S_{1,2,3})$-free, Lemmas 7 and 8 imply that $G[V_i' \cup V_{i+1}'' \cup W_{i-2}']$ has clique-width at most 6. Repeating this argument for each i, we may assume that $V_i' \cup V_{i+1}'' \cup W_{i-2}' = \emptyset$ for every i.

For $i \in \{1, \ldots, 5\}$ let V_i^* be the set of vertices in V_i that are either non-trivial to V_{i+1} or non-trivial to W_{i+2} and let V_i^{**} be the set of the remaining vertices in V_i. For $i \in \{1, \ldots, 5\}$, let W_i^* be the set of vertices that are non-trivial to V_{i+2} and let W_i^{**} be the set of the remaining vertices in W_i.

We claim that every vertex in V_i that is non-trivial to V_{i-1} or that is non-trivial to W_{i-2} is in V_i^{**}. Indeed, if $v \in V_i$ is non-trivial to V_{i-1} then by Condition 3, v is trivial to V_{i+1} and since V_i'' is empty, v must be trivial to W_{i+2}. If $v \in V_i$ is non-trivial to W_{i-2} then v must be trivial to V_{i+1} since V_i' is empty. Moreover, in this case v must also be trivial to W_{i+2}, otherwise, by Condition 2 the vertex v, together with a neighbour of v in each of W_{i+2} and W_{i-2}, would induce a K_3 in G. It follows that every vertex in V_i that is non-trivial to V_{i-1} or that is non-trivial to W_{i-2} is indeed in V_i^{**}. Similarly, for all i, since W_i' is empty, every vertex in W_i that is non-trivial to V_{i-2} is in W_i^{**}.

We say that an edge uv is *insignificant* if u or v is in some set V_i^*, V_i^{**}, W_i^* or W_i^{**} and the other vertex is trivial to this set; all other edges are said to be *significant*. We prove the following claim.

Claim 2. *If $u \in W_i^* \cup V_{i+2}^{**} \cup V_{i+1}^* \cup W_{i-2}^{**}$ and $v \notin W_i^* \cup V_{i+2}^{**} \cup V_{i+1}^* \cup W_{i-2}^{**}$ are adjacent then the edge uv is insignificant.*

To prove this claim suppose, for contradiction, that uv is a significant edge. We split the proof into two cases.

Case 1: $u \in W_i$.

We will show that $v \in V_{i+2}^{**}$ or $v \in V_{i-2}^*$ if $u \in W_i^*$ or $u \in W_i^{**}$, respectively. By Conditions 1, 2, 4 and 5 we know that u is trivial to V_{i-1}, V_{i+1}, W_{i-1}, W_{i+1}, W_{i-2} and W_{i+2}, and that every vertex of V_i is trivial to W_i. Furthermore, u is trivial to $W_i^{**} \setminus \{u\}$ since W_i is independent. Therefore $v \in V_{i-2} \cup V_{i+2}$. Note that v is non-trivial to W_i (by choice of v). If $u \in W_i^*$ then u must be trivial to V_{i-2}, since W_i' is empty. Therefore $v \in V_{i+2}$. Now if $v \in V_{i+2}^*$ then v is non-trivial to V_{i-2} or non-trivial to W_{i-1}. In the first case v is non-trivial to both V_{i-2} and W_i, contradicting the fact that V_{i+2}' is empty. In the second case v has a neighbour $w \in W_{i-1}$. By Condition 2, w is adjacent to u, so $G[u, v, w]$ is a K_3. This contradiction implies that if $u \in W_i^*$ then $v \in V_{i+2}^{**}$, contradicting the choice of v. Now suppose $u \in W_i^{**}$. Then u is trivial to V_{i+2}, so $v \in V_{i-2}$. If $v \in V_{i-2}^{**}$ then v is trivial W_i (by definition of V_{i-2}^{**}). Therefore if $u \in W_i^{**}$ then $v \in V_{i-2}^*$, contradicting the choice of v.

We conclude that for every $i \in \{1, \ldots, 5\}$ the vertex u is not in W_i. Similarly, we may assume $v \notin W_i$. This means that the following case holds.

Case 2: $u \in V_i$, $v \in V_j$ *for some* i, j.

Then $i \neq j$, since V_i is an independent set. By Condition 1, $j \notin \{i-2, i+2\}$. Without loss of generality, we may therefore assume that $j = i+1$. If $u \in V_i^{**}$ then u is trivial to V_{i+1}, so we may assume that $u \in V_i^*$. If $v \in V_{i+1}^*$ then v is non-trivial to V_{i+2}, so by Condition 3 it is trivial to V_i, contradicting the fact that uv is significant. Therefore $v \in V_{i+1}^{**}$, contradicting the choice of v.

We conclude that if for some i, $u \in W_i^* \cup V_{i+2}^{**} \cup V_{i+1}^* \cup W_{i-2}^{**}$ and $v \notin W_i^* \cup V_{i+2}^{**} \cup V_{i+1}^* \cup W_{i-2}^{**}$ are adjacent then the edge uv is insignificant. Hence we have proven Claim 2.

Note that $W_i^*, V_{i+2}^{**}, V_{i+1}^*$ and W_{i-2}^{**} are independent sets. By Condition 1, W_i^* is anti-complete to V_{i+1}^* and V_{i+2}^{**} is anti-complete to W_{i-2}^{**}. Therefore $W_i^* \cup V_{i+1}^*$ and $V_{i+2}^{**} \cup W_{i-2}^{**}$ are independent sets. Thus $G[W_i^* \cup V_{i+2}^{**} \cup V_{i+1}^* \cup W_{i-2}^{**}]$ is an $S_{1,2,3}$-free bipartite graph, which has bounded clique-width by Lemma 5. Applying a bounded number of bipartite complementations (which we may do by Fact 3), we can separate $G[W_i^* \cup V_{i+2}^{**} \cup V_{i+1}^* \cup W_{i-2}^{**}]$ from the rest of the graph. We may thus assume that $W_i^* \cup V_{i+2}^{**} \cup V_{i+1}^* \cup W_{i-2}^{**} = \emptyset$. Repeating this process for each i we obtain the empty graph. This completes the proof. ◀

We can now give the following result, which also implies the $(K_3, P_1 + 2P_2)$-free case.

▶ **Theorem 11.** *For $H \in \{P_1 + P_5, S_{1,2,2}, P_1 + P_2 + P_3\}$, the class of (K_3, H)-free graphs has bounded clique-width.*

Proof Sketch. Let $H \in \{P_1 + P_5, S_{1,2,2}, P_1 + P_2 + P_3\}$ and consider a (K_3, H)-free graph G. We may assume that G is connected, and by Lemma 9, that G contains an induced cycle on five vertices, say $C = v_1 - v_2 - \cdots - v_5 - v_1$. Since G is K_3-free, no vertex v is adjacent to two consecutive vertices of C. Therefore every vertex x of G has at most two neighbours on C, and if x has two neighbours, then they must be non-consecutive vertices of the cycle. We partition the vertices of G that are not on C into a set U of vertices adjacent to no vertices of C, sets W_i of vertices whose unique neighbour in C is v_i and sets V_i of vertices adjacent to v_{i-1} and v_{i+1}. Then, what is left to show is how to modify the graph using operations that preserve boundedness of clique-width, such that in the resulting graph the set U is empty and the partition $V_1, \ldots, V_5, W_1, \ldots, W_5$ satisfies Conditions 1–7 of Lemma 10. For full proof details we refer to [17]. ◀

To prove our main result, we first consider the case where the graph contains a clique on at least four vertices and show that such graphs have bounded clique-width. Theorem 11 implies that $(K_3, P_1 + 2P_2)$-free graphs have bounded clique-width. It is therefore sufficient to consider graphs in the class that contain a K_3, but not a K_4. We show that we can either use operations that preserve boundedness of clique-width to modify the graph into one known to have bounded clique-width or else the graph has a very specific structure, in which case we can show that it has bounded clique-width directly. See [17] for details.

▶ **Theorem 12.** *The class of (diamond, $P_1 + 2P_2$)-free graphs has bounded clique-width.*

References

1. Claudio Arbib and Raffaele Mosca. On (P_5,diamond)-free graphs. *Discrete Mathematics*, 250(1–3):1–22, 2002.
2. Rodica Boliac and Vadim V. Lozin. On the clique-width of graphs in hereditary classes. *Proc. ISAAC 2002, LNCS*, 2518:44–54, 2002.
3. Flavia Bonomo, Maria Chudnovsky, Peter Maceli, Oliver Schaudt, Maya Stein, and Mingxian Zhong. Three-coloring and list three-coloring of graphs without induced paths on seven vertices. preprint, 2015.
4. Flavia Bonomo, Luciano N. Grippo, Martin Milanič, and Martín D. Safe. Graph classes with and without powers of bounded clique-width. *Discrete Applied Mathematics*, 199:3–15, 2016.
5. Andreas Brandstädt, Konrad K. Dabrowski, Shenwei Huang, and Daniël Paulusma. Bounding the clique-width of H-free chordal graphs. *Proc. MFCS 2015 Part II, LNCS*, 9235:139–150, 2015.

6 Andreas Brandstädt, Konrad K. Dabrowski, Shenwei Huang, and Daniël Paulusma. Bounding the clique-width of H-free split graphs. *Discrete Applied Mathematics*, (to appear).

7 Andreas Brandstädt, Joost Engelfriet, Hoàng-Oanh Le, and Vadim V. Lozin. Clique-width for 4-vertex forbidden subgraphs. *Theory of Computing Systems*, 39(4):561–590, 2006.

8 Andreas Brandstädt, Vassilis Giakoumakis, and Frédéric Maffray. Clique separator decomposition of hole-free and diamond-free graphs and algorithmic consequences. *Discrete Applied Mathematics*, 160(4–5):471–478, 2012.

9 Andreas Brandstädt, Tilo Klembt, and Suhail Mahfud. P_6- and triangle-free graphs revisited: structure and bounded clique-width. *Discrete Mathematics and Theoretical Computer Science*, 8(1):173–188, 2006.

10 Andreas Brandstädt, Hoàng-Oanh Le, and Raffaele Mosca. Gem- and co-gem-free graphs have bounded clique-width. *International Journal of Foundations of Computer Science*, 15(1):163–185, 2004.

11 Andreas Brandstädt, Hoàng-Oanh Le, and Raffaele Mosca. Chordal co-gem-free and (P_5,gem)-free graphs have bounded clique-width. *Discrete Applied Mathematics*, 145(2):232–241, 2005.

12 Andreas Brandstädt and Suhail Mahfud. Maximum weight stable set on graphs without claw and co-claw (and similar graph classes) can be solved in linear time. *Information Processing Letters*, 84(5):251–259, 2002.

13 Hajo Broersma, Petr A. Golovach, Daniël Paulusma, and Jian Song. Determining the chromatic number of triangle-free $2P_3$-free graphs in polynomial time. *Theoretical Computer Science*, 423:1–10, 2012.

14 Maria Chudnovsky. Coloring graphs with forbidden induced subgraphs. *Proc. ICM 2014*, IV:291–302, 2014.

15 Maria Chudnovsky, Jan Goedgebeur, Oliver Schaudt, and Mingxian Zhong. Obstructions for three-coloring graphs with one forbidden induced subgraph. *Proc. SODA 2016*, pages 1774–1783, 2016.

16 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

17 Konrad K. Dabrowski, François Dross, and Daniël Paulusma. Colouring diamond-free graphs. *arXiv*, 1512.07849, 2015.

18 Konrad K. Dabrowski, Petr A. Golovach, and Daniël Paulusma. Colouring of graphs with Ramsey-type forbidden subgraphs. *Theoretical Computer Science*, 522:34–43, 2014.

19 Konrad K. Dabrowski, Shenwei Huang, and Daniël Paulusma. Bounding clique-width via perfect graphs. *Proc. LATA 2015, LNCS*, 8977:676–688, 2015.

20 Konrad K. Dabrowski, Vadim V. Lozin, Rajiv Raman, and Bernard Ries. Colouring vertices of triangle-free graphs without forests. *Discrete Mathematics*, 312(7):1372–1385, 2012.

21 Konrad K. Dabrowski and Daniël Paulusma. Classifying the clique-width of H-free bipartite graphs. *Discrete Applied Mathematics*, 200:43–51, 2016.

22 Konrad K. Dabrowski and Daniël Paulusma. Clique-width of graph classes defined by two forbidden induced subgraphs. *The Computer Journal*, (in press).

23 Wolfgang Espelage, Frank Gurski, and Egon Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. *Proc. WG 2001, LNCS*, 2204:117–128, 2001.

24 Jean-Luc Fouquet, Vassilis Giakoumakis, and Jean-Marie Vanherpe. Bipartite graphs totally decomposable by canonical decomposition. *International Journal of Foundations of Computer Science*, 10(04):513–533, 1999.

25 Petr A. Golovach, Matthew Johnson, Daniël Paulusma, and Jian Song. A survey on the computational complexity of colouring graphs with forbidden subgraphs. *Journal of Graph Thoery*, (in press).

26 Martin Grötschel, László Lovász, and Alexander Schrijver. Polynomial algorithms for perfect graphs. *Annals of Discrete Mathematics*, 21:325–356, 1984.

27 Frank Gurski. Graph operations on clique-width bounded graphs. *CoRR*, abs/cs/0701185, 2007.

28 Pinar Heggernes, Daniel Meister, and Charis Papadopoulos. Characterising the linear clique-width of a class of graphs by forbidden induced subgraphs. *Discrete Applied Mathematics*, 160(6):888–901, 2012.

29 Chính T. Hoàng and D. Adam Lazzarato. Polynomial-time algorithms for minimum weighted colorings of $(P_5, \overline{P_5})$-free graphs and similar graph classes. *Discrete Applied Mathematics*, 186:106–111, 2015.

30 Shenwei Huang, Matthew Johnson, and Daniël Paulusma. Narrowing the complexity gap for colouring (C_s, P_t)-free graphs. *The Computer Journal*, 58(11):3074–3088, 2015.

31 Marcin Kamiński, Vadim V. Lozin, and Martin Milanič. Recent developments on graphs of bounded clique-width. *Discrete Applied Mathematics*, 157(12):2747–2761, 2009.

32 Ton Kloks, Haiko Müller, and Kristina Vušković. Even-hole-free graphs that do not contain diamonds: A structure theorem and its consequences. *Journal of Combinatorial Theory, Series B*, 99(5):733–800, 2009.

33 Daniel Kobler and Udi Rotics. Edge dominating set and colorings on graphs with fixed clique-width. *Discrete Applied Mathematics*, 126(2–3):197–221, 2003.

34 Daniel Král', Jan Kratochvíl, Zsolt Tuza, and Gerhard J. Woeginger. Complexity of coloring graphs without forbidden induced subgraphs. *Proc. WG 2001, LNCS*, 2204:254–262, 2001.

35 László Lovász. Coverings and coloring of hypergraphs. *Congressus Numerantium*, VIII:3–12, 1973.

36 Vadim V. Lozin and Dmitriy S. Malyshev. Vertex coloring of graphs with few obstructions. *Discrete Applied Mathematics*, (in press).

37 Vadim V. Lozin and Dieter Rautenbach. On the band-, tree-, and clique-width of graphs with bounded vertex degree. *SIAM Journal on Discrete Mathematics*, 18(1):195–206, 2004.

38 Johann A. Makowsky and Udi Rotics. On the clique-width of graphs with few P_4's. *International Journal of Foundations of Computer Science*, 10(03):329–348, 1999.

39 Dmitriy S. Malyshev. The coloring problem for classes with two small obstructions. *Optimization Letters*, 8(8):2261–2270, 2014.

40 Dmitriy S. Malyshev. Two cases of polynomial-time solvability for the coloring problem. *Journal of Combinatorial Optimization*, 31(2):833–845, 2016.

41 Sang-Il Oum. Approximating rank-width and clique-width quickly. *ACM Transactions on Algorithms*, 5(1):10, 2008.

42 Bert Randerath and Ingo Schiermeyer. Vertex colouring and forbidden subgraphs – a survey. *Graphs and Combinatorics*, 20(1):1–40, 2004.

43 Michaël Rao. MSOL partitioning problems on graphs of bounded treewidth and clique-width. *Theoretical Computer Science*, 377(1–3):260–267, 2007.

44 David Schindl. Some new hereditary classes where graph coloring remains NP-hard. *Discrete Mathematics*, 295(1–3):197–202, 2005.

45 Alan Tucker. Coloring perfect $(K_4 - e)$-free graphs. *Journal of Combinatorial Theory, Series B*, 42(3):313–318, 1987.

Below All Subsets for Some Permutational Counting Problems*

Andreas Björklund

Department of Computer Science, Lund University, Lund, Sweden
andreas.bjorklund@yahoo.se

Abstract

We show that the two problems of computing the permanent of an $n \times n$ matrix of poly(n)-bit integers and counting the number of Hamiltonian cycles in a directed n-vertex multigraph with exp(poly(n)) edges can be reduced to relatively few smaller instances of themselves. In effect we derive the first deterministic algorithms for these two problems that run in $o(2^n)$ time in the worst case. Classic poly$(n)2^n$ time algorithms for the two problems have been known since the early 1960's. Our algorithms run in $2^{n-\Omega(\sqrt{n/\log n})}$ time.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Matrix Permanent, Hamiltonian Cycles, Asymmetric TSP

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.17

1 Introduction

We show that two well-known computationally hard counting problems defined over permutations, admit a strong form of self-reducibility. The problems are:

- Permanent: Given an $n \times n$ matrix M with poly(n)–bit integer elements, compute $\mathrm{per}(M) = \sum_{\sigma \in S_n} \prod_i M_{i,\sigma(i)}$ where S_n is the set of all permutations on n elements.
- HamCycles: Given an n-vertex directed multigraph, compute its number of Hamiltonian cycles, i.e. the number of non-crossing spanning cycles.

For both problems, we show that the solution to an instance of size parameter n can be reduced to a weighted sum of the solutions to poly$(n)2^{n-k}$ instances of size parameter $k < n$ of the same problem. Moreover, this reduction can be carried out in time polynomial in n per generated instance. We use this new relation to derive deterministic $2^{n-\Omega(\sqrt{n/\log n})}$ time algorithms for both Permanent and HamCycles. As a direct corollary we obtain an $Mn^2 2^{n-\Omega(\sqrt{n/log(Mn)})} + M^2 n^4$ time algorithm for Asymmetric TSP in graphs with integer arc weights in $[0, \ldots, M]$.

This is as far as the author knows the first deterministic algorithms that compute these quantities faster than explicitly inspecting at least a constant fraction of all subsets of an n-element set. In particular, no $o(2^n)$ time algorithms were previously known.

Our techniques here are elementary and the presentation is more-or-less self-contained. The main components are inclusion–exclusion counting, polynomial interpolation, and the Chinese remainder theorem. The speed-up is obtained through tabulation.

The two problems have well-known poly$(n)2^n$ time algorithms: Ryser's algorithm based on inclusion–exclusion for the permanent [14] from 1963, and a simple variation of Bellman, Held

* This research was supported in part by the Swedish Research Council grant VR 2012-4730 Exact Exponential–Time Algorithms.

and Karp's dynamic programming algorithm for TSP [3, 9] from 1962. Later a polynomial space inclusion–exclusion algorithm in the same spirit as Ryser's for counting Hamiltonian cycles with the same running time was found [12] in 1977 (and was rediscovered twice [10, 1]).

The question of existence of $O((2-\Omega(1))^n)$ time algorithms for the two problems are well-known open problems. In comparison the recent $O(1.657^n)$ time algorithm for Hamiltonian cycle [5] is randomized, only works for undirected graphs, and cannot even approximate the number of solutions. Recently, Cygan et al. [8] gave an algorithm for Hamiltonicity detection in bipartite directed graphs in $O(1.888^n)$ time. [7] presented an algorithm for computing the parity of the number of Hamiltonian cycles in $O(1.619^n)$ time, and [6] showed that one could reduce Hamiltonicity detection in graphs with few Hamiltonian cycles to the parity problem, and thereby obtained $o(2^n)$ time algorithms when the instance is known to have few solutions. Still, not only have there been no deterministic algorithms running in $o(2^n)$ worst case time for the counting problems, it was not even known how to detect a Hamiltonian cycle in a directed n-vertex graph that fast, probabilistic algorithms included. Nor was it known how to compute the permanent of an $n \times n$ $0-1$ matrix deterministically in $o(2^n)$ time.

Moreover, Knuth asks in exercise 4.6.4.11. [M46] in [11] if it is possible to compute a real $n \times n$-matrix permanent with less than 2^n arithmetic operations. We note that reals of bounded precision can be modeled by large integers, so our algorithm here works also for them. However, a table look-up is not an arithmetic operation, so our algorithm is not exactly what Knuth solicited.

The one general previous improvement over $\mathrm{poly}(n)2^n$ time for any of the two exact counting problems we are aware of is the $2^{n-\Omega(n^{1/3}\log n)}$ *expected* time algorithm for the $0-1$ matrix version of Permanent by Bax and Franklin [2]. Their technique can be extended to work with $O(1)$-bit integers, but probably not beyond that. In contrast, besides being faster and in deterministic time, our algorithm handles $\mathrm{poly}(n)$-bit integers, including negative ones.

The two known $\mathrm{poly}(n)2^n$ time algorithms for the problems based on the principle of inclusion–exclusion, Ryser's [14] and Kohn et al.'s [12] respectively, both use only polynomial space. It is indeed very natural to ask if employing the unexploited resource of using almost as much space as time wouldn't lead to faster algorithms. The problem though with the known approaches above is that there is no evident candidate for what to tabulate. They both sum over too large and typically different combinatorial objects. In the case of Ryser's permanent it is an n-element vector, and in Kohn et al.'s Hamiltonian cycles it is an induced graph on $n/2$ vertices on average.

The key insight here enabling a speed-up from tabulation is that the two problems admit a mapping from the original instances down to a linear combination of not too many much smaller ones. So small in fact that they are bound to coincide, making tabulation worthwhile.

1.1 Overview of the Technique

Consider the Permanent case, the HamCycles is similar. The speed-up is obtained in a series of steps. First we let $k = c\sqrt{n/\log n}$ for a constant c depending on the largest absolute element in the input matrix. Next we employ the existence part of the Chinese remainder theorem to bring matrix elements down to $d \log n$ bits each for some d. That is, we compute the permanent modulo small primes p of size polynomial in n. For each such prime p, we construct $\mathrm{poly}(n)2^{n-k}$ $k \times k$-matrices such that the permanent of the original one is equal to the sum of weighted permanents of all the matrices constructed. This reduction is in itself a two step procedure composed of a reduction to an inclusion–exclusion formula over polynomial matrices, accompanied by polynomial interpolation. We count the occurrences of

each of the smaller matrices in a table. Next we compute the permanent once for each of the different smaller matrices appearing in the sum using the classic $\text{poly}(k)2^k$ time algorithm. We note that there are at most $n^{dk^2} \ll 2^n$ different such matrices of size $k \times k$. The original instance permanent is then computed as a linear combination of all the tabulated matrices' permanent values. Finally, the results for all considered primes p are assembled via the constructive part of the Chinese remainder theorem.

1.2 Organization

In Section 2 we give a self-contained description of the self-reduction, anticipating that this part of the results may be of independent interest. The main results, the $o(2^n)$ algorithms for the two counting problems, are described in Section 3.

2 The Self-Reduction

The two problems Permanent and HamCycles are closely related. At a first glance it appears that the first asks about a property of matrices and the second about graphs, but they can be expressed in the same language. For the purpose of this paper, we will redefine both the Permanent and the HamCycles problem in terms of arc-weighted complete directed graphs to stress their similarity. In the remainder of this paper, the graph $G_n = (V, A)$ will denote the complete directed graph on n vertices V labelled 1 through n.

The set of all permutations on n elements, denoted by S_n, can naturally be partitioned after the number of cycles the permutation describes: A permutation $\sigma \in S_n$ can be interpreted as a directed graph on n vertices, labeled 1 through n, with the arcs $i, \sigma(i)$ for all i. Every vertex has exactly one outgoing and one incoming arc, i.e. the graph is a set of disjoint cycles covering the vertices. We will with S_n^1 denote the subset of S_n of permutations consisting of exactly one such cycle. Hence the permanent can be viewed upon as a sum over cycle covers of a graph, and the Hamiltonian cycles a sum over cycle covers consisting of just one cycle.

In the following it will make sense to be explicitly clear about what ring the computation is over. Thus we extend our problem definitions to:

▶ **Definition 1** (R-Permanent)**.** Given a complete directed graph $G_n = (V, A)$ and a function $f : A \to R$ mapping the arcs to some ring R, the *permanent* of (G, f) over R, denoted $\text{per}(G, f)$, is $\sum_{\sigma \in S_n} \prod_{i=1}^{n} f(i\sigma(i))$.

▶ **Definition 2** (R-HamCycles)**.** Given a complete directed graph $G_n = (V, A)$ and a function $f : A \to R$ mapping the arcs to some ring R, the *hamcycles* of (G, f) over R, denoted $\text{hc}(G, f)$, is $\sum_{\sigma \in S_n^1} \prod_{i=1}^{n} f(i\sigma(i))$.

In the remainder of this section we will prove the following two lemmas:

▶ **Lemma 3.** *Given an instance (G_n, f) to F-Permanent with f mapping arcs to a field F having at least $(n-k)n+1$ elements, and a positive integer $k < n$, one can compute $m = ((n-k)n+1)2^{n-k}$ instances $I_i = (G_k, f_i)$ to F-Permanent and constants $a_i \in F$ for $i = 1, \ldots, m$, so that*

$$\text{per}(G_n, f) = \sum_{i=1}^{m} a_i \, \text{per}(G_k, f_i) \, .$$

Moreover, the constructed smaller instances and constants can be produced in polynomial in n arithmetic operations $+$ and $*$ over F per instance.

▶ **Lemma 4.** *Given an instance (G_n, f) to F-HamCycles with f mapping arcs to a field F having at least $(n-k)k+1$ elements, and a positive integer $k < n$, one can compute $m = ((n-k)k+1)2^{n-k}$ instances $I_i = (G_k, f_i)$ to F-HamCycles and constants $a_i \in F$ for $i = 1, \ldots, m$, so that*

$$\operatorname{hc}(G_n, f) = \sum_{i=1}^{m} a_i \operatorname{hc}(G_k, f_i)\,.$$

Moreover, the constructed smaller instances and constants can be produced in polynomial in n arithmetic operations $+$ and $*$ over F per instance.

2.1 Preliminaries

In a complete directed graph G_n a *walk* of length l is a sequence of not necessarily distinct vertices $(v_0, v_1, \ldots, v_l)$. If $v_0 = v_l$ we say that the walk is a *closed* walk. For a field F and an indeterminate r, we denote by $F[r]$ the polynomial ring over F of polynomials in r with coefficients from F. For a polynomial $p(r) \in F[r]$ we denote by $[r^n]p(r)$ the coefficient of the monomial r^n in $p(r)$.

2.2 Step 1. Inclusion–exclusion

Consider an instance (G_n, f) to either F-Permanent or F-HamCycles for some field F. We fix a subset $K \subseteq V$ of the vertices of size $|K| = k$, called the *kernel* of the reduction. Without loss of generality, we let K be the vertices labeled by $1, 2, \ldots, k$, and hence $V - K$ be the vertices labelled by $k+1, k+2, \ldots, n$.

Our resulting instances will all be over the kernel K, i.e. embedded on the graph G_k. The central idea is to represent the parts of a cycle cover covering the vertices $V - K$, by arcs in G_k between the entry and exit points of the cycles in K. This approach of representing parts of a cycle cover outside a small subgraph by encoding them on the arcs of the subgraph was previously used by the author both in [4] and [5]. The novelty here, is the observation that these reductions can be seen as a mapping to a low degree univariate polynomial, that in step 2 in the next section will be efficiently brought back to the original field.

In this first step, we construct one instance per subset of $V - K$, and use the principle of inclusion–exclusion to relate them to the original instance. The resulting instances will not be over the original field F though. Instead the function f giving weights to the arcs will assign polynomials in one rank indeterminate r to them.

First we define the ranked walks in a vertex subset X. The degree of the indeterminate r counts the number of vertices visited along the walk. For any vertices $u, v \in X \subseteq V$ we let $W_{X,k}(u, v)$ be the ranked walks between vertices u and v visiting k vertices in X. We set

$$W_{X,k}(u,v) = \begin{cases} \sum_{w \in X} W_{X,k-1}(u,w) f(w,v) r & : k > 0 \\ 1 & : k = 0 \wedge u = v \\ 0 & : k = 0 \wedge u \neq v. \end{cases} \tag{1}$$

The ranked walks will be used to make sure all vertices outside the kernel K are visited by the cycle covers in the Permanent case and the Hamiltonian cycles in the HamCycles case. The principle of inclusion–exclusion makes sure crossing walks are cancelled. Since the HamCycles case is somewhat easier technically, we describe it first.

2.2.1 Inclusion–exclusion for HamCycles

We will construct instances of $F[r]$-HamCycles defined on $G_k = (K, A_K)$. We let $f_X : A_K \to F[r]$ for $X \subseteq V - K$ be defined for all $u, v \in K$ as follows

$$f_X(uv) = f(uv) + \sum_{w,z \in X} f(uw) \left(\sum_{i=0}^{n-k-1} W_{X,i}(w,z) \right) f(zv) \cdot r. \tag{2}$$

The point is that $f_X(uv)$ encodes all possible choices between either staying in K by choosing the arc uv directly or taking a detour through $V - K$ consisting of $1, 2, \ldots, n-k$ vertices starting in u and ending in v.

▶ **Lemma 5.** *With G_n, f, K, k, G_k, f_X as above it holds that*

$$\mathrm{hc}(G_n, f) = [r^{n-k}] \sum_{X \subseteq V-K} (-1)^{|V-K-X|} \mathrm{hc}(G_k, f_X)\,.$$

Proof. By the definition of F-HamCycles Def. 2, we have

$$\mathrm{hc}(G_k, f_X) = \sum_{\sigma \in S_k^1} \prod_{i=1}^{k} f_X(i\sigma(i))\,.$$

Expanding f_X via Eq. 2, we get

$$\mathrm{hc}(G_k, f_X) = \sum_{\sigma \in S_k^1} \prod_{i=1}^{k} \left(f(i\sigma(i)) + \sum_{l=1}^{n-k} r^l \sum_{v_1,\ldots,v_l \in X} f(iv_1) \left(\prod_{j=1}^{l-1} f(v_j v_{j+1}) \right) f(v_l \sigma(i)) \right)\,.$$

From the formula above, we see that $[r^{n-k}]hc(G_k, f_X)$ is a sum with terms $\prod_{i=1}^{n} f(v_i v_{i+1})$ for each closed walk $(v_1, v_2, \ldots, v_{n+1})$ with $v_{n+1} = v_1$ where

1. Exactly $n-k$ of $v_1, \ldots, v_n$ belong to X, and
2. Each vertex in K occurs exactly once in $v_1, \ldots, v_n$.

In the inclusion–exclusion summation over $X \subseteq V - K$,

$$\mathrm{hc}(G_n, f) = \sum_{X \subseteq V-K} (-1)^{|V-K-X|} [r^{n-k}] \mathrm{hc}(G_k, f_X)\,,$$

each walk that crosses itself, i.e. has $v_i = v_j$ for some $i < j \leq n$, will be counted an even number of times. Moreover, exactly half of these times it will be added to the sum and the other half it will be subtracted, thereby canceling in the sum. To see why, let $Y = \{v_i | v_i \in V - K\}$ for a crossing walk. Clearly $Y \subset V - K$ since there are precisely $n-k$ vertices from $V - K$ on every contributing walk, and when one occurs at least twice there must be another one that is missing. Since among the subsets Z fulfilling $Y \subseteq Z \subseteq V - K$ there are as many even sized subsets as odd ones the claim follows. Contributing walks that do not cross themselves however, i.e. are Hamiltonian cycles in G, will only be counted once, for $X = V - K$. ◀

2.2.2 Inclusion–exclusion for Permanent

In addition to the ranked walks in $V - K$ we also need to keep track of ranked cycles in $V - K$ for the Permanent. We want to sum over all cycle covers of the input graph G and unlike the HamCycles case we may have vertices in $V - K$ disconnected from K in a cycle

cover. Remember that the vertices in V are labelled $1, 2, \ldots, n$ and associate the natural ordering $<$ of them. We need to define cycles in a cycle cover so that they receive a unique identifier to avoid double counting in our polynomial identity. To this end, we use that every cycle has a minimum vertex under the ordering to define the ranked closed walks *anchored* at $s \in X$ as

$$C_X(s) = 1 + \sum_{i=1}^{n-k} W_{X_{\geq s},i}(s,s)\,. \tag{3}$$

where $X_{\geq s} = \{v | s \leq v \in X\}$, i.e. all vertices in X equal to or larger than s. The cycles anchored at s represents all cycles of length $1, 2, \ldots, n-k$ in $V - K$ where s is the smallest vertex on the cycle. Note in particular that self-loops through s are also included in the sum. The 1 is in the definition of Eq. 3 to take into account the possibility that no cycle is anchored at s in a contributing cycle cover.

▶ **Lemma 6.** *With G_n, f, K, k, G_k, f_X as above it holds that*

$$\operatorname{per}(G_n, f) = [r^{n-k}] \sum_{X \subseteq V-K} (-1)^{|V-K-X|} \operatorname{per}(G_k, f_X) \prod_{s \in X} C_X(s)\,.$$

Proof. By the definition of F-Permanent Def. 1, we have

$$\operatorname{per}(G_k, f_X) \prod_{s \in X} C_X(s) = \sum_{\sigma \in S_k} \prod_{j=1}^{k} f_X(j\sigma(j)) \prod_{i=k+1}^{n} C_X(i)\,.$$

Expanding C_X via Eq. 3 and f_X via Eq. 2, we get

$$\operatorname{per}(G_k, f_X) \prod_{s \in X} C_X(s) =$$
$$\sum_{\sigma \in S_k} \prod_{i=1}^{k} \left(f(i\sigma(i)) + \sum_{l=1}^{n-k} r^l \sum_{v_1,\ldots,v_l \in X} f(iv_1) \left(\prod_{j=1}^{l-1} f(v_j v_{j+1}) \right) f(v_l \sigma(i)) \right)$$
$$\cdot \prod_{i=k+1}^{n} \left(1 + \sum_{l=1}^{n-k} r^l \sum_{\substack{v_1,\ldots,v_l \in X_{\geq i} \\ i = v_1}} f(v_l v_1) \prod_{j=1}^{l-1} f(v_j v_{j+1}) \right)\,.$$

Expanding the formula above into a sum–product formula by identifying terms, we see that

$$[r^{n-k}] \operatorname{per}(G_k, f_X) \prod_{s \in X} C_X(s)\,,$$

is a sum over contributions $\prod_{i=1}^{l} \prod_{uv \in O_i} f(uv)$, for $1 \leq l \leq n$ closed l-long walks $O_i = (v_{i,1}, \ldots, v_{i,m_l}, v_{i,m_l+1})$ with $v_{i,1} = v_{i,m_l+1}$ and $\sum_{i=1}^{l} m_i = n$ where

1. Exactly $n - k$ of the $v_{i,j}$ for $1 \leq i \leq n, 1 \leq j \leq m_i$ belong to X, and
2. Each vertex in K occurs exactly once in the closed walks $O_i, 1 \leq i \leq l$.

In the inclusion–exclusion summation over $X \subseteq V - K$,

$$\operatorname{per}(G_n, f) = \sum_{X \subseteq V-K} (-1)^{|V-K-X|} [r^{n-k}] \operatorname{per}(G_k, f_X) \prod_{s \in X} C_X(s)\,,$$

each set of closed walks $\{O_i\}$ that crosses itself, i.e. has $v_{i1,j1} = v_{i2,j2}$ for some $i1 \neq i2 \vee j1 \neq j2$, will be counted an even number of times. Moreover, exactly half of these times it will be added to the sum and the other half it will be subtracted, thereby canceling in the sum. To see why, again let $Y = \{v_{i,j} | v_{i,j} \in V - K\}$ for a set of closed walks with a crossing. Clearly $Y \subset V - K$ since there are precisely $n - k$ vertices from $V - K$ on every contributing set of closed walks, and when one occurs at least twice there must be another one that is missing. Since among the subsets Z fulfilling $Y \subseteq Z \subseteq V - K$ there are as many even sized subsets as odd ones the claim follows. Contributing sets of closed walks that do not cross themselves, i.e. are cycle covers in G, will only be counted once, for $X = V - K$. ◀

2.3 Step 2. Polynomial Interpolation

In the previous section we related the permanent and the Hamiltonian cycles of an arc weighted graph to smaller graphs with weights over a polynomial ring. We want to bring the small instances to map arcs to the original ring to complete the self-reduction. Unfortunately, we are only able to do this if the original ring is a field, and one that has at least polynomially many elements in the original instance size parameter. In particular, we need the following well-known result:

▶ **Lemma 7** (Lagrange interpolation). *For any set of pairs $\{(r_i, s_i)\}$ with distinct r_i's and $r_i, s_i \in F$ for $i = 1, \ldots, k+1$ where F is a field on at least $k+1$ elements, there is a unique polynomial $p(r)$ in $F[r]$ of degree at most k such that $p(r_i) = s_i$ for all i. Moreover, the polynomial is given by*

$$p(r) = \sum_{i=1}^{k+1} s_i \prod_{j \neq i} \frac{r - r_j}{r_i - r_j} .$$

Specifically, consider an instance (G, f) to F-HamCycles. Via Lemma 5 we see that $\mathrm{hc}(G)$ is related to a coefficient in a polynomial sum of many smaller instances (G_k, f_X) to $F[r]$-HamCycles. We use here that if we know the result in enough points over F we can reconstruct the polynomial via interpolation.

▶ **Lemma 8.** *For every polynomial term $\mathrm{hc}(G_k, f_X)$ in the outer sum in Lemma 5, it is possible to compute $(n-k)k+1$ instances (G_k, f_i) for $i = 1, \ldots, (n-k)k+1$ to the F-HamCycles on k vertices, and constants $a_i \in F$ for $i = 1, \ldots, (n-k)k+1$ so that*

$$[r^{n-k}] \, \mathrm{hc}(G_k, f_X) = \sum_{j=1}^{(n-k)k+1} a_j \, \mathrm{hc}(G_k, f_j) .$$

Proof. Each entry in the codomain of f_X has degree $n - k$ in r by definition of the ranked walks and the definition of f_X in Eq. 2. Since $\mathrm{hc}(G_k, f_X)$ is a sum over the product of k arcs' f_X's, the degree of $\mathrm{hc}(G_k, f_X)$ in r is $(n-k)k$.

Let $r_1, r_2, \ldots, r_m$ be m distinct elements in F and let f_j be equal to f_X evaluated in $r = r_j$. By Lagrange interpolation, it is possible to compute $\mathrm{hc}(G_k, f_X)$ and in particular the coefficient of r_{n-k} from the evaluated polynomial points $\mathrm{hc}(G_k, f_j)$. ◀

The F-Permanent case is similar: consider an instance (G_n, f). Lemma 6 states that $\mathrm{per}(G_n, f)$ is related to a coefficient in a polynomial resulting from a sum of many smaller instances (G_k, f_X) to $F[r]$-Permanent.

▶ **Lemma 9.** *For every polynomial term* $\operatorname{per}(G_k, f_X) \prod_{i=k+1}^{n} C_X(i)$ *in the outer sum in Lemma 6, it is possible to compute* $(n-k)n+1$ *instances* (G_k, f_i) *for* $i = 1, \ldots, (n-k)n+1$ *to the* F-PERMANENT *on* k *vertices, and constants* $a_i \in F$ *for* $i = 1, \ldots, (n-k)n+1$ *so that*

$$[r^{n-k}] \operatorname{per}(G_k, f_X) \sum_{i=k+1}^{n} C_X(i) = \sum_{j=1}^{(n-k)n+1} a_j \operatorname{per}(G_k, f_j)\,.$$

Proof. Each entry in the codomain of f_X has degree $n-k$ by definition of the ranked walks and the definition of f_X in Eq. 2. Since $\operatorname{per}(G_k, f_X)$ is a sum over the product of k arcs f_X's, the degree of $\operatorname{per}(G_k, f_X)$ in r is $(n-k)k$. The degree of $\prod_{i=k+1}^{n} C_X(i)$ is $(n-k)(n-k)$ since every $C_X(i)$ has degree $n-k$ by the definition Eq. 3. Altogether, $\operatorname{per}(G_k, f_X) \prod_{i=k+1}^{n} C_X(i)$ has degree $(n-k)n$.

Let $r_1, r_2, \ldots, r_m$ be m distinct elements in F and let f_j be equal to f_X evaluated in $r = r_j$. Likewise, let b_j be equal to $\prod_{i=k+1}^{n} C_X(i)$ evaluated in $r = r_j$. By Lagrange interpolation, it is possible to compute the coefficent of r^{n-k} in $\operatorname{per}(G_k, f_X) \prod_{i=k+1}^{n} C_X(i)$ from the evaluated polynomial points $b_j \operatorname{per}(G_k, f_j)$. ◀

The self-reduction for F-PERMANENT Lemma 3 follows from the combination of Lemma 6 and Lemma 9, after observing that each $X \subseteq V - K$ and each $r \in 1, \ldots, (n-k)n+1$ corresponds to one small instance. Similarly, the self-reduction for F-HAMCYCLES Lemma 4 follows from Lemma 5 and Lemma 8 with $X \subseteq V - K$ and $r \in 1, \ldots, (n-k)k+1$. It remains to validate the runtime in terms of the number of arithmetic operations used. To compute a small instance (G_k, f_i) in Lemma 3 (Lemma 4 respectively), corresponding to a particular $X \subseteq V - K$ and $r \in 1, \ldots, (n-k)n+1$, we see from the definitions Eqs. 2 and 3 that the instance elements are computed as walks in X for a fixed r. We can compute the elements through the recursive definition of the ranked walks Eq. 1 via dynamic programming in only polynomial in n number of arithmetic operations.

3 The Algorithms

In this section we prove our main theorems:

▶ **Theorem 10.** *Any single* $n \times n$ *matrix instance of* PERMANENT *with* $\operatorname{poly}(n)$*-bit integer elements can be solved deterministically in* $2^{n-\Omega(\sqrt{n/\log n})}$ *time.*

▶ **Theorem 11.** *Any single* n*-vertex directed graph instance of* HAMCYCLES *with* $\exp(\operatorname{poly}(n))$ *number of arcs can be solved deterministically in* $2^{n-\Omega(\sqrt{n/\log n})}$ *time.*

We immediately observe that the above theorem via a standard embedding of the $(\min, +)$-semiring on the integers, can be used to count cycles by weight through polynomial interpolation. In particular, the problem of finding the length of the shortest Hamiltonian cycle, known as the Asymmetric Traveling Salesman problem can be solved by the technique.That is, we introduce yet another indeterminate z, associate an arc of weight w with z^w, and finally solve for the smallest non-zero monomial in the resulting polynomial, see e.g. [12]. Since the evaluated polynomial is of degree at most Mn^2, we get

▶ **Corollary 12.** *The shortest Asymmetric Traveling Salesman Problem route in an* n*-vertex graph with integer arc weights in* $[0, \ldots, M]$ *can be computed in* $Mn^2 2^{n-\Omega(\sqrt{n/\log(Mn)})} + M^2 n^4$ *time.*

On the top level, the idea of the algorithms is to bring the computations down to small finite fields. We next use the self-reductions from Section 2 to transform the input matrix/graph down to so small ones that several of them will be identical. By tabulating which ones of them have been constructed in this process and how often, it then suffices to compute the permanent of the small matrices/the Hamiltonian cycles of the small graphs only once. To make this precise we first need some elementary results from number theory.

3.1 Preliminaries on Modular Arithmetic

The well-known Chinese remainder theorem has two parts, an existence and a constructive one. The existence part states that an integer solution to a set of linear modular equations is uniquely defined in the range between zero and the least common multiple of the moduli. The constructive part describes how to recover the solution given the modular equations. We state them here in a slightly modified form as we will need them

▶ **Lemma 13** (CRT). *Given m distinct primes p_i, and residues $0 \leq a_i < p_i$, $1 \leq i \leq m$,*

- *Existence: There is a unique integer n in $\left\lfloor -\frac{\prod_{i=1}^m p_i}{2} \right\rfloor \leq n < \left\lceil \frac{\prod_{i=1}^m p_i}{2} \right\rceil$ fulfilling $n \equiv a_i (\mod p_i), 1 \leq i \leq m$.*
- *Construction: The integer n can be computed by evaluating $n_+ = \sum_{i=1}^m a_i r_i$ where $r_i = \prod_{j \neq i} p_j \left(\left(\prod_{j \neq i} p_j \right)^{-1} (\mod p_i) \right)$ and then setting $n = n_+$ if $n_+ < \frac{\prod_{i=1}^m p_i}{2}$, and $n = n_+ - \prod_{i=1}^m p$ otherwise.*

We also use the following bound of the prime number theorem to answer how many and large primes we will need to break down a computation using the CRT:

▶ **Lemma 14** (Rosser [13]). *For every integer $n \geq 55$ the number of primes $\pi(n)$ less than or equal to n obey $n/(ln(n)+2) < \pi(n) < n/(ln(n)-4)$.*

3.2 The Algorithm

We will first describe the algorithm for the Permanent case Thm. 10 , and then point out the few changes needed for the HamCycles case Thm. 11. We begin by describing the algorithm in pseudo-code below. Next we will explain the steps in more detail.

Permanent per(G_n, f)

1. Let M be the largest absolute value in the image of f.
2. Let P be the smallest set of primes $> n^2$ such that $\prod_{p \in P} p > 2M^n n!$.
3. Let $k = \lfloor \sqrt{.99n / \log_2 p_{\max}} \rfloor$ where $p_{\max} = \max_{p \in P} p$.
4. For each prime $p \in P$
5. Construct a table T from all $\mathbb{Z}_p^{k \times k}$ matrices to the positive integers, initially set to all zeros.
6. Evaluate $f_{(p)} = f(\mod p)$.
7. Compute $m = (n-k)n2^{n-k}$ instances (G_k, f_j) and constants a_j for $j = 1, \ldots, m$ to $\mathbb{Z}_p$-Permanent such that $\text{per}(G_n, f_{(p)}) = \sum_{j=1}^m a_i \, \text{per}(G_k, f_j)$.
8. For $j = 1, \ldots, m$
9. Let $T(f_j) = T(f_j) + a_j$.
10. Set $sum = 0$.
11. For each g with non-zero table entry $T(g)$

12. Compute $\operatorname{per}(G_k, g)$ using Ryser's permanent algorithm.

13. Let $sum = sum + T(g)\operatorname{per}(G_k, g)(\bmod p)$.

14. Store $per(G_n, f_{(p)}) = sum$

15. Compute the permanent over $\mathbb{Z}$ using the stored $\operatorname{per}(G_n, f_{(p)})$ for all $p \in P$ using the constructive part of CRT.

The existence part of CRT Lemma 13 makes it clear that to compute an integer function solely with the operations $+$ and $*$ over the integers, one can just as well compute it modulo several primes and assemble the result in the end. Both the Permanent and the HamCycles problems are defined as sum–products, so to compute their quantities modulo a prime p, we can replace the input integers with their residues modulo p. Steps 2–6 of the algorithm do precisely that, transform the input integer Permanent instance to instances of $\mathbb{Z}_p$-Permanent for primes p. Step 7 next generates $(n-k)n2^{n-k}$ instances of the $\mathbb{Z}_p$-Permanent problem using the constructive proof for Lemma 3. Steps 8–9 counts the occurrences of each of the different matrices in $\mathbb{Z}_p^{k\times k}$ by keeping track of the total coefficients of each of the smaller matrices' permanents in Lemma 3. Steps 10–14 computes the solution to the $n \times n$-matrix permanent $\operatorname{per}(G_n, f_{(p)})$, and finally step 15 assembles the modular results using the constructive part of the CRT Lemma 13. The correctness of the algorithm follows from Lemma 13 and the self-reduction Lemma 3, after noting that enough primes are chosen in step 2.

To bound the runtime, the only question is how many and how large primes are required, and indirectly, how large tables will be used? The permanent is a sum of $n!$ products of n elements from the input function f. In step 2 of the algorithm we measure the absolute max over all elements used to conclude that $|\operatorname{per}(G_n, f)| \leq M^n n! < 2^{n^c}$ for some positive constant c when the input entries have $\operatorname{poly}(n)$ bits. From Lemma 14 we see that there are at least $m = n^d/(dln(n)+2) - n^2/(2ln(n)-4)$ primes larger than n^2 but smaller than n^d for $n \geq 55$. We want the product of the first m primes larger than n^2, the set of primes P in step 4 of the algorithm, to be larger than $2 \cdot 2^{n^c}$, i.e. $n^{2m} > 2^{n^c+1}$. It is straightforward to note that a constant d depending on c will suffice, in fact using $d = c+3$ is more than enough. Hence $p_{\max}$ in step 3 is bounded by n^d for d constant and k is $\Omega(\sqrt{n/\log n})$.

For each prime $p \in P$ in step 4, we use a table T in step 5–13 with one entry per matrix in $\mathbb{Z}_p^{k\times k}$. An upper bound on the number of matrices in $\mathbb{Z}_p^{k\times k}$ using $p_{\max}$ from step 3 and k from step 4 of the algorithm is $(p_{\max})^{k^2} < 2^{0.99n}$. The runtime of steps 5–9 is easily seen to be $O((n-k)n2^{n-k})$ from the bound on the table T's size and Lemma 3. Computing the permanent of each of the matrices is a $O(k2^k)$ time task with Ryser's algorithm [14], so the total runtime of steps 10–15 is $o(2^{n-k})$. Altogether, the loop at steps 4–14 is run a polynomial number of times, and step 15 is polynomial time, so we get $\operatorname{poly}(n)2^{n-k}$ time in total which is $2^{n-\Omega(\sqrt{n/\log n})}$ time as claimed.

To adjust the algorithm and the proof to counting HamCycles, all we need to do is to replace Lemma 3 for Lemma 4 in step 7 of the algorithm and the analysis, and exchange Ryser's algorithm for the permanent in step 12 for e.g. Bax's [1] Hamiltonian cycle counting algorithm.

Acknowledgments. I thank Thore Husfeldt, Alexander Golovnev, Petteri Kaski, Mikko Koivisto, and Ryan Williams and several anonymous referees for comments on an earlier draft of the paper and stimulating discussions on the subject.

References

1. E. T. Bax. Inclusion and exclusion algorithm for the hamiltonian path problem. *Inform. Process. Lett*, 47:203–207, 1993.
2. E. T. Bax and J. Franklin. A permanent algorithm with $exp[\omega(n^{1/3}/2ln(n))]$ expected speedup for 0-1 matrices. *Algorithmica*, 32:157–162, 2002.
3. R. Bellman. Dynamic programming treatment of the travelling salesman problem. *J. Assoc. Comput. Mach*, 9:61–63, 1962.
4. A. Björklund. Counting perfect matchings as fast as ryser. In *Proceedings of the ACM-SIAM SODA*, pages 914–921, 2012.
5. A. Björklund. Determinant sums for undirected hamiltonicity. *SIAM J. Comput.*, 43:280–299, 2014.
6. A. Björklund, H. Dell, and T. Husfeldt. The parity of set systems under random restrictions with applications to exponential time problems. In *Proceedings of ICALP,*, pages 231–242, 2015.
7. A. Björklund and T. Husfeldt. The parity of directed hamiltonian cycles. In *Proceedings of the IEEE FOCS,*, pages 724–735, 2013.
8. M. Cygan, S. Kratsch, and J. Nederlof. Fast hamiltonicity checking via bases of perfect matchings. In *Proceedings of the ACM STOC*, pages 301–310, 2013.
9. M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *J. Soc. Indust. Appl. Math.*, 10:196–210, 1962.
10. R. M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Oper. Res. Lett.*, 1:49–51, 1982.
11. D. E. Knuth. *The Art of Computer Programming. Vol. 2: Seminumerical Algorithms*. Addison-Wesley, third edition, 1997.
12. S. Kohn, A. Gottlieb, and M. Kohn. A generating function approach to the traveling salesman problem. In *Proceedings of the Annual Conference (ACM'77), Association for Computing Machinery*, pages 294–300, 1977.
13. B. Rosser. Explicit bounds for some functions of prime numbers. *American Journal of Mathematics*, 63:211–232, 1941.
14. H. J. Ryser. *Combinatorial Mathematics*. The Carus mathematical monographs, The Mathematical Association of America, 1963.

Extension Complexity, MSO Logic, and Treewidth*

Petr Kolman[1], Martin Koutecký[2], and Hans Raj Tiwary[3]

1 **Department of Applied Mathematics (KAM) & Institute of Theoretical Computer Science (ITI), Faculty of Mathematics and Physics (MFF), Charles University, Prague, Czech Republic**
kolman@kam.mff.cuni.cz

2 **Department of Applied Mathematics (KAM) & Institute of Theoretical Computer Science (ITI), Faculty of Mathematics and Physics (MFF), Charles University, Prague, Czech Republic**
koutecky@kam.mff.cuni.cz

3 **Department of Applied Mathematics (KAM) & Institute of Theoretical Computer Science (ITI), Faculty of Mathematics and Physics (MFF), Charles University, Prague, Czech Republic**
hansraj@kam.mff.cuni.cz

Abstract

We consider the convex hull $P_\varphi(G)$ of all satisfying assignments of a given MSO_2 formula φ on a given graph G. We show that there exists an extended formulation of the polytope $P_\varphi(G)$ that can be described by $f(|\varphi|, \tau) \cdot n$ inequalities, where n is the number of vertices in G, τ is the treewidth of G and f is a computable function depending only on φ and τ.

In other words, we prove that the extension complexity of $P_\varphi(G)$ is linear in the size of the graph G, with a constant depending on the treewidth of G and the formula φ. This provides a very general yet very simple meta-theorem about the extension complexity of polytopes related to a wide class of problems and graphs.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, F.4.1 Mathematical Logic, G.1.6 Optimization, G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases Extension Complexity, FPT, Courcelle's Theorem, MSO Logic

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.18

1 Introduction

In the '70s and '80s, it was repeatedly observed that various NP-hard problems are solvable in polynomial time on graphs resembling trees. The graph property of *resembling a tree* was eventually formalized as having bounded treewidth, and in the beginning of the '90s, the class of problems efficiently solvable on graphs of bounded treewidth was shown to contain the class of problems definable by the Monadic Second Order Logic (MSO_2) (Courcelle [11], Arnborg et al. [1], Courcelle and Mosbah [13]). Using similar techniques, analogous results for weaker logics were then proven for wider graph classes such as graphs of bounded cliquewidth and rankwidth [12]. Results of this kind are usually referred to as *Courcelle's theorem* for a specific class of structures.

In this paper we study the class of problems definable by the MSO logic from the perspective of extension complexity. While small extended formulations are known for various special classes of polytopes, we are not aware of any other result in the theory of

* This research was partially supported by projects GA15-11559S of GA ČR and 338216 of GA UK.

extended formulations that works on a wide class of polytopes the way Courcelle's theorem works for a wide class of problems and graphs.

Our Contribution. We prove that satisfying assignments of an MSO_2 formula φ on a graph of bounded treewidth can be *expressed* by a "small" linear program. More precisely, there exists a computable function f such that the convex hull – $P_\varphi(G)$ – of satisfying assignments of φ on a graph G on n vertices with treewidth τ can be obtained as the projection of a polytope described by $f(|\varphi|, \tau) \cdot n$ linear inequalities; we call $P_\varphi(G)$ the *MSO polytope*. All our results can be extended to general finite structures where the restriction on treewidth applies to the treewidth of their Gaifman graph [30].

Our proof essentially works by "merging the common wisdom" from the areas of extended formulations and fixed parameter tractability. It is known that dynamic programming can be turned into a compact extended formulation [32, 18], and that Courcelle's theorem can be seen as an instance of dynamic programming [26]; therefore one can expect the polytope of satisfying assignments of an MSO formula of a bounded treewidth graph to be compact.

However, there are a few roadblocks in trying to merge these two folklore wisdoms. For one, while Courcelle's theorem being an instance of dynamic programming in some sense may be obvious to an FPT theorist, it is far from clear to anyone else what that sentence may even mean. On the other hand, being able to turn a dynamic program into a compact polytope may be a theoretical possibility for an expert on extended formulations, but it is by no means an easy statement for an outsider to comprehend. What complicates the matters even further is that the result of Martin et al. [32] is not a result that can be used in a black box fashion. That is, a certain condition must be satisfied to get a compact extended formulation out of a dynamic program. This is far from a trivial task, especially for a theorem like Courcelle's theorem.

The rest of the article is organized as follows. In Section 2 we review some previous work related to Courcelle's theorem and extended formulations. In Section 3 we describe the relevant notions related to polytopes, extended formulations, graphs, treewidth and MSO logic. In Section 4 we prove the existence of compact extended formulations for MSO polytopes parameterized by the length of the given MSO formula and the treewidth of the given graph. In Section 5 we describe how to efficiently construct such a polytope given a tree decomposition of a graph.

2 Related Work

2.1 MSO Logic vs. Treewidth

Because of the wide relevance of the treewidth parameter in many areas (cf. the survey of Bodlaender [5]) and the large expressivity of the MSO and its extensions (cf. the survey of Langer et al. [27]), considerable attention was given to Courcelle's theorem by theorists from various fields, reinterpreting it into their own setting. These reinterpretations helped uncover several interesting connections.

The classical way of proving Courcelle's theorem is constructing a tree automaton A in time only dependent on φ and the treewidth τ, such that A accepts a tree decomposition of a graph of treewidth τ if and only if the corresponding graph satisfies φ; this is the automata theory perspective [11]. Another perspective comes from finite model theory where one can prove that a certain equivalence on the set of graphs of treewidth at most τ has only finitely many (depending on φ and τ) equivalence classes and that it *behaves well* [16]. Another approach proves that a quite different equivalence on so-called extended model checking

games has finitely many equivalence classes [23] as well; this is the game-theoretic perspective. It can be observed that the finiteness in either perspective stems from the same roots.

Another related result is an *expressivity* result: Gottlob et al. [16] prove that on bounded treewidth graphs, a certain subset of the database query language Datalog has the same expressive power as the MSO. This provides an interesting connection between the automata theory and the database theory.

2.2 Extended Formulations

Sellmann, Mercier, and Leventhal [34] claimed to show compact extended formulation for binary Constraint Satisfaction Problems (CSP) for graphs of bounded treewidth, but their proof is not correct [33]. The first two authors of this paper gave extended formulations for CSP that has polynomial size for instances whose constraint graph has bounded treewidth [25] using a different technique. Bienstock and Munoz [3] prove similar results for the approximate and exact version of the problem. In the exact case, Bienstock and Munoz's bounds are slightly worse than those of Kolman and Koutecký [25]. It is worth noting that CSPs are a restricted subclass of problems that can be modeled using MSO logic. Laurent [28] provides extended formulations for the independent set and max cut polytopes of size $O(2^\tau n)$ for n-vertex graphs of treewidth τ and, independently, Buchanan and Butenko [8] provide an extended formulation for the independent set polytope of the same size.

A lot of recent work on extended formulations has focussed on establishing lower bounds in various settings: exact, approximate, linear vs. semidefinite, etc. (See for example [15, 2, 6, 29]). A wide variety of tools have been developed and used for these results including connections to nonnegative matrix factorizations [37], communication complexity [14], information theory [7], and quantum communication [15] among others.

For proving upper bounds on extended formulations, several authors have proposed various tools as well. Kaibel and Loos [19] describe a setting of branched polyhedral systems which was later used by Kaibel and Pashkovich [20] to provide a way to construct polytopes using reflection relations.

A particularly specific composition rule, which we term glued product (cf. Subsection 3.1), was studied by Margot in his PhD thesis [31]. Margot showed that a property called the projected face property suffices to glue two polytopes efficiently. Conforti and Pashkovich [10] describe and strengthen Margot's result to make the projected face property to be a necessary and sufficient condition to describe the glued product in a particularly efficient way.

Martin et al. [32] have shown that under certain conditions, an efficient dynamic programming based algorithm can be turned into a compact extended formulation. Kaibel [18] summarizes this and various other methods.

3 Preliminaries

3.1 Polytopes, Extended Formulations and Extension Complexity

For background on polytopes we refer the reader to Grünbaum [17] and Ziegler [38]. To simplify reading of the paper for the audience that is not working often in the area of polyhedral combinatorics, we provide here a brief glossary of common polyhedral notions that are used in this article.

A *hyperplane* in $\mathbb{R}^n$ is a closed convex set of the form $\{x|a^\intercal x = b\}$ where $a \in \mathbb{R}^n, b \in \mathbb{R}$. A *halfspace* in $\mathbb{R}^n$ is a closed convex set of the form $\{x|a^\intercal x \leqslant b\}$ where $a \in \mathbb{R}^n, b \in \mathbb{R}$. The inequality $a^\intercal x \leqslant b$ is said to define the corresponding halfspace. A *polytope* $P \subseteq \mathbb{R}^n$

is a bounded subset defined by intersection of finite number of halfspaces. A result of Minkowsky-Weyl states that equivalently, every polytope is the convex hull of a finite number of points. Let h be a halfspace defined by an inequality $a^\intercal x \leqslant b$; the inequality is said to be *valid* for a polytope P if $P = P \cap h$. Let h be a halfspace defined by a valid inequality $a^\intercal x \leqslant b$; then, $P \cap \{x | a^\intercal x = b\}$ is said to be a *face* of P.

Note that, taking a to be the zero vector and $b = 0$ results in the face being P itself. Also, taking a to be the zero vector and $b = 1$ results in the empty set. These two faces are often called the *trivial faces* and they are polytopes "living in" dimensions n and -1, respectively. Every face - that is not trivial - is itself a polytope of dimension d where $0 \leqslant d \leqslant n-1$. The zero dimensional faces of a polytope are called its *vertices*, and the $(n-1)$-dimensional faces are called its *facets*.

It is not uncommon to refer to three separate (but related) objects as a face: the actual face as defined above, the valid inequality defining it, and the equation corresponding to the valid inequality. While this is clearly a misuse of notation, the context usually makes it clear as to exactly which object is being referred to.

Let P be a polytope in $\mathbb{R}^d$. A polytope Q in $\mathbb{R}^{d+r}$ is called an *extended formulation* or an *extension* of P if P is a projection of Q onto the first d coordinates. Note that for any linear map $\pi : \mathbb{R}^{d+r} \to \mathbb{R}^d$ such that $P = \pi(Q)$, a polytope Q' exists such that P is obtained by dropping all but the first d coordinates on Q' and, moreover, Q and Q' have the same number of facets.

The *size* of a polytope is defined to be the number of its facet-defining inequalities. Finally, the *extension complexity* of a polytope P, denoted by $\mathrm{xc}(P)$, is the size of its smallest extended formulation. We refer the readers to the surveys [9, 35, 18, 36] for details and background of the subject and we only state three basic propositions about extended formulations here.

▶ **Proposition 1.** *Let P be a polytope with a vertex set $V = \{v_1, \ldots, v_n\}$. Then* $\mathrm{xc}(P) \leqslant n$.

Proof. Let $P = \mathrm{conv}(\{v_1, \ldots, v_n\})$ be a polytope. Then, P is the projection of

$$Q = \left\{ (x, \lambda) \,\middle|\, x = \sum_{i=1}^{n} \lambda_i v_i; \sum_{i=1}^{n} \lambda_i = 1; \lambda_i \geqslant 0 \text{ for } i \in \{1, \ldots, n\} \right\}.$$

It is clear that Q has at most n facets and therefore $\mathrm{xc}(P) \leqslant n$. ◀

▶ **Proposition 2.** *Let P be a polytope obtained by intersecting a set H of hyperplanes with a polytope Q. Then* $\mathrm{xc}(P) \leqslant \mathrm{xc}(Q)$.

Proof. Note that any extended formulation of Q, when intersected with H, gives an extended formulation of P. Intersecting a polytope with hyperplanes does not increase the number of facet-defining inequalities (and only possibly reduces it). ◀

The (cartesian) *product* of two polytopes P_1 and P_2 is defined as

$$P_1 \times P_2 = \mathrm{conv}(\{(x, y) \mid x \in P_1, y \in P_2\}).$$

▶ **Proposition 3.** *Let P_1, P_2 be two polytopes. Then*

$$\mathrm{xc}(P_1 \times P_2) \leqslant \mathrm{xc}(P_1) + \mathrm{xc}(P_2).$$

Proof. Let Q_1 and Q_2 be extended formulations of P_1 and P_2, respectively. Then, $Q_1 \times Q_2$ is an extended formulation of $P_1 \times P_2$. Now assume that $Q_1 = \{x \mid Ax \leqslant b\}$ and $Q_2 = \{y \mid Cy \leqslant d\}$ and that these are the smallest extended formulations of P_1 and P_2, resp. Then

$$Q_1 \times Q_2 = \{(x, y) \mid Ax \leqslant b, Cy \leqslant d\} .$$

That is, we have an extended formulation of $P_1 \times P_2$ of size at most $\mathrm{xc}(P_1) + \mathrm{xc}(P_2)$. ◀

We are going to define the glued product of polytopes, a slight generalization of the usual product of polytopes. We use a case where the extension complexity of the glued product of two polytopes is upper bounded by the sum of the extension complexities of the two polytopes, and use it in Section 4 to describe a small extended formulation for the MSO polytope $P_\varphi(G)$ on graphs with bounded treewidth.

Let $P \subseteq \mathbb{R}^{d_1+k}$ and $Q \subseteq \mathbb{R}^{d_2+k}$ be 0/1-polytopes defined by m_1 and m_2 inequalities and with vertex sets $\mathrm{vert}(P)$ and $\mathrm{vert}(Q)$, respectively. Let $I_P \subseteq \{1, \dots d_1 + k\}$ be a subset of coordinates of size k, $I_Q \subseteq \{1, \dots d_2 + k\}$ be a subset of coordinates of size k, and let $I'_P = \{1, \dots d_1 + k\} \setminus I_P$. For a vector x, and a subset I of coordinates, we denote by $x|_I$ the subvector of x specified by the coordinates I. The *glued product* of P and Q, (glued) with respect to the k coordinates I_P and I_Q, denoted by $P \times_k Q$, is defined as

$$P \times_k Q = \mathrm{conv}\left(\left\{(x|_{I'_P}, y) \in \mathbb{R}^{d_1+d_2+k} \mid x \in \mathrm{vert}(P), y \in \mathrm{vert}(Q), x|_{I_P} = y|_{I_Q}\right\}\right) .$$

We adopt the following convention while discussing glued products in the rest of this article. In the above scenario, we say that $P \times_k Q$ is obtained by gluing P and Q along the k coordinates I_P of P with the k coordinates I_Q of Q. If, for example, these coordinates are named z in P and w in Q, then we also say that P and Q have been glued along the z and w coordinates and we refer to the coordinates z and w as the *glued coordinates*. In the special case that we glue along the last k coordinates, the definition of the glued product simplifies to

$$P \times_k Q = \mathrm{conv}\left(\left\{(x, y, z) \in \mathbb{R}^{d_1+d_2+k} \mid (x, z) \in \mathrm{vert}(P), (y, z) \in \mathrm{vert}(Q)\right\}\right) .$$

This notion was studied by Margot [31] who provided a sufficient condition for being able to write the glued product in a specific (and efficient) way from the descriptions of P and Q. We will use this particular way in Lemma 1. The existing work [31, 10], however, is more focused on characterizing exactly when this particular method works. We do not need the result in its full generality and therefore we only state a very specific version of it that is relevant for our purposes; for the sake of completeness, we also provide a proof of it.

▶ **Lemma 1** (Gluing lemma). *Let P and Q be 0/1-polytopes and let the k (glued) coordinates in P be labeled $z_1, \dots, z_k$, and the k (glued) coordinates in Q be labeled $w_1, \dots, w_k$. Suppose that $\mathbf{1}^\intercal z \leqslant 1$ is valid for P and $\mathbf{1}^\intercal w \leqslant 1$ is valid for Q. Then $\mathrm{xc}(P \times_k Q) \leqslant \mathrm{xc}(P) + \mathrm{xc}(Q)$.*

Proof. Let (x', z', y', w') be a point from $P \times Q \cap \{(x, z, y, w) | z = w\}$. Observe that the point (x', z') is a convex combination of points $(x', 0), (x', e_1), \dots, (x', e_k)$ from P with coefficients $(1 - \sum_{i=1}^k z'_i), z'_1, z'_2, \dots, z'_k$ where e_i is the i-th unit vector. Similarly, the point (y', w') is a convex combination of points $(y', 0), (y', e_1), \dots, (y', e_k)$ from Q with coefficients $(1 - \sum_{i=1}^k w'_i), w'_1, w'_2, \dots, w'_k$. Notice that for every $j \in [k]$, (x'_j, e_j, y'_j) is a point from the glued product. As $w_i = z_i$ for every $i \in [k]$, we conclude that $(x', w', z') \in P \times_k Q$. Thus, by Proposition 2 the extension complexity of $P \times_k Q$ is at most that of $P \times Q$ which is at most $\mathrm{xc}(P) + \mathrm{xc}(Q)$ by Proposition 3. ◀

3.2 Graphs and Treewidth

For notions related to the treewidth of a graph and nice tree decomposition, in most cases we stick to the standard terminology as given in the book by Kloks [22]; the only deviation is in the leaf nodes of the nice tree decomposition where we assume that the bags are empty. For a vertex $v \in V$ of a graph $G = (V, E)$, we denote by $\delta(v)$ the set of neighbors of v in G, that is, $\delta(v) = \{u \in V \mid \{u, v\} \in E\}$.

A *tree decomposition* of a graph $G = (V, E)$ is a tree T in which each *node* $a \in T$ has an assigned set of vertices $B(a) \subseteq V$ (called a *bag*) such that $\bigcup_{a \in T} B(a) = V$ with the following properties:

- for any $\{u, v\} \in E$, there exists a node $a \in T$ such that $u, v \in B(a)$.
- if $v \in B(a)$ and $v \in B(b)$, then $v \in B(c)$ for all nodes c on the path from a to b in T.

The *treewidth* $tw(T)$ *of a tree decomposition* T is the size of the largest bag of T minus one. The *treewidth* $tw(G)$ *of a graph* G is the minimum treewidth over all possible tree decompositions of G.

A *nice tree decomposition* is a tree decomposition with one special node r called the *root* in which each node is one of the following types:

- *Leaf node*: a leaf a of T with $B(a) = \emptyset$.
- *Introduce node*: an internal node a of T with one child b for which $B(a) = B(b) \cup \{v\}$ for some $v \in B(a)$.
- *Forget node*: an internal node a of T with one child b for which $B(a) = B(b) \setminus \{v\}$ for some $v \in B(b)$.
- *Join node*: an internal node a with two children b and c with $B(a) = B(b) = B(c)$.

For a vertex $v \in V$, we denote by $top(v)$ the topmost node of the nice tree decomposition T that contains v in its bag. For any graph G on n vertices, a nice tree decomposition of G with at most $8n$ nodes can be computed in time $\mathcal{O}(n)$ [4, 22].

Given a graph $G = (V, E)$ and a subset of vertices $\{v_1, \ldots, v_d\} \subseteq V$, we denote by $G[v_1, \ldots, v_d]$ the subgraph of G induced by the vertices $v_1, \ldots, v_d$. Given a tree decomposition T and a node $a \in V(T)$, we denote by T_a the subtree of T rooted in a, and by G_a the subgraph of G induced by all vertices in bags of T_a, that is, $G_a = G[\bigcup_{b \in V(T_a)} B(b)]$. Throughout this paper we assume that for every graph, its vertex set is a subset of $\mathbb{N}$. We define the following operator σ: for any set $U = \{v_1, v_2, \ldots, v_l\} \subseteq \mathbb{N}$, $\sigma(U) = (v_{i_1}, v_{i_2}, \ldots, v_{i_l})$ such that $v_{i_1} < v_{i_2} \cdots < v_{i_l}$.

For an integer $m \geq 0$, an $[m]$-*colored graph* is a pair $(G, \vec{V})$ where $G = (V, E)$ is a graph and $\vec{V} = (V_1, \ldots, V_m)$ is an m-tuple of subsets of vertices of G called an $[m]$-*coloring of* G. For integers $m \geq 0$ and $\tau \geq 0$, an $[m]$-*colored* τ-*boundaried graph* is a triple $(G, \vec{V}, \vec{p})$ where $(G, \vec{V})$ is an $[m]$-colored graph and $\vec{p} = (p_1, \ldots, p_\tau)$ is a τ-tuple of vertices of G called a *boundary* of G. If the tuples $\vec{V}$ and $\vec{p}$ are clear from the context or if their content is not important, we simply denote an $[m]$-colored τ-boundaried graph by $G^{[m],\tau}$. For a tuple $\vec{p} = (p_1, \ldots, p_\tau)$, we denote by p the corresponding set, that is, $p = \{p_1, \ldots, p_\tau\}$.

Two $[m]$-colored τ-boundaried *graphs* $(G_1, \vec{V}, \vec{p})$ and $(G_2, \vec{U}, \vec{q})$ are *compatible* if the function $h : \vec{p} \to \vec{q}$, defined by $h(p_i) = q_i$ for each i, is an isomorphism of the induced subgraphs $G_1[p_1, \ldots, p_\tau]$ and $G_2[q_1, \ldots, q_\tau]$, and if for each i and j, $p_i \in V_j \Leftrightarrow q_i \in U_j$.

Given two compatible $[m]$-colored τ-boundaried graphs $G_1^{[m],\tau} = (G_1, \vec{U}, \vec{p})$ and $G_2^{[m],\tau} = (G_2, \vec{W}, \vec{q})$, the *join* of $G_1^{[m],\tau}$ and $G_2^{[m],\tau}$, denoted by $G_1^{[m],\tau} \oplus G_2^{[m],\tau}$, is the $[m]$-colored τ-boundaried graph $G^{[m],\tau} = (G, \vec{V}, \vec{p})$ where

- G is the graph obtained by taking the disjoint union of G_1 and G_2, and for each i, identifying the vertex p_i with the vertex q_i and keeping the label p_i for it;
- $\vec{V} = (V_1, \ldots, V_m)$ with $V_j = U_j \cup W_j$ and every q_i replaced by p_i, for each j;

- $\vec{p} = (p_1, \ldots, p_\tau)$ with p_i being the node in $V(G)$ obtained by the identification of $p_i \in V(G_1)$ and $q_i \in V(G_2)$, for each i.

Because of the choice of referring to the boundary vertices by their names in $G_1^{[m],\tau}$, it does not always hold that $G_1^{[m],\tau} \oplus G_2^{[m],\tau} = G_2^{[m],\tau} \oplus G_1^{[m],\tau}$; however, the two structures are isomorphic and equivalent for our purposes (see below).

3.3 Monadic Second Order Logic and Types of Graphs

In most cases, we stick to standard notation as given by Libkin [30]. A *vocabulary* σ is a finite collection of *constant* symbols $c_1, c_2, \ldots$ and *relation* symbols $P_1, P_2, \ldots$. Each relation symbol P_i has an associated arity r_i. A *σ-structure* is a tuple $\mathcal{A} = (A, \{c_i^{\mathcal{A}}\}, \{P_i^{\mathcal{A}}\})$ that consists of a universe A together with an interpretation of the constant and relation symbols: each constant symbol c_i from σ is associated with an element $c_i^{\mathcal{A}} \in A$ and each relation symbol P_i from σ is associated with an r_i-ary relation $P_i^{\mathcal{A}} \subseteq A^{r_i}$.

To give an example, a graph $G = (V, E)$ can be viewed as a σ_1-structure $(V, \emptyset, \{E\})$ where E is a symmetric binary relation on $V \times V$ and the vocabulary σ_1 contains a single relation symbol. Alternatively, for another vocabulary σ_2 containing three relation symbols, one of arity two and two of arity one, one can view a graph $G = (V, E)$ also as a σ_2-structure $I(G) = (V_I, \emptyset, \{E_I, L_V, L_E\})$, with $V_I = V \cup E$, $E_I = \{\{v, e\} \mid v \in e, e \in E\}$, $L_V = V$ and $L_E = E$; we will call $I(G)$ the *incidence graph* of G. In our approach we will make use of the well known fact that the treewidths of G and $I(G)$, viewed as a σ_1- and σ_2- structures as explained above, differ by one at most [24].

The main subject of this paper are formulas for graphs in monadic second order logic (MSO) which is an extension of first order logic that allows quantification over monadic predicates (i.e., over sets of vertices). By MSO_2 we denote the extension of MSO that allows in addition quantification over sets of edges. As every MSO_2 formula φ over σ_1 can be turned into an MSO formula φ' over σ_2 such that for every graph G, $G \models \varphi$ if and only if $I(G) \models \varphi'$ [folklore], for the sake of presentation we restrict our attention, without loss of generality, to MSO formulae over the σ_2 vocabulary. To further simplify the presentation, without loss of generality (cf. [21]) we assume that the input formulae are given in a variant of MSO that uses only set variables (and no element variables).

An important kind of structures that are necessary in the proofs in this paper are the $[m]$-colored τ-boundaried graphs. An $[m]$-colored τ-boundaried graph $G = (V, E)$ with boundary $p_1, \ldots, p_\tau$ colored with $V_1, \ldots, V_m$ is viewed as a structure $(V_I, \{p_1, \ldots, p_\tau\}, \{E_I, L_V, L_E, V_1, \ldots, V_m\})$; for notational simplicity, we stick to the notation $G^{[m],\tau}$ or $(G, \vec{V}, \vec{p})$. The corresponding vocabulary is denoted by $\sigma_{m,\tau}$.

A variable X is *free* in φ if it does not appear in any quantification in φ. If $\vec{X}$ is the tuple of all free variables in φ, we write $\varphi(\vec{X})$. A variable X is *bound* in φ if it is not free. By $qr(\varphi)$ we denote the *quantifier rank* of φ which is the number of quantifiers of φ when transformed into the prenex form (i.e., all quantifiers are at the beginning of the formula). We denote by $\mathrm{MSO}[k, \tau, m]$ the set of all MSO formulae φ over the vocabulary $\sigma_{\tau,m}$ with $qr(\varphi) \leq k$.

Two $[m]$-colored τ-boundaried graphs $G_1^{[m],\tau}$ and $G_2^{[m],\tau}$ are $\mathrm{MSO}[k]$-*elementarily equivalent* if they satisfy the same $\mathrm{MSO}[k, \tau, m]$ formulae; this is denoted by $G_1^{[m],\tau} \equiv_k^{MSO} G_2^{[m],\tau}$. The main tool in the model theoretic approach to Courcelle's theorem, that will also play a crucial role in our approach, can be stated as the following theorem.

▶ **Theorem 2** (follows from Proposition 7.5 and Theorem 7.7 [30]). *For any fixed $\tau, k, m \in \mathbb{N}$, the equivalence relation $\equiv_k^{MSO}$ has a finite number of equivalence classes.*

Let us denote the equivalence classes of the relation $\equiv_k^{MSO}$ by $\mathcal{C} = \{\alpha_1 \ldots, \alpha_w\}$, fixing an ordering such that α_1 is the class containing the empty graph. Note that the size of $\mathcal{C}$ depends only on k, m and τ, that is, $|\mathcal{C}| = f(k, m, \tau)$ for some computable function f. For a given MSO formula φ with m free variables, we define an *indicator function* $\rho_\varphi : \{1, \ldots, |\mathcal{C}|\} \to \{0, 1\}$ as follows: for every i, if there exists a graph $G^{[m],\tau} \in \alpha_i$ such that $G^{[m],\tau} \models \varphi$, we set $\rho_\varphi(i) = 1$, and we set $\rho_\varphi(i) = 0$ otherwise; note that if there exists a graph $G^{[m],\tau} \in \alpha_i$ such that $G^{[m],\tau} \models \varphi$, then $G'^{[m],\tau} \models \varphi$ for every $G'^{[m],\tau} \in \alpha_i$.

For every $[m]$-colored τ-boundaried graph $G^{[m],\tau}$, its *type*, with respect to the relation $\equiv_k^{MSO}$, is the class to which $G^{[m],\tau}$ belongs. We say that *types* α_i and α_j are *compatible* if there exist two $[m]$-colored τ-boundaried graphs of types α_i and α_j that are compatible; note that this is well defined as all $[m]$-colored τ-boundaried graphs of a given type are compatible. For every $i \geq 1$, we will encode the type α_i naturally as a binary vector $\{0, 1\}^{|\mathcal{C}|}$ with exactly one 1, namely with 1 on the position i.

An important property of the types and the join operation is that the type of a join of two $[m]$-colored τ-boundaried graphs depends on their types only.

▶ **Lemma 3** (Lemma 7.11 [30] and Lemma 3.5 [16]). *Let $G_a^{[m],\tau}$, $G_{a'}^{[m],\tau}$, $G_b^{[m],\tau}$ and $G_{b'}^{[m],\tau}$ be $[m]$-colored τ-boundaried graphs such that $G_a^{[m],\tau} \equiv_k^{MSO} G_{a'}^{[m],\tau}$ and $G_b^{[m],\tau} \equiv_k^{MSO} G_{b'}^{[m],\tau}$. Then $(G_a^{[m],\tau} \oplus G_b^{[m],\tau}) \equiv_k^{MSO} (G_{a'}^{[m],\tau} \oplus G_{b'}^{[m],\tau})$.*

The importance of the lemma rests in the fact that for determination of the type of a join of two $[m]$-colored τ-boundaried graphs, it suffices to know only a *small* amount of information about the two graphs, namely their types. The following two lemmas deal in a similar way with the type of a graph in other situations.

▶ **Lemma 4** (implicitly in [16]). *Let $(G_a, \vec{X}, \vec{p})$, $(G_b, \vec{Y}, \vec{q})$ be $[m]$-colored τ-boundaried graphs and let $(G_{a'}, \vec{X'}, \vec{p'})$, $(G_{b'}, \vec{Y'}, \vec{q'})$ be $[m]$-colored $(\tau+1)$-boundaried graphs with $G_a = (V, E)$, $G_{a'} = (V', E')$, $G_b = (W, F)$, $G_{b'} = (W', F')$ such that for some $v \notin V$ and $w \notin W$*
1. *$(G_a, \vec{X}, \vec{p}) \equiv_k^{MSO} (G_b, \vec{Y}, \vec{q})$;*
2. *$V' = V \cup \{v\}$, $\delta(v) \subseteq p$, $\vec{p}$ is a subtuple of $\vec{p'}$ and $(G_{a'}[V], \vec{X'}[V], \vec{p'}[V]) = (G_a, \vec{X}, \vec{p})$;*
3. *$W' = W \cup \{w\}$, $\delta(w) \subseteq q$, $\vec{q}$ is a subtuple of $\vec{q'}$ and $(G_{b'}[W], \vec{Y'}[W], \vec{q'}[W]) = (G_b, \vec{Y}, \vec{q})$;*
4. *$(G_{a'}, \vec{X'}, \vec{p'})$ and $(G_{b'}, \vec{Y'}, \vec{q'})$ are compatible.*
Then $(G_{a'}, \vec{X'}, \vec{p'}) \equiv_k^{MSO} (G_{b'}, \vec{Y'}, \vec{q'})$.

▶ **Lemma 5** (implicitly in [16]). *Let $(G_a, \vec{X}, \vec{p})$, $(G_b, \vec{Y}, \vec{q})$ be $[m]$-colored τ-boundaried graphs and let $(G_{a'}, \vec{X'}, \vec{p'})$, $(G_{b'}, \vec{Y'}, \vec{q'})$ be $[m]$-colored $(\tau+1)$-boundaried graphs with $G_a = (V, E)$, $G_{a'} = (V', E')$, $G_b = (W, F)$, $G_{b'} = (W', F')$ such that*
1. *$(G_{a'}, \vec{X'}, \vec{p'}) \equiv_k^{MSO} (G_{b'}, \vec{Y'}, \vec{q'})$;*
2. *$V \subseteq V'$, $|V'| = |V| + 1$, $\vec{p}$ is a subtuple of $\vec{p'}$ and $(G_{a'}[V], \vec{X'}[V], \vec{p'}[V]) = (G_a, \vec{X}, \vec{p})$;*
3. *$W \subseteq W'$, $|W'| = |W| + 1$, $\vec{q}$ is a subtuple of $\vec{q'}$ and $(G_{b'}[W], \vec{Y'}[W], \vec{q'}[W]) = (G_b, \vec{Y}, \vec{q})$.*
Then $(G_a, \vec{X}, \vec{p}) \equiv_k^{MSO} (G_b, \vec{Y}, \vec{q})$.

3.4 Feasible Types

Suppose that we are given an MSO formula φ over σ_2 with m free variables and a quantifier rank at most k, a graph G of treewidth at most τ, and a nice tree decomposition T of the graph G.

For every node of T we are going to define certain types and tuples of types as *feasible*. For a node $b \in V(T)$ of any kind (leaf, introduce, forget, join) and for $\alpha \in \mathcal{C}$, we say that α is a *feasible type of the node b* if there exist an $[m]$-coloring $\vec{X} = (X_1, \ldots, X_m)$ of G_b such

that $(G_b, \vec{X}, \sigma(B(b)))$ is of type α; we say that $\vec{X}$ *realizes type* α *on the node* b. We denote the set of feasible types of the node b by $\mathcal{F}(b)$.

For an *introduce* node $b \in V(T)$ with a child $a \in V(T)$ (assuming that v is the new vertex), for $\alpha \in \mathcal{F}(a)$ and $\beta \in \mathcal{F}(b)$, we say that (α, β) is a *feasible pair of types for* b if there exist $\vec{X} = (X_1, \ldots, X_m)$ and $\vec{X'} = (X'_1, \ldots, X'_m)$ realizing types α and β on the nodes a and b, respectively, such that for each i, either $X'_i = X_i$ or $X'_i = X_i \cup \{v\}$. We denote the set of feasible pairs of types of the introduce node b by $\mathcal{F}_p(b)$.

For a *forget* node $b \in V(T)$ with a child $a \in V(T)$ and for $\beta \in \mathcal{F}(b)$ and $\alpha \in \mathcal{F}(a)$, we say (α, β) is a *feasible pair of types for* b if there exists $\vec{X}$ realizing β on b and α on a. We denote the set of feasible pairs of types of the forget node b by $\mathcal{F}_p(b)$.

For a *join* node $c \in V(T)$ with children $a, b \in V(T)$ and for $\alpha \in \mathcal{F}(c)$, $\gamma_1 \in \mathcal{F}(a)$ and $\gamma_2 \in \mathcal{F}(b)$, we say that $(\gamma_1, \gamma_2, \alpha)$ is a *feasible triple of types for* c if γ_1, γ_2 and α are mutually compatible and there exist $\vec{X^1}, \vec{X^2}$ realizing γ_1 and γ_2 on a and b, respectively, such that $\vec{X} = (X^1_1 \cup X^2_1, \ldots, X^1_m \cup X^2_m)$ realizes α on c. We denote the set of feasible triples of types of the join node c by $\mathcal{F}_t(c)$.

We define an *indicator function* $\mu : \mathcal{C} \times V(G) \times \{1, \ldots, m\} \to \{0, 1\}$ such that $\mu(\beta, v, i) = 1$ if and only if there exists $\vec{X} = (X_1, \ldots, X_m)$ realizing the type β on the node $top(v) \in V(T)$ and $v \in X_i$.

4 Extension Complexity of the MSO Polytope

For a given MSO formula $\varphi(\vec{X})$ over σ_2 with m free set variables $X_1, \ldots, X_m$, we define a polytope of satisfying assignments on a given graph G, represented as a σ_2 structure $I(G) = (V_I, \emptyset, \{E_I, L_V, L_E\})$ with domain of size n, in a natural way. We encode any assignment of elements of $I(G)$ to the sets $X_1, \ldots, X_m$ as follows. For each X_i in φ and each v in V_I, we introduce a binary variable y^v_i. We set y^v_i to be one if $v \in X_i$ and zero otherwise. For a given 0/1 vector y, we say that y *satisfies* φ if interpreting the coordinates of y as described above yields a satisfying assignment for φ. The polytope of satisfying assignments, also called the *MSO polytope*, is defined as

$$P_\varphi(G) = \operatorname{conv}\left(\{y \in \{0,1\}^{nm} \mid y \text{ satisfies } \varphi\}\right).$$

▶ **Theorem 6** (Extension Complexity of the MSO Polytope). *For every graph G and for every MSO formula φ, $\operatorname{xc}(P_\varphi(G)) = f(|\varphi|, \tau) \cdot n$ where f is some computable function, $\tau = tw(G)$ and $n = |V_I|$.*

Proof. Let T be a fixed nice tree decomposition of treewidth τ of the given graph G represented as $I(G)$ and let k denote the quantifier rank of φ and m the number of free variables of φ. Let $\mathcal{C}$ be the set of equivalence classes of the relation $\equiv^{MSO}_k$. For each node b of T we introduce $|\mathcal{C}|$ binary variables that will represent a feasible type of the node b; we denote the vector of them by t_b (i.e., $t_b \in \{0,1\}^{|\mathcal{C}|}$). For each introduce and each forget node b of T, we introduce additional $|\mathcal{C}|$ binary variables that will represent a feasible type of the child (descendant) of b; we denote the vector of them by d_b (i.e., $d_b \in \{0,1\}^{|\mathcal{C}|}$). Similarly, for each join node b we introduce additional $|\mathcal{C}|$ binary variables, denoted by l_b, that will represent a feasible type of the left child of b, and other $|\mathcal{C}|$ binary variables, denoted by r_b, that will represent a feasible type of the right child of b (i.e., $l_b, r_b \in \{0,1\}^{|\mathcal{C}|}$).

We are going to describe inductively a polytope in the dimension given (roughly) by all the binary variables of all nodes of the given nice tree decomposition. Then we show that its extension complexity is small and that a properly chosen face of it is an extension of $P_\varphi(G)$.

First, for each node b of T, depending on its type, we define a polytope P_b as follows:

- b is a *leaf*. P_b consists of a single point $P_b = \{\overbrace{100\ldots0}^{|\mathcal{C}|}\}$.
- b is an *introduce* or *forget* node. For each feasible pair of types $(\alpha_i, \alpha_j) \in \mathcal{F}_p(b)$ of the node b, we create a vector $(d_b, t_b) \in \{0,1\}^{2|\mathcal{C}|}$ with $d_b[i] = t_b[j] = 1$ and all other coordinates zero. P_b is defined as the convex hull of all such vectors.
- b is a *join* node. For each feasible triple of types $(\alpha_h, \alpha_i, \alpha_j) \in \mathcal{F}_t(b)$ of the node b, we create a vector $(l_b, r_b, t_b) \in \{0,1\}^{3|\mathcal{C}|}$ with $l_b[h] = r_b[i] = t_b[j] = 1$ and all other coordinates zero. P_b is defined as the convex hull of all such vectors.

It is clear that for every node b in T, the polytope P_b contains at most $|\mathcal{C}|^3$ vertices, and, thus, by Proposition 1 it has extension complexity at most $\mathrm{xc}(P_b) \leqslant |\mathcal{C}|^3$. Recalling our discussion in Section 3 about the size of $\mathcal{C}$, we conclude that there exists a function f such that for every $b \in V(T)$, it holds that $\mathrm{xc}(P_b) \leqslant f(|\varphi|, \tau)$.

To obtain an extended formulation for $P_\varphi(G)$, we first glue these polytopes together, starting in the leaves of T and processing T in a bottom up fashion. We create polytopes Q_b for each node b in T recursively as follows:

- If b is a leaf then $Q_b = P_b$.
- If b is an introduce or forget node, then $Q_b = Q_a \times_{|\mathcal{C}|} P_b$ where a is the child of b and the gluing is done along the coordinates t_a in Q_a and d_b in P_b.
- If b is a join node, then we first define $R_b = Q_a \times_{|\mathcal{C}|} P_b$ where a is the left child of b and the gluing is done along the coordinates t_a in Q_a and l_b in P_b. Then Q_b is obtained by gluing R_b with Q_c along the coordinates t_c in Q_c and r_b in R_b where c is the right child of b.

The following lemma states the key property of the polytopes Q_b's.

▶ **Lemma 7.** *For every vertex y of the polytope Q_b there exist an $[m]$-coloring $\vec{X} = (X_1, \ldots, X_m)$ of G_b such that $(G_b, \vec{X}, \sigma(B(b)))$ is of type α where α is the unique type such that the coordinate of y corresponding to the binary variable $t_b(\alpha)$ is equal to one.*

Proof. The proof is by induction, starting in the leaves of T and going up towards the root. For leaves, the lemma easily follows from the definition of the polytopes P_b's.

For the inductive step, we consider an inner node b of T and we distinguish three cases:

- If b is a join node, then the claim for b follows from the inductive assumptions for the children of b, definition of a feasible triple, definition of the polytope P_b, Lemma 3 and the construction of the polytope Q_b.
- If b is an introduce node or a forget node, respectively, then, analogously, the claim for b follows from the inductive assumption for the child of b, definition of a feasible pair, definition of the polytope P_b, Lemma 4 or Lemma 5, respectively, and the construction of the polytope Q_b. ◀

Let c be the root node of the tree decomposition T. Consider the polytope Q_c. From the construction of Q_c, our previous discussion and the Gluing lemma, it follows that $\mathrm{xc}(Q_c) \leqslant \sum_{b \in V(T)} \mathrm{xc}(P_b) \leqslant f(|\varphi|, \tau) \cdot \mathcal{O}(n)$. It remains to show that a properly chosen face of Q_c is an extension of $P_\varphi(G)$. We start by observing that $\sum_{i=1}^{|\mathcal{C}|} t_c[i] \leq 1$ and $\sum_{i=1}^{|\mathcal{C}|} \rho_\varphi(i) \cdot t_c[i] \leq 1$, where ρ_φ is the indicator function, are valid inequalities for Q_c.

Let Q_φ be the face of Q_c corresponding to the valid inequality $\sum_{i=1}^{|\mathcal{C}|} \rho_\varphi(i) \cdot t_c[i] \leq 1$. Then, by Lemma 7, the polytope Q_φ represents those $[m]$-colorings of G for which φ holds. The corresponding feasible assignments of φ on G are obtained as follows: for every vertex $v \in V(G)$ and every $i \in \{1, \ldots, m\}$ we set $y_i^v = \sum_{j=1}^{|\mathcal{C}|} \mu(\alpha_j, v, i) \cdot t_{top(v)}[j]$. The sum is 1 if

and only if there exists a type j such that $t_{top(v)}[j] = 1$ and at the same time $\mu(\alpha_j, v, i) = 1$; by the definition of the indicator function μ in Subsection 3.4, this implies that $v \in X_i$. Thus, by applying the above projection to Q_φ we obtain $P_\varphi(G)$, as desired.

It is worth mentioning at this point that the polytope Q_c depends only on the quantifier rank k of φ and the number of free variables of φ. The dependence on the formula φ itself only manifests in the choice of the face Q_φ of Q_c that projects to $P_\varphi(G)$. ◀

▶ **Corollary 8.** *The extension complexity of the convex hull of all satisfying assignments of a given MSO_2 formula φ on a given graph G of bounded treewidth is linear in the size of the graph G.*

5 Efficient Construction of the MSO Polytope

In the previous section we have proven that $P_\varphi(G)$ *has* a compact extended formulation but our definition of feasible tuples and the indicator functions μ and ρ_φ did not explicitly provide a way how to actually *obtain* it efficiently. That is what we do in this section. We also briefly mention some implications of our results for optimization versions of Courcelle's theorem.

As in the previous section we assume that we are given a graph G of treewidth τ and an MSO formula φ with m free variables and quantifier rank k. We start by constructing a nice tree decomposition T of G of treewidth τ in linear time.

Let $\mathcal{C}$ denote the set of equivalence classes of $\equiv_k^{MSO}$. Because $\mathcal{C}$ is finite and its size is independent of the size of G (Theorem 2), for each class $\alpha \in \mathcal{C}$, there exists an $[m]$-colored τ-boundaried graph $(G^\alpha, \vec{X}^\alpha, \vec{p^\alpha})$ of type α whose size is upper-bounded by a function of k, m and τ. For each $\alpha \in \mathcal{C}$, we fix one such graph, denote it by $W(\alpha)$ and call it the *witness of* α. Let $\mathcal{W} = \{W(\alpha) \mid \alpha \in \mathcal{C}\}$. The witnesses make it possible to compute the indicator function ρ_φ: for every $\alpha \in \mathcal{C}$, we set $\rho_\varphi(\alpha) = 1$ if and only if $W(\alpha) \models \varphi$, and we set $\rho_\varphi(\alpha) = 0$ otherwise.

▶ **Lemma 9** (implicitly in [16] in the proof of Theorem 4.6 and Corollary 4.7)**.** *The set $\mathcal{W}$ and the indicator function ρ_φ can be computed in time $f(k, m, \tau)$, for some computable function f.*

It will be important to have an efficient algorithmic test for MSO$[k, \tau]$-elementary equivalence. This can be done using the Ehrenfeucht-Fraïssé games:

▶ **Lemma 10** (Theorem 7.7 [30])**.** *Given two $[m]$-colored τ-boundaried graph $G_1^{[m],\tau}$ and $G_2^{[m],\tau}$, it can be decided in time $f(m, k, \tau, |G_1|, |G_2|)$ whether $G_1^{[m],\tau} \equiv_k^{MSO} G_2^{[m],\tau}$, for some computable function f.*

▶ **Corollary 11.** *Recognizing the type of an $[m]$-colored τ-boundaried graph $G^{[m],\tau}$ can be done in time $f(m, k, \tau, |G|)$, for some computable function f.*

Now we describe a linear time construction of the sets of feasible types, pairs and triples of types $\mathcal{F}(b)$, $\mathcal{F}_p(b)$ and $\mathcal{F}_t(b)$ for all relevant nodes b in T. In the initialization phase, we construct the set $\mathcal{W}$ and the indicator function ρ_φ using the algorithm from Lemma 9. The rest of the construction is inductive, starting in the leaves of T and advancing in a bottom up fashion towards the root of T. The idea is to always replace a possibly *large* graph $G_a^{[m],\tau}$ of type α by the *small* witness $W(\alpha)$ when computing the set of feasible types for the parent of the node a.

Leaf node. For every leaf node $a \in V(T)$ we set $\mathcal{F}(a) = \{\alpha_1\}$. Obviously, this corresponds to the definition in Section 3.

Introduce node. Assume that $b \in V(T)$ is an introduce node with a child $a \in V(T)$ for which $\mathcal{F}(a)$ has already been computed, and $v \in V(G)$ is the new vertex. For every $\alpha \in \mathcal{F}(a)$, we first produce a τ'-boundaried graph $H^{\tau'} = (H^\alpha, \vec{q})$ from $W(\alpha) = (G^\alpha, \vec{X^\alpha}, \vec{p^\alpha})$ as follows: let $\tau' = |\vec{p^\alpha}| + 1$ and H^α be obtained from G^α by attaching to it a new vertex in the same way as v is attached to G_a. The boundary $\vec{q}$ is obtained from the boundary $\vec{p^\alpha}$ by inserting in it the new vertex at the same position that v has in the boundary of $(G_a, \sigma(B(a)))$. For every subset $I \subseteq \{1, \ldots, m\}$ we construct an $[m]$-coloring $\vec{Y}^{\alpha,I}$ from $\vec{X^\alpha}$ by setting $Y_i^{\alpha,I} = X_i^\alpha \cup \{v\}$, for every $i \in I$, and $Y_i^{\alpha,I} = X_i^\alpha$, for every $i \notin I$. Each of these $[m]$-colorings $\vec{Y}^{\alpha,I}$ is used to produce an $[m]$-colored τ'-boundaried graph $(H^\alpha, \vec{Y}^{\alpha,I}, \vec{q})$ and the types of all these $[m]$-colored τ'-boundaried graphs are added to the set $\mathcal{F}(b)$ of feasible types of b, and, similarly, the pairs (α, β) where β is a feasible type of some of the $[m]$-colored τ'-boundaried graph $(H^\alpha, \vec{Y}^{\alpha,I}, \vec{q})$, are added to the set $\mathcal{F}_p(b)$ of all feasible pairs of types of b. The correctness of the construction of the sets $\mathcal{F}(b)$ and $\mathcal{F}_p(b)$ for the node b of T follows from Lemma 4.

Forget node. Assume that $b \in V(T)$ is a forget node with a child $a \in V(T)$ for which $\mathcal{F}(a)$ has already been computed and that the d-th vertex of the boundary $\sigma(B(a))$ is the vertex being forgotten. We proceed in a similar way as in the case of the introduce node. For every $\alpha \in \mathcal{F}(a)$ we produce an $[m]$-colored τ'-boundaried graph $(H^\alpha, \vec{Y}^\alpha, \vec{q})$ from $W(\alpha) = (G^\alpha, \vec{X}^\alpha, \vec{p}^\alpha)$ as follows: let $\tau' = |\vec{p^\alpha}| - 1$, $H^\alpha = G^\alpha$, $\vec{Y}^\alpha = \vec{X}^\alpha$ and $\vec{q} = (p_1, \ldots, p_{d-1}, p_{d+1}, \ldots, p_{\tau'+1})$. For every $\alpha \in \mathcal{F}(a)$, the type β of the constructed graph is added to $\mathcal{F}(b)$, and, similarly, the pairs (α, β) are added to $\mathcal{F}_p(b)$. The correctness of the construction of the sets $\mathcal{F}(b)$ and $\mathcal{F}_p(b)$ for the node b of T follows from Lemma 5.

Join node. Assume that $c \in V(T)$ is a join node with children $a, b \in V(T)$ for which $\mathcal{F}(a)$ and $\mathcal{F}(b)$ have already been computed. For every pair of compatible types $\alpha \in \mathcal{F}(a)$ and $\beta \in \mathcal{F}(b)$, we add the type γ of $W(\alpha) \oplus W(\beta)$ to $\mathcal{F}(c)$, and the triple (α, β, γ) to $\mathcal{F}_t(c)$. The correctness of the construction of the sets $\mathcal{F}(c)$ and $\mathcal{F}_t(c)$ for the node b of T follows from Lemma 3.

It remains to construct the indicator function μ. We do it during the construction of the sets of feasible types as follows. We initialize μ to zero. Then, every time we process a node b in T and we find a new feasible type β of b, for every $v \in B(b)$ and for every i for which d-th vertex in the boundary of $W(\beta) = (G^\beta, \vec{X}, \vec{p})$ belongs to X_i, we set $\mu(\beta, v, i) = 1$ where d is the order of v in the boundary of $(G_b, \sigma(B(b))$. The correctness follows from the definition of μ and the definition of feasible types.

Concerning the time complexity of the inductive construction, we observe, exploiting Corollary 11, that for every node b in T, the number of steps, the sizes of graphs that we worked with when dealing with the node b, and the time needed for each of the steps, depends on k, m and τ only. We summarize the main result of this section in the following theorem.

▶ **Theorem 12.** *Under the assumptions of Theorem 6, the polytope $P_\varphi(G)$ can be constructed in time $f'(|\varphi|, \tau) \cdot n$, for some computable function f'.*

5.1 Courcelle's Theorem and Optimization

It is worth noting that even though linear time *optimization* versions of Courcelle's theorem are known, our result provides a linear size LP for these problems out of the box. Together with a polynomial algorithm for solving linear programming we immediately get the following:

▶ **Theorem 13.** *Given a graph G on n vertices with treewidth τ, a formula $\varphi \in MSO$ with m free variables and real weights w_v^i, for every $v \in V(G)$ and $i \in \{1, \ldots, m\}$, the problem*

$$opt \left\{ \sum_{v \in V(G)} \sum_{i=1}^{m} w_v^i \cdot y_v^i \;\middle|\; y \text{ satisfies } \varphi \right\}$$

where opt is min *or* max*, is solvable in time polynomial in the input size.*

Acknowledgements. We thank the anonymous reviewers for pointing out existing work and shorter proof of the Glueing lemma, among various other improvements.

References

1 S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, June 1991.

2 D. Avis and H. R. Tiwary. On the extension complexity of combinatorial polytopes. In *Proc. ICALP(1)*, pages 57–68, 2013.

3 D. Bienstock and G. Munoz. LP approximations to mixed-integer polynomial optimization problems. *ArXiv e-prints*, January 2015. `arXiv:1501.00288`.

4 H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proc. STOC*, pages 226–234, 1993.

5 H. L. Bodlaender. Treewidth: characterizations, applications, and computations. In *Proc. of WG*, volume 4271 of *LNCS*, pages 1–14. Springer, 2006.

6 G. Braun, S. Fiorini, S. Pokutta, and D. Steurer. Approximation limits of linear programs (beyond hierarchies). *Math. Oper. Res.*, 40(3):756–772, 2015.

7 G. Braun, R. Jain, T. Lee, and S. Pokutta. Information-theoretic approximations of the nonnegative rank. *Electronic Colloquium on Computational Complexity*, 20:158, 2013.

8 A. Buchanan and S. Butenko. Tight extended formulations for independent set, 2014. Available on Optimization Online. URL: `http://www.optimization-online.org/DB_HTML/2014/09/4540.html`.

9 M. Conforti, G. Cornuéjols, and G. Zambelli. Extended formulations in combinatorial optimization. *Annals of Operations Research*, 204(1):97–143, 2013.

10 M. Conforti and K. Pashkovich. The projected faces property and polyhedral relations. *Mathematical Programming*, pages 1–12, 2015.

11 B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

12 B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique width. In *Proc. of WG*, volume 1517 of *LNCS*, pages 125–150. Springer, 1998.

13 B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109(1–2):49–82, 1 March 1993.

14 Y. Faenza, S. Fiorini, R. Grappe, and H. R. Tiwary. Extended formulations, nonnegative factorizations, and randomized com. protocols. *Math. Program.*, 153(1):75–94, 2015.

15 S. Fiorini, S. Massar, S. Pokutta, H. R. Tiwary, and Ronald de Wolf. Exponential lower bounds for polytopes in combinatorial optimization. *J. ACM*, 62(2):17, 2015.

16 G. Gottlob, R. Pichler, and F. Wei. Monadic datalog over finite structures with bounded treewidth. In *Proc. PODS*, pages 165–174, 2007.

17 B. Grünbaum. *Convex Polytopes*. Wiley Interscience Publ., London, 1967.

18 V. Kaibel. Extended formulations in combinatorial optimization. *Optima*, 85:2–7, 2011.

19 V. Kaibel and A. Loos. Branched polyhedral systems. In *Proc. IPCO*, volume 6080 of *LNCS*, pages 177–190. Springer, 2010.

20 V. Kaibel and K. Pashkovich. Constructing extended formulations from reflection relations. In *Proc. IPCO*, volume 6655 of *LNCS*, pages 287–300. Springer, 2011.

21 L. Kaiser, M. Lang, S. Leßenich, and Ch. Löding. A Unified Approach to Boundedness Properties in MSO. In *Proc. of CSL*, volume 41 of *LIPIcs*, pages 441–456, 2015.

22 T. Kloks. *Treewidth: Computations and Approximations*, volume 842 of *LNCS*. Springer, 1994.

23 J. Kneis, A. Langer, and P. Rossmanith. Courcelle's theorem – A game-theoretic approach. *Discrete Optimization*, 8(4):568–594, 2011.

24 P. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. In *Proc. PODS*, 1998.

25 P. Kolman and M. Koutecký. Extended formulation for CSP that is compact for instances of bounded treewidth. *Electr. J. Comb.*, 22(4):P4.30, 2015.

26 S. Kreutzer. Algorithmic meta-theorems. In *Proc. of IWPEC*, volume 5018 of *LNCS*, pages 10–12. Springer, 2008.

27 A. Langer, F. Reidl, P. Rossmanith, and S. Sikdar. Practical algorithms for MSO model-checking on tree-decomposable graphs. *Computer Science Review*, 13-14:39–74, 2014.

28 M. Laurent. Sums of squares, moment matrices and optimization over polynomials. In *Emerging applications of algebraic geometry*, pages 157–270. Springer, 2009.

29 J. R. Lee, P. Raghavendra, and D. Steurer. Lower bounds on the size of semidefinite programming relaxations. In *Proc. STOC*, pages 567–576, 2015.

30 L. Libkin. *Elements of Finite Model Theory*. Springer, Berlin, 2004.

31 F. Margot. *Composition de polytopes combinatoires: une approche par projection*. PhD thesis, École polytechnique fédérale de Lausanne, 1994.

32 R. K. Martin, R. L. Rardin, and B. A. Campbell. Polyhedral characterization of discrete dynamic programming. *Oper. Res.*, 38(1):127–138, February 1990.

33 M. Sellmann. The polytope of tree-structured binary constraint satisfaction problems. In *Proc. CPAIOR*, volume 5015 of *LNCS*, pages 367–371. Springer, 2008.

34 M. Sellmann, L. Mercier, and D. H. Leventhal. The linear programming polytope of binary constraint problems with bounded tree-width. In *Proc. CPAIOR*, volume 4510 of *LNCS*, pages 275–287. Springer, 2007.

35 F. Vanderbeck and L. A. Wolsey. Reformulation and decomposition of integer programs. In *50 Years of Integer Programming 1958-2008*, pages 431–502. Springer, 2010.

36 L. A. Wolsey. Using extended formulations in practice. *Optima*, 85:7–9, 2011.

37 M. Yannakakis. Expressing combinatorial optimization problems by linear programs. *J. Comput. Syst. Sci.*, 43(3):441–466, 1991.

38 G. M. Ziegler. *Lectures on Polytopes*, volume 152 of *Graduate Texts in Mathematics*. Springer, 1995.

Optimal Online Escape Path Against a Certificate*

Elmar Langetepe[1] and David Kübel[2]

1 **University of Bonn, Institute of Computer Science I, Bonn, Germany**
elmar.langetepe@cs.uni-bonn.de
2 **University of Bonn, Institute of Computer Science I, Bonn, Germany**
kuebel@cs.uni-bonn.de

Abstract

More than fifty years ago Bellman asked for the best escape path within a known fores t but for an unknown starting position. This deterministic finite path is the shortest path that leads out of a given environment from any starting point. There are some worst case positions where the full path length is required. Up to now such a fixed *ultimate optimal escape path* for a known shape for any starting position is only known for some special convex shapes (i.e., circles, strips of a given width, fat convex bodies, some isosceles triangles).

Therefore, we introduce a different, simple and intuitive escape path, the so-called *certificate path* which makes use of some additional information w.r.t. the starting point s. This escape path depends on the starting position s and takes the distances from s to the outer boundary of the environment into account. Because of this, in the above convex examples the certificate path *always* (for any position s) leaves the environment earlier than the ultimate escape path.

Next we assume that neither the precise shape of the environment nor the location of the starting point is known, we have much less information. For a class of environments (convex shapes and shapes with kernel positions) we design an *online* strategy that always leaves the environment. We show that the path length for leaving the environment is always shorter than 3.318764 the length of the corresponding certificate path. We also give a lower bound of 3.313126 which shows that for the above class of environments the factor 3.318764 is (almost) tight.

1998 ACM Subject Classification F.1.2 Modes of Computation, Online computation, F.2.2 Nonnumerical Algorithms and Problems, Geometrical problems and computations, G.1.6 Optimization

Keywords and phrases Search games, online algorithms, escape path, competitive analysis, spiral conjecture

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.19

1 Introduction

We consider the following motion planning task. Let us assume that we are given a simple polygon P and a starting point s inside P. We would like to design a simple path starting at s that finally hits the boundary and leaves the polygon. In the sense of a game, we can choose a path but then an adversary can rotate the polygon P around s so that the path will leave the polygon very late.

Since we know the distances to the boundary we apply a simple intuitive strategy for this problem. The *certificate path* is the best combination of a line segment l and an arc of length $l\alpha$ along the circle of radius l around the starting point. So this path simply checks an

* Partially supported by the National Science Foundation, NSF grant CCF 1017539; see also [17].

angular portion of the environment for a distance l. For a given starting point the certificate path is the best (shortest) such path that guarantees to hit the boundary. Altogether the certificate path is a very simple *escape path* for given s and P (if an adversary can only rotate P around s).

In turn, for any given unknown starting position s inside an unknown polygon we would like to design an *online strategy* (with less information) that is never much worse than the length of the above certificate path. In this paper we show that for a class of environments there is a spiral strategy that leaves any such polygon and approximates the length of the certificate path within a ratio of 3.318674. We also prove that this is an (almost) tight bound. There is no other strategy that always attains a better ratio against the length of the certificate path.

This optimal online approximation is restricted to the following class of environments. We assume that in any direction from the unknown starting point only one boundary point exists. The distance to the boundary points still remains unknown. This subsumes any unknown convex environment (for any unknown starting position) and also unknown star-shaped environments (for any unknown starting point inside the kernel). The motivation of comparing an online escape path for special polygons (star-shaped) and special starting positions (inside the kernel) with a path that is computed with some additional but not complete information (certificate path) stems from the following observation.

For a given known polygonal shape and an unknown starting point we can also define the *ultimate optimal escape path*. This path will lead out of the environment for any starting point and any rotation of the polygon. The ultimate optimal escape path is the shortest finite path with this property. The clue is that only the polygon is known but neither the starting position nor the rotation around the starting position. The path is motivated by the situation of swimming in the fog in a pool whose shape is known. Because it is foggy, the starting point and the rotation around the starting point is not known. Unfortunately, ultimate optimal escape paths have been found only for a few special convex shapes (circles, strips of given width, fat convex bodies, isosceles triangles, ...). It is unrealistic to think that such paths will be found for more complicated convex or star-shaped environments.

Fortunately, for the few known ultimate optimal escape paths the above defined certificate path is not only a good approximation, we can even show that the certificate path *beats* the ultimate escape paths for *any* starting point in these examples. Therefore, we can argue that the certificate path for any star-shaped polygon and any starting point inside the kernel can serve as a substitute for the unknown ultimate optimal escape path.

The paper is organized as follows. In the next section we present the related work. After introducing the certificate path in Section 3 and showing different justifications for the measure in Section 4, we present and analyse a strategy with path length not larger than 3.318674 times the length of the certificate path in Section 5. The strategy is a logarithmic spiral attained by keeping aware of two extremes of the certificate. Optimizing the spiral for two extremes is also different from classical logarithmic spiral constructions where we normally optimize against a single distance (shortest path). In Section 6 we present a general lower bound that also proves that the given strategy is almost optimal for the restricted cases. No other strategy will have a better ratio than 3.313126 against the length of the certificate path. Proving lower bounds is a tedious task, the construction and the analysis might be interesting in its own right.

2 Related work

The *Swimming-in-the-fog* problem is a game where two players, a searcher and a hider, compete with each other. The searcher tries to reach the boundary of a known shape from its starting point along a single finite path, while the hider can rotate and translate the environment so that the path of the searcher will cross the boundary as late as possible. For a given shape the shortest finite path that always leads out of the given environment can be denoted as an *ultimate optimal escape* path as mentioned before.

Search games have been studied in many variations in the last 60 years since the first work by Koopman in 1946. The book by Gal [13] and the reissue by Alpern and Gal [1] gives a comprehensive overview of such search game problems also for unknown environments.

The above problem goes at least back to 1956 and to Bellman [3] who similarly asked for the shortest escape path within a known forest but for an unknown starting point. Since then the problem has attracted a lot of attention. Unfortunately until today, the problem could be solved only for very special convex environments (circles, strips of given width, rectangles, fat convex bodies, isosceles triangles); see for example the monograph of Finch and Wetzel [9].

For circles and fat convex bodies it was shown that the diameter is the ultimate optimal escape path; see Finch and Wetzel [9]. For the strip of width l the ultimate optimal escape path was found by Zalgaller [24, 25]. For the simple equilateral triangle of side length 1 the Zig-Zag path of Besicovitch [4] of length ≈ 0.981981 is optimal; see also [5]. Furthermore, in 1961 Gluss [14] introduced the problem of searching for a circle C of given radius s and given distance r away from the start A. Two different cases can be considered, either A is inside C or not. Interestingly, in the latter case and for $s = 1$ a certificate path with length $l = r$ and an arc of length $2\pi \cdot l$ is the best one can do.

It is unrealistic to think that such ultimate optimal escape paths will be achieved for more complicated environments. As an alternative in this paper for a given environment and a given starting point s we have introduced a simple and natural *certificate path* that is computed individually for any starting point and takes the distance distribution from s to the boundary into account. Fortunately, for the cases above we are able to show that our certificate path outperforms the corresponding ultimate escape paths for any possible starting point. The certificate path always leaves the environment earlier. The proofs are given in the full version of the paper; see [20].

The use of alternative comparison measures has some tradition. For example for the problem of searching for a point in a polygon and competing against the shortest path there is no competitive strategy. So for this case also other comparison measures have been suggested; see Fleischer et al. [11] or Koutsoupias et al. [18]. Additionally, comparing the online strategy to the shortest path to the boundary is often a very difficult task. For example, the spiral conjecture for searching for a single line or a single ray against the shortest path is still open. In this sense our result might be considered as an intermediate step.

We further assume that the precise shape of the environment and the position of the starting point is not known. We are searching for a good *online* approximation that competes with the certificate path which is computed with more information. We make use of the competitive framework. That is, we compare the length of the online escape path from a starting point to the boundary to the length of the certificate path to the boundary computed for the known environment and starting point. The competitive framework was introduced by Sleator and Tarjan [23], and used in many settings since then; see for example the survey by Fiat and Woeginger [7] or, for the field of online robot motion planning, see the surveys [15, 21].

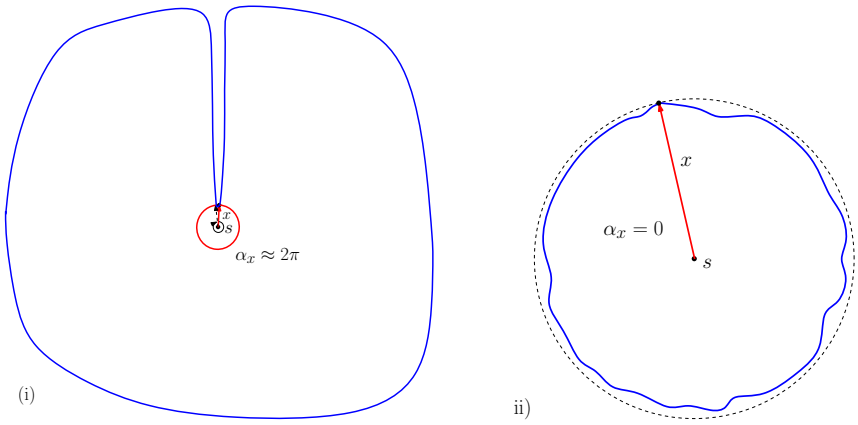


Figure 1 Two extreme situation for reaching the boundary with a circular arc. On the right-hand side of the figure, the radial maximal distance from s to the boundary is almost the same in any direction. So it suffices to move in an arbitrary direction of maximal distance, which is optimal. On the left-hand side of the figure only the distance to some few boundary points is very small, but much larger to most of the others. Therefore, a reasonable path checks the small distance with a circular arc of length approximately 2π. In both cases $x(1 + \alpha_x)$ is minimal among all such circular strategies.

Our optimal online approximation is restricted to the following class of environments. We assume that in any direction from the unknown starting point only one boundary point exists. The distance to the boundary points still remains unknown. This subsumes any unknown convex environment (for any unknown starting position) and also unknown star-shaped environments (for any unknown starting point inside the kernel). In this sense the certificate is also a natural extension of the discrete performance measure Kirkpatrick [16] mentioned in the discrete case of searching for the end of a set of m given lists of unknown length. In his setting it is sufficient to reach the end of only one list. In our configuration this means, that we have exactly 2π directions of unknown distance and it is sufficient to reach the shoreline in a single point. The corresponding relationship is shown in Section 4.

We will see that our solution is a specific logarithmic spiral. In general, logarithmic spirals are natural candidates for optimal competitive search strategies, but in almost all cases the optimality remains a conjecture; see [2, 6, 8, 10, 13]. In [19] the optimality of spiral search was shown for searching a point in the plane with a radar. Many other conjectures are still open. For example Finch and Zhu [10] considered the problem of searching for a line in the plane, the *relevant conjecture that the family of logarithmic spirals contains the minimal path remains open.*

3 The certificate path

Assume that you are located in an unknown environment and would like to reach its boundary. Formally, for the environment we consider a closed Jordan curve B that subdivides the Euclidean plane into exactly two regions. The starting point s lies inside the inner region, say P. The task is to reach a point on the boundary B as soon as possible.

If you have some idea about the distance x from s to the boundary B but nothing more, it is very intuitive to move along the circle of radius x around the starting point. Therefore, a reasonable strategy moves toward this circle along a shortest path (by radius x) in some

direction and then follows the circle in either clockwise or counterclockwise direction until the boundary is met. Let us denote this behaviour a *circular strategy*. If we hit the boundary after moving an arc α_x along the circle, the overall path length is given by $x(1+\alpha_x)$.

We would like to use such a circular strategy of small path length. In the sense of a game, the adversary can only rotate the environment around the starting point and the certificate path guarantees to hit the boundary for any rotation.

3.1 Extreme cases and general definition

Let us first consider two somehow extreme examples of the above intuitive idea as given in Figure 1. If the distance from s to the boundary is almost the same in any direction (similar to a circle), a line segment with maximal distance to the boundary (roughly the radius of the circle) will always hit the boundary and is indeed a very good escape path for any direction; see Figure 1(ii). The movement along an arc is not necessary in this case. In other words, α_x equals 0. We check a single direction for the largest distance.

On the other hand, if the distance to the boundary is very large w.r.t. almost all directions from s, but is relatively small (distance x) for some few directions, a segment of length x and a circular arc of length $x\alpha_x$ with $\alpha_x \approx 2\pi$ will hit the boundary for any starting direction of the segment x; see Figure 1(i). The overall path length $x(1+\alpha_x)$ is also comparatively small. The certificate path checks a small distance for many (almost all) directions.

Now, consider a more general environment modelled by a simple polygon P and a fixed starting point s in P as given in Figure 2(ii). For convenience, we make use of an example, where any boundary point b of P is *visible* from s, i.e. the segment sb lies fully inside P. Or the other way round, s lies inside the kernel of P. Note, that the certificate can also be computed for more general polygons as shown in the full version of the paper [20].

For the polygon P and for any radial direction $\phi \in [0, 2\pi]$ from s, we consider the boundary point $p_{s,\phi}$ on P in direction ϕ. This gives a *radial distance function* $f(\phi) := |s\, p_{s,\phi}|$ as depicted in Figure 2(i).

Now, let $p_{s,\phi}$ be a point with distance $x := |s\, p_{s,\phi}|$ in direction ϕ. For any circle $C_s(x)$ with radius x around s such that $C_s(x)$ hits the boundary of P, there will be some maximal arc $\alpha_s(x)$ so that the above simple circular strategy is successful. Note, that this is independent from the starting direction for x. We are looking for the maximum circle segment of $C_s(x)$ that fully lies inside P.

Let $\Pi_s(x)$ denote the *certificate path for distance* x of maximal length $x(1+\alpha_s(x))$. The interpretation is that independent from the starting direction for x, this finite path will always touch the boundary. The adversary can only rotate the environment in order to attain a worst case length of $x(1+\alpha_x)$. The path $\Pi_s(x)$ can be found in the plot of the radial distance function; see Figure 2(i). It consists of two segments, starting with a vertical segment of length x and ending with a horizontal segment of length α_x. For any starting angle this path will touch the boundary of the distance function.

In turn, the overall *certificate path* Π_s in P for a given starting point s is the shortest certificate path $\Pi_s(x)$ among all distances x. That is, the certificate for P and s is:

$$\Pi_s := \min_x \Pi_s(x) = \min_x x(1+\alpha_s(x)) \ .$$

For both extreme situations in Figure 1, the presented paths equal the overall certificate paths for the given environments.

If parts of the boundary are not visible from s, there is more than one boundary point in some directions. In this case we can also compute the radial distances in a continuous way

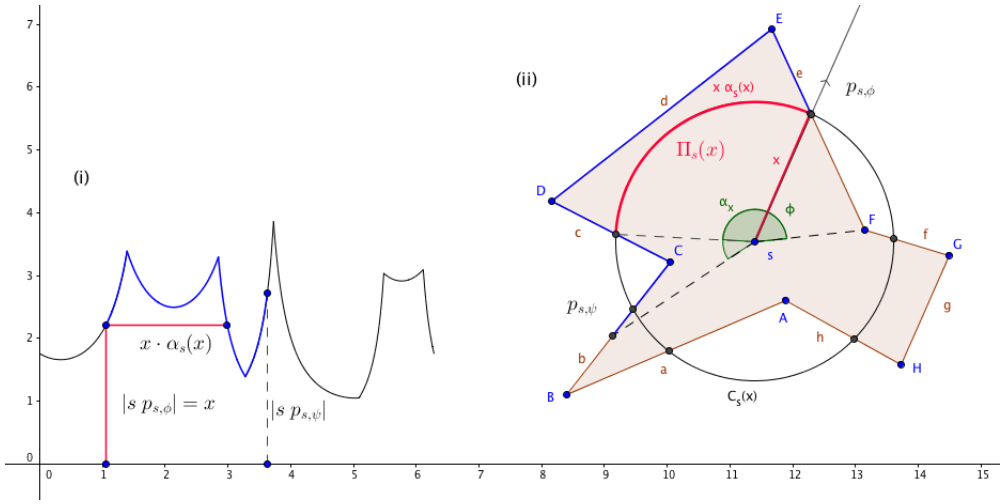


Figure 2 (ii) Consider the polygon P and a starting point s. Let us assume that we radially sweep the boundary of P (starting from point F with angle 0) in counterclockwise order and calculate the distance from the boundary to s for any angle. (i) shows this *radial distance function* of the boundary of P from s in polar coordinates for the interval $[0, 2\pi]$. The blue sub-curve corresponds to the blue boundary part in (ii). The certificate path $\Pi_s(x)$ for distance x is the longest path that successfully checks the distance x by a circular strategy. This means that it hits the boundary for any starting direction ϕ of x in P. In the polar-coordinate setting in (i) this is a path with two line segments of length x and α_x that always hits the boundary of the radial distance function independent from the starting angle ϕ. For the case where the boundary of P is not totally visible an example is given in the full version of the paper [20].

and obtain a radial distance curve. The definition of the certificate path remains exactly the same. An example is given in the full version of the paper; see [20].

4 Justification of the certificate

The certificate path is an intuitive and simple way of leaving an environment and can be computed in polynomial time by computing a lower envelope of upper envelopes. We can interpret the certificate as a path that balances depth-first and breadth-first search for the starting position s in a way that the resulting path is as short as possible. That way, it outperforms the ultimate optimal escape path at any given starting position for all known cases as proved in the full version of this paper; see [20].

Furthermore, the certificate is closely related to a discrete cost measure that Kirkpatrick introduces in [16]. He analyses the problem of digging for oil at m different locations s_i, where $|s_i|$ denotes the (unknown) distance to the source of the oil at the corresponding location. In this scenario, no extra costs arise for switching the location. The challenge is to find a strategy that reaches one source of oil while assuring a small overall digging effort.

At first, Kirkpatrick considers (partially informed) strategies. Those are given all distances from the top to the sources of oil, but not the corresponding location: In case the distances $|s_i|$ have about the same length at all locations, he states that a depth-first searching strategy is certainly effective. Thus, a single location can be chosen for digging, as Figure 3(i) indicates. Although at the chosen location, the distance to the source might be greatest, the digging costs are almost optimal. In case the distance to the source of a single location is significantly

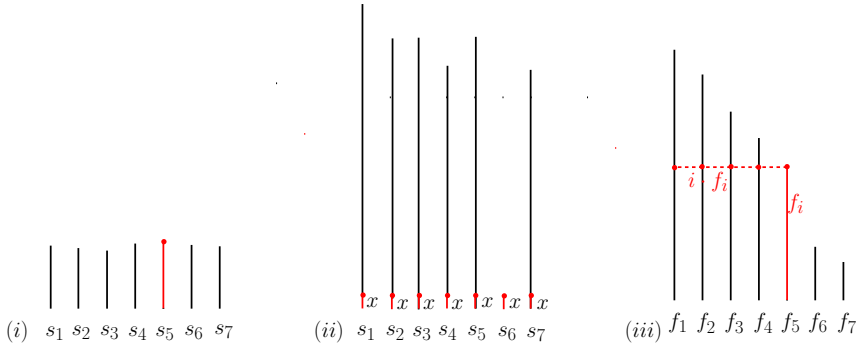


Figure 3 Online searching for the end of a segment (or digging for oil) for $m = 7$ segments of unknown length. There are two extreme cases: (i) All segments have about the same length. It is reasonable to move along an arbitrary segment up to the end, which is almost optimal. (ii) One segment is significantly shorter than all other segments. One will find the end of a shortest segment by checking all segments with its length. (iii) In case the length of each segment is known, but not the corresponding number of segment. There is always an optimal strategy: Assume that $f_1 \geq f_2 \geq \cdots \geq f_m$ is the decreasing order of the length of all segments. An optimal strategy explores i (arbitrary) segments up to depth f_i, where i is chosen so that $i \cdot f_i = \min_{1 \leq k \leq m} k \cdot f_k$.

shorter than all others, a breadth-first searching strategy performs best. Figure 3(ii) shows that digging at every location with a certain effort x still achieves a small overall effort of $x \cdot m$ in the worst case. These two extreme situations are similar to the cases outlined in Section 3.1 and depicted in Figure 1. For the general case, Kirkpatrick suggests to use a hybrid strategy. If $f_1 \geq f_2 \geq \cdots \geq f_m$ denotes the sorted set of distances, he suggests to choose i so that $i \cdot f_i$ is minimal. The hybrid strategy digs at i (arbitrarily chosen) locations up to the same depth f_i. In the worst case, this strategy reaches a source at the last location with a final effort of $i \cdot f_i$; see Figure 3(iii). Among all such partially informed strategies, this hybrid strategy is certainly optimal and achieves a maximum digging effort of $\lambda := i \cdot f_i$. Similar to this hybrid strategy, we defined the certificate path in the previous section. The certificate path can also be considered as a mixture of depth-first and breadth-first searching. However, the certificate path models a motion. The effort of the digging strategy to explore a certain depth depends on the product of the number chosen locations and the digging depth. In contrast to this, the effort of the certificate path depends on the sum of the searching depth and width. Consequently, the certificate path is a stronger cost measure than the equivalent of the hybrid digging strategy in the plane.

During the further analysis, Kirkpatrick compares a totally uninformed digging strategy to the optimal hybrid strategy. He proves that this strategy approximates the hybrid strategy in $O(\lambda \log(\min(m, \lambda))$ and shows that this factor is tight. Similar to his approach, we compare the certificate path to a totally uninformed spiral strategy and obtain a constant competitive ratio. David Kirkpatrick [17] brought up the question what happens in a continuous setting. Note, that the game is a quite different in this case, because we have to take the movements in the plane into account and we also require a starting orientation.

5 Online approximation of the certificate path

We are searching for a reasonable escape strategy in an unknown environment. As shown in the previous section, the certificate path and its length is a reasonable candidate for

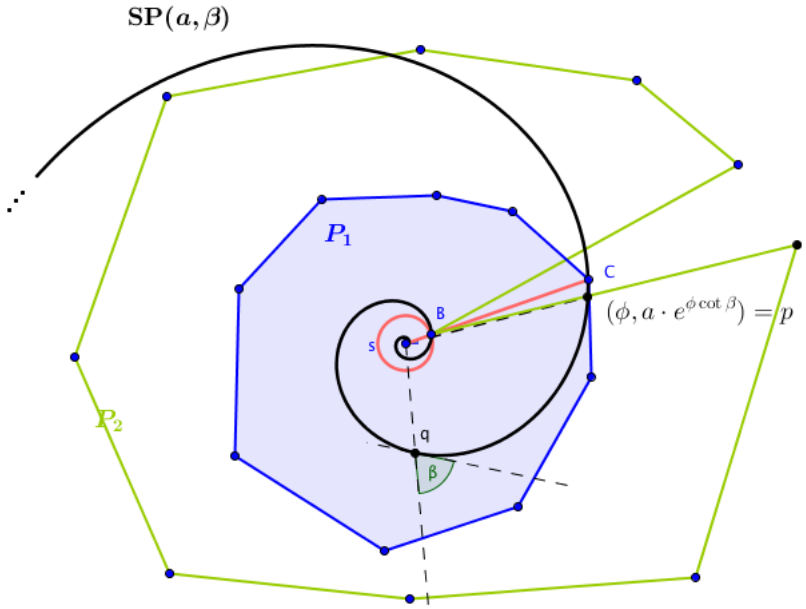


Figure 4 We apply a spiral strategy for unknown polygons and an unknown starting point s in the kernel. The eccentricity β is chosen so that the two extreme cases have the same ratio. For both polygons P_1 and P_2, the strategy passes the boundary at point $p = (\phi, a \cdot e^{\phi \cot \beta})$ close to C. The path length of the strategy for leaving the polygons is roughly the same. The certificate for P_1 has length $|s\ C|$ (checking the maximal distance to the boundary of P_1), whereas the certificate for P_2 has length $|s\ B|(1 + 2\pi)$ with $|s\ C| = e^{2\pi \cot \beta}|s\ B|$ (checking the smallest distance to the boundary of P_2 with a full circle). We can construct such examples for any point p on the spiral.

comparisons. Let us assume that $x(1+\alpha_x)$ is the length of the certificate for some polygon P and for an arbitrary distance x. We can assume that $\alpha_x \in [0, 2\pi]$. This holds since the shortest distance d_s from s to the boundary always results in a candidate $d_s(1 + 2\pi)$. All other reasonable distances x are larger than d_s and $\alpha_x \leq 2\pi$ holds for the optimal x.

Similar to the considerations of Kirkpatrick (see Section 4), we would like to guarantee that we leave the polygon P if we have overrun the distance x more than α_x times. This means that the boundary should not wind arbitrarily around s. Therefore, we restrict our consideration to a position s in the kernel of a star-shaped polygon so that there is a single (unknown) distance to the outer boundary in any direction.

In this case we apply the following logarithmic spiral strategy. A logarithmic spiral can be defined by polar coordinates $(\varphi, a \cdot e^{\varphi \cot(\beta)})$ for $\varphi \in (-\infty, \infty)$, a constant a and an eccentricity β as shown in Figure 4. For an angle ϕ, the path length of the spiral up to point $(\phi, a \cdot e^{\phi \cot(\beta)})$ is given by $\frac{a}{\cos \beta} e^{\phi \cot(\beta)}$.

For our purpose we choose β so that the two extreme cases of the certificate attain the same ratio; see Figure 4. We can assume that the certificate of the environment is $x(1 + \alpha_x)$ for an arbitrary distance x and an angle $\alpha_x \in [0, 2\pi]$. Since the spiral strategy checks the distances in a monotonically increasing and periodical way, there has to be some angle ϕ so that $x = a \cdot e^{(\phi - \alpha_x) \cot(\beta)}$ holds. This means that in the worst case, the spiral strategy will leave the environment at point $p = (\phi, a \cdot e^{\phi \cot(\beta)})$ with path length $\frac{a}{\cos \beta} \cdot e^{\phi \cot(\beta)}$. Exactly α_x distances of length x have been exceeded, which means that the boundary has been reached. (Note that, this might not hold for points outside the kernel.)

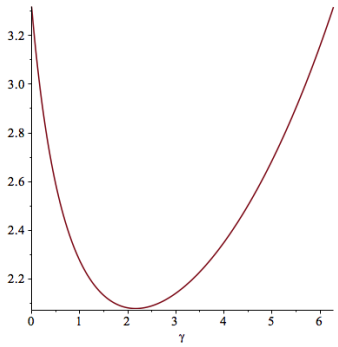


Figure 5 The graph of the ratio function f of Equation (3) for the spiral strategy with eccentricity $\beta \approx 1.26471$. The two extreme cases 0 and 2π have the same ratio ≈ 3.318674 and all other ratios are strictly smaller.

We would like to choose β so that the two extreme cases $\alpha_x = 0$ and $\alpha_x = 2\pi$ have the same ratio. Thus, we are searching for an angle β so that

$$\frac{\frac{a}{\cos\beta} \cdot e^{\phi \cot\beta}}{a \cdot e^{\phi \cot\beta}(1+0)} = \frac{\frac{a}{\cos\beta} \cdot e^{\phi \cot\beta}}{a \cdot e^{(\phi-2\pi)\cot\beta}(1+2\pi)} \quad \Leftrightarrow \tag{1}$$

$$1 = \frac{e^{2\pi \cot\beta}}{1+2\pi} \tag{2}$$

holds. The right-hand side of Equation (1) shows the case where $x_2 = a \cdot e^{(\phi-2\pi)\cot(\beta)}$ and $\alpha_{x_2} = 2\pi$ gives the certificate and the left-hand side shows the case that $x_1 = a \cdot e^{\phi \cot(\beta)}$ and $\alpha_{x_1} = 0$ gives the certificate $x_i(1+\alpha_{x_i})$, respectively. In both cases the spiral will detect the boundary latest at point $p = (\phi, a \cdot e^{\phi \cot(\alpha)})$, because the spiral checks 2π distances larger than or equal to x_2 and at least one distance x_1. Figure 4 shows the construction of corresponding polygons P_1 and P_2.

The solution of Equation (2) gives $\beta = \operatorname{arccot}\left(\frac{\ln(2\pi+1)}{2\pi}\right) = 1.264714\ldots$ and the ratio is $\frac{1}{\cos\beta} = 3.3186738\ldots$. Fortunately, for all other values $x = a \cdot e^{(\phi-\gamma)\cot\beta}$ and $\alpha_x = \gamma$ for $\gamma \in (0, 2\pi)$ the ratio is smaller than these two extremes. The overall ratio function is

$$f(\gamma) = \frac{\frac{a}{\cos\beta} \cdot e^{\phi \cot\beta}}{a \cdot e^{(\phi-\gamma)\cot\beta}(1+\gamma)} = \frac{e^{\gamma \cot\beta}}{\cos\beta(1+\gamma)} \text{ for } \gamma \in [0, 2\pi] \tag{3}$$

and Figure 5 shows the plot of all possible ratios of the spiral strategy with eccentricity β.

Altogether, we have the following result.

▶ **Theorem 1.** *There is a spiral strategy for any unknown starting point s inside the kernel of an unknown environment P that always hits the boundary with path length smaller than* 3.318674 *times the length of the corresponding certificate for s and P.*

Proof. Assume that the certificate of P and s is given by $x(1+\alpha_x)$. We can set $\gamma := \alpha_x$ and we will also find an angle ϕ so that $x = a \cdot e^{(\phi-\gamma)\cot\beta}$ holds. At point $p = (\phi, a \cdot e^{\phi \cot\beta})$ the spiral has subsumed an arc of angle γ with distances x, so the spiral strategy will leave P at p in the worst case. (Note that, if the start point is not inside the kernel, this might not be true!) The ratio is given by $f(\gamma)$ as in (3) and Figure 5. In the worst case for the strategy γ is either 0 or 2π for the ratio 3.318674, respectively. ◀

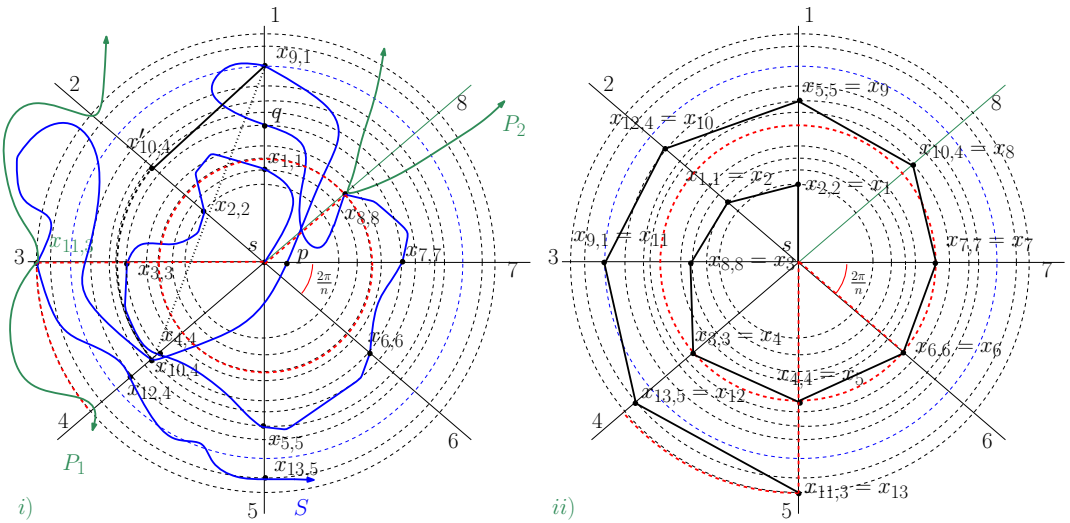


Figure 6 i) The strategy S results in a sequence S' that represents the visits on n rays successively. There will be a next entry x_{i,j_i} if the strategy exceeds the distance on ray j_i. For two successive extensions on the same ray only the last entry is registered in S'. For the subsequence $S'_{13} = (x_{1,1}, x_{2,2}, \ldots, x_{13,5})$ there will be a last visit on each ray, a minimal distance $x_m = x_{8,8}$ on ray 8 and a maximal distance $x_M = x_{11,3}$ on ray 3. These values gives rise to the construction of certificates for S as sketched by the polygons P_1 and P_2. There are polygons P_1 and P_2 with certificates $x_M(1+\frac{2\pi}{n})$ and $x_m(1+2\pi)$ and so far S has not been escaped from neither P_1 nor P_2. In S'_{13} the direct distance between two successive points, for example $x_{9,1}$ and $x_{10,4}$, is shorter than the original path length on S and we can further shorten the distance by assuming that $x_{10,4}$ is on the neighbouring ray as depicted by $x'_{10,4}$.

ii) We sort the entries of S'_k into a sequence X_k and visit the rays in increasing distance and periodic order. The path length of X_k is not larger than the original path length of S_k and the corresponding certificates for the maximal and minimal value $x'_M = x_M$ and $x'_m > x_m$ are not smaller. Thus, the sum of the corresponding ratios gives a lower bound for the sum of the original ratios.

We have designed a spiral strategy for some reasonable environments. In the next section, we give a lower bound that shows that this strategy is (almost) optimal for these environments.

Note that a spiral strategy for the online approximation of the certificate path of an arbitrary unknown polygon and position cannot be competitive in general. The polygon might itself wind around the spiral. The ratio against the certificate might be arbitrarily large, consequently. In more general environments other online strategies have to be applied. A potential strategy might be a connected sequence of circles C_i with exponentially increasing radii r^i. This online strategy should result in a constant competitive ratio. Obtaining the optimal strategy for such cases might be complicated and gives rise to future work.

6 Lower bound construction: Online strategy against the certificate

▶ **Theorem 2.** *Any strategy that escapes from an unknown environment P in unknown position s will achieve a competitive factor of at least* 3.313126 *against the length of a corresponding certificate for s and P in the worst-case.*

Proof. Let us assume that a strategy S is given that attains a better ratio C in the worst case. We consider a bunch of n rays emanating from s with equidistant angle $\frac{2\pi}{n}$ as depicted in Figure 6 for $n = 8$.

The strategy S will successively extend the distances from s also along the rays. Let the sequence $S' = (x_{1,j_1}, x_{2,j_2}, x_{3,j_3}, \ldots)$ describe successive visits of the n rays by the strategy. In x_{i,j_i} the entry i stands for the order and j_i stands for the ray. In S' we only register a visit on ray j_i, if it exceeds the previous visit on the ray j_i. Furthermore, if the distance at ray j_i is exceeded in two successive entries we do not register the first visit in the sequence S'.

In Figure 6(i) we have registered 13 successive visits x_{i,j_i}. Here for example the visit of ray 7 at point q between $x_{9,1}$ and $x_{10,4}$ was not registered in S' because it does not improve the distance of the former visit $x_{7,7}$. Additionally, the visit of ray 1 at point q just before $x_{9,1}$ improved the distance $x_{1,1}$ but it was further improved on $x_{9,1}$ and in between no other ray was improved. For any continuous strategy S we will find such an infinite sequence S'. Let x_{i,j_i} denote the visit and also the distance to the starting point s.

Let us assume that we stop the strategy S at of some ray j_k where the distance was just exceeded on ray j_k, so S' has k steps. Let S_k denote the sub-strategy of S and S'_k the corresponding subsequence. There will be at least two ratios for S_k that correspond to values of S'_k as follows. In S'_k we consider the last visits on each ray which gives the corresponding maximal visited distance to s for each ray. There will be an overall maximal distance x_M on some ray M_j and a minimal distance x_m on some other ray m_j. In Figure 6(i) we have stopped the strategy S at $x_{13,5}$ on ray 5 and in S'_{13} the minimal distance for the last *round* is given by $x_m = x_{8,8}$ on ray 8 and the maximal distance is given by $x_M = x_{11,3}$ on ray 3.

We can construct polygons P_1 and P_2 so that $x_M(1 + \frac{2\pi}{n})$ is a certificate for P_1 and $x_m(1 + 2\pi)$ is a certificate for P_2. In the first case all other rays have been visited with depth smaller or equal to x_M and we build a polygon P_1 outside S_k that visits any ray at $x_M - \epsilon$ and ray M_j at x_M. This means that a circular strategy with x_M and an arc of length $x_M \frac{2\pi}{n}$ will be sufficient and gives the certificate for P_1 (or at least an upper bound for the certificate of P_1). See for example the polygon P_1 sketched in Figure 6(i) for the maximal visit $x_M = x_{13,5}$. On the other hand for the minimal value x_m we construct a polygon P_2 that hits x_m on ray m_j but runs arbitrarily far away from S_k in any other direction. Thus, $x_m(1 + 2\pi)$ gives the certificate (or at least an upper bound for the certificate of P_2). See for example the polygon P_2 sketched in Figure 6(i) for the minimal visit $x_m = x_{8,8}$. We do not expect that S_k has already detected these polygons but S finally will. So the ratio of the path length $|S_k|$ over $x_m(1 + 2\pi)$ and also the ratio of the path length $|S_k|$ over $x_M(1 + \frac{2\pi}{n})$ give lower bounds for the strategy S. Note that the half of the sum of the two ratios cannot exceed C because otherwise at least one has to be greater than C.

For any such stop we will sort the values of S'_k in a sequence X_k and we will visit the n rays in a monotone and periodic way by sequence X_k connecting the points by line segments; see Figure 6(ii). We can prove that the overall path length of X_k cannot be larger than $|S_k|$.

This can be seen as follows. Successively visiting the points of S'_k in a polygonal chain is already a short cut for S_k. This polygonal path for S'_k might move between two successive values x_{i,j_i} to $x_{i+1,j_{i+1}}$ where j_i and j_{i+1} are not neighbouring rays. In this case we can further short cut the length of the chain of S'_k by just counting a movement from x_{i,j_i} to the distance $x_{i+1,j_{i+1}}$ on one of the directly neighbouring rays. For example in Figure 6(i) the segment from $x_{9,1}$ and $x'_{10,4}$ improves the path length from $x_{9,1}$ and $x_{10,4}$ but passes ray 2 and 3. We only count the distance between $x_{9,1}$ and $x'_{10,4}$ on the neighbouring ray which further improves the length. This means that for a lower bound on the overall path length we can also consider a path that visits two neighbouring rays with angle $\frac{2\pi}{n}$ successively from one to the other with the corresponding depth values x_{i,j_i} stemming from S'_k. By triangle inequality it can be shown that the shortest path that visit all the depth of a sequence S'_k on two rays by changing from one ray to the other in any step, visits the two rays successively

in an increasing order. A similar argument was applied by one author of this article in [19] where a detailed proof of this property is given in the Appendix of [19]. Finally, we can rearrange the path of S'_k to a path that visits the rays in a periodic and monotone way.

Altogether, we have translated the strategy S_k in a discrete strategy $X_k = (x_1, x_2, \ldots, x_k)$ with k entries on n rays that visit the rays in a periodic order such that x_i visits ray $i \bmod n$ and with overall shorter path length; see Figure 6(ii). Consider the corresponding certificates of this new strategy in comparison to the original strategy. For the smallest value on the last round and the largest value on the last round we will obtain a certificate path $x_k(1+\frac{2\pi}{n})$ which is the same for the previous maximal value $x_M = x_k$ and a certificate path $x_{k-n+1}(1+2\pi)$ which is never smaller than $x_m(1+2\pi)$ for the minimal value $x_m \leq x_{n-k+1}$. The minimal value can only increase since we sorted the values of S'_k. For example in Figure 6(ii) we have $k = 13$ and the minimal value in the last round is given by $x_6 = x_{6,6}$ which is larger than $x_m = x_{8,8}$. Altogether, the sum of these two ratios in the periodic and monotone setting is always smaller than the sum of the ratios in the original setting.

Finally, we would like to find a periodic and monotone strategy that optimizes the sum of exactly such ratios in this discrete version. This optimal strategy will perform at least as good as any strategy obtained by the above reconstruction. Thus, the optimal value for the sum is a lower bound for the sum of two ratios in the original setting.

For optimizing the sum for an arbitrary strategy we use an infinite sequence of values $X = (x_1, x_2, \ldots)$ and we define the following functionals

$$F_k^1(X) = \frac{\sum_{i=1}^{k-1} \sqrt{x_i^2 - 2\cos\left(\frac{2\pi}{n}\right) x_i x_{i+1} + x_{i+1}^2}}{x_k(1+\frac{2\pi}{n})} \tag{4}$$

and

$$F_k^2(X) = \frac{\sum_{i=1}^{k-1} \sqrt{x_i^2 - 2\cos\left(\frac{2\pi}{n}\right) x_i x_{i+1} + x_{i+1}^2}}{x_{k-n+1}(1+2\pi)} \tag{5}$$

that represent the ratios. We are looking for a sequence X so that

$$\inf_Y \sup_k F_k^1(Y) + F_k^2(Y) = D \text{ and } \sup_k F_k^1(X) + F_k^2(X) = D$$

holds which shows that D is the best sum ratio that we can achieve.

Optimizing such discrete functionals can be done by the method proposed by Gal; see also Gal [12, 13], Alpern and Gal [1], and an adaption of Schuierer [22]. It is shown that under certain prerequisites there will be an optimal exponential strategy $x_i = a^i$. The main requirement is that the functional has to fulfil a unimodality property. This means that the piecewise sum of two strategies X and Y is never worse than one of the single strategies. This should also hold for a scalar multiplication of a single strategy. So any linear combination of strategies that are bounded by a constant will remain bounded by the maximal bound. The proof of Gal shows that in this case we can always combine bounded strategies so that we finally get arbitrarily close to an exponential strategy that has the same bound; see the full proof of Gal in [13] Appendix 2, Theorem 1.

We can easily show that the requirements for the main Theorem of Gal are fulfilled for both functionals $F_k^1(X)$ and $F_k^2(X)$. For a similar functional a detailed proof of this property was given in the Appendix of [19].

Now let us assume that we have an optimal strategy X for the sum, say $F_k^1(X) + F_k^2(X)$. This means that both functionals will also be bounded by constants D_1 and D_2 w.r.t. X. We make use of linear combination of X but apply them independently to the functionals $F_k^1(X)$

and $F_k^2(X)$. The Theorem of Gal shows that we will get arbitrarily close to an exponential strategy $x_i = a^i$ that is not worse than X for both $F_k^1(X)$ and $F_k^2(X)$. This means that $x_i = a^i$ is also not worse than X for the sum functional.

Altogether, it is allowed to search for the best strategy $x_i = a^i$ and we have to optimize

$$\frac{\sum_{i=1}^{k-1}\sqrt{a^{2i} - 2\cos\left(\frac{2\pi}{n}\right)a^{2i+1} + a^{2i+2}}}{a^k(1+\frac{2\pi}{n})} + \frac{\sum_{i=1}^{k-1}\sqrt{a^{2i} - 2\cos\left(\frac{2\pi}{n}\right)a^{2i+1} + a^{2i+2}}}{a^{k-n+1}(1+2\pi)} \iff$$

$$\sum_{i=1}^{k-1} a^i\left(\frac{\sqrt{1 - 2\cos\left(\frac{2\pi}{n}\right)a + a^2}}{a^k(1+\frac{2\pi}{n})}\right) + \sum_{i=1}^{k-1} a^i\left(\frac{\sqrt{1 - 2\cos\left(\frac{2\pi}{n}\right)a + a^2}}{a^{k-n+1}(1+2\pi)}\right). \tag{6}$$

For Equation (6) we resolve the geometric serie part and simplify the expression to the minimization of

$$g_n(a) := \frac{1}{a-1}\left(\frac{\sqrt{1 - 2\cos\left(\frac{2\pi}{n}\right)a + a^2}}{(1+\frac{2\pi}{n})}\right) + \frac{a^{n+1}}{a-1}\left(\frac{\sqrt{1 - 2\cos\left(\frac{2\pi}{n}\right)a + a^2}}{(1+2\pi)}\right). \tag{7}$$

We minimize Equation (6) by numerical means. For any number of rays n a minimal value of $g_n(a)$ gives a lower bound on the sum of two ratios in the original problem. So we can choose n as large as we want. We minimize $g_n(a)$ by numerical means using Maple. For example for $n = 28000000000$ we obtain $a = 1.0000000006809\ldots$ and $g(a) = 6.62521\ldots$ This means that for an arbitrary strategy of the original problem there will always be at least one ratio larger than $\frac{6.6252}{2} = 3.313126$ which finishes the proof. ◀

7 Conclusion

We have introduced a new, simple and intuitive performance measure for the comparison against an online escape path for an unknown environment. The measure outperforms the (few) known ultimate optimal escape paths of convex environments and is also sort of a generalization of a discrete list searching approach by Kirkpatrick.

For a more general class of environments, we presented an online spiral strategy that approximates the measure within an (almost) optimal factor of ≈ 3.318674. Different to classical results the spiral optimizes against two extremes. It was shown that the factor is almost tight by constructing a lower bound that also holds for arbitrary environments. Additionally, one of the very few cases where the optimality of spiral search is verified.

Future work can be done by considering randomization. Additionally, we would like to prove the strong conjecture that the certificate path is indeed always better than the shortest escape path for all environments (even when the optimal escape path is not known).

Acknowledgements. We would like to thank the anonymous referees for their helpful comments and suggestions.

References

1 Steve Alpern and Shmuel Gal. *The Theory of Search Games and Rendezvous*. Kluwer Academic Publications, 2003.

2 R. Baeza-Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Inform. Comput.*, 106:234–252, 1993.

3 Richard Bellman. Minimization problem. *Bull. Amer. Math. Soc.*, 62(3):270, 1956.

4 A. S. Besicovitch. On arcs that cannot be covered by an open equilateral triangle of side 1. *The Mathematical Gazette*, 49(369):pp. 286–288, 1965.

5 P. Coulton and Y. Movshovich. Besicovitch triangles cover unit arcs. *Geometriae Dedicata*, 123(1):79–88, 2006. doi:10.1007/s10711-006-9107-7.

6 Andrea Eubeler, Rudolf Fleischer, Tom Kamphans, Rolf Klein, Elmar Langetepe, and Gerhard Trippen. Competitive online searching for a ray in the plane. In Sándor Fekete, Rudolf Fleischer, Rolf Klein, and Alejandro López-Ortiz, editors, *Robot Navigation*, number 06421 in Dagstuhl Seminar Proceedings, 2006.

7 Amos Fiat and Gerhard Woeginger, editors. *On-line Algorithms: The State of the Art*, volume 1442 of *Lecture Notes Comput. Sci.* Springer-Verlag, 1998.

8 Steven R. Finch. The logarithmic spiral conjecture, 2005.

9 Steven R. Finch and John E. Wetzel. Lost in a forest. *The American Mathematical Monthly*, 111(8):pp. 645–654, 2004.

10 Steven R. Finch and Li-Yan Zhu. Searching for a shoreline. arXiv:math/0501123v1, 2005.

11 Rudolf Fleischer, Tom Kamphans, Rolf Klein, Elmar Langetepe, and Gerhard Trippen. Competitive online approximation of the optimal search ratio. *Siam J. Comput.*, pages 881–898, 2008.

12 S. Gal and D. Chazan. On the optimality of the exponential functions for some minmax problems. *SIAM J. Appl. Math.*, 30:324–348, 1976.

13 Shmuel Gal. *Search Games*, volume 149 of *Mathematics in Science and Engeneering*. Academic Press, New York, 1980.

14 Brian Gluss. The minimax path in a search for a circle in a plane. *Naval Research Logistics Quarterly*, 8(4):357–360, 1961. doi:10.1002/nav.3800080404.

15 Christian Icking, Thomas Kamphans, Rolf Klein, and Elmar Langetepe. On the competitive complexity of navigation tasks. In H. Bunke and et al., editors, *Sensor Based Intelligent Robots*, volume 2238 of *LNCS*, pages 245–258. Springer, 2002.

16 David Kirkpatrick. Hyperbolic dovetailing. In Amos Fiat and Peter Sanders, editors, *Algorithms – ESA 2009*, volume 5757 of *Lecture Notes in Computer Science*, pages 516–527. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-04128-0_46.

17 David Kirkpatrick. Personal communication, 2015. Workshop on Geometric Problems on Sensor Networks and Robots, Supported by NSF grant CCF 1017539, Organized by Peter Brass (CCNY) and Jon Lenchner (IBM Research).

18 Elias Koutsoupias, Christos H. Papadimitriou, and Mihalis Yannakakis. Searching a fixed graph. In *Proc. 23th Internat. Colloq. Automata Lang. Program.*, volume 1099 of *Lecture Notes Comput. Sci.*, pages 280–289. Springer, 1996.

19 Elmar Langetepe. On the optimality of spiral search. In *SODA 2010: Proc. 21st Annu. ACM-SIAM Symp. Disc. Algor.*, pages 1–12, 2010.

20 Elmar Langetepe and David Kübel. Optimal online escape path against a certificate. Technical Report arXiv:1604.05972, University of Bonn, 2016.

21 N. S. V. Rao, S. Kareti, W. Shi, and S. S. Iyengar. Robot navigation in unknown terrains: introductory survey of non-heuristic algorithms. Technical Report ORNL/TM-12410, Oak Ridge National Laboratory, 1993.

22 S. Schuierer. Lower bounds in on-line geometric searching. *Comput. Geom. Theory Appl.*, 18:37–53, 2001.

23 D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

24 Viktor A Zalgaller. How to get out of the woods. *On a problem of Bellman (in Russian), Matematicheskoe Prosveshchenie*, 6:191–195, 1961.

25 Viktor A Zalgaller. A question of Bellman. *Journal of Mathematical Sciences*, 131(1):5286–5306, 2005.

Lagrangian Duality based Algorithms in Online Energy-Efficient Scheduling

Nguyen Kim Thang*

IBISC, Université d'Evry-Val-d'Essonne, Evry, France

Abstract

We study online scheduling problems in the general energy model of speed scaling with power down. The latter is a combination of the two extensively studied energy models, speed scaling and power down, toward a more realistic one. Due to the limits of the current techniques, only few results have been known in the general energy model in contrast to the large literature of the previous ones.

In the paper, we consider a Lagrangian duality based approach to design and analyze algorithms in the general energy model. We show the applicability of the approach to problems which are unlikely to admit a convex relaxation. Specifically, we consider the problem of minimizing energy with a single machine in which jobs arrive online and have to be processed before their deadlines. We present an α^α-competitive algorithm (whose the analysis is tight up to a constant factor) where the energy power function is of typical form $z^\alpha + g$ for constants $\alpha > 2$ and $g \geq 0$. Besides, we also consider the problem of minimizing the weighted flow-time plus energy. We give an $O(\alpha/\ln \alpha)$-competitive algorithm; that matches (up to a constant factor) to the currently best known algorithm for this problem in the restricted model of speed scaling.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Online Scheduling, Energy Minimization, Speed Scaling and Power-down, Lagrangian Duality

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.20

1 Introduction

Energy-efficient algorithms [1] have gained considerable interest in the algorithmic community in the last decade. Many results and techniques have been developed to reduce energy while optimizing some objectives, especially in the domain of scheduling. There are two widely studied models in energy-aware scheduling: *power down* and *speed scaling*. In the power down model, a machine could be set in one of several states, which vary from low-power states to high-power ones and there are transition costs from one state to another. Depending on the required tasks to be performed, an algorithm has to decide when to make a transition and to which states to switch. The goal is to minimize the total energy consumption. In the speed scaling model, there is no state but now one can choose an appropriate speed to process required tasks. The energy power of a machine is a convex function of its speed. An algorithm needs to specify the machine speed and a policy to process jobs in order to optimize some quality of service and the consumed energy. Each model captures partly (but complementarily) a more realistic setting — the speed scaling with power down model. In the latter, a machine could be set in different states and its speed could also be varied as a function of required tasks.

* Research supported by the ANR project OATA n° ANR-15-CE40-0015-01.

A power management scheduling problem in a realistic setting is an *online* problem [1], meaning that at any time the scheduler is not aware of future events and the decision is irrevocable. The standard performance measure of an algorithm is the *competitive ratio*, defined as the worst ratio between the cost of the algorithm and that of the optimal solution. As having been raised in [1], an important direction is to design efficient algorithms (in term of competitive ratio) for online problems in the speed scaling and power down model (general energy model for short). Attempting efficient algorithms for problems in the general energy model, one encounters several limits of current tools. Hence there are only a few works on the model [2, 7, 13] in contrast to a large literature of previous energy models.

Popular tools in online computation are the charging scheme and the potential function method. The idea of the methods is to show that an algorithm behaves well in an amortized sense. However, such methods yield little insight about the nature of problems. Recently, different approaches based on the duality of mathematical programming to design and analyze online scheduling have been proposed [3, 12, 17]. The approaches reveal the nature of such the problems, hence lead to algorithms which are usually simple and competitive [3, 12, 17, 11, 14, 15, 6, 4]. An essential starting point of those approaches (except [17]) is to formulate a linear or convex relaxation for a given problem. However, problems in the general energy model unlikely admit convex program and that represents the main obstacle to design competitive algorithms. In the paper, we show an approach to bypass this difficulty.

1.1 The Model, Approaches and Contribution

Model. We are given a single machine that can be set either in the *sleep state* or in the *active state*. In the active state, the machine can be either in *working state* in which some job is currently processed or in *idle state* in which no job is currently executed. Each transition of the machine from the sleep state to the active one has cost A, which represents the *wake-up* cost. In the sleep state, the energy consumption of the machine is 0. In the active state, the machine can choose an arbitrary speed $s(t)$ to execute jobs. Hence, if $s(t) > 0$ then the machine is in the working state; otherwise if $s(t) = 0$, the machine is idle.

The power energy consumption of the machine at time t in its active state is $P(s(t)) = s(t)^\alpha + g$ where $\alpha > 2$ and $g = P(0) \geq 0$. The consumed energy (without wake-up cost) of the machine is $\int_0^\infty P(s(t))dt$ where the integral is taken during the machine's active periods. We decompose the latter into *dynamic energy* $\int_0^\infty s(t)^\alpha dt$ and *static energy* $\int_0^\infty g dt$ (where again the integrals are taken during active periods). We call the model as the *general energy model*.

Jobs arrive over time and could be processed preemptively, i.e., a job could be interrupted and resumed later. At any time, the scheduler has to determine the state and the machine speed (if it is active) and also a policy to execute jobs. In the paper, we study the following problems.

In the first problem, each job j has a released time r_j, a deadline d_j, a processing volume p_j. A job j has to be fully processed in $[r_j, d_j]$. The objective is to minimize the total energy consumption (the static, dynamic energy and the wake-up cost).

In the second problem, each job j has released time r_j, weight w_j and requires p_j units of processing volume. The flow-time of a job j is $C_j - r_j$ where C_j is the completion time of the job. The objective is to minimize the total weighted flow-time plus the total energy (including the wake-up cost).

Lagrangian Duality Approach

To overcome the issues in the general energy model, we follow our duality approach presented in [17]. The approach has been applied to non-convex relaxations for several problems in [17]. However, such the problems admit linear relaxations with some lost factors[1] and the consideration of non-convex relaxations permits improvement on the competitive ratio. In this paper, we study the approach for non-convex problems, i.e., the problems unlikely admit a convex relaxation with bounded integrality gap. To the best of our knowledge, it is the first time online algorithms have been designed based on non-convex relaxations.

We first briefly summarize the high level idea of the approach in [17]. Given a problem, formulate a relaxation which is *not* necessarily convex and its Lagrangian dual. Next construct dual variables such that the Lagrangian dual has objective value within a desired factor of the primal one (due to some algorithm). Then by the standard Lagrangian weak duality[2] in mathematical programming, the competitive ratio follows. Since the Lagrangian weak duality also holds in the context of calculus of variations, the approach could be applied for the unknowns which are not only variables but also functions.

Let $L(x, \lambda)$ be the Lagrangian function with primal and dual variables x and λ, respectively. Let $\mathcal{X}$ and $\mathcal{Y}$ are feasible sets of x and λ. Intuitively, the approach could be seen as a game between an algorithm and an adversary. The algorithm chooses dual variables $\lambda^* \in \mathcal{Y}$ in such a way that whatever the choice (strategy) of the adversary, the value $\min_{x\in\mathcal{X}} L(x, \lambda^*)$ is always within a desirable factor c of the objective due to the algorithm. We emphasize that $\min_{x\in\mathcal{X}} L(x, \lambda^*)$ is taken over x feasible solutions of the primal.

An advantage of the approach is the flexibility of the formulation. As convexity is not required, we can study a (non-convex) formulation of a given problem. The main core of the approach is to determine the dual variables and to prove the desired competitive ratio. Determining such dual variables is the crucial step in the analysis. Sometimes, the dual variables have intuitive interpretations that inspire their construction.

It is worthy to note that in the analyses (of both problems), we consider mathematical programs in which the machine state variable remains 0-1 (without being relaxed). An advantage of keeping the variables integer, which is allowed due to the flexibility of the approach, is that we can additionally use charging-scheme-liked arguments. Hence, the analyses are carried out by benefiting properties from both mathematical programming (weak duality) and amortized method (charging scheme).

Our Results

1. For the problem of minimizing the total consumed energy, we formulate a natural *non-convex* formulation using the Dirac delta function. The Dirac function is useful to represent the wake-up cost — an issue that causes the problems in the general energy model to be non-convex. We present an α^α-competitive algorithm and the analysis follows the Lagrangian duality approach described above. In the construction of dual variables, a key step of the analysis, we define these variables in such a way that they subtly capture the marginal increase of the energy consumption. Note that the analysis is tight since our algorithm, in the restricted speed scaling model (without static energy and wake-up cost), turns out to be the OA algorithm, which has competitive ratio exactly α^α [10]. Hence, even the competitive ratio has been only slightly improved from $\max\{4, \alpha^\alpha + 2\}$ [13]

[1] factors polynomial in $1/\epsilon$ in the resource augmentation model in which the machine has speed $1 + \epsilon$.
[2] For completeness, the weak duality is given in the appendix.

to α^α, it suggests that the duality-based approach is seemingly a right tool for online scheduling. Besides, the formulation and the analysis give basics to study the second problem.

2. For the problem of minimizing energy plus weighted flow-time, we derive an $O(\alpha/\ln\alpha)$-competitive algorithm that matches the currently best known competitive ratio (up to a constant factor) for the same problem in the restricted speed scaling model (where the wake-up cost and the static energy cost are 0). The dual variables and the analysis are built upon the salient ideas from the ones in the previous problem but in a more involved manner. Informally, the dual solutions are constructed in order to balance the weighted flow-time cost and the energy cost at any moment in the schedule (the same idea of previous algorithms in the speed scaling model). However, in the general energy model this cannot be guaranteed for every moment in the schedule. Hence, we introduce an additional term to dual variables that covers the moments where the two costs are not balanced. Intuitively, the additional term represents the loss due to the non-convexity of the problem. It turns out that the loss in term of competitive ratio is only a constant factor.
   Due to the space constraint, the analysis is partly given. The full version can be found on the website of the author.

1.2 Related work

Anand et al. [3] proposed studying online scheduling by linear (convex) programming and dual fitting. By this approach, they gave simple algorithms and simple analyses with improved performance for problems where the analyses based on potential functions are complex or it is unclear how to design such functions. Gupta et al. [12] gave a primal-dual algorithm for a class of scheduling problems with cost function $f(z) = z^\alpha$. In [17] we generalized the approach in [3] and proposed to study online scheduling by non-convex programming and the weak Lagrangian duality. Using that technique, [17] derive competitive algorithms for problems related to weighted flow-time. The approaches based on duality become more and more popular. Subsequently, many competitive algorithms have been designed for different problems in online scheduling [3, 12, 17, 11, 14, 15, 6].

For the online problem of minimizing the energy consumption in the model of speed scaling, Bansal et al. [10] gave a $2(\frac{\alpha}{\alpha-1})^\alpha e^\alpha$-competitive algorithm. Later on, Bansal et al. [8] showed that no deterministic algorithm has better competitive ratio than e^α/α. In the general energy model, Irani et al. [16] were the first who studied the problem and they derived an algorithm with competitive ratio $(2^{2\alpha-2}\alpha^\alpha + 2^{\alpha-1} + 2)$. Subsequently, Han et al. [13] presented an algorithm which is $\max\{4, \alpha^\alpha + 2\}$-competitive. In offline setting, the problem is recently showed to be NP-hard [2]. Moreover, they [2] also gave a 1.171-approximation algorithm, which improved the 2-approximation algorithm in [16]. If the instances are agreeable then the problem is polynomial [7]. Recently, Antoniadis et al. [5] have given a FPTAS for the problem.

To the best of our knowledge, the objective of minimizing the total weighted flow-time plus energy has not been studied in the speed scaling with power down energy model. However, this objective has been widely studied in speed scaling energy model. Bansal et al. [9] gave an $O(\alpha/\log\alpha)$-competitive algorithm for weighted flow-time plus energy in a single machine where the energy function is s^α. Based on linear programming and dual-fitting, Anand et al. [3] proved an $O(\alpha^2)$-competitive algorithm for unrelated machines. Subsequently, Nguyen [17] and Devanur and Huang [11] presented an $O(\alpha/\log\alpha)$-competitive algorithms for unrelated machines by dual fitting and primal dual approaches, respectively. It turns out

that the different approaches lead to the same algorithm. To the best of our knowledge, no competitive algorithm is known in the general energy model for this problem.

2 Minimizing Energy in Speed Scaling with Power Down Model

In this section, we study the problem of minimizing the total energy. We formulize the problem as a mathematical program. In such a program, we need to incorporate an information about the machine states and the transition cost from the sleep state to the active one. Here we make use of the properties of the Heaviside step function and the Dirac delta function to encode the machine states and the transition cost. Recall that the Heaviside step function $H(t) = 0$ if $t < 0$ and $H(t) = 1$ if $t \geq 0$. Then $H(t)$ is the integral of the Dirac delta function δ (i.e., $H' = \delta$) and it holds that $\int_{-\infty}^{+\infty} \delta(t)dt = 1$. Now let $F(t)$ be a function indicating whether the machine is in active state at time t, i.e., $F(t) = 1$ if the machine is active at t and $F(t) = 0$ if it is in the sleep state. Assume that the machine initially is in the sleep state. Then $A\int_0^{+\infty} |F'(t)|dt$ equals twice the transition cost of the machine (a transition from the active state to the sleep state costs 0 while by the term $A\int_0^{+\infty} |F'(t)|dt$, it costs A).

Let $s_j(t)$ be variable representing the speed of job j at time t. The problem could be formulated as the following (non-convex) program.

$$\min \quad \int_0^{\infty} P\Big(\sum_j s_j(t)\Big)F(t)dt + \frac{A}{2}\int_0^{+\infty} |F'(t)|dt \tag{1}$$

$$\text{subject to} \quad \int_{r_j}^{d_j} s_j(t)F(t)dt \geq p_j \qquad \forall j$$

$$s_j(t) \geq 0,\ F(t) \in \{0,1\} \qquad \forall j,t$$

Observe that each time a job is executed, the machine has to be in the active state. The first constraint ensures that every job j must be fully processed during $[r_j, d_j]$. Note that we do not relax the variable $F(t)$. The objective function consists of corresponding terms to the energy cost during the active periods and the wake-up cost. Recall that $P(z) = z^{\alpha} + g$.

2.1 Algorithm and Dual Variable Construction

Define the *critical* speed $s^c = \arg\min_{s>0} P(s)/s$. It has been observed in [7] that in any algorithm, it would better to set the machine speed at least s^c whenever it executes some job. Let $0 < \beta \leq 1$ be some constant to be chosen later.

Let $s^*(t)$ and $s_j^*(t)$ be the machine speed and the speed of job j at time t by the algorithm, respectively. In the algorithm, we maintain variables, called *virtual* speeds, $s(t)$ and $s_j(t)$. Intuitively, job j would be processed by speed $s_j(t)$ at time t (and the machine would process jobs by speed $s(t)$) *if* the wake-up cost equals A and the parameter $g = 0$. However, it is not the case so the algorithm will process jobs by different speeds but it is the function related to the virtual speeds.

During the execution of the algorithm, we also maintain a set of *active jobs*. Informally, a job is *active* if it has been released but has not been processed by the algorithm. Initially, set auxiliary variables $s(t)$ and $s_j(t)$ equal 0 for every time t and jobs j. If a job is released then it is marked as *active*.

Let τ be the current moment. Set $s(t) \leftarrow s^*(t)$ for every $t \geq \tau$. Consider currently active jobs in the earliest deadline first (EDF) order. (The set of active jobs may include new released job and jobs that have been released before τ but have not been

processed.) For every active job j and $\tau \le t \le d_j$, increase continuously $s_j(t)$ for all $t \in \arg\min_{t'} P'(s(t'))$ and update simultaneously $s(t) \leftarrow s(t) + s_j(t)$ until $\int_{r_j}^{d_j} s_j(t')dt' = p_j$.

Now consider different states of the machine at the current time τ. We distinguish three different states: (1) in *working state* the machine is active and is executing some jobs; (2) in *idle state* the machine is active but its speed equals 0; and (3) in *sleep state* the machine is inactive.

In working state. If $s(\tau) > 0$ then set the machine speed $s^*(t) \leftarrow \max\{s(t), s^c\}$ for $t \ge \tau$ as long as pending jobs exists. Additionally, mark all currently pending jobs as inactive. Otherwise (if $s(\tau) = 0$), switch the machine to the idle state.

In idle state. If $s(\tau) \ge s^c$ then switch to the working state.

If $s^c > s(\tau) > 0$. Do not execute any job; however, mark all currently pending jobs as active. Intuitively, we delay the execution of such jobs until some moment where the machine has to run at speed s^c in order to complete these jobs (assuming that there is no new job released).

Otherwise, if the total duration of idle state from the last wake-up equals A/g then switch to the sleep state.

In sleep state. If $s(\tau) \ge s^c$ then switch to the working state.

Dual variables. Consider a job j and the virtual machine speed $s(t, r_j)$. If $s(t, r_j) > 0$ for every $t \in [r_j, d_j]$, set λ_j such that $\lambda_j p_j/\beta$ equals the marginal increase of the *dynamic* energy due to the arrival of job j. If $s(t, r_j) = 0$ for some moment $t \in [r_j, d_j]$, define λ_j such that $\lambda_j p_j$ equals the marginal increase of the *dynamic and static* energy due to the arrival of job j (assuming no new job will be released later).

2.2 Analysis

The Lagrangian dual of (1) is $\max_{\lambda \ge 0} \min_{s,F} L(s, F, \lambda)$ where the minimum is taken over (s, F) feasible solutions of the primal and L is the following Lagrangian function

$$\begin{aligned} L(s,F,\lambda) &= \int_0^\infty P\Big(\sum_j s_j(t)\Big)F(t)dt + \frac{A}{2}\int_0^{+\infty} |F'(t)|dt + \sum_j \lambda_j \Big(p_j - \int_{r_j}^{d_j} s_j(t)F(t)dt\Big) \\ &\ge \sum_j \lambda_j p_j - \sum_j \int_{r_j}^{d_j} s_j(t)F(t)\Big(\lambda_j - \frac{P(s(t))}{s(t)}\Big)\mathbb{1}_{\{s(t)>0\}}\mathbb{1}_{\{F(t)=1\}}dt + \frac{A}{2}\int_0^{+\infty}|F'(t)|dt \end{aligned} \tag{2}$$

where $s(t) = \sum_j s_j(t)$.

By weak duality, the optimal value of the primal is always larger than the one of the corresponding Lagrangian dual. In the following, with the chosen values of dual variables, we bound the Lagrangian dual value in function of the algorithm cost and show the competitive ratio.

Let $s^*(t, r_j)$ be the machine speed at time $t \ge r_j$ settled by the algorithm at time r_j. In other words, $s^*(t, r_j)$ would be the machine speed at time t if there is no new released job after job j. Similarly, let $s_j^*(t, r_j)$ be the speed of job j at time t settled by the algorithm at time r_j.

▶ **Lemma 1.** *Let j be an arbitrary job.*

1. *If $s^*(t, r_j) > 0$ for every $t \in [r_j, d_j]$ then $\lambda_j \le \beta P'(s^*(t))$ for every $t \in [r_j, d_j]$.*
2. *Moreover, if $s^*(t, r_j) = 0$ for some $t \in [r_j, d_j]$ then $\lambda_j = P(s^c)/s^c$.*

Proof. We prove the first claim. For any time t, speed $s^*(t)$ is non-decreasing as long as new jobs arrive. Hence, it is sufficient to prove the claim assuming that no other job is released after j, i.e., $\lambda_j \leq \beta P'(s^*(t, r_j))$. The marginal increase in the dynamic energy due to the arrival of j could be written as

$$\frac{1}{\beta}\lambda_j p_j = \int_{r_j}^{d_j} \Big[P(s^*(t,r_j)) - P\big(s^*(t,r_j) - s_j^*(t,r_j)\big)\Big]dt \leq \int_{r_j}^{d_j} P'(s^*(t,r_j))s_j^*(t)dt$$

$$= \min_{r_j \leq t \leq d_j} P'(s^*(t,r_j)) \int_{r_j}^{d_j} s_j^*(t,r_j)dt = \min_{r_j \leq t \leq d_j} P'(s^*(t,r_j)) \cdot p_j$$

where $\min P'(s^*(t,r_j))$ is taken over $t \in [r_j, d_j]$ such that $s_j^*(t,r_j) > 0$. The inequality is due to the convexity of P and the second equality follows the algorithm (that increase the speed of job j at $\arg\min P'(s(t))$ for $r_j \leq t \leq d_j$). So the first claim follows.

We are now showing the second claim. By the algorithm, the fact that $s^*(t, r_j) = 0$ for some $t \in [r_j, d_j]$ means that job j will be processed at speed s^c in some interval $[a, b] \subset [r_j, d_j]$ (assuming that no new job is released after r_j). The marginal increase in the energy is $P(s^c)(b-a)$ while p_j could be expressed as $s^c(b-a)$. Therefore, $\lambda_j = P(s^c)/s^c$. ◀

▶ **Theorem 2.** *The algorithm has competitive ratio at most* $\max\{4, \alpha^\alpha\} = \alpha^\alpha$ *for* $\alpha > 2$.

Proof. Let E_1^* be the dynamic energy of the algorithm schedule. Due to the definition of λ_j's and $0 < \beta \leq 1$ we have $E_1^* = \int_0^\infty [P(s^*(t)) - P(0)]dt \leq \sum_j \lambda_j p_j/\beta$.

Let E_2^* be the static energy plus the wake-up energy of the algorithm, i.e., $E_2^* = \int_0^\infty P(0)F^*(t)dt + \frac{A}{2}\int_0^\infty |(F^*)'(t)|dt$ where $F^*(t)$ is the corresponding state (active or sleep) of the machine at time t by the algorithm. We will lower bound the Lagrangian dual objective by E_1^* and E_2^*.

By Lemma 1 (second statement), for every job j such that $s^*(t, r_j) = 0$ for some $t \in [r_j, d_j]$, $\lambda_j = \frac{P(s^c)}{s^c}$. By the definition of the critical speed, $\lambda_j \leq \frac{P(z)}{z}$ for any $z > 0$. Therefore,

$$\sum_j \int_{r_j}^{d_j} s_j(t)F(t)\left(\lambda_j - \frac{P(s(t))}{s(t)}\right)dt \leq 0 \tag{3}$$

where in the sum is taken over jobs j such that $s^*(t, r_j) = 0$ for some $t \in [r_j, d_j]$.

Define

$$L_1(s,\lambda) := \sum_j \lambda_j p_j - \sum_j \int_{r_j}^{d_j} s_j(t)F(t)\left(\lambda_j - \frac{P(s(t))}{s(t)}\right)\mathbb{1}_{\{s(t)>0\}}\mathbb{1}_{\{F(t)=1\}}dt$$

which is the right-hand side of (2) without the wake-up term.

Let $\bar{s}(t) \in \arg\max_z z\beta P'(s^*(t)) - P(z)$. We have

$$\begin{aligned}
&L_1(s,\lambda)\\
&\geq \beta E_1^* - \max_{s,F} \sum_j \int_{r_j}^{d_j} s_j(t)F(t)\left[\beta P'(s^*(t)) - \frac{P(s(t))}{s(t)}\right]\mathbb{1}_{\{s(t)>0\}}\mathbb{1}_{\{F(t)=1\}}\mathbb{1}_{\{s^*(t)>0\}}dt\\
&\geq \beta E_1^* - \max_{s} \int_0^\infty s(t)\left[\beta P'(s^*(t)) - \frac{P(s(t))}{s(t)}\right]\mathbb{1}_{\{s(t)>0\}}\mathbb{1}_{\{F(t)=1\}}\mathbb{1}_{\{s^*(t)>0\}}dt\\
&\geq \beta E_1^* - \int_0^\infty \Big[\beta P'(s^*(t))\bar{s}(t) - P(\bar{s}(t))\Big]\mathbb{1}_{\{s(t)>0\}}\mathbb{1}_{\{F(t)=1\}}\mathbb{1}_{\{s^*(t)>0\}}dt\\
&\geq \beta E_1^* - \frac{1}{2}\int_0^\infty \Big[\beta P'(s^*(t))\bar{s}(t) - P(\bar{s}(t))\Big]\mathbb{1}_{\{s^*(t)>0\}}dt\\
&\qquad - \frac{1}{2}\int_0^\infty \Big[\beta P'(s^*(t))\bar{s}(t) - P(\bar{s}(t))\Big]\mathbb{1}_{\{F(t)=1\}}dt
\end{aligned}$$

where in the first line, the sum is taken over jobs j such that $s^*(t, r_j) > 0$ for all $t \in [r_j, d_j]$. Note that if $s^*(t, r_j) > 0$ then $s^*(t) \geq s^*(t, r_j) > 0$. The first inequality follows (3) and Lemma 1 (first statement). The second inequality holds since $F(t) \in \{0, 1\}$ and $s(t) \geq \sum_j s_j(t)$ where again the sum is taken over jobs j such that $s^*(t, r_j) > 0$ for all $t \in [r_j, d_j]$. The third inequality is due to the first order derivative and $\bar{s}(t)$ maximizes function $z\beta P'(s^*(t)) - P(z)$ (so $\bar{s}(t)$ is the solution of equation $P'(z(t)) = \beta P'(s^*(t))$).

As the energy power function $P(z) = z^\alpha + g$ where $\alpha > 2$ and $g \geq 0$, $\bar{s}(t)^{\alpha-1} = \beta(s^*(t))^{\alpha-1}$. Therefore,

$$\begin{aligned}
L_1(s, \lambda) &\geq \beta E_1^* - \frac{1}{2}\int_0^\infty \Big(\beta\alpha(s^*(t))^{\alpha-1}\bar{s}(t) - (\bar{s}(t))^\alpha - g\Big)\mathbb{1}_{\{s^*(t)>0\}}dt \\
&\qquad - \frac{1}{2}\int_0^\infty \Big(\beta\alpha(s^*(t))^{\alpha-1}\bar{s}(t) - (\bar{s}(t))^\alpha - g\Big)\mathbb{1}_{\{F(t)=1\}}dt \\
&= \beta E_1^* - \int_0^\infty (\alpha-1)\beta^{\alpha/(\alpha-1)}(s^*(t))^\alpha dt + \frac{1}{2}\int_0^\infty g\mathbb{1}_{\{s^*(t)>0\}}dt + \frac{1}{2}\int_0^\infty g\mathbb{1}_{\{F(t)=1\}}dt \\
&= \Big[\beta - (\alpha-1)\beta^{\alpha/(\alpha-1)}\Big]E_1^* + \frac{1}{2}\int_0^\infty g\mathbb{1}_{\{s^*(t)>0\}}dt + \frac{1}{2}\int_0^\infty g\mathbb{1}_{\{F(t)=1\}}dt
\end{aligned}$$

Choose $\beta = 1/\alpha^{\alpha-1}$, we have that

$$L(s, F, \lambda) \geq \frac{1}{\alpha^\alpha}E_1^* + \frac{1}{2}\int_0^\infty g\mathbb{1}_{\{s^*(t)>0\}}dt + \frac{1}{2}\int_0^\infty g\mathbb{1}_{\{F(t)=1\}}dt + \frac{A}{2}\int_0^\infty |F'(t)|dt$$

In order to prove the theorem, we prove the following claim.

▶ **Claim 3.** *Define*

$$L_2(F) := \frac{1}{2}\int_0^\infty g\mathbb{1}_{\{s^*(t)>0\}}dt + \frac{1}{2}\int_0^\infty g\mathbb{1}_{\{F(t)=1\}}dt + \frac{A}{2}\int_0^\infty |F'(t)|dt$$

Then, $L_2(F) \geq E_2^*/4$ for any feasible solution (s, F) of the relaxation.

We first show how to deduce the theorem assuming the claim. By the claim, the dual

$$L(s, F, \lambda) \geq E_1^*/\alpha^\alpha + L_2(F) \geq E_1^*/\alpha^\alpha + E_2^*/4$$

whereas the primal is $E_1^* + E_2^*$. Thus, the competitive ratio is at most $\max\{4, \alpha^\alpha\}$. In the remaining, we prove the claim by amortized arguments.

Proof of Claim. Consider the algorithm schedule. An *end-time* u is a moment in the schedule such that the machine switches from the idle state to the sleep state. Conventionally, the first end-time in the schedule is 0. Partition the time line into phases. A *phase* $[u, v)$ is a time interval such that u, v are two consecutive end-times. Observe that in a phase, the schedule has transition cost A and there is always a new job released in a phase (otherwise the machines would not switch to non-sleep state). We will prove the claim on every phase. In the following, we are interested in phase $[u, v)$ and whenever we mention $L_2(F)$, it refers to $\frac{1}{2}\int_u^v g\mathbb{1}_{\{s^*(t)>0\}}dt + \frac{1}{2}\int_u^v g\mathbb{1}_{\{F(t)=1\}}dt + \frac{A}{2}\int_u^v |F'(t)|dt$.

By the algorithm, the static energy of the schedule during the idle time is A, i.e., $\int_u^v g\mathbb{1}_{\{s^*(t)=0\}}dt = A$. Let (s, F) be an arbitrary feasible primal solution.

If during $[u, v)$, the machine according to the solution (s, F) makes a transition from sleep state to non-sleep state (i.e., $F(t') = 0$ and $F(t'') = 1$ for some $u \leq t' < t'' < v$) or

inversely then

$$\begin{aligned} L_2(F) &\geq \frac{1}{2}\int_u^v g\mathbb{1}_{\{s^*(t)>0\}}dt + \frac{A}{2} \\ &\geq \frac{1}{2}\int_u^v g\mathbb{1}_{\{s^*(t)>0\}}dt + \frac{1}{4}\left(\int_u^v g\mathbb{1}_{\{s^*(t)=0\}}dt + A\right) \geq \frac{1}{4}E_2^*\big|_{[u,v)}. \end{aligned}$$

If during $[u, v)$, the machine following solution (s, F) makes no transition (from non-sleep static to sleep state or inversely) then $F(t) = 1$ during $[u, v)$ in order to process jobs released in the phase. Therefore,

$$\begin{aligned} L_2(F) &\geq \frac{1}{2}\int_u^v g\mathbb{1}_{\{s^*(t)>0\}}dt + \frac{1}{2}\int_u^v g\mathbb{1}_{\{F(t)=1\}}dt = \frac{1}{2}\int_u^v g\mathbb{1}_{\{s^*(t)>0\}}dt + \frac{1}{2}\int_u^v g dt \\ &\geq \frac{1}{2}\int_u^v g\mathbb{1}_{\{s^*(t)>0\}}dt + \frac{1}{4}\int_u^v g\mathbb{1}_{\{s^*(t)=0\}}dt + \frac{A}{4} \\ &\geq \frac{1}{4}\left(\int_u^v g\mathbb{1}_{\{s^*(t)>0\}}dt + \int_u^v g\mathbb{1}_{\{s^*(t)=0\}}dt + A\right) = \frac{1}{4}E_2^*\big|_{[u,v)} \end{aligned}$$

where the second inequality follows the algorithm: as the machine switches to sleep state at time v, it means that the total idle duration in $[u, v)$ incurs a cost A. ◀

The proof of the claim completes the theorem proof. ◀

3 Minimizing Energy plus Weighted Flow-Time in Speed Scaling with Power Down Model

In this section, we study the problem of minimizing total weighted flow-time plus energy. Let $F(t)$ be a function indicating whether the machine i is in active state at time t, i.e., $F(t) = 1$ if the machine is active at t and $F(t) = 0$ if it is in the sleep state. Let $s_j(t)$ be the variable that represents the speed of job j at time t. Let C_j be a variable representing the completion time of j. Similar as the previous section, the problem can be formulized as the following (non-convex) program.

$$\begin{aligned} \min \int_0^\infty 2P\left(\sum_j s_j(t)\right)F(t)dt + 2\sum_j\left(\int_{r_j}^{C_j} s_j(t)F(t)dt\right)\frac{w_j}{p_j}(C_j - r_j) \\ + A\int_0^\infty |F'(t)|dt \qquad (4) \end{aligned}$$

$$\begin{aligned} \text{subject to} \quad & \int_{r_j}^{C_j} s_j(t)F(t)dt = p_j \quad \forall j \\ & s_j(t) \geq 0, \quad F(t) \in \{0, 1\} \quad \forall j, t \end{aligned}$$

The first constraint ensures that every job j must be completed by some moment C_j which is its completion time. In the objective function, the first and second terms represent twice the consumed energy and the total weighted flow-time, respectively. Note that in the second term, $\int_{r_j}^{C_j} s_j(t)F(t)dt = p_j$ by the constraints. The last term stands for twice the transition cost.

Notations. We say that a job j is *pending* at time t if it has not been completed, i.e., $r_j \leq t < C_j$. At time t, denote $q_j(t)$ the *remaining* processing volume of job j. The *total weight* of pending jobs at time t is denoted as $W(t)$. The *density* of a job j is $\delta_j = w_j/p_j$. Recall that the *critical speed* $s^c \in \arg\min_{z\geq 0} P(z)/z$. As $P(z) = z^\alpha + g$, by the first order condition, s^c satisfies $(\alpha - 1)(s^c)^\alpha = g$.

3.1 The Algorithm

We first describe intuitively the ideas of the algorithm. In the speed scaling model, all previous algorithms explicitly or implicitly balance the weighted flow-time of jobs and the consumed energy to process such jobs. That could be done by setting the machine speed at any time t proportional to some function of the total weight of pending jobs (precisely, proportional to $W(t)^{1/\alpha}$ where $W(t)$ is the total weight of pending jobs). Our algorithm follows the idea of balancing the weighted flow-time and the energy. However, in the general energy model, the algorithm would not be competitive if the speed is always set proportionally to $W(t)^{1/\alpha}$ since the static energy might be large due to the long active periods of the machine. Hence, even if the total weight of pending jobs on the machine is small, in some situation the speed is maintained larger than $W(t)^{1/\alpha}$. In fact, it will be set to be the critical speed s^c.

An issue while dealing with the general model is to determine the state of the machine at a given time (active or inactive). If the total weight of pending jobs is small and the machine is active for a long time, then the static energy is large. Otherwise if pending jobs remain for long time then the weighted flow-time is large. The algorithm, together with dual variables, are constructed in order to bypass this difficulty.

Description of algorithm

At any time t, we distinguish different states of the machine: the *working state* (the machine is active and currently processes some job), the *idle state* (the machine is active but currently processes no job) and the *sleep state*. At time t, we (re)compute the total weight of pending jobs and consider different scenarios corresponding to the current machine state.

In working state. If $\alpha W(t)^{(\alpha-1)/\alpha} > P(s^c)/s^c$ then the machine speed is set as $W(t)^{1/\alpha}$. Otherwise, the speed is set as s^c. At any time, the machine processes the highest density job among the pending ones.

In idle state. **1.** If $\alpha W(t)^{(\alpha-1)/\alpha} > P(s^c)/s^c$ then switch to the working state.

2. If $0 < \alpha W(t)^{\frac{\alpha-1}{\alpha}} \leq P(s^c)/s^c$ then make a plan to process pending jobs with speed (exactly) s^c in non-increasing order of their density. The plan consists of a single block (with no idle time) and the block length could be explicitly computed (given the processing volumes of all jobs and speed s^c). Hence, the total consumed energy in the plan can be computed and it is independent of the starting time of the plan.

Choose the starting time of the plan in such a way that the total energy consumption in the plan equals the total weighted flow-time of all jobs in the plan. There always exists such starting time since if the plan begins immediately at the current time, the energy is larger than the weighted flow-time; and inversely if the starting time is large enough, the weighted flow-time dominates the energy consumption.

At the starting time of a plan, switch to the working state. (Note that the plan together with its starting time could be changed due to the arrival of new jobs.)

3. Otherwise, if the total duration of idle state from the last wake-up equals A/g then switch to sleep state.

In sleep state. Use the same policy as the first two steps of the idle state.

3.2 Analysis

The Lagrangian dual of program (4) is $\max \min_{s,C,F} L$ where L is the corresponding Lagrangian function and the maximum is taken over dual variables. We need to choose appropriate dual variables and prove that for any feasible solution (s, C, F) of the primal, the Lagragian dual is bounded by a desired factor from the primal.

Dual variables

Denote the dual variables corresponding to the first constraints of (4) as λ_j's. Set all dual variables (corresponding to the primal (4)) except λ_j's equal 0. The values of dual variables λ_j's is defined as follows.

Fix a job j. At the arrival of a job j, rename pending jobs as $\{1, \ldots, k\}$ in non-increasing order of their densities, i.e., $p_1/w_1 \leq \ldots \leq p_k/w_k$ (note that p_a/w_a is the inverse of job a's density). Denote $W_a = w_a + \ldots + w_k$ for $1 \leq a \leq k$.

Define λ_j such that

$$\lambda_j p_j = w_j \sum_{a=1}^{j} \frac{q_a(r_j)}{W_a^{1/\alpha}} + W_{j+1} \frac{q_j(r_j)}{W_j^{1/\alpha}} + P(s^c) \frac{q_j(r_j)}{s^c} \tag{5}$$

Note that $q_j(r_j) = p_j$. If job j is processed with speed larger than s^c then the first term stands for the weighted flow-time of j and the second term represents an upper bound on the increase of the weighted flow-time of jobs with density smaller than δ_j due to the arrival of j. Observe that as j arrives, the jobs with higher density than δ_j are completed earlier and the ones with smaller density than δ_j may have higher flow-time. Here, the second term in (5) captures the marginal change in the total weighted flow-time. The third term in (5) is introduced in order to cover energy consumption during the execution periods of job j if it is processed by speed s^c. That term is necessary since during such periods the energy consumption and the weighted flow-time are not balanced.

The Lagrangian function $L(s, C, F, \lambda)$ with the chosen dual variables becomes

$$\begin{aligned}
&A \int_0^\infty |F'(t)|dt + 2 \int_0^\infty P\left(\sum_j s_j(t)\right) F(t)dt + 2 \sum_j \delta_j (C_j - r_j) \int_{r_j}^{C_j} s_j(t) F(t) dt \\
&\qquad + \sum_j \lambda_j \left(p_j - \int_{r_j}^{C_j} s_j(t) F(t) dt\right) \\
&= \sum_j \lambda_j p_j + A \int_0^\infty |F'(t)|dt + \sum_j \int_{r_j}^{C_j} \delta_j (C_j - r_j) s_j(t) F(t) dt \\
&\qquad - \sum_j \int_{r_j}^{C_j} s_j(t) F(t) \left(\lambda_j - 2\frac{P(s(t))}{s(t)} - \delta_j (C_j - r_j)\right) dt
\end{aligned}$$

Notations

We denote $s^*(t)$ the machine speed at time t by the algorithm. So by the algorithm, if $s^*(t) > 0$ then $s^*(t) \geq s^c$. Let $\mathcal{E}_1^*$ and $\mathcal{E}_2^*$ be the total dynamic and static energy consumed by the algorithm schedule, respectively. In other words, $\mathcal{E}_1^* = \int_0^\infty (s^*(t))^\alpha dt$ and $\mathcal{E}_2^* = \int_0^\infty g$ where the integral is taken over all moments t where the machine is active (either in working or in idle states). Additionally, let $\mathcal{E}_3^*$ be the total transition cost of the machine. Moreover, let $\mathcal{F}^*$ be the total weighted flow-time due to the algorithm.

We relate the energy cost of the algorithm schedule and the chosen values of dual variables by the following lemma. Note that by definition of λ_j's, we have that $\sum_j \lambda_j p_j \geq \mathcal{F}^*$.

▶ **Lemma 4.** *It holds that* $2\mathcal{E}_1^* + 2\mathcal{E}_2^* \geq \mathcal{F}^*$ *and* $\sum_j \lambda_j p_j \geq \mathcal{E}_1^*$. *Consequently,* $\sum_j \lambda_j p_j \geq \frac{7}{8}\mathcal{E}_1^* + \frac{1}{16}\mathcal{F}^* - \frac{1}{8}\mathcal{E}_2^*$.

The following lemma is crucial in the analysis.

▶ **Lemma 5.** *Let j be an arbitrary job. Then, for every $t \geq r_j$*

$$\lambda_j - \delta_j(t - r_j) \leq \max\left\{\frac{\alpha}{\alpha-1}W(t)^{\frac{\alpha-1}{\alpha}} + \frac{P(s^c)}{s^c}, 2\frac{P(s^c)}{s^c}\right\}$$

▶ **Theorem 6.** *The algorithm is $O(\alpha/\ln\alpha)$-competitive.*

Proof. Recall that the dual has value at least $\min L(s, C, F, \lambda)$ where the minimum is taken over (s, C, F) feasible solution of the primal. The goal is to lower bound the Lagrangian function.

$$L(s,C,F,\lambda) = \sum_j \lambda_j p_j + A\int_0^\infty |F'(t)|dt + \sum_j \int_{r_j}^{C_j} \delta_j(C_j - r_j)s_j(t)F(t)dt$$
$$- \sum_{i,j}\int_{r_j}^{C_j} s_j(t)F(t)\left(\lambda_j - 2\frac{P(s(t))}{s(t)} - \delta_j(C_j - r_j)\right)\mathbb{1}_{\{s(t)>0\}}dt \tag{6}$$

for any feasible primal solution (s, C, F).

Fix a feasible primal solution (s, C, F). Define $L_1(s, C, F, \lambda)$ as

$$\sum_j \int_{r_j}^{C_j} s_j(t)F(t)\left(\lambda_j - 2\frac{P(s(t))}{s(t)} - \delta_j(C_j - r_j)\right)\mathbb{1}_{\{s(t)>0\}}dt$$

▶ **Claim 7.** *Let (s, C, F) be an arbitrary feasible solution of the primal. Then,*

$$L_1(x,s,C,F,\lambda) \leq \frac{1}{(\alpha-1)^{\frac{1}{\alpha-1}}}\mathcal{F}^* - \frac{1}{2}\int_0^\infty g\mathbb{1}_{\{F(t)>0\}}dt - \frac{1}{2}\int_0^\infty g\mathbb{1}_{\{s^*(t)>0\}}dt$$

▶ **Claim 8.** *Let (s, C, F) be an arbitrary feasible solution of the primal. Define*

$$L_2(F) := \sum_j \int_{r_j}^{C_j} \delta_j(C_j - r_j)s_j(t)F(t)dt + A\int_0^\infty |F'(t)|dt$$
$$+ \frac{1}{2}\int_0^\infty g\mathbb{1}_{\{F(t)>0\}}dt + \frac{1}{2}\int_0^\infty g\mathbb{1}_{\{s^*(t)>0\}}dt$$

Then, $L_2(F) \geq (\mathcal{E}_2^* + \mathcal{E}_3^*)/4$.

We first show how to prove the theorem assuming the claims. By (6), we have

$$\begin{aligned}
L(s,C,F,\lambda) &\geq \sum_j \lambda_j p_j + A\int_0^\infty |F'(t)|dt + \sum_j \int_{r_j}^{C_j} \delta_j(C_j - r_j)s_j(t)F(t)dt \\
&\quad - \sum \frac{1}{(\alpha-1)^{1/(\alpha-1)}}\mathcal{F}^* + \frac{1}{2}\int_0^\infty g\mathbb{1}_{\{F(t)>0\}}dt + \frac{1}{2}\int_0^\infty g\mathbb{1}_{\{s^*(t)>0\}}dt \\
&\geq \sum_j \lambda_j p_j - \frac{1}{(\alpha-1)^{1/(\alpha-1)}}\mathcal{F}^* + \frac{1}{4}\mathcal{E}_2^* + \frac{1}{4}\mathcal{E}_3^* \\
&\geq \left(1 - \frac{1}{(\alpha-1)^{1/(\alpha-1)}}\right)\left(\frac{7}{8}\mathcal{E}_1^* + \frac{1}{16}\mathcal{F}^* - \frac{1}{8}\mathcal{E}_2^*\right) + \frac{1}{4}\mathcal{E}_2^* + \frac{1}{4}\mathcal{E}_3^* \\
&\geq \left(1 - \frac{1}{(\alpha-1)^{1/(\alpha-1)}}\right)\left(\frac{7}{8}\mathcal{E}_1^* + \frac{1}{16}\mathcal{F}^*\right) + \frac{1}{8}\mathcal{E}_2^* + \frac{1}{4}\mathcal{E}_3^* \\
&\geq \frac{\ln(\alpha-1)}{2(\alpha-1)}\left(\frac{7}{8}\mathcal{E}_1^* + \frac{1}{16}\mathcal{F}^*\right) + \frac{1}{8}\mathcal{E}_2^* + \frac{1}{4}\mathcal{E}_3^*
\end{aligned}$$

where the first and second inequalities are due to Claim 7 and Claim 8, respectively. The third inequality follows Lemma 4 and $\sum_j \lambda_j p_j \geq \mathcal{F}^*$. The last inequality is due to the fact that $2 \geq (\alpha-1)^{\frac{1}{\alpha-1}} \geq 1 + \frac{\ln(\alpha-1)}{\alpha-1}$ for every $\alpha > 2$.

Besides, the primal objective is at most $2(\mathcal{F}^* + \mathcal{E}_1^* + \mathcal{E}_2^* + \mathcal{E}_3^*)$. Hence, the competitive ratio is $O(\alpha/\ln\alpha)$. ◀

▶ **Theorem 9.** *The algorithm is $O(\alpha/\ln\alpha)$-competitive.*

4 Conclusion

In this paper, we have shown that the Lagrangian duality approach is appropriate to study certain problems which unlikely admit a convex formulation. For many optimization problems, it is challenging to come up with a strong formulation in which the integral constraint of variables is relaxed and the integrality gap is relatively small. The Lagrangian duality approach gives the flexibility to study directly certain problems without relaxing the integrality and without linear/convex formulation. As mentioned earlier and having observed in the analyses, by the approach, one can benefit from techniques in mathematical programming and amortized analysis. It would be interesting to see more work in this direction. For concrete questions, the problems studied in the paper are open for unrelated machine environment. One would expect the existence of algorithms with similar competitive ratio (up to a constant factor).

Acknowledgement. We thank anonymous reviewers for their useful feedbacks that improve the presentation of the paper.

References

1 Susanne Albers. Energy-efficient algorithms. *Commun. ACM*, 53(5):86–96, 2010.
2 Susanne Albers and Antonios Antoniadis. Race to idle: new algorithms for speed scaling with a sleep state. *ACM Transactions on Algorithms (TALG)*, 10(2):9, 2014.
3 S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *Proc. 23rd ACM-SIAM Symposium on Discrete Algorithms*, pages 1228–1241, 2012.

4 Spyros Angelopoulos, Giorgio Lucarelli, and Kim Thang Nguyen. Primal-dual and dual-fitting analysis of online scheduling algorithms for generalized flow time problems. In *Proc. 23rd European Symposium of Algorithms (ESA)*, pages 35–46. Springer, 2015.

5 Antonios Antoniadis, Chien-Chung Huang, and Sebastian Ott. A fully polynomial-time approximation scheme for speed scaling with sleep state. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1102–1113. SIAM, 2015.

6 Yossi Azar, Nikhil R. Devanur, Zhiyi Huang, and Debmalya Panigrahi. Speed scaling in the non-clairvoyant model. In *Proc. 27th ACM on Symposium on Parallelism in Algorithms and Architectures*, pages 133–142, 2015.

7 Evripidis Bampis, Christoph Dürr, Fadi Kacem, and Ioannis Milis. Speed scaling with power down scheduling for agreeable deadlines. *Sustainable Computing: Informatics and Systems*, 2(4):184–189, 2012.

8 Nikhil Bansal, Ho-Leung Chan, Dmitriy Katz, and Kirk Pruhs. Improved bounds for speed scaling in devices obeying the cube-root rule. *Theory of Computing*, 8(1):209–229, 2012.

9 Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. In *Proc. 20th ACM-SIAM Symposium on Discrete Algorithms*, pages 693–701, 2009.

10 Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1), 2007.

11 Nikhil R. Devanur and Zhiyi Huang. Primal dual gives almost optimal energy efficient online algorithms. In *Proc. 25th ACM-SIAM Symposium on Discrete Algorithms*, 2014.

12 Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Online primal-dual for non-linear optimization with applications to speed scaling. In *Proc. 10th Workshop on Approximation and Online Algorithms*, pages 173–186, 2012.

13 Xin Han, Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. Deadline scheduling and power management for speed bounded processors. *Theor. Comput. Sci.*, 411(40-42):3587–3600, 2010.

14 Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive algorithms from competitive equilibria: Non-clairvoyant scheduling under polyhedral constraints. In *STOC*, 2014.

15 Sungjin Im, Janardhan Kulkarni, Kamesh Munagala, and Kirk Pruhs. Selfishmigrate: A scalable algorithm for non-clairvoyantly scheduling heterogeneous processors. In *Proc. 55th IEEE Symposium on Foundations of Computer Science*, 2014.

16 Sandy Irani, Sandeep K. Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Transactions on Algorithms*, 3(4), 2007.

17 Nguyen Kim Thang. Lagrangian duality in online scheduling with resource augmentation and speed scaling. In *Proc. 21st European Symposium on Algorithms*, pages 755–766, 2013.

Online Dominating Set*

Joan Boyar[1], Stephan J. Eidenbenz[2], Lene M. Favrholdt[3], Michal Kotrbčík[4], and Kim S. Larsen[5]

1 **University of Southern Denmark, Odense, Denmark**
joan@imada.sdu.dk
2 **Los Alamos National Laboratory, Los Alamos, USA**
eidenben@lanl.gov
3 **University of Southern Denmark, Odense, Denmark**
lenem@imada.sdu.dk
4 **University of Southern Denmark, Odense, Denmark**
kotrbcik@imada.sdu.dk
5 **University of Southern Denmark, Odense, Denmark**
kslarsen@imada.sdu.dk

Abstract

This paper is devoted to the online dominating set problem and its variants on trees, bipartite, bounded-degree, planar, and general graphs, distinguishing between connected and not necessarily connected graphs. We believe this paper represents the first systematic study of the effect of two limitations of online algorithms: making irrevocable decisions while not knowing the future, and being incremental, i.e., having to maintain solutions to all prefixes of the input. This is quantified through competitive analyses of online algorithms against two optimal algorithms, both knowing the entire input, but only one having to be incremental. We also consider the competitive ratio of the weaker of the two optimal algorithms against the other. In most cases, we obtain tight bounds on the competitive ratios. Our results show that requiring the graphs to be presented in a connected fashion allows the online algorithms to obtain provably better solutions. Furthermore, we get detailed information regarding the significance of the necessary requirement that online algorithms be incremental. In some cases, having to be incremental fully accounts for the online algorithm's disadvantage.

1998 ACM Subject Classification F.1.2 [Modes of Computation] Online Computation, G.2.2 [Graph Theory] Graph Algorithms, I.1.2 [Algorithms] Analysis of Algorithms

Keywords and phrases online algorithms, dominating set, competitive analysis, graph classes, connected graphs

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.21

1 Introduction

We consider online versions of a number of NP-complete graph problems, *dominating set* (DS), and variants hereof. Given an undirected graph $G = (V, E)$ with vertex set V and edge set E, a set $D \subseteq V$ is a *dominating set* for G if for all vertices $u \in V$, either $u \in D$ (containment) or there exists an edge $\{u, v\} \in E$, where $v \in D$ (dominance). The objective is to find a dominating set of minimum cardinality.

In the variant *connected* dominating set (CDS), we add the requirement that D be connected (if G is not connected, D should be connected for each connected component

* Supported in part by the Danish Council for Independent Research and the Villum Foundation.

of G). In the variant *total* dominating set (TDS), every vertex must be dominated by another, corresponding to the definition above with the "containment" option removed. We also consider *independent* dominating set (IDS), where we add the requirement that D be independent, i.e., if $\{u, v\} \in E$, then $\{u, v\} \not\subseteq D$. In both this introduction and the preliminaries section, when we refer to dominating set, the statements are relevant to all the variants unless explicitly specified otherwise.

The study of dominating set and its variants dates back at least to seminal books by König [19], Berge [3], and Ore [21]. The concept of domination readily lends itself to modelling many conceivable practical problems. Indeed, at the onset of the field, Berge [3] mentions a possible application of keeping all points in a network under surveillance by a set of radar stations, and Liu [20] notes that the vertices in a dominating set can be thought of as transmitting stations that can transmit messages to all stations in the network. Several monographs are devoted to domination [14], total domination [15], and connected domination [12], and we refer the reader to these for further details.

We consider *online* [5] versions of these problems. More specifically, we consider the vertex-arrival model where the vertices of the graph arrive one at a time and with each vertex, the edges connecting it to previous vertices are also given. The online algorithm must maintain a dominating set, i.e., after each vertex has arrived, D must be a dominating set for the subgraph given so far. In particular, this means that the first vertex must always be included in the solution, except for the case of total dominating set. Since the graph consisting of a single vertex does not have a total dominating set at all, we allow an online algorithm for TDS to *not* include isolated vertices in the solution, unlike the other variants of DS. If the online algorithm decides to include a vertex in the set D, this decision is irrevocable. Note, however, that not just a new vertex but also vertices given previously may be added to D at any time. An online algorithm must make this decision without any knowledge about possible future vertices.

Defining the nature of the irrevocable decisions is a modelling issue, and one could alternatively have made the decision that also the act of *not* including the new vertex in D should be irrevocable, i.e., not allowing algorithms to include already given vertices in D at a later time. The main reason for our choice of model is that it is much better suited for applications such as routing in wireless networks for which domination is intensively studied; see for instance [10] and the citations thereof. Indeed, when domination models a (costly) establishment of some service, there is no reason why *not* establishing a service at a given time should have any inherent costs or consequences, such as preventing one from doing so later. Furthermore, the stricter variant of irrevocability results in a problem for which it becomes next to impossible for an online algorithm to obtain a non-trivial result in comparison with an optimal offline algorithm. Consider, for example, an instance where the adversary starts by giving a vertex followed by a number of neighbors of that vertex. If the algorithm ever rejects one of these neighbors, the remaining part of the sequence will consist of neighbors of the rejected vertex and the neighbors must all be selected. This shows that, using this model of irrevocability, online algorithms for DS or TDS would have to select at least $n - 1$ vertices, while the optimal offline algorithm selects at most two. For CDS it is even worse, since rejecting any vertex could result in a nonconnected dominating set. A similar observation is made in [18] for this model; their focus is on a different model, where the vertices are known in advance, and all edges incident to a particular vertex are presented when that vertex arrives.

An online algorithm can be seen as having two characteristics: it maintains a feasible solution at any time, and it has no knowledge about future requests. We also define a larger

class of algorithms: An *incremental* algorithm is an algorithm that maintains a feasible solution at any time. It may or may not know the whole input from the beginning.

We analyze the quality of online algorithms for the dominating set problems using *competitive analysis* [22, 16]. Thus, we consider the size of the dominating set an online algorithm computes up against the result obtained by an optimal offline algorithm, Opt.

As something a little unusual in competitive analysis, we are working with two different optimal algorithms. This is with the aim of investigating whether it is predominantly the requirement to maintain feasible solutions or the lack of knowledge of the future which makes the problem hard. Thus, we define Opt^{INC} to be an *optimal incremental* algorithm and Opt^{OFF} to be an *optimal offline* algorithm, i.e., it is given the entire input, and then produces a dominating set for the whole graph. The reason for this distinction is that in order to properly measure the impact of the knowledge of the future, it is necessary that it is the sole difference between the algorithm and Opt. Therefore, Opt has to solve the same problem and hence the restriction on Opt^{INC}. While such an attention to comparing algorithms to an appropriate Opt already exists in the literature, to the best of our knowledge the focus also on the comparison of different optimum algorithms is a novel aspect of our work. Previous results requiring the optimal offline algorithm to solve the same problem as the online algorithm include (1) [7] which considers *fair* algorithms that have to accept a request whenever possible, and thus require Opt to be fair as well, (2) [8] which studies *k-bounded-space* algorithms for bin packing that have at any time at most k open bins and requires Opt to also adhere to this restriction, and (3) [4] which analyzes the performance of online algorithms for a variant of bin packing against a *restricted offline optimum* algorithm that knows the future, but has to process the requests in the same order as the algorithm under consideration.

Given an input sequence I and an algorithm Alg, we let $\text{Alg}(I)$ denote the size of the dominating set computed by Alg on I, and we define Alg to be *c-competitive* if there exists a constant α such that for all input sequences I, $\text{Alg}(I) \leq c\,\text{Opt}(I) + \alpha$, where Opt may be Opt^{INC} or Opt^{OFF}, depending on the context. The (asymptotic) *competitive ratio* of Alg is the infimum over all such c and we denote this $\mathbb{CR}^{\text{INC}}(\text{Alg})$ and $\mathbb{CR}^{\text{OFF}}(\text{Alg})$, respectively. In some results, we use the strict competitive ratio, i.e., the inequality above holds without an additive constant. For these results, when the strict result is linear in n, we write the asymptotic competitive ratio in Table 2 without any additive constant.

We consider the four dominating set problem variants on various graph types, including trees, bipartite, bounded-degree (letting Δ denote the maximum degree), and to some extent planar graphs. In all cases, we also consider the online variant where the adversary is restricted to giving the vertices in such a manner that the graph given at any point in time is connected. In this case, the graph is called *always-connected*. One motivation is that graphs in applications such as routing in networks are most often connected. The connectivity assumption allows us to obtain provably better bounds on the performance of online algorithms, at least compared to Opt^{OFF}, and these bounds are of course more meaningful for the relevant applications.

The results for online algorithms are summarized in Tables 1 and 2. The results for Opt^{INC} against Opt^{OFF} are identical to the results of Table 2, except that for DS on trees, $\mathbb{CR}^{\text{OFF}}(\text{Opt}^{\text{INC}}) = 2$ and for DS on always-connected planar graphs, $\mathbb{CR}^{\text{OFF}}(\text{Opt}^{\text{INC}}) = \lceil n/2 \rceil$. The results are discussed in the conclusion.

Table 1 Bounds on the competitive ratio of any online algorithm with respect to $\textsc{Opt}^{\text{INC}}$.

Graph class	DS	CDS	TDS	IDS
Trees	2	1		1
Bipartite	$[n/4, n/2]$			
Always-connected bipartite	$n/4$			
Bounded degree	$[\frac{\Delta}{2}; \Delta + 1]$	$[\frac{\Delta}{2}; \Delta]$	$[\frac{\Delta}{2}; \Delta]$	
Always-connected bounded degree		$[\frac{\Delta}{2}; \Delta - 1]$		

Table 2 Bounds on the competitive ratio of any online algorithm with respect to $\textsc{Opt}^{\text{OFF}}$.

Graph class	DS	CDS	TDS	IDS
Trees	$[2; 3]$	1	2	n
Bipartite	n		$n/2$	
Always-connected bipartite	$n/2$			
Bounded degree	$[\Delta; \Delta + 1]$	$\Delta + 1$	$[\Delta - 1; \Delta]$	Δ
Always-connected bounded degree	$[\frac{\Delta}{2}; \Delta + 1]$	$[\Delta - 2; \Delta - 1]$		$[\Delta - 1; \Delta]$
Planar	n		$n/2$	n
Always-connected planar				

2 Preliminaries

Since we are studying online problems, the order in which vertices are given is important. Throughout the paper, we will assume that the indices of the vertices of G, $v_1, \ldots, v_n$, indicate the order in which they are given to the online algorithm, and we use $\textsc{Alg}(G)$ to denote the size of the dominating set computed by Alg using this ordering. When no confusion can occur, we implicitly assume that the dominating set being constructed by an online algorithm Alg is denoted by D. We use the phrase *select a vertex* to mean that the vertex in question is added to the dominating set in question. We use G_i to denote the subgraph of G induced by $\{v_1, \ldots, v_i\}$. We let D_i denote the dominating set constructed by Alg after processing the first i vertices of the input. When no confusion can occur, we sometimes implicitly identify a dominating set D and the subgraph it induces. For example, we may say that D *has k components* or *is connected*, meaning that the subgraph of G induced by D has k components or is connected, respectively.

Online algorithms must compute a solution for all prefixes of the input seen by the algorithm. Given the irrevocable decisions, this can of course affect the possible final sizes of a dominating set. When we want to emphasize that a bound is derived under this restriction, we use the word *incremental* to indicate this, i.e., if we discuss the size of an incremental dominating set D of G, this means that $D_1 \subseteq D_2 \subseteq \cdots \subseteq D_n = D$ and that D_i is a dominating set of G_i for each i. Note in particular that any incremental algorithm, including $\textsc{Opt}^{\text{INC}}$, for DS, CDS, or IDS must select the first vertex.

Throughout the text, we use standard graph-theoretic notation. In particular, the *path on n vertices* is denoted P_n. A *star* with n vertices is the complete bipartite graph $K_{1,n-1}$. A *leaf* is a vertex of degree 1, and an *internal* vertex is a vertex of degree at least 2. We use $\mathrm{c}(G)$ to denote the number of components of a graph G. The size of a minimum dominating set of a graph G is denoted by $\gamma(G)$. We use indices to indicate variants, using $\gamma_C(G)$, $\gamma_T(G)$, and $\gamma_I(G)$ for connected, total, and independent dominating set, respectively. This is an

alternative notation for the size computed by $\textsc{Opt}^{\text{OFF}}$. We also use these indices on $\textsc{Opt}^{\text{INC}}$ to indicate which variant is under consideration. We use Δ to denote the maximum degree of the graph under consideration. Similarly, n denotes the number of vertices in the graph.

In many of the proofs of lower bounds on the competitive ratio, when the path, P_n, is considered, either as the entire input or as a subgraph of the input, we assume that it is given in the *standard order*, the order where the first vertex given is a leaf, and each subsequent vertex is a neighbor of the vertex given in the previous step. When the path is a subgraph of the input graph, we often extend this standard order of the path to an *adversarial order* of the input graph – a fixed ordering of the vertices that yields an input attaining the bound.

In some online settings, we are interested in connected graphs, where the vertices are given in an order such that the subgraph induced at any point in time is connected. In this case, we use the term *always-connected*, indicating that we are considering a connected graph G, and all the partial graphs G_i are connected. We implicitly assume that trees are always-connected and we drop the adjective. Since all the classes we consider are hereditary (that is, any induced subgraph also belongs to the class), no further restriction of partial inputs G_i is necessary. In particular, these conventions imply that for trees, the vertex arriving at any step (except the first) is connected to exactly one of the vertices given previously, and since we consider unrooted trees, we can think of that vertex as the parent of the new vertex.

3 The Cost of Being Online

In this section we focus on the comparison of algorithms bound to the same irrevocable decisions. We do so by comparing any online algorithm with $\textsc{Opt}^{\text{INC}}$ and $\textsc{Opt}^{\text{OFF}}$, investigating the role played by the (absence of) knowledge of the future. We start by using the size of a given dominating set to bound the sizes of some connected or incremental equivalents.

▶ **Theorem 1.** *Let G be always-connected, let S be a dominating set of G, and let R be an incremental dominating set of G. Then the following hold:*

1. *There is a connected dominating set S' of G such that $|S'| \leq |S| + 2(\mathrm{c}(S) - 1)$.*
2. *There is an incremental connected dominating set R' of G such that $|R'| \leq |R| + \mathrm{c}(R) - 1$.*
3. *If G is a tree, there is an incremental dominating set R'' of G such that $|R''| \leq |S| + \mathrm{c}(S)$.*

Moreover, all three bounds are tight for infinitely many graphs.

Proof. The proof of 1. can be found in the full version of the paper [6].

To prove 2., we label the components of R in the order in which their first vertices arrive. Thus, let $C_1, \ldots, C_k$ be the components of R, and, for $1 \leq i \leq k$, let v_{j_i} be the first vertex of C_i that arrives. Assume that v_{j_i} arrives before $v_{j_{i+1}}$ for each $i = 1, \ldots, k-1$. We prove that for each component C_i of R, there is a path of length 2 joining v_{j_i} with C_h in G_{j_i} for some $h < i$, i.e., a path with only one vertex not belonging to either component. Let $P = v_{l_1}, \ldots, v_{l_m}, v_{j_i}$ be a shortest path in G_{j_i} connecting v_{j_i} and some component C_h, $h < i$, and assume for the sake of contradiction that $m \geq 3$. In G_{j_i}, the vertex v_{l_3} is not adjacent to a vertex in any component $C_{h'}$, where $h' < i$, since in that case a shorter path would exist. However, since vertices cannot be unselected as the online algorithm proceeds, it follows that in G_{l_3}, v_{l_3} is not dominated by any vertex, which is a contradiction. Thus, selecting just one additional vertex at the arrival of v_{i_j} connects C_i to an earlier component, and the result follows inductively. To see that the bound is tight, observe that the optimal incremental connected dominating set of P_n has $n-1$ vertices, while for even n, there is an incremental dominating set of size $n/2$ with $n/2$ components.

To obtain 3., consider an algorithm $\textsc{Alg}$ processing vertices greedily, while always selecting all vertices from S. That is, v_1 and all vertices of S are always selected, and when a vertex v

not in S arrives, it is selected if and only if it is not dominated by already selected vertices, in which case it is called a *bad* vertex. Clearly, ALG produces an incremental dominating set, R'', of G.

To prove the upper bound on $|R''|$, we gradually mark components of S. For a bad vertex v_i, let v be a vertex from S dominating v_i, and let C be the component of S containing v. Mark C. To prove the claim it suffices to show that each component of S can be marked at most once, since each bad vertex leads to some component of S being marked.

Assume for the sake of contradiction that some component, C, of S is marked twice. This happens because a vertex v of C is adjacent to a bad vertex b, and a vertex v' (not necessarily different from v) of C is adjacent to some later bad vertex b'. Since G is always-connected and b' was bad, b and b' are connected by a path not including v'. Furthermore, v and v' are connected by a path in C. Thus, the edges $\{b, v\}$ and $\{b', v'\}$ imply the existence of a cycle in G, contradicting the fact that it is a tree.

To see that the bound is tight, let $v_1, \ldots, v_m$, $m \equiv 2 \pmod 6$, be a path in the standard order. Let G be obtained from P_m by attaching m pendant vertices (new vertices of degree 1) to each of the vertices $v_2, v_5, v_8, \ldots, v_m$, where the pendant vertices arrive in arbitrary order, though respecting that G should be always-connected. Each minimum incremental dominating set of G contains each of the vertices $v_2, v_5, v_8, \ldots, v_m$, the vertex v_1, and one of the vertices v_{3i} and v_{3i+1} for each i, and thus it has size $2(m+1)/3$. On the other hand, the vertices $v_2, v_5, v_8, \ldots, v_m$ form a dominating set S of G with $\mathrm{c}(S) = (m+1)/3$. ◀

Theorem 1 is best possible in the sense that none of the assumptions can be omitted. Indeed, Proposition 20 implies that it is not even possible to bound the size of an incremental (connected) dominating set in terms of the size of a (connected) dominating set, much less to bound the size of an incremental connected dominating set in terms of the size of a dominating set. Therefore, 1. and 2. in Theorem 1 cannot be combined even on bipartite planar graphs. The situation is different for trees: Corollary 10 1. essentially leverages the fact that any connected dominating set D on a tree can be produced by an incremental algorithm without increasing the size of D.

▶ **Proposition 2.** *For any graph G, there is a unique incremental independent dominating set.*

Proof. We fix G and proceed inductively. The first vertex has to be selected due to the online requirement. When the next vertex, v_{i+1}, is given, if it is dominated by a vertex in D_i, it cannot be selected, since then D_{i+1} would not be independent. If v_{i+1} is not dominated by a vertex in D_i, then v_{i+1} or one of its neighbors must be selected. However, none of v_{i+1}'s neighbors can be selected, since if they were not selected already, then they are dominated, and selecting one of them would violate the independence criteria. Thus, v_{i+1} must be selected. In either case, D_{i+1} is uniquely defined. ◀

Since a correct incremental algorithm is uniquely defined by this proposition by a forced move in every step, OPT^{INC} must behave exactly the same. This fills the column for independent dominating set in Table 1.

We let PARENT denote the following algorithm for trees. The algorithm selects the first vertex. When a new vertex v arrives, if v is not already dominated by a previously arrived vertex, then the parent vertex that v is adjacent to is added to the dominating set. For connected dominating set on trees, PARENT is 1-competitive, even against OPT^{OFF}:

▶ **Proposition 3.** *For any tree T,* Parent(T) *outputs a connected dominating set of T and*

$$\textsc{Parent}(T) = \begin{cases} \gamma_C(T)+1 & \text{if } v_1 \text{ is a leaf of } T \\ \gamma_C(T) & \text{otherwise.} \end{cases}$$

Proof. For trees with at least two vertices, Parent selects the internal vertices plus at most one leaf. Clearly, the size of the minimal connected dominating set of any tree T equals the number of its internal vertices. ◀

To show that for TDS on trees, Parent is 1-competitive against $\textsc{Opt}^{\text{INC}}$, we prove:

▶ **Lemma 4.** *For any incremental total dominating set D for an always-connected graph G, all D_i are connected.*

Proof. For the sake of a contradiction, suppose that for some i, the set D_i induces a subgraph of G with at least two components, and let i be the smallest index with this property. It follows that the vertex v_i constitutes a singleton component of the subgraph induced by D_i. Thus, v_i cannot be dominated by any other vertex of D_i, contradicting that the solution was incremental. ◀

▶ **Corollary 5.** *For any tree T on n vertices,*

$$\textsc{Opt}_T^{\text{INC}}(T) = \textsc{Opt}_C^{\text{INC}}(T) = \begin{cases} \text{int}(T)+1 & \text{if } v_1 \text{ is a leaf of } T \\ \text{int}(T) & \text{otherwise,} \end{cases}$$

where $\text{int}(T)$ *is the number of internal vertices of T. Consequently, when given in the standard order* $\textsc{Opt}_C^{\text{INC}}(P_n) = \textsc{Opt}_T^{\text{INC}}(P_n) = n-1$ *for every* $n \geq 3$.

▶ **Proposition 6.** *For any $n \in \mathbb{Z}^+$ and P_n given in the standard order,* $\textsc{Opt}^{\text{INC}}(P_n) = \lceil n/2 \rceil$.

▶ **Proposition 7.** *For any online algorithm* Alg *for DS and $n > 0$, there is a tree T with n vertices such that the dominating set constructed by* Alg *for T has at least $n-1$ vertices.*

Proof. We prove that the adversary can maintain the invariant that at most one vertex is not included in the solution of Alg. The algorithm has to select the first vertex, so the invariant holds initially. When presenting a new vertex v_i, the adversary checks whether all vertices given so far are included in Alg's solution. If this is the case, v_i is connected to an arbitrary vertex, and the invariant still holds. Otherwise, v_i is connected to the unique vertex not included in D_{i-1}. Now v_i is not dominated, so Alg must select an additional vertex. ◀

▶ **Proposition 8.** *For any always-connected bipartite graph G, the smaller partite set of G (plus, possibly, the vertex v_1) forms an incremental dominating set.*

As a corollary of Proposition 7 and Proposition 8, we get the following result.

▶ **Corollary 9.** *For any online algorithm* Alg *for DS on trees,* $\mathbb{CR}^{\text{INC}}(\textsc{Alg}) \geq 2$.

▶ **Corollary 10.** *For trees, the following hold.*

1. *For DS,* $\mathbb{CR}^{\text{INC}}(\textsc{Parent}) = 2$ *and* $\mathbb{CR}^{\text{OFF}}(\textsc{Parent}) = 3$.
2. *For CDS,* $\mathbb{CR}^{\text{INC}}(\textsc{Parent}) = \mathbb{CR}^{\text{OFF}}(\textsc{Parent}) = 1$.
3. *For TDS,* $\mathbb{CR}^{\text{INC}}(\textsc{Parent}) = 1$ *and* $\mathbb{CR}^{\text{OFF}}(\textsc{Parent}) = 2$.

We extend the Parent algorithm for graphs that are not trees as follows. When a vertex v_i, $i > 1$, arrives, which is not already dominated by one of the previously presented vertices, Parent selects *any* of the neighbors of v_i in G_i.

▶ **Proposition 11.** *For any always-connected graph G, the set computed by* Parent *on G is an incremental connected dominating set of G.*

Proof. We prove the claim by induction on n. Since Parent always selects v_1, the statement holds for $n = 1$. Consider the graph G_i, for some $i > 1$, and assume that D_{i-1} is an incremental connected dominating set of G_{i-1}. If v_i is already dominated by a vertex in D_{i-1}, then Parent keeps D unchanged (that is, $D_i = D_{i-1}$) and thus D_i is an incremental connected dominating set of G_i. If v_i is not dominated by D_{i-1}, then Parent chooses a neighbor v of v_i in G_{i-1}. Clearly, this implies that D_i is an incremental dominating set of G_i. Since D_{i-1} is an incremental connected dominating set of G_{i-1} and the vertex v is adjacent to the only component of D_{i-1}, D_i is connected, which concludes the proof. ◀

▶ **Proposition 12.** *For DS and CDS on always-connected bipartite graphs, $\mathbb{CR}^{\text{OFF}}(\text{Parent}) \leq n/2$.*

▶ **Proposition 13.** *Let G be a graph with n vertices and maximum degree Δ. For any graph G, $\gamma_C(G) \geq \gamma(G) \geq n/(\Delta+1)$ and $\gamma_T(G) \geq n/\Delta$.*

Proposition 13 implies that any algorithm computing an incremental dominating set is no worse than $(\Delta+1)$-competitive.

▶ **Corollary 14.** *For any algorithm* Alg *for DS, $\mathbb{CR}^{\text{OFF}}(\text{Alg}) \leq \Delta + 1$. Furthermore, for any algorithm* Alg *for TDS, $\mathbb{CR}^{\text{OFF}}(\text{Alg}) \leq \Delta$.*

▶ **Proposition 15.** *For any algorithm* Alg *for CDS, $\mathbb{CR}^{\text{OFF}}(\text{Alg}) \leq \Delta - 1$.*

In the next result and in Proposition 19 in Section 4 we use *layers* in an always-connected graph G defined by letting L assign layer numbers to vertices in the following manner. Let $L(v_1) = 0$ and for $i > 1$, $L(v_i) = 1 + \min\{L(v_j) \mid v_j \text{ is a neighbor of } v_i \text{ in } G_i\}$.

Our next aim is to show that for always-connected bipartite graphs, there is an $n/4$-competitive algorithm against Opt^{INC}. This is achieved by considering the following *first parent* algorithm, denoted FirstParent, which generalizes Parent. For DS and CDS, the algorithm FirstParent always selects v_1 and for each vertex v_i, $i > 1$, if v_i is not dominated by one of the already selected vertices, it selects a neighbor of v_i with the smallest layer number. For TDS, we add the following to FirstParent, so that the dominating set produced is total: If, when v_i arrives, v_i and v_j $(j < i)$ are the only vertices of a component of size 2, then besides v_j, FirstParent also selects v_i.

▶ **Theorem 16.** *For DS, CDS, and TDS on always-connected bipartite graphs, we have $\mathbb{CR}^{\text{INC}}(\text{FirstParent}) \leq n/4$ for $n \geq 4$.*

Proof. We consider DS and CDS first. Since FirstParent is an instantiation of Parent, Proposition 11 implies that the incremental dominating set constructed by FirstParent is connected. Therefore, the fact that for any graph G with at least three vertices $\text{Opt}^{\text{INC}}(G) \leq \text{Opt}_T^{\text{INC}}(G) \leq \text{Opt}_C^{\text{INC}}(G) + 1$ implies that it is sufficient to prove that FirstParent is $n/4$-competitive against Opt^{INC}. Furthermore, we only need to consider the case $\text{Opt}^{\text{INC}}(G) < 4$, since otherwise FirstParent is trivially $n/4$-competitive. Since G is bipartite, there are no edges between vertices of a single layer. Our first aim is to bound the number of layers.
Claim: If $\text{Opt}^{\text{INC}}(G) < 4$, then G has at most 6 layers.

To establish the claim, we prove that if an always-connected graph G has $2k+1$ layers, then $\text{Opt}^{\text{INC}}(G) > k$. For the sake of contradiction, suppose that there exist graphs G that are always-connected with $2k+1$ layers such that $\text{Opt}^{\text{INC}}(G) \leq k$, and among all such graphs

choose one, G, with the smallest number of vertices. Since any dominating set contains at least one vertex, we have $k \geq 1$. Let D be an incremental dominating set of G with $|D| \leq k$ and let l be the largest integer such that G_l has $2k-1$ layers. Since G is the smallest counterexample, we have $\textsc{Opt}^{\text{INC}}(G_l) \geq k$. Recall that D_l is defined as $D \cap G_l$. The fact that D is an incremental dominating set implies that D_l is a dominating set of G_l. We claim that $|D_l| = k$, since otherwise D_l would be an incremental dominating set of G_l with $|D_l| < k$, contradicting the fact that $\textsc{Opt}^{\text{INC}}(G_l) \geq k$. The fact that $|D_l| = k$ is equivalent to $D \subseteq V(G_l)$ and, in particular, $L(v) \leq 2k-1$ for each vertex v from D. Let w be a vertex of G such that $L(w) = 2k+1$, such a vertex exists since G has $2k+1$ layers. By the definition of layers the vertex w does not have a neighbor in any of the first $2k-1$ layers and thus is not adjacent to any vertex of D, contradicting the fact that D is a dominating set of G. This concludes the proof of the claim.

In the rest of the proof, we distinguish several cases according to the number of layers of G. If there are at most two layers, then FirstParent selects only the root v_1 and the result easily follows. Let l_i denote the size of the i-th layer and s_i the number of vertices selected by FirstParent from the i-th layer. For convenience, we will ignore the terms s_0 and l_0, both of which are one, which is viable since we are dealing with the asymptotic competitive ratio. Because FirstParent can add a vertex from the i-th layer to the dominating set only when a (non-dominated) vertex from the $(i+1)$-st layer arrives, we have

$$s_i \leq l_{i+1}. \tag{Ai}$$

Clearly,

$$s_i \leq l_i. \tag{Bi}$$

The letter i in equations (A) and (B) indicates the layer for which the equation is applied. If there are precisely three layers, then $\textsc{Opt}^{\text{INC}}(G) \geq 2$ and we must prove that $s_1 + s_2 \leq n/2$. However, $s_2 = 0$, and $s_1/2 \leq l_1/2$ by $(B1)$ and $s_1/2 \leq l_2/2$ by $(A1)$. Adding the last two inequalities yields $s_1 \leq l_1/2 + l_2/2 = n/2$, as required.

We use the same idea as for three layers also in the cases of four and five layers, albeit the counting is slightly more complicated. First we deal separately with the case where $\textsc{Opt}^{\text{INC}}(G) = 2$, and, consequently, there are four layers. Note that the two vertices in the optimal solution are necessarily in layers 0 and 2, and it follows that $l_2 = 1$. Furthermore, $(A1)$ implies that $s_1 \leq 1$ and $(B2)$ implies that $s_2 \leq 1$. Since $s_3 = 0$, FirstParent always selects at most 3 vertices, which yields the desired result. Assume now that $\textsc{Opt}^{\text{INC}}(G) \geq 3$ and therefore, our aim is to prove that $\textsc{FirstParent}(G) \leq 3n/4$. Adding 1/4 times $(A1)$, 3/4 times $(B1)$, 1/2 times $(A2)$, and 1/2 times $(B2)$ yields

$$s_1 + s_2 \leq 3l_1/4 + 3l_2/4 + l_3/2. \tag{1}$$

If there are four layers, then $s_3 = 0$ and the right-hand side of (1) satisfies $3l_1/4+3l_2/4+l_3/2 \leq 3(l_1+l_2+l_3)/4 = 3n/4$, which yields the desired result. If there are five layers, we add 3/4 times $(A3)$ and 1/4 times $(B3)$ to (1), which gives $s_1+s_2+s_3 \leq 3(l_1+l_2+l_3+l_4)/4 = 3n/4$, as required. The last remaining case is that of six layers and $\textsc{Opt}^{\text{INC}}(G) = 3$, which is dealt with similarly to that of four layers and $\textsc{Opt}^{\text{INC}}(G) = 2$. In particular, the vertices selected by $\textsc{Opt}^{\text{INC}}$ necessarily lie in layers 0, 2, and 4, and thus $l_0 = l_2 = l_4 = 1$. Now observing that $s_5 = 0$ and adding (Bi) for all even i to (Ai) for $i = 1$ and $i = 3$ yields that $\textsc{FirstParent}(G) \leq 5$, which implies the result in the always-connected case.

For TDS, the additional vertices accepted by FirstParent must by accepted by any incremental online algorithm, so the result also holds for TDS. ◀

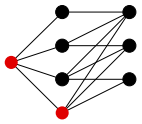


Figure 1 A two-layer construction; the minimum connected dominating set is depicted in red (Proposition 18).

▶ **Proposition 17.** *For DS, CDS, and TDS, we have* $\mathbb{CR}^{\text{INC}}(\textsc{FirstParent}) \leq n/2$ *for* $n \geq 2$.

Proof. Since for any graph, FirstParent constructs an incremental dominating set, we need to consider only the cases where $\textsc{Opt}^{\text{INC}}(G) \leq 1$, $\textsc{Opt}_C^{\text{INC}}(G) \leq 1$, and $\textsc{Opt}_T^{\text{INC}}(G) \leq 1$. For TDS, either G has no edges, in which case the empty set of vertices is a feasible solution constructed both by $\textsc{Opt}_T^{\text{INC}}$ and FirstParent, or G contains an edge, in which case $\textsc{Opt}_T^{\text{INC}}(G) \geq 2$ and the bound follows. Since $\textsc{Opt}^{\text{INC}}(G) \leq \textsc{Opt}_C^{\text{INC}}(G)$, it is sufficient to consider the case where $\textsc{Opt}^{\text{INC}}(G) = 1$. If, at any point, G_i has more than one component, then $\textsc{Opt}^{\text{INC}}(G_i) \geq 2$. Thus, if $\textsc{Opt}^{\text{INC}}(G_i) = 1$, G is a star and is always-connected. Thus, the center vertex must arrive as either the first or second request, so $\textsc{FirstParent}(G) \leq 2 \leq n$. ◀

▶ **Proposition 18.** *For any online algorithm* Alg *for DS, CDS, or TDS on always-connected bipartite graphs,* $\mathbb{CR}^{\text{INC}}(\textsc{Alg}) \geq n/4$ *and* $\mathbb{CR}^{\text{INC}}(\textsc{Alg}) \geq \Delta/2$.

Proof. We prove that for any online algorithm Alg for DS, CDS, or TDS and for any integer $\Delta \geq 2$, there is a bipartite graph G with maximum degree Δ such that $\textsc{Alg}(G) = \Delta \geq n/2$ and $\textsc{Opt}^{\text{INC}}(G) = \textsc{Opt}_C^{\text{INC}}(G) = \textsc{Opt}_T^{\text{INC}}(G) = 2$. Consider the graph consisting of a root v, Δ vertices $u_1, \ldots, u_\Delta$ adjacent to the root and constituting the first layer, and an additional $\Delta - 1$ vertices $w_1, \ldots, w_{\Delta-1}$, which will be given in that order, constituting the second layer, with adjacencies as follows: For $i = 1, \ldots, \Delta - 1$, the i-th vertex w_i of the second layer is adjacent to $\Delta - i + 1$ vertices of the first layer in such a way that we obtain the following strict set containment of sets of neighbors of these vertices: $N(w_i) \supset N(w_{i+1})$ for all $i = 1, \ldots, \Delta - 2$. An example of this construction for $\Delta = 4$ is depicted in Figure 1. After the entire first layer is presented to the algorithm, the vertices of the first layer are indistinguishable to the algorithm and $D_{\Delta+1}$ does not necessarily contain more than one vertex. For each $i = 1, \ldots, \Delta - 1$, the neighbors of w_i are chosen from the first layer in such a way that $N(w_{i-1}) \supset N(w_i)$, the degree of w_i is $\Delta - i + 1$, and $N(w_i)$ contains as many vertices not contained in the dominating set constructed by Alg so far as possible. Consider the situation when the vertex w_i arrives. It is easy to see that if the set $N(w_i)$ does not contain a vertex from the dominating set constructed so far, then Alg must select at least one additional vertex at this time. The last observation implies that Alg selects at least $\Delta - 1$ vertices from the first and second layer, plus the root.

Since there is a vertex u in the first layer that is adjacent to all vertices in the second layer, $\{u, v\}$ is an incremental connected dominating set of G, which concludes the proof. ◀

4 The Cost of Being Incremental

This section is devoted to comparing the performance of incremental algorithms and $\textsc{Opt}^{\text{OFF}}$. Since $\textsc{Opt}^{\text{OFF}}$ performs at least as well as $\textsc{Opt}^{\text{INC}}$ and $\textsc{Opt}^{\text{INC}}$ performs at least as well as any online algorithm, each lower bound in Table 2 is at least the maximum of the corresponding

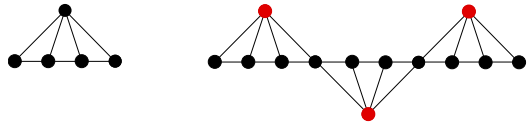


Figure 2 A fan with $\Delta = 4$ (left; Proposition 24) and an alternating fan with $k = 3$ and $\Delta = 4$ (right; Proposition 25).

lower bound in Table 1 and the corresponding lower bound for $\mathbb{CR}^{\text{OFF}}(\text{OPT}^{\text{INC}})$. Similarly, each upper bound in Table 1 and corresponding upper bound for $\mathbb{CR}^{\text{OFF}}(\text{OPT}^{\text{INC}})$ is at least the corresponding upper bound in Table 2. In both cases, we mention only bounds that cannot be obtained in this way from cases considered already.

The following result generalizes the idea of Proposition 8.

▶ **Proposition 19.** *For DS on always-connected graphs,* $\mathbb{CR}^{\text{OFF}}(\text{OPT}^{\text{INC}}) \leq n/2$.

Proof. For a fixed ordering of G, consider the layers $L(v)$ assigned to vertices of G. It is easy to see that the set of vertices in the even layers is an incremental solution for DS and similarly for the set of vertices in odd layers plus the vertex v_1. Therefore, OPT^{INC} can select the smaller of these two sets, which necessarily has at most $n/2$ vertices. ◀

▶ **Proposition 20.** *The following hold for the strict competitive ratio:*

- *For DS on bipartite planar graphs,* $\mathbb{CR}^{\text{OFF}}(\text{OPT}^{\text{INC}}) \geq n - 1$ *and* $\mathbb{CR}^{\text{OFF}}(\text{OPT}^{\text{INC}}) \geq \Delta$.
- *For CDS on bipartite planar graphs,* $\mathbb{CR}^{\text{OFF}}(\text{OPT}^{\text{INC}}) \geq n$.

▶ **Proposition 21.** *For IDS and for the strict competitive ratio,* $\mathbb{CR}^{\text{OFF}}(\text{OPT}^{\text{INC}}) \geq \Delta$ *and* $\mathbb{CR}^{\text{OFF}}(\text{OPT}^{\text{INC}}) \geq n - 1$.

▶ **Proposition 22.** *For IDS on always-connected graphs,* $\Delta - 1 \leq \mathbb{CR}^{\text{OFF}}(\text{OPT}^{\text{INC}}) \leq \Delta$.

Theorem 1 3. implies the following bound on the performance of OPT^{INC} on trees.

▶ **Corollary 23.** *For DS on trees,* $\mathbb{CR}^{\text{OFF}}(\text{OPT}^{\text{INC}}) \leq 2$.

All of the following results are lower bounds. Specific examples of the families of graphs used to obtain these lower bounds are depicted in the following figures; the details of the proofs appear in the full paper [6].

A *fan* of degree Δ is the graph obtained from a path P_Δ by addition of a vertex v that is adjacent to all vertices of the path, as in Figure 2. The adversarial order of a fan is defined by the standard order of the underlying path, followed by the vertex v.

▶ **Proposition 24.** *For always-connected planar graphs (and, thus, also on general planar graphs), the following strict competitive ratio results hold.*

- *For DS,* $\mathbb{CR}^{\text{OFF}}(\text{OPT}^{\text{INC}}) \geq n/2$.
- *For CDS,* $\mathbb{CR}^{\text{OFF}}(\text{OPT}^{\text{INC}}) \geq n - 2$.
- *For TDS,* $\mathbb{CR}^{\text{OFF}}(\text{OPT}^{\text{INC}}) \geq n/2 - 1$.

An *alternating fan* with k fans of degree Δ consists of k copies of the fan of degree Δ, where the individual copies are joined in a path-like manner by identifying some of the vertices of degree 2, as in Figure 2. Thus, $n = k(\Delta + 1) - (k - 1)$ and $k = (n - 1)/\Delta$. The adversarial order of an alternating fan is defined by the concatenation of the adversarial orders of the underlying fans.

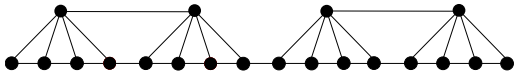


Figure 3 A modular bridge with $k = 4$ and $\Delta = 5$ (Proposition 26).

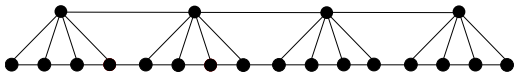


Figure 4 A bridge with $k = 4$ and $\Delta = 6$ (Proposition 27).

▶ **Proposition 25.** *For DS on always-connected graphs,* $\mathbb{CR}^{\text{OFF}}(\text{OPT}^{\text{INC}}) \geq (\Delta - 1)/2$.

A *modular bridge of degree* Δ *with* k *sections*, where k is even, is the graph obtained from a path on $k(\Delta - 1)$ vertices, with an additional k *chord* vertices. There is a perfect matching on the chord vertices $u_1, \ldots, u_k$ with u_{2i} is adjacent to u_{2i-1} for all $i = 1, \ldots, k/2$. Furthermore, the i-th chord vertex is adjacent to the vertices of the i-th *section*; see Figure 3 for an example. The adversarial order of a modular bridge is defined by the standard order of the path, followed by the chord vertices in any order.

▶ **Proposition 26.** *For TDS on always-connected graphs,* $\mathbb{CR}^{\text{OFF}}(\text{OPT}^{\text{INC}}) \geq \Delta - 1$.

A *bridge of degree* Δ *with* k *sections* is obtained from a modular bridge of degree $\Delta - 1$ with k sections by joining vertices u_{2i} and u_{2i+1} by an edge for each $i = 1, \ldots, k/2 - 1$; see Figure 4 for an example. The adversarial order of a bridge is identical with the adversarial order of the underlying modular bridge.

▶ **Proposition 27.** *For CDS on always-connected graphs,* $\mathbb{CR}^{\text{OFF}}(\text{OPT}^{\text{INC}}) \geq \Delta - 2$.

A *rotor of degree* Δ, where $\Delta \geq 2$ is even, is a graph obtained from a star, $K_{1,\Delta}$, on $\Delta + 1$ vertices by adding the edges of a perfect matching on the pendant vertices, as in Figure 5. The adversarial order of a rotor G of degree Δ is any fixed order such that G_{2i} is a graph with a perfect matching for each $i = 1, \ldots, \Delta/2$ and the central vertex of the original star is the last vertex to arrive.

▶ **Proposition 28.** *For CDS,* $\mathbb{CR}^{\text{OFF}}(\text{OPT}^{\text{INC}}) \geq \Delta + 1$*, and for TDS,* $\mathbb{CR}^{\text{OFF}}(\text{OPT}^{\text{INC}}) \geq \Delta/2$.

For any $n \geq 2$, the *two-sided fan of size* n is the graph obtained from a path on $n - 2$ vertices by attaching two additional vertices, one to the even-numbered vertices of the path and the other to the odd-numbered vertices of the path. The adversarial order of a two-sided fan is defined by the standard order of the path, followed by the two additional vertices. See Figure 5 for an illustration of a two-sided fan of size 10.

▶ **Proposition 29.** *For any incremental algorithm* ALG *for CDS or TDS on always-connected bipartite graphs,* $\mathbb{CR}^{\text{OFF}}(\text{ALG}) \geq (n - 3)/2$ *holds for the strict competitive ratio.*

5 Conclusion and Open Problems

Online algorithms for four variants of the dominating set problem are compared using competitive analysis to OPT^{INC} and OPT^{OFF}, two reasonable alternatives for the optimal algorithm having knowledge of the entire input. Several graph classes are considered, and tight results are obtained in most cases.

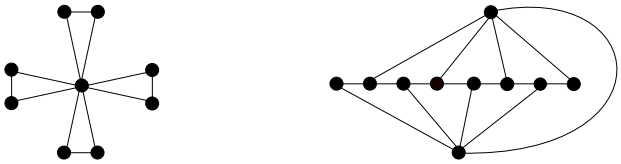


Figure 5 The rotor of degree 8 (left, Proposition 28) and two-sided fan of size 10 (right, Proposition 29).

The difference between OPT^{INC} and OPT^{OFF} is that OPT^{INC} is required to maintain an incremental solution (as any online algorithm), while OPT^{OFF} is only required to produce an offline solution for the final graph. The algorithms are compared to both OPT^{INC} and OPT^{OFF}, and OPT^{INC} is compared to OPT^{OFF}, in order to investigate why all algorithms tend to perform poorly against OPT^{OFF}. Is this due to the requirement to be incremental, or is it because of the lack of knowledge of the future?

Inspecting the results in the tables, perhaps the most striking conclusion is that the competitive ratios of any online algorithm and OPT^{INC}, respectively, against OPT^{OFF}, are almost identical. This indicates that the requirement to maintain an incremental dominating set is a severe restriction, which can be offset by the full knowledge of the input only to a very small extent. On the other hand, when we restrict our attention to online algorithms against OPT^{INC}, it turns out that the handicap of not knowing the future still presents a barrier, leading to competitive ratios of the order of n or Δ in most cases.

One could reconsider the nature of the irrevocable decisions, which originally stemmed from practical applications. Which assumptions on irrevocability are relevant for practical applications, and which irrevocability components make the problem hard from an online perspective? We expect that these considerations will apply to many other online problems as well.

There is relatively little difference observed between three of the variants of dominating set considered: dominating set, connected dominating set, and total dominating set. In fact, the results for total dominating set generally followed directly from those for connected dominating set as a consequence of Lemma 4. The results for independent dominating set were significantly different from the others. It can be viewed as the minimum maximal independent set problem since any maximal independent set is a dominating set. This problem has been studied in the context of investigating the performance of the greedy algorithm for the independent set problem. In fact, the unique incremental independent dominating set is the set produced by the greedy algorithm for independent set.

In yet another orthogonal dimension, we compare the results for various graph classes. Dominating set is a special case of set cover and is notoriously difficult in classical complexity, being NP-hard [17], $W[2]$-hard [11], and not approximable within $c \log n$ for any constant c on general graphs [13]. On the positive side, on planar graphs, the problem is FPT [1], admits a PTAS [2], and is approximable within $\log \Delta$ on bounded degree graphs [9]. On the other hand, the relationship between the performance of online algorithms and structural properties of graphs is not particularly well understood. In particular, there are problems where the absence of knowledge of the future is irrelevant; examples of such problems in this work are CDS and TDS on trees, and IDS on any graph class. As expected, for bounded degree graphs, the competitive ratios are of the order of Δ, but closing the gap between $\Delta/2$ and Δ seems to require additional ideas. On the other hand, for planar graphs, the problem, rather surprisingly, seems to be as difficult as the general case when compared to OPT^{OFF}.

When online algorithms for planar graphs are compared to OPT^{INC}, we suspect there might be an algorithm with constant competitive ratio. At the same time, this case is the most notable open problem directly related to our results. Drawing inspiration from classical complexity, one could consider more specific graph classes in the quest for understanding exactly what structural properties make the problem solvable. From this perspective, our consideration of planar, bipartite, and bounded degree graphs is a natural first step.

Acknowledgment. The authors would like to thank the anonymous referees for constructive comments.

References

1 J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier. Fixed parameter algorithms for dominating set and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002.

2 B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994.

3 C. Berge. *Theory of Graphs and its Applications*. Meuthen, London, 1962.

4 M. Böhm, J. Sgall, and P. Veselý. Online colored bin packing. In E. Bampis and O. Svensson, editors, *12th International Workshop on Approximation and Online Algorithms (WAOA)*, volume 8952 of *Lecture Notes in Computer Science*, pages 35–46. Springer, 2015.

5 A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

6 J. Boyar, S. J. Eidenbenz, L. M. Favrholdt, M. Kotrbčík, and K. S. Larsen. Online dominating set. Technical Report arXiv:1604.05172 [cs.DS], arXiv, 2016.

7 J. Boyar and K. S. Larsen. The seat reservation problem. *Algorithmica*, 25(4):403–417, 1999.

8 M. Chrobak, J. Sgall, and G. J. Woeginger. Two-bounded-space bin packing revisited. In C. Demetrescu and M. M. Halldórsson, editors, *19th Annual European Symposium (ESA)*, volume 6942 of *Lecture Notes in Computer Science*, pages 263–274. Springer, 2011.

9 V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

10 B. Das and V. Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *IEEE International Conference on Communications (ICC)*, volume 1, pages 376–380, 1997.

11 R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

12 D.-Z. Du and P.-J. Wan. *Connected Dominating Set: Theory and Applications*. Springer, New York, 2013.

13 U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

14 T. W. Haynes, S. Hedetniemi, and P. Slater. *Fundamentals of Domination in Graphs*. Marcel Dekker, New York, 1998.

15 M. Henning and A. Yao. *Total Domination in Graphs*. Springer, New York, 2013.

16 A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:79–119, 1988.

17 R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

18 G.-H. King and W.-G. Tzeng. On-line algorithms for the dominating set problem. *Information Processing Letters*, 61(1):11–14, 1997.

19 D. König. *Theorie der Endlichen und Unendlichen Graphen*. Chelsea, New York, 1950.

20 C. L. Liu. *Introduction to Combinatorial Mathematics*. McGraw-Hill, New York, 1968.

21 O. Ore. *Theory of Graphs*, volume 38 of *Colloquium Publications*. American Mathematical Society, Providence, 1962.

22 D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

Sorting Under Forbidden Comparisons

Indranil Banerjee[1] and Dana Richards[2]

1 **Department Of Computer Science, George Mason University, Fairfax, USA**
`ibanerje@gmu.edu`
2 **Department Of Computer Science, George Mason University, Fairfax, USA**
`richards@cs.gmu.edu`

Abstract

In this paper we study the problem of sorting under forbidden comparisons where some pairs of elements may not be compared (forbidden pairs). Along with the set of elements V the input to our problem is a graph $G(V, E)$, whose edges represents the pairs that we can compare in constant time. Given a graph with n vertices and $m = \binom{n}{2} - q$ edges we propose the first non-trivial deterministic algorithm which makes $O((q+n)\log n)$ comparisons with a total complexity of $O(n^2 + q^{\omega/2})$, where ω is the exponent in the complexity of matrix multiplication. We also propose a simple randomized algorithm for the problem which makes $\widetilde{O}(n^2/\sqrt{q+n}+n\sqrt{q})$ probes with high probability. When the input graph is random we show that $\widetilde{O}(\min(n^{3/2}, pn^2))$ probes suffice, where p is the edge probability.

1998 ACM Subject Classification F.2.2 Sorting and searching

Keywords and phrases Sorting, Random Graphs, Complexity

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.22

1 Introduction

Comparison based sorting algorithms is one of the most studied areas in theoretical computer science. The majority of the efforts have been focused on the uniform comparison cost model. Arbitrary non-uniform cost models can make trivial problems non-trivial, like finding the minimum [10, 16] . Thus it makes sense to consider a more structured cost. For example, a common cost model is the monotone[1] cost model. As shown in [16] the best one can do is to get an algorithm that is within a logarithmic factor of a cost optimal algorithm. However, the 1-∞ cost model in this paper is not monotonic. This model has comparison cost of 1 or ∞. A pair with cost ∞ is considered a "forbidden pair". The set of pairs with comparison cost 1, defines an undirected graph, $G(V, E)$, where V is the set of keys and E represents the allowed comparisons. We call G the comparison graph. Define E_f to be the set of forbidden pairs (edges). Let $|V| = n$ and $|E_f| = q$.

Next we define the query model used in this paper. We don't get charged for checking whether an edge exists but are only charged for the comparisons made. The number of comparisons made or rather asked to the oracle is naturally defined as the *comparison complexity* or the *probe complexity*. No non-trivial ITB for the probe complexity is known in the standard decision tree model. We believe that the model is too weak for this purpose. For example, given a comparison graph G the number of different acyclic orientations of G gives an upper bound on the number of possible answers as each correspond to a unique

[1] By monotone we mean that the cost of comparing a pair is a monotone function of the values of the pair.

partial order. Since identifying G (up to isomorphism by verifying edges) is free and G has $\leq \prod_{v \in V}(d_v + 1) \leq n^n$[15] number of acyclic orientations we have the ITB of $\Omega(n \log n)$ for this problem. We believe this bound to be weak for this problem. The matter is further complicated if one is also given the guarantee that the graph G is *sortable*. We say G is sortable if G can be totally sorted. This restriction further reduces the number of possible answers for graphs with small number of edges. For example if G has $= n - 1$ edges then we can determine the unique total order by just making one comparison. Since any acyclic orientation of the edges of G must give a Hamiltonian path and G has $= n - 1$ edges, the edges must link consecutive vertices in the unknown order. A solitary probe is then used to determine the direction of this ordering. In this paper we take G to be arbitrary and not necessarily sortable. Hence by sorting G we mean determining the orientations of all the edges of G which may only get us a partial order on the vertices.

1.1 Prior Results

The problem of sorting with forbidden pairs is still open for the most part. It is closely related to the problem of partial sorting under a relation determining oracle. In this model we are given a set P of elements and a oracle $\mathcal{O}_r$ which is used to determine the relations between pairs of elements in P. The goal is to determine all the valid relations. Number of queries made to $\mathcal{O}_r$ is defined as the *query complexity*. Since there are $\Omega(2^{n^2/4})$[11] labeled posets with n elements, it immediately follows that the information theoretic bound (ITB) for the query complexity is $\Omega(n^2)$. This has been investigated for width bounded posets in [12], where the authors show that if P has width at most w (size of the largest anti-chain) then the ITB for the query complexity is $\Omega((w + \log n)n)$. They presented a query optimal algorithm for width bounded posets whose total complexity is $O(nw^2 \log \frac{n}{w})$. This algorithm can be generalized for any poset with an additional $\log w$ factor added to the the query complexity. Their results were the first major extension in this line of research after the seminal work by Faigle and Turán[14] which only showed the existence of such an algorithm. Another similar problem is the *local sorting* problem. In this problem V is an ordered set and for each $(u, v) \in E$ we want to determine their relative order. The problem is to determine if this can be done without sorting the entire set V, since the ITB for this problem is $\Omega(n \log (m + n)/n)$ in the standard comparison tree model (where m is the number of edges G). Currently no non-trivial deterministic algorithm is known for this problem. However, there is a randomized algorithm which makes optimal number of comparison with high probability [9]. Another related problem is the *partial order production problem*, where given a set T with an unknown total order we are interested in determining the partial order of another set S by comparing pairs in T. The goal is do this with minimum number of comparisons. The reader is referred to the survey by Cardinal et. al [8] which discusses some of these and other related problems in detail.

An example of a problem that uses the probe complexity model is the nuts and bolts problem. This is strictly not a sorting problem rather a matching one. In this problem one is given two sets of elements, a set of nuts and a set bolts. Elements in each set have distinct sizes and for each nut it is guaranteed that there exists a unique bolt of same size. Matching is performed by comparing a nut with a bolt. However, pairs of nuts or pairs of bolts cannot be compared. So in this case $G = K(N, B)$ is a complete bipartite graph with edges from the set of nuts N to the set of bolts B. This problem has been solved in the mid 1990s [3, 20]. The existence of a $O(n \log n)$ time deterministic algorithm was proved for it using the theory on bipartite expanders [3]. In the context of randomized algorithms, this problem has been studied in [17, 4]. The authors in [17] proposed a randomized algorithm that sorts G with

a probe complexity of $\widetilde{O}(n^{3/2})$ with high probability[2]. However their implementation uses as a sub-routine a poly-time uniform sampling algorithm to sample points from a convex polytope[13]. The authors did not discuss the exact bound on the total time complexity in their paper. At each step the algorithm either finds a *balancing edge*[3] or finds a subset of elements that can be sorted quickly. For an arbitrary G it is not guaranteed that a *balancing edge* always exists. However, when G is the complete graph there always exists a *balancing edge* that reduces the number of linear extension at-least by a factor of 8/11 [19].

1.2 Our Results

In this paper we propose the first non-trivial deterministic algorithm under the probe complexity model as well as a randomized algorithm. The results are expressed in terms of n and q. Expressing the results in terms of the number of forbidden edges fits naturally with the problem. First of all q and w are related, where w is the width of the poset P_G found after sorting G. We have $q \geq \#$ of incomparable pairs in $P_G \geq \binom{w}{2}$. Hence, $w = O(\sqrt{q})$. Although we cannot directly compare the probe complexity used in this paper with the query complexity in [12] it gives a better sense of the relatedness of the two models. Secondly, in the absence of any other structural properties of the input graph G, q gives a good indication of how difficult it is to sort G. For example, when $q = O(\log n)$, it is easy to see that one can sort in $O(n \log n)$ total time. To do this we pick an arbitrary pair of non-adjacent vertices and take out one of them, removing it from the graph. We do the same thing with the remaining graph until the graph remaining is a clique. It is clear that we had to take out at most $O(\log n)$ vertices. Then we sort this graph with $O(n \log n)$ comparisons and merge the vertices we had remove previously by probing all the remaining undirected edges, which is at most $O(n \log n)$. On the other extreme, if $|E| = \binom{n}{2} - q = O(n)$ then it can be shown that we need to make $\Omega(|E|)$ probes to determine the partial order, since the complete bipartite graph $K(A, B)$ with $|A| \ll |B|$ has many acyclic orientations [18, 15]. So in this case one has to probe most of the allowed edges.

The main contributions of this paper are as follows:

- Given a comparison graph G we propose a deterministic algorithm that sorts G with $O((q + n) \log n)$ probes. The total complexity of our algorithm is $O(n^2 + q^{\omega/2})$, where $\omega \in [2, 2.38]$ is the exponent in the complexity of matrix multiplication. We start by finding a set of large enough cliques in G and use its elements to determine a good pivot. This algorithm is applied recursively to induced subgraphs of G to generate a collection of partial orders. We then merge these partial orders in the final steps.
- We propose a randomized algorithm which sorts G with $O(n^2/\sqrt{n+q} + n\sqrt{q})$ probes with high probability. We use a random graph model for this purpose. The method uses only elementary techniques and unlike in Huang, et. al[17] has a total run time of $O(n^\omega)$ in the worst case.
- When G is a random graph with edge probability p we show that one can sort G with high probability using only $\widetilde{O}(\min(n^{3/2}, pn^2))$ probes.

The rest of this paper is organized as follows: in section 1.2 we introduce some definitions and lemmas. Section 2 details the proposed deterministic algorithm. In section 3 we introduce the randomized algorithm and its extension to random graphs.

---

[2] By *high probability* we mean that the probability tends to 1 as $n \to \infty$.

[3] An edge in G revealing whose orientation is guaranteed to reduce the number of linear extensions of the current partial order by a constant fraction. The pair of vertices incident to this edge is referred to as a *balancing pair*.

1.3 Definitions

Recall $G(V, E)$ is the input graph on the set V of elements to be sorted. A pair of vertices (u, v) can be compared if $(u, v) \in E$, otherwise, we say the pair is forbidden and is in E_f. The graph G is given to us by our adversary. Let G_i be the graph after i-edges have been oriented and P_i be the associated partial order. We denote the degree of a vertex v by $d(v)$ and $n(v) = n - 1 - d(v)$ is the number of vertices that are not adjacent to v. The set of neighbors of a vertex v is denoted by $N(v)$. We use the notation $E(A, B)$ we denote the set of edges between the sets of vertices $A, B \subset V$. We also define the little-o notation to remove any ambiguity from our exposition.

▶ **Definition 1.** If $f(n) \in o(g(n))$ then $f(n) \in O(g(n))$ but $f(n) \notin \Omega(g(n))$.

The following lemmas can be easily proven, hence we omit their proofs.

▶ **Lemma 2.** *Let $\{f_1(n), f_2(n), ..., f_k(n)\}$ be a finite set of non-negative monotonically increasing functions in n such that for $g(n)$:*
1. $\forall i\ f_i(n) \in o(g(n))$
2. $\sum_i f_i(n) \leq cg(n)$
If $F(n) = \sum_i f_i^2(n)$ then $F(n) \in o(g^2(n))$.

▶ **Lemma 3.** *Let $T(n) = \sum_{i=1}^{k} T(n_i) + f(n)$ where $\sum_i n_i \leq \delta n$ for some $0 < \delta < 1$ and $f(n) \in o(n^2)$. Then, $T(n) \in o(n^2)$.*

2 A Deterministic Algorithm For Restricted Sorting

First we look at a simple case where $q = O(n)$. We will use some of the main ideas from this algorithm to extend it to the general case. This initial algorithm will have a worse probe complexity than the main algorithm. In this algorithm we shall do case analysis based on whether a certain quantity is $o(n)$ or not. We acknowledge that this is not an algorithmic test. However, we use it in this algorithm to establish a framework for the second algorithm, which uses a traditional test and does not affect the claims made in this paper.

2.1 A Restricted Case

Assume $q \leq cn$ for some constant c. Let $R = \{v \in V \mid n(v) > c_1\}$ for some constant c_1, then $|R| \leq (2c/c_1)n$. This is obvious from the fact that $\sum_v n(v) \leq 2cn$. We choose $c_1 = 4c$. Let $S = V \setminus R$ and $G[S]$ be the induced subgraph generated by S. We have $|S| \geq n/2$ and if $v \in S$ then $n(v) \leq c_1$.

▶ **Claim 4.** *There exists a subset $X \subset S$ such that $|X| \geq \frac{n}{2(4c+1)}$ and $G[X]$ is a complete graph.*

Proof. We construct X explicitly. We start with $X = \{u\}$, where u is an arbitrary vertex in S. We pick successive vertices from S iteratively. Let v be last vertex to be added to X. Since v has at least $|S| - c_1$ neighbors, whenever we pick a neighbor of v from S to add to X we loose at most $c_1 + 1$ vertices (including the vertex we picked). Hence if we pick neighbors of v the size of X is at least $|S|/(c_1 + 1) \geq n/2(4c + 1)$. ◀

Clearly the above procedure runs in $O(n^2)$ time and makes no comparisons. Now we are ready to describe our algorithm. The main algorithm is recursive and we have two levels of recursion. We shall break up the algorithm into several steps.

2.1.1 Initial Sorting

Given the input graph G, let X be a clique, with $|X| \geq n/2(4c+1)$ (Claim 1). Let $Y = V \setminus X$. Note that $|Y| \leq n - n/2(4c+1) = (8c+1/8c+2)n$. Now we sort X using $O(n \log n)$ comparisons as $G[X]$ is a complete graph. We can use a standard comparison based sorting algorithms for this purpose. Now we have two possibilities:

Case 1. If $|Y| = o(n)$, then we probe all edges of $G[Y]$ and $G[Y, X]$, where $G[Y, X]$ is the induced bipartite graph generated by the sets Y and X. Then we take the transitive closure of the resulting relations, which does not need any additional probes. It can be easily seen that the number of probe made in the previous step is $o(n^2)$. For the sake of contradiction if we assume that it is not so then $|X||Y| + |Y|^2/2 \geq dn^2$ for some d. Which implies $|Y| \geq dn$, since $|X| + |Y|/2 \leq n$. But then, $|Y| = \Omega(n)$, which is not true according to our earlier assumption. So, in this case we would have sorted V by making only $o(n^2)$ probes.

Case 2. Otherwise $|Y| \geq \delta n$, for some constant δ. In this case we recursively partition Y based on elements from X. We call this the partition step.

2.1.2 Partition Step

We will recursively partition both X and Y. To keep track of the current partition depth we rename X to X_{00} and Y to Y_{00}. We pick m_{00} the median of X_{00} (after X_{00} is sorted). Since $X_{00} \subset S$ we have $n(m_{00}) \leq c_1$. So m_{00} will be comparable to all but at most c_1 elements of Y_{00}. Let,

$$A_{00} = \{v \in Y_{00} |\ v \in N(m_{00})\}$$

and $B_{00} = Y_{00} \setminus A_{00}$. Note $|B_{00}| \leq c_1$. Now let U_{00} be the subset of A_{00} whose elements are $\geq m_{00}$ and the set L_{00} accounts for the rest of $A_{00} \setminus m_{00}$. Let X_{10} and X_{11} be the elements of X_{00} that are $<$ and $\geq$ to m_{00} respectively. We recursively partition the sets U_{00} and L_{00} using the medians of X_{10} and X_{11}.The B-sets are kept for later processing. We rename the sets U_{00} and L_{00} to Y_{10} and Y_{11}. So, the pairs (X_{10}, Y_{10}) and (X_{11}, Y_{11}) are processed as above generating the sets A_{10}, A_{11}, B_{10} and B_{11}. We continue doing this until the size of the X-set is $\leq c_2$, where c_2 is some constant. At this point we don't know the size of the Y-set paired with it. There are two cases we need to consider:

Case 1. $|Y| = o(n)$**:** Then we probe all the edges of $G[Y]$ and $G[X, Y]$ which uses at most $c_2|Y| + \binom{|Y|}{2}$ number of comparisons.

Case 2. $|Y| \geq \delta n$**:** Then we have $|Y| \geq \delta n$ for constant δ. Hence the graph $G[Y]$ can have at most $\leq (c/\delta)|Y|$ missing edges. This satisfies our initial premise that the number of missing edges in $G[Y]$ is linear in the number of vertices. Hence we can apply our initial strategy recursively[4]. That is we first find a large enough clique (which according to Claim 1 must exist) and then use it to partition the rest of the set Y.

Let us visualize using a partial recursion tree T (see Fig.1 below). We shall call T the partial recursion tree for reasons that will soon be clear. At the root we have the pair (X_{00}, Y_{00}). It has two children node (X_{10}, Y_{10}) and (X_{11}, Y_{11}) each having two children of

[4] Note that (c/δ) is an absolute constant. If the input graph has at most cn missing edges we apply the procedure recursively to subgraphs whose number of missing edges are at most (c/δ) times the number of vertices in the subgraph at any level of recusrion. This (c/δ) factor is not successively multiplied within each level of recursion.

their own and so on. Now at each level, the size of the X-set gets halved. So, the number of levels in T is at most $O(\log n)$. However, the Y-sets need not get divided with equal proportions. So, at the frontier (the deepest level) we will have nodes of the above two types, depending on the size of their corresponding Y-sets. Let the collection of these frontier nodes be partitioned in two sets Φ and Ψ corresponding to case 1 and case 2 respectively.

We can conclude that the total number of probes needed to compute all relations in Φ is $o(n^2)$. This follows from Lemma 1. Here we can map the size of the Y-sets of the nodes in the collection Φ to the functions $f_i(n)$. We know that the total elements in the union of these Y-sets is $\leq |Y_{00}| \leq (8c + 1/8c + 2)n$. The total number of probes will be $F(n)$ in worst case. What is the total number of probes on the internal nodes of T? We know that in the internal nodes we compare the median of the X-set with the elements of the A-set, which takes $|A|$ probes. Since the union of these A-sets cannot exceed the total number of vertices in $G(n)$, at each level of T we do at most $O(n)$ probes, totaling to $O(n \log n)$ probes over all the internal nodes.

Unlike the nodes in Φ, the nodes in Ψ recursively call the initial strategy using the input graph $G[Y]$. Let the probe complexity of our initial strategy be $Q(n)$. Then the recursion for Q is as follows:

$$Q(n) = \sum_{i=1}^{|\Psi|} Q(n_i) + o(n^2)$$

Here we assume that the nodes in Ψ are indexed according to some arbitrary order. We can solve this recurrence using Lemma 2 giving $Q(n) \in o(n^2)$, since $\sum_{i=1}^{|\Psi|} n_i \leq (8c + 1/8c + 2)n$. Note here that $|\Psi|$ is bounded by a constant since the size of the Y-sets are $\Omega(n)$.

We call $\hat{T}$ the full tree. All leaf nodes in $\hat{T}$ are in Φ. It is straightforward to show that $\hat{T}$ has $O(\log^2 n)$ levels. Since any of the leaf nodes of T has $|Y| \leq \beta n$ (where $\beta = (8c+1/8c+2)$), its subtree in $\hat{T}$ can have at most $\alpha \log \beta n = \alpha \log n - \alpha\beta$ levels, and any of its leaves having at most $\alpha \log n - 2\alpha\beta$ levels and so on for some constant α.

2.1.3 Merge Step

Once we have completed building $\hat{T}$ we proceed with the final stage of our algorithm. Recall that during the forward partition step we had generated many of these B-sets in the internal nodes of $\hat{T}$. Now we start from the leaves of $\hat{T}$ and proceed upwards. Each pair of leaf nodes l, r sharing a common parent p, sends a partial order to it (computed as in case 1). When we merge this two partial orders we know that no extra probes are needed since they have already been split by the median of the X-set of p. What remains is to probe all edges between the B-set in p and elements in this partial order (which constitutes the set of elements $A \cup X$ of the node p) as well as the edges in $G[B]$. Then we pass the resulting partial order to the parent of p, and so on. Since the size of the B-sets are bounded by c_1 (at any level in $\hat{T}$), total number of probes we make is then $\leq c_1 \sum_i (|A_i| + |X_i| + c_1)$. The sum is taken over all the nodes in that level. Hence this is bounded by $c_1 n$, so at each level we do at most $O(n)$ probes in the backward merging step. Since there are at most $O(\log^2 n)$ levels, it totals to $O(n \log^2 n)$ additional probes. Adding this to the probe cost of partitioning in the forward step does not effect the total probe complexity, which was $o(n^2)$. The final step is to compute the transitive closure of the resulting set of relations, which can be done without any additional probing. Since computing the transitive closure is equivalent to boolean matrix multiplication[21] the total complexity is $O(n^\omega)$.

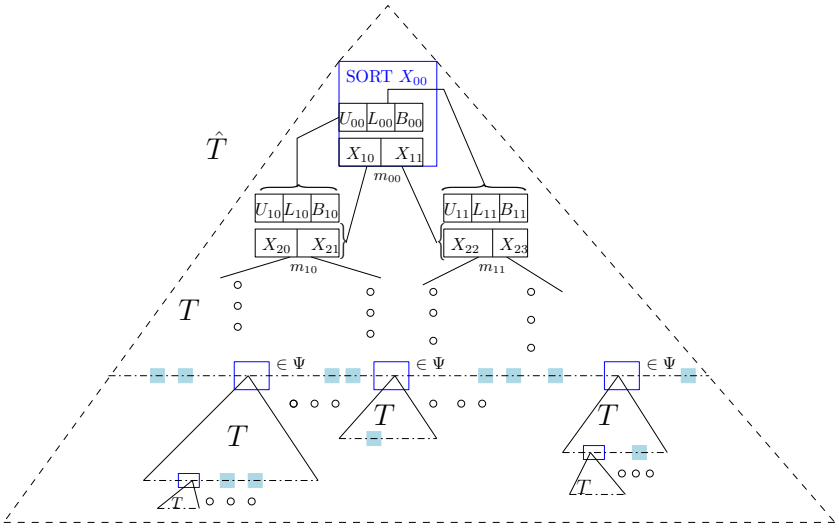


Figure 1 Visualizing the steps. At the bottom of T the shaded boxes represents the Φ-nodes and the blue rectangles the Ψ-nodes. The outer dashed triangle represents the full tree $\hat{T}$. The tree $\hat{T}$ is created during the partitioning step and in the merge step we start from the deepest leaves of $\hat{T}$ and move upwards.

2.2 The General Case

We will define the sets R and S analogously to section 2.1. We have $R = \{v \in V \mid n(v) > c_1 q/n\}$ for some constant c_1. With $c_1 = 4$, we get $|R| \leq \delta_1 n$ where $\delta_1 \leq 2/c_1 = 1/2$. Hence $|S| \geq (1 - \delta_1)n \geq n/2$. Now we will apply Claim 1 successively to construct a "big-enough" set $X \subset S$ which we will use to find an approximate median of V. This set X consists of disjoint subsets X_i such that $G[X_i]$ is a clique.

2.2.1 Constructing X

Let us define $S_i = S \setminus \bigcup_{j=1}^{i} X_j$. We construct the first clique $X_1 \subset S$ using the method detailed in Claim 1. There are two cases:

Case 1. $q < n$: In this case we can show that $|X_1| \geq (n/2)/(c_1 q/n + 1) \geq n/10$. We take the first $n/10$ elements and keep the rest for the second round. Now we construct the second clique X_2 from S_1 which has at least $2n/25$ vertices. We let $X = X_1 \cup X_2$. Hence X has at least $9n/50$ vertices.

Case 2. $q \geq n$: In this case we have $|X_1| \geq (n/2)/(c_1 q/n + 1) \geq n^2/10q$. Again we take $|X_1| = (1/10)n^2/q$ discarding some vertices if necessary. Similarly we construct $X_2 \subset S_1$. It can be shown that $|X_2| \geq (n^2/10q)(1 - n/5q)$ and we keep $(n^2/10q)(1 - n/5q)$ vertices in X_2 and the rest are discarded to be processed the next round. In general for the clique X_r we have $|X_r| \geq (n^2/10q)(1 - n/5q)^{r-1}$. Now we let $X = \bigcup_{i=1}^{r} X_i$. We will show that $|X| \geq \delta_2 n$ for some constant $\delta_2 > 0$. We let $r = 5q/n + 1$. Then we have

$$|X_r| \geq (n^2/10q)(1 - n/5q)^{r-1} \geq (n^2/10q)(1 - n/5q)^{5q/n} > 3n^2/100q$$

since $q \geq n$. Hence, $|X| = \sum_{i=1}^{r} |X_i| \geq r|X_r| \geq (9/50)n$, giving $\delta_2 = 9/50$. Now for each X_i $(1 \leq i \leq r)$ we keep a subset Y_i of size $|X_r|$ and throw away the rest. Clearly, for each i, the induced sub-graph $G[Y_i]$ is also a clique. Let $Y = \bigcup_{i=1}^{r} Y_i$. We also have $|Y| \geq (9/50)n$.

2.2.2 Computing An Approximate Median Of V

We shall compute an approximate median with respect to all the vertices (the set V) and not just the set S. We will find a median element that divides the set V in constant proportions. This can be done easily using the set Y. For each Y_i we find its median using $\Theta(|Y_i|)$ probes since $G[Y_i]$ is a complete graph. Let this median be m_i and M be the set of these r medians. Since $m_i \in S$, $n(m_i) \leq 4q/n$. We define the upper set of $m \in M$ with respect to a set $A \subset V$ (m may not be a member of A) as $U(m, A) = \{a \in A \mid a > m\}$. Similarly we define the lower set $L(m, A)$. We want to compute the sets $U(m, Y)$ and $L(m, Y)$. However, m may not be neighbors of all the elements in Y. So we compute approximate upper and lower sets by probing all the edges in $E(\{m\}, Y \setminus \{m\})$. These sets are denoted by $\widetilde{U}(m, Y)$ and $\widetilde{L}(m, Y)$ respectively. It is easy to see that there exists some $m \in M$ which divides Y into sets of roughly equal sizes (their sizes are a constant factor of each other). In fact the median of M is such an element. However the elements in M may not all be neighbors of each other hence we will approximate m using the ranks of the elements in M with respect to the set Y (which is $|\widetilde{L}(m, Y)|$). Next we prove that the element m^* is an approximate median of M, picked using the above procedure, is also an approximate median of Y.

▶ **Claim 5.** *The element m^* picked as described above is an approximate median of Y.*

Proof. First we show that the median of M is an approximate median of Y. This can be easily verified. Let us take the elements in M in sorted order $(m_1, ..., m_r)$, so the median of M is $m_{\lfloor r/2 \rfloor}$. Now $L(m_{\lfloor r/2 \rfloor}, Y) \geq \bigcup_{i=1}^{\lfloor r/2 \rfloor} L(m_i, Y_i)$. Since, the sets Y_i are disjoint and $L(m_i, Y_i) \geq |X_r|/2$, we have $|L(m_{\lfloor r/2 \rfloor}, Y)| \geq |X_r|r/4$ (ignoring the floor). Similarly we can show that $|U(m_{\lfloor r/2 \rfloor}, Y)| \geq |X_r|r/4$. Hence $m_{\lfloor r/2 \rfloor}$ is an approximate median of Y. Now we show that $| \, |L(m^*, Y)| - |L(m_{\lfloor r/2 \rfloor}, Y)| \, | < 4q/n$. Consider the sorted order of elements in M according to $|\widetilde{L}(m^*, Y)|$. Since each element in $m \in M$ has at most $4q/n$ missing neighbors in Y, we have $| \, |\widetilde{L}(m, Y)| - |L(m, Y)| \, | < 4q/n$. So the rank of an element in the sorted order is at most $4q/n$ less than its actual rank. Thus an element m^* picked as the median of M using its approximate rank $|\widetilde{L}(m, Y)|$ cannot be more than $4q/n$ apart from $m_{\lfloor r/2 \rfloor}$ in the sorted order of Y. Hence

$$|L(m^*, Y)| \geq |X_r|r/4 - 4q/n \geq 9n/200 - 4q/n \geq n/40 \tag{1}$$

whenever $n^2 \geq 200q$. In an identical manner we can show that $|U(m^*, Y)| \geq n/40$. Hence, m^* is an approximate median of Y. When $q < n$ we just take m^* as the median with the higher $|\widetilde{L}(\cdot, Y)|$ value, which guarantees $|L(m^*, Y)| \geq n/40$ whenever $n^2 \geq 800q/13$. So we take $n^2 \geq 200q$ to cover both the cases. ◀

It immediately follows that m^* is also an approximate median of V with both $|L(m^*, V)|$ and $|U(m^*, V)|$ lower bounded by $n/40$. Lastly, we note that the above process of computing an approximate median makes $\Theta(q + n)$ probes. This follows from the fact that computing the medians makes $\Theta(n)$ probes in total and for each of the $\leq 5q/n + 1$ medians we make $O(n)$ probes.

2.2.3 A Divide-And-Conquer Approach

Now that we have computed an approximate median of V we proceed with an recursive approach. Let m^* be the median. As in section 2.1 we partition V into three sets U, L and B. The U and L are the upper and lower sets with respect to m^*. B is the set of vertices that do not fall into either, that is, they are non-neighbors of m^*. Since $m^* \in S$ we have $|B| \leq 4q/n$.

We recursively proceed to partially sort the sets U and L with the corresponding graphs $G[U]$ and $G[L]$ and keep B for later processing (as we did in the merging step previously). Like before we can imagine a recursion tree T. Let E_{f_P} be the set forbidden edges in $G[P]$. We take $n_P = |P|$ and $q_P = |E_{f_P}|$. For each node $P \in T$ there are two cases:

Case 1. When $n_P^2 \geq 200q_P$ we recursively sort P. In this case we can guarantee that the approximate median m_P^* of P will satisfy equation (1). That is both $|L(m_P^*, P)|$ and $|U(m_P^*, P)|$ is $\geq n_P/40$.

Case 2. Otherwise we probe all edges in $G[P]$. In this case P will become a leaf node in T.

It can be easily seen that the depth of the recursion tree is bounded by $O(\log n)$ since at each internal node P of T we pass sets of constant proportions (where the size of the larger of the two set is upper bounded by $(39/40)n_P$) to its children nodes.

2.2.4 Merge Step

In this step we start with the leaves of T and proceed upwards. A parent node P gets two partial orders from its left and right children respectively. Then it probes all the edges between its B-set and these partial orders to generate a new partial order and pass it on to its own parent. This step works exactly as the "merge step" of the previous algorithm. Only difference is that the B-sets here may not be of constant size but of size $\leq 4q/n$.

2.2.5 Probe Complexity

We can determine the probe complexity by looking at the recursion tree T. First we compute it for the forward partition step. At each internal node of T we compute a set of medians and pick one element from it appropriately chosen. Then we partition the set of elements at the node by probing all edges between the selected element and rest of the elements in the node. As mentioned before this only takes $\Theta(q_P + n_P)$ probes for some internal node P. We assume that all the leaves of T are at the same depth, otherwise we can insert internal dummy nodes and make it so. At each level of T the sum total of all the vertices in every node is $\leq n$ and the sum total of the forbidden edges is $\leq q$. Hence we do $O(q+n)$ probes at any internal level of T. So for a total of $O(\log n)$ internal levels in T the number of probes done is $((q+n)\log n)$ in the forward partition step. If P is a leaf node then we probe all edges in $G[P]$. There are at most $\binom{n_P}{2} - q_P$ edges in $G[P]$. Since P is a leaf node, according equation 1, $n_P^2 < 200q_P$. Hence we make $\binom{n_P}{2} - q_P = O(q_P)$ probes. Summing this over all the leaves gives a total of $O(q)$ probes. Hence the total probe complexity during the forward step is $O((q+n)\log n)$.

Now we look at the merging step. Merging happens only at the internal nodes. Lets look at an arbitrary internal level of T. At each node P of this level we probe all the edges in $E(B_P, U_P \cup L_P \cup m_P^*)$ and in $G[B_P]$. Note that we do not have to make any probes between U and L as they were already separated by the approximate median m_P^*. Hence the total number of probes made in this node is $\leq (|U_P|+|L_P|+|B_P|+1)|B_P| \leq (n_P)(4q_P/n_P) \leq 4q_P$. Summing over all the nodes at any given level gives us $O(q)$ as the probe complexity per level. So the total probe complexity in the merging stage is $O(q \log n)$. Hence, combining the probes made during the partition step and the merge step we see that the total probes needed to sort V is $O((q+n)\log n)$.

2.2.6 Total Complexity

Now we look at the total complexity of the previous procedure. Again the analysis is divided into forward step and the merge step. In the forward step at each node P we perform

$O(n_P^2)$ operations. This includes computing the degrees, finding the cliques, computing the approximate median. So at any level of T, regardless of it being an internal level or not, we perform $O(n^2)$ operations. Hence it totals to $O(n^2 \log n)$ operations in the forward step. However this is a conservative estimate and we can remove the $\log n$ factor as argued below: we can define the recurrence for the forward computation as,

$$T(n) = \begin{cases} T(n/40) + T(39n/40) + O(n^2) & n^2 \geq 200q \\ O(q) \text{ Otherwise} \end{cases} \tag{2}$$

This follows from the previous discussion. If we don't recurse on a node we guarantee that $n_P^2 < 200q_p$ for that node. Hence, we have $T(n) = O(n^2 + q)$ using the Akra-Bazzi method[2]. In the merge step, we only make $O(q_P)$ comparisons at any given node. We compute transitive closures only at the leaves. However for any leaf P we have $n_P^2 < 200q_P$. Hence computing the transitive closure of $G[P]$ takes $O(q_P^{\omega/2})$ time. Hence, the total complexity of the above procedure is $O(n^2 + q^{\omega/2})$. We summarize the results in this section with the following theorem:

▶ **Theorem 6.** *Given a graph $G(V, E)$ of n vertices having q forbidden edges, one can compute the partial order of V with $O((q + n) \log n)$ comparisons and in total $O(n^2 + q^{\omega/2})$ time.*

Proof. Follows from the discussions in this section. ◀

3 A Randomized Algorithm

In this section we look at a more direct way of sorting by making random probes. The proposed method is inspired by the literature on two-step oblivious parallel sorting [1, 7] algorithms, in particular on a series of studies by Bollobás and Brightwell showing certain sparse graphs can be used to construct efficient sorting networks [6, 5]. It was shown that if a graph satisfies certain properties then probing its edges and taking the transitive closure of the results would yield large number of relations. Then we just probe the remaining edges that are not oriented, which is guaranteed (with high probability) to be a "small" set.

The main idea is as follows: Let $\mathcal{H}_n$ be a collection of undirected graphs on n vertices having certain properties. A transitive orientation of a graph $H(V, E) \in \mathcal{H}_n$ is an ordering of V and the induced orientation of the edges of H based on that ordering. Let σ be an ordering on V and $P(H, \sigma)$ be the partial order generated by this ordering σ on H. It is a partial order since H may not be sortable. Let $\mathcal{P} = P(H, \sigma)$ and $t(P)$ be the number of incomparable pairs in $\mathcal{P}$. We want H to be such that $t(p)$ is small. If that is the case then $\mathcal{P}$ will have many relations and if H is sparse then we can probe all the edges of H and afterwards we will be left with probing only a small number of pairs. These are pairs which were not oriented during the first round of probing and after the transitive closure computation. A graph H is *useful* to our purpose if every transitive orientation of H results in many relations. We want to find a collection $\mathcal{H}_n$ such that every graph in it is useful with high probability.

We extend the results in [6, 5] to show that a collection of certain conditional random graphs are useful, with high probability. In our case this random graph will be a spanning subgraph of the input graph G. Here we recall an important result from [6] (Theorem 7) which we will use in our proof.

▶ **Theorem 7** ([6]). *If G is any graph on n vertices and G satisfies the following property:*
Q1 *Any two subsets A, B of vertices having size l have at least one edge between them.*
Then, the number of incomparable pairs in $P(G, \sigma)$ is at most $O(nl \log l)$ for any σ.

The input graph G is chosen by our adversary. However, we show that any random spanning subgraph of G with an appropriate edge probability will satisfy Q1 with high probability. Let $H_{n,p}(G)$ be a random spanning subgraph of G, where $H_{n,p}(G)$ has the same vertex set as G and a pair of vertices in $H_{n,p}(G)$ has an edge between them with probability p if they are adjacent in G, otherwise they are also non-adjacent in $H_{n,p}(G)$. All we need to prove is that any random spanning subgraph $H_{n,p}(G)$ given G with n-vertices and edge probability p will satisfy Q1 with high probability. Since G has at most q forbidden edges any two subsets of vertices A, B (not necessarily distinct) of size l must have at least $\binom{l}{2} - q$ edge between them. Let E_{AB} be the event that the pair (A, B) is bad (they have no edges between them), then the probability $S_{n,p}$ that there exists a bad pair is:

$$S_{n,p} := \mathbb{P}(\sum_{i,j} E_{A_i B_j}) \leq \sum_{i,j} \mathbb{P}(E_{A_i B_j}) \leq \sum_{i,j} (1-p)^{e(A_i, B_j)} \tag{3}$$

where the sum is taken over all such $\binom{n}{l}^2$ pairs of subsets, and the number of edges between the two sets A and B in G is $e(A, B) \geq \binom{l}{2} - q$. So we have,

$$\begin{aligned} S_{n,p} &\leq \binom{n}{l}^2 (1-p)^{\binom{l}{2}-q} \leq \binom{n}{l}^2 e^{-p(\binom{l}{2}-q)} \quad \text{Since, } e^{-x} \geq 1 - x \\ &\leq \left(\frac{en}{l}\right)^{2l} e^{-p(\binom{l}{2}-q)} \leq \exp(2l(\log en/l) - p(\binom{l}{2} - q)) \end{aligned}$$

Hence $S_{n,p} \to 0$ as $n \to \infty$ whenever $\exp(2l(\log en/l) - p(\binom{l}{2} - q)) = o(1)$. Given $q < \binom{n}{2}$ it is always possible to find appropriate values for p and l as functions of q and n such that $S_{n,p} = o(1)$. Given some value for the pair (p, l), we see that in the first round we make $O(pn^2)$ probes with high probability and in the second round $O(nl \log l)$ probes (for the remaining unoriented edges) again with high probability. So the total probe complexity is $\widetilde{O}(pn^2 + nl)$. With some further algebra it can be shown that this is $\widetilde{O}(n^2/\sqrt{q+n} + n\sqrt{q})$. We summarize this section with the following theorem:

▶ **Theorem 8.** *Given a graph G on n vertices and q forbidden edges one can determine the partial order on G with high probability in two steps by probing only $\widetilde{O}(n^2/\sqrt{q+n} + n\sqrt{q})$ edges in total and in $O(n^\omega)$ time.*

Proof. Follows from the preceding discussions. ◀

3.1 When G Is A Random Graph

The above technique can easily be extended for the case when the input graph is random. Let $G_{n,p}$ be the input graph having n-vertices and an uniform edge probability p. For such a graph we can use equation (3) to bound $S_{n,p}$ as follows:

$$S_{n,p} \leq \binom{n}{l}^2 (1-p)^{l^2} \leq \exp(-pl^2 + 2l \log n)$$

Hence, we can choose any $l > 2 \log n/p$ such that $S_{n,p} \to 0$ as $n \to \infty$. Let $l = 3 \log n/p$. Using Theorem 2 we have $t(G_{n,p}) = \widetilde{O}(nl) = \widetilde{O}(n/p)$. Since $G_{n,p}$ has $pn^2/2$ edges (with

high probability) the critical value of p when $t(G_{n,p}) = pn^2/2$ is $\widetilde{O}(1/\sqrt{n})$. Let this be $\hat{p}$. Hence if $p > \hat{p}$, we can sort by making only $\widetilde{O}(n^{3/2})$ comparisons. Since given $G_{n,p}$ we can construct an induced subgraph $G_{n,\hat{p}}$ and use it as the random graph in our previous construction. Otherwise we just probe all the edges which makes $O(pn^2)$ comparisons. Thus we can sort $G_{n,p}$ with at most $\widetilde{O}(\min(n^{3/2}, pn^2))$ comparisons with high probability. Hence, we get an elementary technique to sort a random graph with at most $\widetilde{O}(n^{3/2})$ comparisons. The algorithm in [17] has a slightly better bound of $\widetilde{O}(n^{7/5})$ comparisons. However, the total runtime of the algorithm in [17] is only polynomially bounded when p is small. In our algorithm we need compute the transitive closure only twice making it run in $O(n^\omega)$ total time.

Acknowledgements. We thank the anonymous reviewers for their constructive comments, which helped us to improve the paper.

References

1 Miklós Ajtai, János Komlós, William Steiger, and Endre Szemerédi. Almost sorting in one round. *Randomness and Computation*, 5:117–125, 1989.

2 Mohamad Akra and Louay Bazzi. On the solution of linear recurrence equations. *Comp. Opt. and Appl.*, 10(2):195–210, 1998. URL: http://dx.doi.org/10.1023/A:1018373005182, doi:10.1023/A:1018373005182.

3 Noga Alon, Manuel Blum, Amos Fiat, Sampath Kannan, Moni Naor, and Rafail Ostrovsky. Matching nuts and bolts. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia.*, pages 690–696, 1994. URL: http://dl.acm.org/citation.cfm?id=314464.314673.

4 Stanislav Angelov, Keshav Kunal, and Andrew McGregor. Sorting and selection with random costs. In *LATIN 2008: Theoretical Informatics, 8th Latin American Symposium, Búzios, Brazil, April 7-11, 2008, Proceedings*, pages 48–59, 2008. URL: http://dx.doi.org/10.1007/978-3-540-78773-0_5, doi:10.1007/978-3-540-78773-0_5.

5 Béla Bollobás and Graham Brightwell. Graphs whose every transitive orientation contains almost every relation. *Israel Journal of Mathematics*, 59(1):112–128, 1987.

6 Béla Bollobás and Graham R. Brightwell. Transitive orientations of graphs. *SIAM J. Comput.*, 17(6):1119–1133, 1988. URL: http://dx.doi.org/10.1137/0217072, doi:10.1137/0217072.

7 Béla Bollobás and Moshe Rosenfeld. Sorting in one round. *Israel Journal of Mathematics*, 38(1-2):154–160, 1981.

8 Jean Cardinal and Samuel Fiorini. On generalized comparison-based sorting problems. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 164–175, 2013. URL: http://dx.doi.org/10.1007/978-3-642-40273-9_12, doi:10.1007/978-3-642-40273-9_12.

9 Jean Cardinal, Samuel Fiorini, Gwenaël Joret, Raphaël M. Jungers, and J. Ian Munro. An efficient algorithm for partial order production. *SIAM J. Comput.*, 39(7):2927–2940, 2010. URL: http://dx.doi.org/10.1137/090759860, doi:10.1137/090759860.

10 Moses Charikar, Ronald Fagin, Venkatesan Guruswami, Jon M. Kleinberg, Prabhakar Raghavan, and Amit Sahai. Query strategies for priced information. *J. Comput. Syst. Sci.*, 64(4):785–819, 2002. URL: http://dx.doi.org/10.1006/jcss.2002.1828, doi:10.1006/jcss.2002.1828.

11 SD Chatterji. The number of topologies on n points, kent state university. *NASA Technical Report*, 1966.

12 Constantinos Daskalakis, Richard M. Karp, Elchanan Mossel, Samantha Riesenfeld, and Elad Verbin. Sorting and selection in posets. *SIAM J. Comput.*, 40(3):597–622, 2011. URL: http://dx.doi.org/10.1137/070697720, doi:10.1137/070697720.

13 Martin E. Dyer, Alan M. Frieze, and Ravi Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. *J. ACM*, 38(1):1–17, 1991. URL: http://doi.acm.org/10.1145/102782.102783, doi:10.1145/102782.102783.

14 Ulrich Faigle and György Turán. Sorting and recognition problems for ordered sets. *SIAM J. Comput.*, 17(1):100–113, 1988. URL: http://dx.doi.org/10.1137/0217007, doi:10.1137/0217007.

15 Wayne Goddard, Claire Kenyon, Valerie King, and Leonard J. Schulman. Optimal randomized algorithms for local sorting and set-maxima. *SIAM J. Comput.*, 22(2):272–283, 1993. URL: http://dx.doi.org/10.1137/0222020, doi:10.1137/0222020.

16 Anupam Gupta and Amit Kumar. Sorting and selection with structured costs. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 416–425, 2001. URL: http://dx.doi.org/10.1109/SFCS.2001.959916, doi:10.1109/SFCS.2001.959916.

17 Zhiyi Huang, Sampath Kannan, and Sanjeev Khanna. Algorithms for the generalized sorting problem. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 738–747, 2011. URL: http://dx.doi.org/10.1109/FOCS.2011.54, doi:10.1109/FOCS.2011.54.

18 Nabil Kahale and Leonard J. Schulman. Bounds on the chromatic polynomial and on the number of acyclic orientations of a graph. *Combinatorica*, 16(3):383–397, 1996. URL: http://dx.doi.org/10.1007/BF01261322, doi:10.1007/BF01261322.

19 Jeff Kahn and Michael Saks. Balancing poset extensions. *Order*, 1(2):113–126, 1984.

20 János Komlós, Yuan Ma, and Endre Szemerédi. Matching nuts and bolts in o(n log n) time. *SIAM J. Discrete Math.*, 11(3):347–372, 1998. URL: http://dx.doi.org/10.1137/S0895480196304982, doi:10.1137/S0895480196304982.

21 John E. Savage. *Models of computation - exploring the power of computing*. Addison-Wesley, 1998.

Total Stability in Stable Matching Games

Sushmita Gupta[1], Kazuo Iwama[2], and Shuichi Miyazaki[3]

1 **Institute of Informatics, University of Bergen, Bergen, Norway**
sushmita.gupta@gmail.com
2 **Research Institute for Mathematical Sciences, Kyoto University, Kyoto, Japan**
iwama@kuis.kyoto-u.ac.jp
3 **Academic Center for Computing and Media Studies, Kyoto University, Kyoto, Japan**
shuichi@media.kyoto-u.ac.jp

Abstract

The stable marriage problem (**SMP**) can be seen as a typical game, where each player wants to obtain the best possible partner by manipulating his/her preference list. Thus the set **Q** of preference lists submitted to the matching agency may differ from **P**, the set of true preference lists. In this paper, we study the stability of the stated lists in **Q**. If **Q** is not Nash equilibrium, i.e., if a player can obtain a strictly better partner (with respect to the preference order in **P**) by only changing his/her list, then in the view of standard game theory, **Q** is vulnerable. In the case of **SMP**, however, we need to consider another factor, namely that all valid matchings should not include any "blocking pairs" with respect to **P**. Thus, if the above manipulation of a player introduces blocking pairs, it would prevent this manipulation. Consequently, we say **Q** is *totally stable* if either **Q** is a Nash equilibrium or if any attempt at manipulation by a single player causes blocking pairs with respect to **P**. We study the complexity of testing the total stability of a stated strategy. It is known that this question is answered in polynomial time if the instance $(\mathbf{P}, \mathbf{Q})$ always satisfies $\mathbf{P} = \mathbf{Q}$. We extend this polynomially solvable class to the general one, where **P** and **Q** may be arbitrarily different.

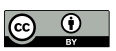
1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases stable matching, Gale-Shapley algorithm, manipulation, stability, Nash equilibrium

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.23

1 Introduction

Matching under preferences is an extensively studied area of theoretical and empirical research that has a wide range of applications in economics and social sciences. One of the most popular and standard problems in this field is the *stable marriage problem* (**SMP**) introduced by Gale and Shapley [4], where there are two parties; a set of men and a set of women. Each man has a list that orders women according to his preference, and similarly each woman also has a preference ordering for the men. The goal is to find a *stable matching*. A matching is said to be *stable* if it does not have a *blocking pair*, that is, a pair of a man and a woman who prefer each other to their current matching partners. Existence of blocking pairs poses threat to the "stability of the marriage". It is important to note that a matching may be stable in terms of one set of preference lists, but not in terms of another. Hence, whenever there is a possibility of confusion when referring to a stable matching or blocking pairs, we will specify the corresponding preference lists, like "**Q**-stable" and "**Q**-blocking pairs" for a set **Q** of preference lists. It is well-known that there may be more than one stable matching for a



given set of preference lists, where the *men-optimal* and *women-optimal* stable matchings represent the two extremes. These matchings derive their names from the property that, in the men-optimal (women-optimal) stable matching, every man (woman) receives the best possible partner among all the stable matchings. It is well-known that the men-proposing Gale-Shapley algorithm (GS-M, for short) finds the men-optimal stable matching in linear time [4, 6]. Throughout this paper, we focus on only GS-M.

SMP can be seen as a game, in the sense that each player is selfish and wants to obtain the best possible partner, even at the cost of stealing one from the other players. Thus, several game-theoretic issues come into play in this scenario. These issues have been studied mainly from the point of view of economic and market theories (see Roth and Sotomayor [12] for detailed discussions). A fundamental axiom of the selfish game is that players cheat to maximize their individual outcomes. There are a lot of work in this context. Dubins and Freedman [2] showed that, when GS-M is used, no man or a subset of men can misstate their true preference and thereby improve the outcome for all its members. On the other hand, women can manipulate to get better partners when GS-M is used (see [6] for example). Teo *et al.* [13] considered a manipulation by a single woman, and gave an $\mathcal{O}(n^3)$ time algorithm that computes the best matching partner of w who is attainable by applying GS-M to one of the $n!$ permutations of her preference list.

The above approach assumes that the stated preference lists are the same as the true preference lists. As observed above, however, this may be true for men but not for women; a woman w may use a list $Q(w)$ in the game, which is different from her true preference list $P(w)$, in the hope of obtaining a better partner than she obtains when using $P(w)$. This motivates us to consider the case where there are true preferences $\mathbf{P} = (P(M), P(W))$ and stated strategy $\mathbf{Q} = (P(M), Q(W))$, where $P(M)$ and $P(W)$ are the sets of true preference lists of men and women, respectively, and $Q(W)$ is the set of stated preference lists of women. Our aim is to check if $\mathbf{Q}$ is *Nash equilibrium*. A strategy $\mathbf{Q}$ is said to be a Nash equilibrium with respect to strategy $\mathbf{P}$, if no single player can get a strictly better partner (in terms of the true preference list in $\mathbf{P}$) by changing his/her list in $\mathbf{Q}$ while all others use their respective lists in $\mathbf{Q}$.

In this case, we need to consider another factor. Suppose that a woman w, with the true preference $P(w)$ and the stated preference $Q(w)$, may be able to obtain a better partner by using $Q'(w)$ than the one she obtains when $Q(w)$ is used. However, if the resulting matching (which is of course stable with respect to the used preferences) is unstable with respect to $\mathbf{P}$, the matching may be broken and w may lose a partner, so we cannot say this manipulation successful. Therefore, for w's manipulation to be successful, we have to add the condition that the resulting matching is stable with respect to $\mathbf{P}$.

Model. In this article, we study **SMP** in the following game theoretic model. Our game consists of four elements,

- **(i)** a true strategy $\mathbf{P}$,
- **(ii)** an arbitrary stated strategy $\mathbf{Q}$, and
- **(iii)** two different notions of stability for $\mathbf{Q}$:
  - **(a)** one based on the absence of $\mathbf{P}$-blocking pairs, and
  - **(b)** the other based on Nash equilibrium with respect to $\mathbf{P}$.

In this paper, we incorporate all these elements in our model, making it the most general model for stable marriage problems. Throughout this paper, we will *only* deal with strategies that contain the complete and strict preference lists of every man and woman, and so the misstated preferences can only be a permutation of the true preference list.

Men	$P(M)$				Women	$P(W)$				$Q(W)$			
$1:$	a	b	c	d	$a:$	2	3	$\underline{1}$	4	$\underline{3}$	4	1	2
$2:$	b	a	c	d	$b:$	1	$\underline{2}$	3	4	$\underline{1}$	2	3	4
$3:$	b	c	a	d	$c:$	2	$\underline{3}$	4	1	$\underline{2}$	3	4	1
$4:$	a	d	b	c	$d:$	$\underline{4}$	1	2	3	$\underline{4}$	1	2	3

True strategy $\mathbf{P} = (P(M), P(W))$ and stated strategy $\mathbf{Q} = (P(M), Q(W))$.

Figure 1 The men-optimal $\mathbf{Q}$-stable matching is not $\mathbf{P}$-stable, has a blocking pair $(2, a)$.

For a given a strategy $\mathbf{Q} = (P(M), Q(W))$, $\mu_{\mathbf{Q}}$ denotes its men-optimal stable matching. Also, for a player a, we define $\mathscr{S}_{(\mathbf{Q},a)} = \{(Q(-a), Q'(a)) \mid Q'(a)$ is a permutation of $Q(a)\}$ to be the family of strategies obtained by changing the list of a in $\mathbf{Q}$, while all others retain their respective lists in $\mathbf{Q}$. A strategy $\mathbf{Q}$ with respect to the (true) strategy $\mathbf{P}$ is said to be *totally stable* if (a) $\mu_{\mathbf{Q}}$ is $\mathbf{P}$-stable, and (b) for any woman $w \in W$, if $\mathbf{Q}' \in \mathscr{S}_{(\mathbf{Q},w)}$, then either $\mu_{\mathbf{Q}'}$ is not $\mathbf{P}$-stable or $\mu_{\mathbf{Q}}(w) \succeq \mu_{\mathbf{Q}'}(w)$ in $P(w)$, i.e., $\mathbf{Q}'$ yields no better partner for w in terms of $\mathbf{P}$. In plain words, there is no woman who can improve her outcome by changing her list in $\mathbf{Q}$, while the resulting matching is $\mathbf{P}$-stable.

There is a long history of research on the stability of the first kind (discussed while introducing our model), that is matchings that have no blocking pair with respect to a given (fixed) strategy. This includes a beautiful mathematical structure that describes the entire set of stable matchings (see [6], e.g.). However, this knowledge is only applicable when there is only *one* strategy (i.e., $\mathbf{P} = \mathbf{Q}$, in our model). The situation is considerably different when $\mathbf{P} \neq \mathbf{Q}$.

We illustrate the difference between the cases $\mathbf{P} = \mathbf{Q}$ and $\mathbf{P} \neq \mathbf{Q}$ by way of a small example presented in Figure 1. Strategies $\mathbf{P}$ and $\mathbf{Q}$ differ only in the list of a, and the men-optimal $\mathbf{P}$-stable matching is $\mu_{\mathbf{P}} = \{(1, a), (2, b), (3, c), (4, d)\}$. Note that the men-optimal $\mathbf{Q}$-stable matching $\mu_{\mathbf{Q}} = \{(1, b), (2, c), (3, a), (4, d)\}$ is preferred by a to $\mu_{\mathbf{P}}$, in terms of both $\mathbf{P}$ and $\mathbf{Q}$, since $3 \succ 1$ in $P(a)$ and $Q(a)$. By definition, $\mu_{\mathbf{Q}}$ is $\mathbf{Q}$-stable, but as evidenced by the blocking pair $(2, a)$, it is not $\mathbf{P}$-stable.

In comparison, our knowledge about the stability of the second kind, mentioned in our model, is considerably less. As mentioned earlier, it has been known for a long time that when GS-M is used, stating their true preferences is the best strategy for men, but a woman may be able to obtain a better matching partner by manipulating her true list. All these improvements are in terms of the manipulating player's true list. The initial research into the strategic manipulation by women primarily dealt with strategies that were obtained by truncating a player's true list. The possibility of manipulation by permuting the true list (assumed to contain all n players of the opposite kind) has been known for decades (see [6, pg 65]), but to the best of our knowledge, its time complexity was not analysed until Teo *et al.* [13], who gave an $\mathcal{O}(n^3)$ time algorithm to compute the best matching partner of a given woman w. An immediate corollary is that we can test whether the true strategy $\mathbf{P}$ is itself a Nash equilibrium in time $\mathcal{O}(n^4)$. However, [13] does not discuss the stability of the matching obtained by manipulation.

In our case, the situation becomes much more complicated since we have to consider two distinct strategies $\mathbf{P}$ and $\mathbf{Q}$, their respective stabilities, and the interaction between them. We illustrate this scenario by way of a small example in Figure 2. The men-optimal $\mathbf{Q}$-stable matching $\mu_{\mathbf{Q}}$ matches $(1, a)$. The algorithm from [13] applied to a and $\mathbf{Q}$ yields $Q'(a) = [3, 4, 1, 2]$. But, as shown in the earlier example, $(2, a)$ is a $\mathbf{P}$-blocking pair. Thus, seeking efficient algorithms for testing the total stability of a stated strategy seems to be a

Men	$P(M)$				Women	$P(W)$				$Q(W)$			
1 :	a	b	c	d	a :	2	3	$\underline{1}$	4	3	2	$\underline{1}$	4
2 :	b	a	c	d	b :	1	$\underline{2}$	3	4	1	$\underline{2}$	3	4
3 :	b	c	a	d	c :	2	$\underline{3}$	4	1	2	$\underline{3}$	4	1
4 :	a	d	b	c	d :	$\underline{4}$	1	2	3	$\underline{4}$	1	2	3

True strategy $\mathbf{P} = (P(M), P(W))$ and stated strategy $\mathbf{Q} = (P(M), Q(W))$

Figure 2 Algorithm in [13] applied to $\mathbf{Q}$ yields a matching that is not $\mathbf{P}$-stable.

nice algorithmic challenge in the field of matchings under preferences.

By combining [13] and [12, Thm 4.16], we can obtain a polynomial time algorithm to test the total stability when $\mathbf{P} = \mathbf{Q}$. In other words our problem is polynomial time tractable for the special case of $\mathbf{P} = \mathbf{Q}$. The main goal of this paper is to extend this tractability result to the most general setting of $\mathbf{P} \neq \mathbf{Q}$.

1.1 Our Contribution

We show that total stability can be tested in polynomial time. As stated in the Introduction, men do not have any incentive to misstate their true preference ([2] and [12, Theorem 4.10]); consequently, our analysis only considers manipulations by women. Specifically, for two strategies $\mathbf{P} = (P(M), P(W))$ and $\mathbf{Q} = (P(M), Q(W))$ containing complete lists for every man and woman, with $\mathbf{P}$ assumed to be the true strategy, our output should be **No** if there is a woman w and a strategy $\mathbf{Q}' = (Q(-w), Q'(w)) \in \mathscr{S}_{(\mathbf{Q},w)}$ (the family of strategies derived from $\mathbf{Q}$ where only w permutes her preference list) such that (a) $\mu_{\mathbf{Q}'}$ is $\mathbf{P}$-stable, and (b) w obtains a strictly better partner in $\mu_{\mathbf{Q}'}$, in terms of $P(w)$ (notationally expressed as $\mu_{\mathbf{Q}'}(w) \succ \mu_{\mathbf{Q}}(w)$, in $P(w)$). If there is no such strategy $\mathbf{Q}' \in \mathscr{S}_{(\mathbf{Q},w)}$ for any woman w, then the answer should be **Yes**.

The obvious brute-force method is to check all the $n!$ permutations as the list $Q'(w)$; once $Q'(w)$ is fixed, computing $\mu_{\mathbf{Q}'}$ and checking if it is $\mathbf{P}$-stable can be done in $\mathcal{O}(n^2)$ time. For a polynomial time algorithm, we need to do the following two computations without examining all the $n!$ permutations for $Q'(w)$: (i) obtaining all possible (by changing only her list) partners of w who are better (in terms of P) than w's partner in $\mu_{\mathbf{Q}}$, and (ii) for a man m found in (i), we need to search for a permutation $Q'(w)$ such that w is matched to m and $\mu_{\mathbf{Q}'}$ is $\mathbf{P}$-stable.

We wish to point out that the result in [13] is not enough for the first task, since it only gives the best partner in terms of $\mathbf{Q}$, and the matching outcome may not be $\mathbf{P}$-stable in general, as depicted by Figure 1. Note that if a is satisfied with the second best partner, 2, then she could use a list $[2, 3, 4, 1]$ and the resulting matching, $(a, 2), (b, 1), (c, 3), (d, 4)$, is now $\mathbf{P}$-stable. Thus it does not suffice to merely detect the best possible partner of a. Also, note that potential partners of women other than w may be arbitrary and there can be (exponentially) many lists for w that result in matching w to m, some of which may yield a $\mathbf{P}$-stable matching, while others may not, even if m is fixed as a target.

The basic idea of our algorithm is as follows: For task (ii), we obtain a result (Theorem 5) which proves that if two permutations $Q'_1(w)$ and $Q'_2(w)$ are available as $Q'(w)$ (both matching w to m), then $\mu_{\mathbf{Q}'_1} = \mu_{\mathbf{Q}'_2}$. Hence, if any one list gives a $\mathbf{P}$-stable outcome, then so do the others. Thus, for algorithmic purposes, it is enough to consider *any* arbitrary single permutation as $Q'(w)$. We believe that this result is interesting in its own right, and could be useful in other scenarios. For task (i) we can use an algorithm based on the same idea as [13], but its correctness proof is quite different from the original one, that heavily relies on

the fact that they are only interested in the best manipulated partner of w. For our purpose, we need to detect all attainable partners.

1.2 Related Work

In addition to [13] that we have discussed in details, there are other relevant works that we can point to.

For a stated strategy $\mathbf{Q}$, Dubins and Freedman [2] proved that there is no coalition C of men who have a manipulation strategy $\mathbf{P}' = (P(-C), P'(C))$, so that the outcome is $\mathbf{P}'$-stable and is strictly better than $\mu_{\mathbf{P}}$ in terms of $\mathbf{P}$, for each $m \in C$. Demange *et al.* [1] extend this result to include women in the coalition C, showing that there is no $\mathbf{P}'$-stable matching μ' such that every player $a \in C$ prefers $\mu'(a)$ (in terms of $P(a)$) to his/her partner in *every* $\mathbf{P}$-stable matching.

For truncation strategies, it was shown by Gale and Sotomayor [5] that if there are at least two $\mathbf{P}$-stable matchings, then there is a woman w who has a unilateral manipulation strategy $\mathbf{Q}' \in \mathscr{S}_{(\mathbf{Q},w)}$ that gives a strictly better outcome than $\mu_{\mathbf{P}}$. If $C = W$, then there is a truncation strategy $\mathbf{P}' = (P(M), P'(W))$ such that $\mu_{\mathbf{P}'}$ is the women-optimal $\mathbf{P}$-stable matching. Considerable work on truncation strategies have been undertaken (see [3, 11] for motivations and applications). In fact, up until the late 1980s, analyses of manipulation strategies of women centred almost exclusively around truncation strategies.

Immorlica and Mahidian [7] show that with high probability, truthfulness is the best strategy for any individual player, assuming everybody else is being truthful as well. In their model, the men's preference lists may have ties but the lengths are bounded by a constant, and are drawn from an arbitrary probability distribution, while the women's lists are arbitrary and complete.

Kobayashi and Matsui [8] consider the possibility that a coalition C of women have a manipulation strategy $\mathbf{P}' = (P(M), P'(W))$ containing complete lists, such that $\mu_{\mathbf{P}'}$ yields specific partnerships for the members of C. The situation manifests in two specific forms, depending on the nature of the input. In the first case, the input consists of the complete lists of all men, a partial matching (some agents may be unmatched) μ', and complete lists of the subset of women who are unmatched in μ', denoted by $W \setminus C$. The problem is to test whether there exists a permutation strategy for each woman in C, such that for the combined strategy $\mathbf{P}' = (P(-C), P'(C))$, $\mu_{\mathbf{P}'}$ is a perfect matching that extends μ'. In the second case, the input consists of the lists of all men, a perfect matching μ, and lists for women in $W \setminus C$. The problem is to test if there are permutation strategies for the women in C such that strategy $\mathbf{P}' = (P(-C), P'(C))$ yields $\mu'_{\mathbf{P}} = \mu$. They present polynomial-time algorithms for both problems.

Pini *et al.* [9] show that for an arbitrary instance of **SMP**, there is a stable matching mechanism for which it is NP-hard to find a manipulation strategy.

Roth [10] had shown that if a strategy $\mathbf{Q} = (P(M), Q(W))$ is a Nash equilibrium with respect to $\mathbf{P}$, then the matching $\mu_{\mathbf{Q}}$ is $\mathbf{P}$-stable. The proof discussed in [12] allows women to truncate their preference lists as a means of manipulation. This result holds even if all players are restricted to using complete lists in their true, stated and manipulated strategies. We use this result without a proof, as it is similar to the second approach described in [12, pg 101].

2 Preliminaries

We will always use M to denote the set of n men $\{m_1, m_2, \ldots, m_n\}$ and W the set of n women $\{w_1, w_2, \ldots, w_n\}$. Our matching mechanism is the men-proposing Gale-Shapley algorithm

(GS-M in short), which proceeds as follows: On the men's side, a man who is not yet matched to a woman, *proposes* to the woman who is at the top of his current list, which is obtained by removing all the women who have already rejected him. On the woman's side, when a woman w receives a proposal from a man m, she *accepts* the proposal if it is her first proposal, or if she prefers m to her current partner m'. If w prefers her current partner m' to m, then w *rejects* m. If m is rejected by w, then m will start issuing proposals, and this process continues until there is no man left who is unmatched. For more details, see [6]. At any stage of GS-M, if there are two or more unmatched men, then we set the convention that in this group the man with the smallest index is the first to propose. This removes the possibility of arbitrary tie-breaking, and thus, makes the algorithm purely deterministic.

A *strategy* $\mathbf{Q}$ is a set of *preference lists* (or simply *lists*) of all the men in M and all the women in W. For a person x in $M \cup W$, $Q(x)$ denotes the x's list in the strategy $\mathbf{Q}$. For a given strategy $\mathbf{Q}$, suppose that *only* w changes her list from $Q(w)$ to $Q'(w)$. We denote the resulting strategy by $\mathbf{Q}' = (Q(-w), Q'(w))$, and use $\mathscr{S}_{(\mathbf{Q},w)}$ to denote the family of all such strategies $\mathbf{Q}'$. Note that all lists considered in this article are complete, i.e., they are permutations of n men or n women.

Let $\mathbf{Q}$ be a strategy. If w prefers m_1 to m_2 in $Q(w)$, then we write "$m_1 \succ m_2$ in $Q(w)$". We use $m_1 \succeq m_2$ if $m_1 \succ m_2$ or $m_1 = m_2$. Let μ be a (perfect) matching between M and W. Then $\mu(p)$ denotes the partner of a person p. A pair (m, w) of a man and a woman is called a $\mathbf{Q}$-*blocking pair* if $w \succ \mu(m)$ in $Q(m)$ and $m \succ \mu(w)$ in $Q(w)$. We say that μ is $\mathbf{Q}$-*stable* if there is no $\mathbf{Q}$-blocking pair.

For a strategy $\mathbf{Q}$, $\mu_{\mathbf{Q}}$ denotes the man-optimal stable matching, computed by the Gale-Shapley algorithm. If a man m proposes to a woman w during this procedure, then we say that m is *active* in $Q(w)$ (formally speaking we should say m is active in $Q(w)$ during the computation of $\mu_{\mathbf{Q}}$, but for the sake of brevity, we will omit strategy $\mathbf{Q}$ when it is obvious from the context.)

Recall that a woman w changes her list $Q(w)$ for the purpose of manipulation. For a subset $M' \subseteq M$, let I be an ordering of men in M'. Then, $Q(I; w)$ denotes a permutation of $Q(w)$, where the men in M' are at the front in the order in which they appear in I. An ordered (sub)list, such as I, is called a *tuple*, and for any given tuple I, we define $Q(I; w) = [I, Q(w) \backslash I]$. For example, if $Q(w) = [1, 2, 3, 4, 5, 6]$ and $I = [5, 2]$, then $Q(I; w) = [5, 2, 1, 3, 4, 6]$. Now we are ready to introduce our main concept.

We are given a strategy $\mathbf{Q}$ and a *true* strategy $\mathbf{P}$. Then for a woman $w \in W$, a strategy $\mathbf{Q}' \in \mathcal{Q}_w$ is said to be a *unilateral manipulation strategy* of w, if $\mu_{\mathbf{Q}'}(w) \succ \mu_{\mathbf{Q}}(w)$ in $P(w)$, i.e., w strictly prefers the outcome of $\mu_{\mathbf{Q}'}$ to $\mu_{\mathbf{Q}}$ with respect to her true preference/strategy. If, furthermore, $\mu_{\mathbf{Q}'}$ is a $\mathbf{P}$-stable matching, then $\mathbf{Q}'$ is said to be a $\mathbf{P}$-*stable manipulation strategy* of w. A strategy $\mathbf{Q}$ is said to be totally stable if there does not exist a $w \in W$ who has a $\mathbf{P}$-stable manipulation strategy $(Q(-w), Q'(w)) \in \mathcal{Q}_w$. In this paper, we consider the following problem.

Problem: Total Stability
Input: True strategy $\mathbf{P} = (P(M), P(W))$ and stated strategy $\mathbf{Q} = (P(M), Q(W))$
Question: Is $\mathbf{Q}$ totally stable?

3 Listing active men

Now our goal is to design an algorithm that, for two given strategies, a stated strategy $\mathbf{Q}$ and a true strategy $\mathbf{P}$, answers if $\mathbf{Q}$ is totally stable. To do so, we first design an algorithm that

Algorithm 1: $\mathcal{A}(\mathbf{Q}, w)$ [13]

Input: Strategy $\mathbf{Q} = (P(M), Q(W))$, and a woman $w \in W$
Output: Sets $N_w(\mathbf{Q}) = \{m \in M \mid \exists\, \mathbf{Q}' \in \mathscr{S}_{(\mathbf{Q},w)}$ that yields $\mu_{\mathbf{Q}'}(w) = m\}$, and
$L_w(\mathbf{Q}) = \{Q'(m; w) \mid m \in N_w(\mathbf{Q}),\ \mathbf{Q}' = (Q(-w), Q'(m; w))$ yields $\mu_{\mathbf{Q}'}(w) = m\}$

Let x_1 be the first active man in $Q(w)$
Let $N \leftarrow \{x_1\}$ and $L \leftarrow \{Q(x_1; w)\}$
Explore$(Q(x_1; w))$
return (N, L)

Procedure **Explore**$(Q'(x, I; w))$

Let $A \leftarrow \{$men who are active in $Q'(x, I; w)$ after $x\}$
foreach $y \in A \setminus N$ **do**
  $N \leftarrow N \cup \{y\}$ and $L \leftarrow L \cup \{Q'(y, x, I; w)\}$
  Explore$(Q'(y, x, I; w))$

outputs the set $N_w(\mathbf{Q})$ of all possible partners m of a given fixed woman w such that there is a (manipulated) strategy $\mathbf{Q}' = (Q(-w), Q'(w))$, for which the men-optimal stable matching will match w to m. By using this algorithm n times, we can obtain $N_{w_1}(\mathbf{Q}), \ldots, N_{w_n}(\mathbf{Q})$. The use of this set to prove our main result is explained in the next section.

Let us consider Algorithm 1, which is basically the same as the one given by Teo *et al.* [13]: Suppose that $Q(w) = [1, 2, 3, 4, 5, 6, 7, 8]$ and the first proposal comes from man 5. Then the algorithm adds 5 to N and calls **Explore**$(Q(5; w))$: it executes the men-proposing Gale-Shapley algorithm (GS-M, in short) after moving 5 to the front of the list $Q(w)$. In general, procedure **Explore** takes as a parameter $Q(x, I; w)$, a preference list of w. As per our notation, x is at the front of this list, followed by the sublist I and then the rest of the men, thus, defining the strategy $\mathbf{Q}' = (Q(-w), Q(x, I; w))$. **Explore**$(Q(x, I; w))$ executes GS-M for the strategy $\mathbf{Q}'$ and produces the set of men A who propose to w after x. Now for each $y \in A$, we check if y is "new" (i.e., not yet in N). If so, we add y to N and call **Explore** recursively after moving y to the top of $Q(x, I; w)$; else, we do nothing.

Since **Explore** is called only once for each man in N, its time complexity is obviously at most $n \times T(\text{GS})$, where $T(\text{GS})$ is the time complexity of one execution of GS-M, thus, overall it is $\mathcal{O}(n^3)$. The nontrivial part is the correctness of the argument, which we shall prove now.

▶ **Theorem 1.** *For a strategy* $\mathbf{Q}$ *and a woman* $w \in W$*, Algorithm 1 produces* $N = \{m \in M \mid \exists \mathbf{Q}' \in \mathcal{Q}_w,\ s.t.\ \mu_{\mathbf{Q}'}(w) = m\}$ *and for each* $m \in N$*, a list* $Q(m, I; w)$ *such that, for some partial list* I*,* m *is active in* $Q(I; w)$*.*

Proof. Let $Q'(w)$ be an arbitrary permutation of n men and $\mathbf{Q}'$ the strategy $(Q(-w), Q'(w))$. It is enough to prove if a man $x \in M$ proposes to w during the computation of $\mu_{\mathbf{Q}'}$ (i.e., x is active in $Q'(w)$), then x is added to N during the execution of Algorithm 1.

Here we need two new definitions: Suppose that $x_1, x_2, \ldots, x_t$ is a sequence of men who proposed to w (in this order) during the computation of $\mu_{\mathbf{Q}'}$. Then this sequence is called an *active sequence* for $Q'(w)$, denoted by $\text{AS}'(w)$. Also define $y_1, y_2, \ldots, y_s$ as a maximal subsequence of $\text{AS}'(w)$ such that $y_1 = x_1$ and for $i \geq 2$, y_i is the first element after y_{i-1} such that $y_i \succ y_{i-1}$ in $Q'(w)$. This is called the *increasing active subsequence* for $Q'(w)$ and is denoted by $\text{IAS}'(w)$. As an example, let $Q'(w) = [1, 2, 3, 4, 5, 6, 7, 8, 9]$

and $\text{AS}'(w) = 5, 6, 3, 4, 2, 8$. Then $\text{IAS}'(w) = 5, 3, 2$. Now consider a different list $Q''(w) = [1, 2, 3, 5, 9, 8, 4, 6, 7]$, thus, $Q'(w) \neq Q''(w)$. However, we can observe that the active sequence and the increasing active subsequence for $Q''(w)$ are identical to those of $Q'(w)$, for the following reasons. The lists in $\mathbf{Q}'$ and $\mathbf{Q}''$ are the same except that of w's, so the first proposal for w must come from the same man regardless of w's list. Since the man 5 is accepted by w in both executions, the next proposal should also be from the same man 6. Now since $5 \succ 6$ in both $Q'(w)$ and $Q''(w)$, 6 is rejected in both $Q'(w)$ and $Q''(w)$ and thus, the next proposal must also be same, and so on. This observation leads us to the following lemma.

▶ **Lemma 2.** *For strategies $\mathbf{Q}', \mathbf{Q}'' \in \mathcal{Q}_w$, let $x_1, x_2, \ldots, x_p$ and $u_1, u_2, \ldots, u_q$ denote the active sequences for $Q'(w)$ and $Q''(w)$ respectively, and let $y_1, y_2, \ldots, y_s$ and $v_1, v_2, \ldots, v_t$ denote the corresponding increasing active subsequence. Then, the following conditions must hold.*

(a) *$x_1 = y_1 = u_1 = v_1$.*

(b) *For an arbitrary l ($l \leq p$ and $l \leq q$), we consider the prefixes of the active sequences up to position l and the prefixes of the corresponding increasing active subsequences, denoted by $y_1, ..., y_j$ and $v_1, ..., v_j$. Then, if $x_i = u_i$ for all $i \leq l$ and $y_k = v_k$ for all $k \leq j$, then $x_{l+1} = u_{l+1}$.*

Proof. By definition, $x_1 = y_1$ and $u_1 = v_1$. Recall that all the lists in Q' and Q'' are the same except those for w. Furthermore, we use a fixed tie-breaking protocol in the deterministic GS algorithm. Hence, $x_1 = u_1$ follows directly.

To prove condition (b), let $y_2 = x_{i_1+1}, y_3 = x_{i_2+1}, \ldots$, and so on. Then we can write $\text{AS}'(w)$ as follows, where $x_2, \ldots, x_{i_1}, x_{i_1+2}, \ldots, x_{i_2}, \ldots$ may be empty.

$$\text{AS}'(w) = y_1, x_2, \ldots, x_{i_1}, y_2, x_{i_1+2}, \ldots, x_{i_2}, \ldots, y_j, x_{i_{j-1}+2}, \ldots, x_l, x_{l+1}, \ldots$$

Now one can see that y_1 is accepted, all of $x_2 \ldots, x_{i_1}$ are rejected since they are after y_1 in the list by definition. This continues as y_2 is accepted, $x_{i_1+2}, \ldots, x_{i_2}$ rejected, and so on. Now, consider $\text{AS}''(w)$, depicted below.

$$\text{AS}''(w) = v_1, u_2, \ldots, u_{i_1}, v_2, u_{i_1+2}, \ldots, u_{i_2}, \ldots, v_j, u_{i_{j-1}+2}, \ldots, u_l, u_{l+1}, \ldots$$

By the assumption, these two sequences are identical up to position l, so acceptance or rejection for each proposal follows identically, as discussed above. Therefore, the *configuration* (see below) of GS-M for $\mathbf{Q}'$ at the moment when x_l proposes to w and the configuration for $\mathbf{Q}''$ when u_l proposes to w are exactly the same. A configuration consists of (i) the lists of all men (recall that some entries are removed when proposals are rejected), (ii) the set of single men, and (iii) the current temporal matching partner of each woman. (Formally this should be shown by induction, but it is straightforward and is omitted). Also the acceptance/rejection for x_l and u_l is the same. Thus in either case, the execution of the (deterministic) GS-M is exactly the same for $\mathbf{Q}'$ and $\mathbf{Q}''$ until w receives proposal from x_{l+1} and u_{l+1}, respectively. Hence, x_{l+1} and u_{l+1}, should be equal and the lemma is proved. ◀

Now let us look at the execution sequence of Algorithm 1 while comparing it with the execution sequence of GS-M on $\mathbf{Q}'$. Let the active sequence and increasing active sequence for $\mathbf{Q}'$ be $\text{AS}'(w) = x_1, x_2, \ldots, x_p$ and $\text{IAS}'(w) = y_1, y_2, \ldots, y_s$, respectively. By Lemma 2, the first proposal to w is always y_1, so the algorithm starts with **Explore**$(Q(y_1; w))$ (we simply say the algorithm invokes $Q(y_1; w)$), and $N = \{y_1\}$, at the very beginning.

Now we note that it is quite easy to see that the active sequence for $Q(y_1; w)$ should be $y_1, x_2, \ldots, x_{i_1}, \ldots$, i.e., it should be identical to that of $Q'(w)$ up to the position i_1, with i_1 defined as in the proof of Lemma 2. The reason is as follows. We already know the first active man is always y_1 and that is also the first symbol in the increasing active sequence of both. Thus we can use Lemma 2 to conclude that the second symbol should also be the same, since x_2 is not in $\mathrm{IAS}'(w)$, meaning that it is rejected, which is also the same in $Q(y_1; w)$ since y_1 is at the top of the sequence. Thus the third symbol is the same in both and so on up to position i_1. Then the next symbol in $\mathrm{AS}'(w)$ is y_2 and it is also active in $Q(y_1; w)$, meaning $Q(y_2, y_1; w)$ is invoked by the algorithm. (The algorithm also invokes $Q(x_2, y_1; w)$, $Q(x_3, y_1; w), \ldots, Q(x_{i_1}, y_1; w)$, but these are not important for us at this moment.)

We again consider the active sequence for $Q(y_2, y_1; w)$ and by the same argument presented earlier, we can conclude that it is identical to $\mathrm{AS}'(w)$ up to position i_2 and so y_3 is found to be an active man. Hence, $Q(y_3, y_2, y_1; w)$ is invoked if y_3 was not already present in N. Continuing like this, we note that if $Q(y_s, y_{s-1}, \ldots, y_1; w)$ is invoked, then we are done since its active sequence is identical to that of $Q'(w)$. However, this case happens only if each y_i $(2 \leq i \leq s)$, is a brand new active man found during the invocation of $Q(y_{i-1}, \ldots, y_1; w)$. If one of them is not new then the subsequent lists are not invoked, and yet, we are assured due to Lemma 3 that Algorithm 1 will detect all the active men in $Q'(w)$.

Lemma 3 is rather surprising and may be of independent interest. For two lists $Q'(w)$ and $Q''(w)$, that are distinct and arbitrary orderings on men, we assume nothing about the execution of GS-M on the two lists except that a particular man x is active in both lists. Yet, we are able to show that a man who proposes to w when $Q'(x; w)$ is used must also propose when $Q''(x; w)$ is used. This result allows us to focus solely on active men that have been discovered in the current invocation of **Explore**, thereby restricting the number of recursion steps to $\mathcal{O}(n)$.

▶ **Lemma 3.** *For two distinct strategies $\mathbf{Q}'$ and $\mathbf{Q}''$ in $\mathscr{S}_{(\mathbf{Q},w)}$, suppose that x is active in both $Q'(w)$ and $Q''(w)$. Then a man who is active in $Q'(x; w)$ is also active in $Q''(x; w)$.*

Proof. Consider the strategies $\mathbf{Q}_1 = (Q'(-w), Q'(x; w))$ and $\mathbf{Q}_2 = (Q''(-w), Q''(x; w))$. Let y denote a man who is active in $Q'(x; w)$ but not in $Q''(x; w)$. Then, $\mu_{\mathbf{Q_2}}(y) \succ w \succ \mu_{\mathbf{Q_1}}(y)$ in $P(y)$.

Note that $\mu_{\mathbf{Q_1}}(w) = x$ and $\mu_{\mathbf{Q_2}}(w) = x$. Clearly, any $\mathbf{Q}_1$-blocking pair in $\mu_{\mathbf{Q_2}}$ must involve w, as otherwise it would also be a $\mathbf{Q}_2$-blocking pair. However, since x is at the top of w's list in $\mathbf{Q}_1$, w cannot be in a $\mathbf{Q_1}$-blocking pair, implying that $\mu_{\mathbf{Q_2}}$ is a $\mathbf{Q}_1$-stable matching. Since $\mu_{\mathbf{Q_1}}$ is the men-optimal stable matching for $\mathbf{Q}_1$, we have that $\mu_{\mathbf{Q_1}}(y) \succeq \mu_{\mathbf{Q_2}}(y)$ in $P(y)$. This contradicts the fact we have shown earlier, and hence y must be active in $Q''(x; w)$. ◀

The next lemma completes the proof. We give one more notation, where y_is are men defined in Lemma 2.

$$Q_1(w) = Q_1(y_1; w), \text{ and } Q_{j+1}(w) = [y_{j+1}, Q_j(w) \setminus \{y_j\}], \text{ for } 1 \leq j \leq s-1.$$

▶ **Lemma 4.** *For each j $(1 \leq j \leq s)$, the algorithm invokes $Q(y_j, I; w)$ for some tuple I, and each man in $\{x_{i_{j-1}+2}, \ldots, x_{i_j}, y_{j+1}\}$ is active in it.*

Proof. We prove this result by induction on y_i. The base case has been already proved, since for y_1, it has been shown earlier that $Q(y_1; w)$ is invoked at the beginning and every man in $\{x_2, \ldots, x_{i_1}, y_2\}$ is active in $Q(y_1; w)$ after y_1.

Suppose that the induction hypothesis holds for y_t, where $t \leq s-2$, i.e., for some tuple I, $Q(y_t, I; w)$ is invoked, and each man in $\{x_{i_{t-1}+2}, \ldots, x_{i_t}, y_{t+1}\}$ is active in it. We

will complete the proof by showing that the hypothesis holds for y_{t+1}. If y_{t+1} is "new", i.e., it is added to N during the invocation of $Q(y_t, I; w)$, then $Q(y_{t+1}, y_t, I; w)$ is invoked subsequently. Using the fact that y_{t+1} is active in both $Q_{t+1}(w)$ and $Q(y_t, I; w)$, and all the men in $\{x_{i_t+2}, \ldots, y_{t+2}\}$ are active in $Q_{t+1}(w)$ after y_{t+1}, Lemma 3 applied to each of them implies that they are also active in $Q(y_{t+1}, y_t, I; w)$. Hence, for this case, the hypothesis is proved for y_{t+1}.

If y_{t+1} is already in N when $Q(y_t, I; w)$ is invoked, then for some tuple I', y_{t+1} should have been added to N during the invocation of $Q(I'; w)$. Thus, $Q(y_{t+1}, I'; w)$ would have been invoked prior to $Q(y_t, I; w)$. Using the same argument (on $Q_{t+1}(w)$ and $Q(y_{t+1}, I'; w)$) that we used for the earlier case, we conclude that even for this case, the hypothesis holds for y_{t+1}. ◀

Thus, we have shown that all the men in $\mathrm{AS}'(w)$ are active somewhere during the execution of the algorithm and thus, all are present in N at the end of the execution. This completes the proof of Theorem 1. ◀

4 Algorithm to test if a strategy is totally stable

In this section, we consider the problem of deciding, for a given true strategy $\mathbf{P}$ and a stated strategy $\mathbf{Q}$, whether $\mathbf{Q}$ is totally stable. We show that this problem is solvable in time $\mathcal{O}(n^4)$. Algorithm 2 uses Algorithm 1 as a subroutine.

Algorithm 2: Algorithm for Total Stability.

Input: True strategy $\mathbf{P}$, stated strategy $\mathbf{Q}$, and the set of women W.
Output: Answers "Yes", if $\mathbf{Q}$ is totally stable, else "No".

```
foreach w ∈ W do
    Run Algorithm 1 on input (Q, w) to obtain N_w(Q) and L_w(Q)
    Let 𝒩̃ ← {m ∈ N_w(Q) s.t. w prefers m to μ_Q(w), in P(w)}
    foreach m ∈ 𝒩̃ do
        Let Q(m, I; w) ∈ L_w(Q) be the list that yields (m, w) as a matched pair
        Let μ be the men-optimal (Q(−w), Q(m, I; w))-stable matching
        if μ is P-stable then
            return "No"
return "Yes"
```

The following result is of independent interest as it implies that Algorithm 1 on the input $(\mathbf{Q}, w)$ generates *all matchings* that can be attained by changing w's preference list. In particular, it answers if for any given $w \in W$, there exists a strategy $\mathbf{Q}' \in \mathscr{S}_{(\mathbf{Q},w)}$ such that $\mu_{\mathbf{Q}'}(w) \succ \mu_{\mathbf{Q}}(w)$ in $P(w)$, and the matching $\mu_{\mathbf{Q}'}$ is $\mathbf{P}$-stable matching. As a consequence, using Algorithm 1 as a subroutine, Algorithm 2 is able to solve Total Stability.

▶ **Theorem 5.** *For any (fixed) man $\overline{m}$, if there exists a permutation $Q'(w)$ of a woman w's list $Q(w)$ such that the strategy $\mathbf{Q}' = (Q(-w), Q'(w))$ yields a matching that matches w to $\overline{m}$, then that matching is unique.*

Proof. Suppose that $Q'(w)$ (defined in the theorem) is an arbitrary strategy of w to obtain $\overline{m}$, and let μ' denote the men-optimal $\mathbf{Q}'$-stable matching. Algorithm 1 applied to input $(\mathbf{Q}, w)$

computes a list $Q(\overline{m}, I; w)$ such that w attains $\overline{m}$ by the strategy $\mathbf{Q}^* = (Q(-w), Q(\overline{m}, I; w))$. Note that $\overline{m}$ appears at the front of the list $Q^*(w) = Q(\overline{m}, I; w)$. Let μ^* be the men-optimal $\mathbf{Q}^*$-stable matching. Our goal is to show that $\mu' = \mu^*$.

▶ **Claim 6.** *For each m, $\mu^*(m) \succeq \mu'(m)$ in $Q'(m)$.*

Proof. We begin by showing that μ' is $\mathbf{Q}^*$-stable. We know that μ' is $\mathbf{Q}'$-stable, and $\mathbf{Q}'$ and $\mathbf{Q}^*$ differ only in w's list. Hence, if there is a $\mathbf{Q}^*$-blocking pair in μ', then it must contain w. However, this is impossible since w is matched with $\overline{m}$, who is at the front of the list $Q^*(w)$. Therefore, μ' must be $\mathbf{Q}^*$-stable

Since μ^* is the men-optimal $\mathbf{Q}^*$-stable matching and μ' is a $\mathbf{Q}^*$-stable matching, we have that, for each man m, $\mu^*(m) \succeq \mu'(m)$ in $Q^*(m)$. Since $Q^*(m) = Q'(m)$ for each man m, the claim is proved. ◀

▶ **Claim 7.** *For each m, $\mu'(m) \succeq \mu^*(m)$ in $Q'(m)$.*

Proof. As for Claim 6, we begin by showing that μ^* is $\mathbf{Q}'$-stable. Suppose that it is not. Then there is a $\mathbf{Q}'$-blocking pair in μ^*, and it includes w for the same reason as in the proof of Claim 6. Let (m', w) denote a $\mathbf{Q}'$-blocking pair in μ^*. Then, $w \succ \mu^*(m')$ in $Q'(m')$, and $m' \succ \mu^*(w)$ in $Q'(w)$.

Using Claim 6, we have $w \succ \mu'(m')$ in $Q'(m')$, and $\mu^*(w) = \mu'(w) = \overline{m}$ by definition. Hence, (m', w) is a $\mathbf{Q}'$-blocking pair in μ', a contradiction. Again, for the same reason as in the proof of Claim 6, we can conclude that $\mu'(m) \succeq \mu^*(m)$ in $Q'(m)$ for each man m. ◀

By Claims 6 and 7, we have $\mu'(m) = \mu^*(m)$ for each man m. Thus, $\mu' = \mu^*$, completing the proof of Theorem 5. ◀

▶ **Theorem 8.** *Algorithm 2 solves* Total Stability *in $\mathcal{O}(n^4)$ time.*

Proof. Suppose that Algorithm 2 outputs "No". Then, it implies that a $\mathbf{P}$-stable manipulation strategy was found, and therefore $\mathbf{Q}$ is not totally stable. For the opposite direction, suppose that $\mathbf{Q}$ is not totally stable and there exists a woman w who has a manipulation strategy $\mathbf{Q}'$ such that $\mu_{\mathbf{Q}'}$ is $\mathbf{P}$-stable. Then, man $\mu_{\mathbf{Q}'}(w)$ is added to $\widetilde{\mathcal{N}}$ when Algorithm 1 is executed on the input $(\mathbf{Q}, w)$. By Theorem 5 the matching $\mu_{\mathbf{Q}'}$ is uniquely defined, i.e., there is a unique matching resulting from a manipulation strategy of w that results in the matched pair $(w, \mu_{\mathbf{Q}'}(w))$. Since $\mu_{\mathbf{Q}'}$ is $\mathbf{P}$-stable, Algorithm 2 will output "No." This proves the correctness of Algorithm 2.

We claim that the time complexity of Algorithm 1 is $\mathcal{O}(n^3)$. This is because the size of the set N is at most n, and is computed iteratively by executing GS-M once for each man $m \in N$. Since the running time of GS-M is $\mathcal{O}(n^2)$, the running time of Algorithm 1 is $\mathcal{O}(n^3)$. Algorithm 2 executes Algorithm 1 for each woman $w \in W$. Hence, the running time of Algorithm 2 is $\mathcal{O}(n^4)$. ◀

5 Conclusions

We leave the question of manipulation by a group of women as an avenue of further research. In particular, we would like to answer in polynomial time (1) if a stated strategy $\mathbf{Q}$ is totally stable against manipulations by a subset of women, and (2) if a given subset of women $W' \subseteq W$ have a manipulation strategy that yields a $\mathbf{P}$-stable matching and gives each of them a better partner than the one given by the stated strategy. In this article, we solved both of these problems for the special case of manipulation by one woman acting on her own.

Acknowledgments. The authors would like to thank the anonymous reviewers for their valuable comments. This work was supported by JSPS KAKENHI Grant Numbers 24500013 and 25240002.

References

1 G. Demange, D. Gale, and M. Sotomayor. A further remark on the stable matching problem. *Discrete Applied Mathematics*, 16:217–222, 1987.
2 L. E. Dubins and D. A. Freedman. Machiavelli and the Gale-Shapley algorithm. *The American Mathematical Monthly*, 88(7):485–494, 1981.
3 L. Ehlers. Truncation strategies in matching markets. *Mathematics of Operations Research*, 33(2):327–335, 2008.
4 D. Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.
5 D. Gale and M. Sotomayor. Ms. Machiavelli and the Gale-Shapley algorithm. *American Mathematical Monthly*, 92(4):261–268, 1985.
6 D. Gusfield and R. W. Irving. *The Stable Marriage Problem-Structure and Algorithm.* The MIT Press, 1989.
7 N. Immorlica and M. Mahidian. Marriage, honesty and stability. In *Proceedings of SODA'05*, pages 53–62, 2005.
8 H. Kobayashi and T. Matsui. Cheating strategies for the Gale-Shapley algorithm with complete preference lists. *Algorithmica*, 58:151–169, 2010.
9 M. S. Piny, F. Rossi, K. B. Veneble, and T. Walsh. Manipulation complexity and gender neutrality in in stable marriage procedures. *Auton. Agent Multi-Agent Systems*, 22(183–199), 2011.
10 A. E. Roth. Misrepresentation and stability in the marriage problem. *Journal of Economic Theory*, 34:383–387, 1984.
11 A. E. Roth and U. G. Rothblum. Truncation strategies in matching markets-in search of advice for participants. *Econometrica*, 67(1):21–43, 1999.
12 A. E. Roth and M. Sotomayor. *Two-Sided Matching: A Study in Game Theoretic Modeling and Analysis.* Cambridge Univ. Press, 1990.
13 C-P. Teo, J. Sethuraman, and W-P. Tan. Gale-Shapley stable marriage problem revisited: Strategic issues and applications. *Management Science*, 47(9):1252–1267, 2001.

Estimating The Makespan of The Two-Valued Restricted Assignment Problem

Klaus Jansen[1], Kati Land[*2], and Marten Maack[3]

1 **Institute of Computer Science, University of Kiel, Kiel, Germany**
kj@informatik.uni-kiel.de
2 **Institute of Computer Science, University of Kiel, Kiel, Germany**
kla@informatik.uni-kiel.de
3 **Institute of Computer Science, University of Kiel, Kiel, Germany**
mmaa@informatik.uni-kiel.de

Abstract

We consider a special case of the scheduling problem on unrelated machines, namely the Restricted Assignment Problem with two different processing times. We show that the configuration LP has an integrality gap of at most $\frac{5}{3} \approx 1.667$ for this problem. This allows us to estimate the optimal makespan within a factor of $\frac{5}{3}$, improving upon the previously best known estimation algorithm with ratio $\frac{11}{6} \approx 1.833$ due to Chakrabarty, Khanna, and Li [2].

1998 ACM Subject Classification F.2.2 Sequencing and Scheduling, G.1.6 Optimization

Keywords and phrases unrelated scheduling, restricted assignment, configuration LP, integrality gap, estimation algorithm

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.24

1 Introduction

Scheduling on unrelated machines is a problem where we are given a set J of jobs and a set M of machines, and the processing time of job $j \in J$ on machine $i \in M$ is given by p_{ij}. The task is to find an assignment $\sigma\colon J \to M$, called the schedule, that minimizes the makespan, i.e. the maximum load $\max_{i \in M} \sum_{j \in \sigma^{-1}(i)} p_{ij}$ of a machine.

Lenstra, Shmoys, and Tardos [6] proved that it is NP-hard to approximate the makespan with a factor less than 1.5. On the other hand, they presented an algorithm with approximation ratio 2. The algorithm uses a rounding procedure for the following, natural linear programming formulation, which is commonly known as the *assignment linear program (LP)*:

$$\sum_{i \in M} x_{ij} = 1 \quad \text{for each } j \in J \tag{1}$$

$$\sum_{j \in J} p_{ij} x_{ij} \leq T \quad \text{for each } i \in M \tag{2}$$

$$x_{ij} = 0 \quad \text{for each } i \in M, j \in J \text{ with } p_{ij} > T \tag{3}$$

$$x_{ij} \geq 0 \quad \text{for each } i \in M, j \in J. \tag{4}$$

Here, T is the desired makespan. We denote the above LP by $\mathrm{ALP}(T)$. It is clear that there is a schedule with makespan T if and only if $\mathrm{ALP}(T)$ has an integral solution. Note that equation 3 strengthens the otherwise intuitive formulation by forbidding fractional assignments if the whole job cannot be feasibly processed on a machine.

* Research was supported by German Research Foundation (DFG) project JA 612/15-1.

Closing this gap between approximability and inapproximability is a major open problem in scheduling theory. Since 1990, the approximation ratio was slightly improved to $2-1/|M|$ [7]. Because no substantial progress has been made for more than 20 years, the focus has shifted towards special cases of the problem. One important special case is the so called Restricted Assignment Problem, where for each job $j \in J$ there is a number p_j such that $\{p_{ij} \mid i \in M\} \subseteq \{p_j, \infty\}$. A natural interpretation is that for each job j there is a set $\mathrm{M}(j) \subseteq M$ of machines on which j may be processed, and the processing time is the same on each of these machines. The restricted assignment case may look easier than the general problem, but the inapproximability bound of $1.5-\varepsilon$ still holds, even if we further restrict that $|\mathrm{M}(j)| \leq 2$ and $p_j \in \{1,2\}$ for each job j [3].

A breakthrough was achieved by Svensson [9], who considered another, stronger LP formulation, the *configuration LP*. To introduce it, we require some notation. For each set $J' \subseteq J$ of jobs we define $\mathrm{p}(J') = \sum_{j \in J'} p_j$. We also abbreviate $\mathrm{p}(j) = \mathrm{p}(\{j\})$ for a single job j. A *configuration* for a machine i is a set $C \subseteq J$ with $\sum_{j \in C} p_{ij} \leq T$ We denote the set of all configurations for machine i that respect the target makespan T by $\mathcal{C}(i,T)$. The configuration LP $\mathrm{CLP}(T)$ is defined as

$$\sum_{C \in \mathcal{C}(i,T)} x_{i,C} \leq 1 \quad \text{for each } i \in M \tag{5}$$

$$\sum_{i \in M} \sum_{\substack{C \in \mathcal{C}(i,T) \\ j \in C}} x_{i,C} \geq 1 \quad \text{for each } j \in J \tag{6}$$

$$x_{i,C} \geq 0 \quad \text{for each } i \in M,\, C \in \mathcal{C}(i,T). \tag{7}$$

The first constraint enforces that at most one configuration is assigned to each machine, and the second constraint ensures that each job is completely assigned. Svensson [9] proved that the integrality gap of the configuration LP is at most $33/17 \approx 1.941$ for the Restricted Assignment Problem.

Usually, the *integrality gap* of an integer linear program is defined by $\sup_I \frac{\mathrm{OPT}(I)}{\mathrm{OPT}_{\mathrm{LP}}(I)}$, where $\mathrm{OPT}(I)$ and $\mathrm{OPT}_{\mathrm{LP}}(I)$ denote the optimal integer and fractional solutions of the LP. In this case however, we have a feasibility program. One therefore defines $\mathrm{OPT}(I) = \min\{T \mid \mathrm{CLP}(T) \text{ has a feasible integer solution}\}$ and $\mathrm{OPT}_{\mathrm{LP}}(I)$ analogously. Indeed, with this definition, $\mathrm{OPT}(I)$ is equal to the makespan of an optimal schedule. We will write OPT or $\mathrm{OPT}_{\mathrm{LP}}$ instead of $\mathrm{OPT}(I)$ and $\mathrm{OPT}_{\mathrm{LP}}(I)$ if the instance is clear from the context.

Even though the number of variables in $\mathrm{CLP}(T)$ may be exponentially large, one can find an approximate solution it in polynomial time via its dual [1]: If we interpret $\mathrm{CLP}(T)$ as maximizing a zero objective function, the dual is given by

$$\min \sum_{i \in M} y_i - \sum_{j \in J} z_j \tag{8}$$

$$y_i \geq \sum_{j \in C} z_j \qquad \text{for each } i \in M,\, C \in \mathcal{C}(i,T) \tag{9}$$

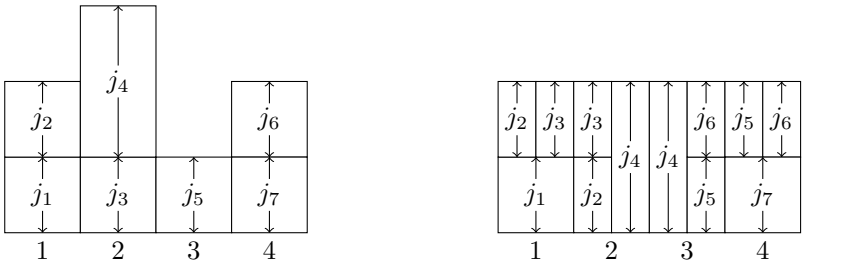
$$z_j \geq 0 \qquad \text{for each } j \in J \tag{10}$$

Finding a violated constraint of the dual is equivalent to $|M|$ knapsack problems, and we can find an approximate solution to these knapsack problems using an FPTAS. Using this FPTAS as separation oracle, a solution to $\mathrm{CLP}(T)$ can be found. This solution then may contain configurations C with $T < \mathrm{p}(C) \leq (1+\varepsilon)T$, where $\varepsilon > 0$ is the chosen precision.

Performing a binary search for the best target makespan T, we can therefore estimate $\mathrm{OPT}_{\mathrm{LP}}(I)$ within a factor $1+\varepsilon$ in polynomial time for arbitrary small $\varepsilon > 0$. Using Svenssons

Table 1 An instance with integrality gap 3/2.

Job	1	2	3	4	5	6	7
p_j	1	1	1	2	1	1	1
M(j)	1	1,2	1,2	2,3	3,4	3,4	4



(a) An optimal integral solution (b) An optimal fractional solution

Figure 1 Solutions for the instance given in Table 1.

bound on the integrality gap then allows us to estimate the optimum makespan OPT(I) within a factor of $33/17 + \varepsilon$ in polynomial time, where $\varepsilon > 0$ is again an arbitrary small constant. It is a major open problem to find a polynomial rounding procedure whose approximation guarantee matches the integrality gap.

Better results have been obtained when the instances have further restrictions. For example, if $|\mathrm{M}(j)| \leq 2$ for each $j \in J$, there is a 1.75-approximation [3]. Recently, Huang and Ott [4] gave improved algorithms for the case that the constraint $|\mathrm{M}(j)| \leq 2$ only applies to big jobs.

We will in particular investigate the case of only two types of jobs: small jobs with processing time s and big jobs with processing time b. Even in this case the integrality gap is at least $\frac{3}{2}$, as the instance given in Table 1 and its solutions depicted in Figure 1 show.

Svensson [9] proved that the integrality gap of the configuration LP in this case is at most $\frac{5}{3} + s$ if $b = \mathrm{OPT}_{\mathrm{LP}} = 1$. Kolliopoulos and Moysoglou [5] pointed out that this bound also holds for $\mathrm{OPT}_{\mathrm{LP}} < 2b$ and generalizes to $\frac{5}{3} + \frac{s}{\mathrm{OPT}_{\mathrm{LP}}}$ when $b < \mathrm{OPT}_{\mathrm{LP}} < 2b$. Note that if $\mathrm{OPT}_{\mathrm{LP}} \geq 2b$, the analysis of Lenstra, Shmoys, and Tardos [6] bounds the integrality gap of the assignment LP by 1.5, and the configuration LP is at least as strong.

Kolliopoulos and Moysoglou [5] developed a $(2 - \frac{s}{b})$-approximation for the case that b is a multiple of s. By rounding general instanes to this form when $\frac{s}{b} \geq 0.2$ and using Svenssons result when $\frac{s}{b} < 0.2$, the makespan can be estimated within a factor of 1.883. Recently, Chakrabarty, Khanna, and Li [2] found a $(2 - \frac{s}{b})$-approximation for the general case and therefore improved the estimation ratio to 1.833. They also presented a constructive algorithm with approximation ratio $2 - \delta$ for a very small $\delta > 0$.

Our Contribution

In section 2, we conduct a tighter analysis of Svensson's [9] local search algorithm that depends on the structure of the fractional solution. In particular, we distinguish the cases that $\mathrm{OPT}_{\mathrm{LP}} - b$ or $\mathrm{OPT}_{\mathrm{LP}}$ is a multiple of s. Note that either $\mathrm{OPT}_{\mathrm{LP}} - b$ or $\mathrm{OPT}_{\mathrm{LP}}$ must be a multiple of s, because a configuration of length $\mathrm{OPT}_{\mathrm{LP}}$ either contains one big job, or only small jobs. As result, we present better bounds on the integrality gap of the configuration LP. A peculiarity of these bounds is that they are piecewise linear functions with infinitely

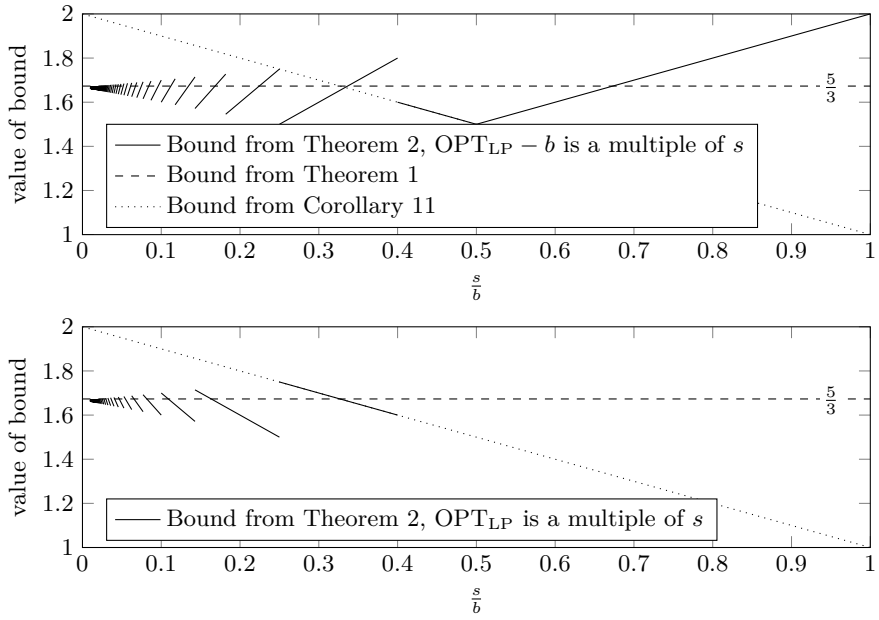


Figure 2 Comparison of bounds on the integrality gap for $\mathrm{OPT}_{\mathrm{LP}} = b$.

many discontinuities. Our bounds are largest in the case $\mathrm{OPT}_{\mathrm{LP}} = b$, for which they are depicted in Figure 2.

We improve this bound further in Sections 3 and 4 and obtain our main result:

▶ **Theorem 1.** *The integrality gap of the configuration LP for the Restricted Assignment Problem with two different processing times is at most* $\frac{5}{3}$.

This bound also allows us to estimate the optimal makespan within a factor of $\frac{5}{3} \approx 1.667$, improving upon the previously best possible ratio 1.833 [2].

The proof of Theorem 1 is split in two parts. If $\frac{s}{b} \leq \frac{1}{3}$, we study the shape of the bounds we obtained in section 2 more closely in section 3. Our idea is to modify instances for which the bound from section 2 is larger than $\frac{5}{3}$ by scaling one of the processing times s and b such that we get a better bound for the modified instance. We then bound the integrality gap of the original instance in terms of the integrality gap of the newly constructed instance.

In section 4, we present an algorithm with approximation ratio $2 - \frac{s}{b}$. The algorithm is based on solving an augmented assignment LP and rounding the solution using a technique due to Shmoys and Tardos [8]. As a corollary, the integrality gap of the augmented assignment LP and the (stronger) configuration LP is bounded by $2 - \frac{s}{b}$, which proves Theorem 1 if $\frac{s}{b} \geq \frac{1}{3}$. We should note that the approximation ratio $2 - \frac{s}{b}$ was independently obtained by Chakrabarty, Khanna, and Li [2] using different methods, but they do not use LPs, and therefore cannot derive any bound on the integrality gap. Moreover, our algorithm has an additive approximation guarantee of $\mathrm{OPT}_{\mathrm{LP}} + b - s$, which might be of independent interest.

2 Bounding the Integrality Gap by Local Search

In this section we present an improved bound on the integrality gap of the configuration LP. Our proof requires that each configuration in the optimal fractional solution contains at most one big job. To satisfy the first condition, it is sufficient (but not necessary) that

$\mathrm{OPT}_{\mathrm{LP}} < 2b$. The argumentation also works for a restricted variant of the problem where only one big job per configuration is permitted. We further distinguish whether $\mathrm{OPT}_{\mathrm{LP}} - b$ or $\mathrm{OPT}_{\mathrm{LP}}$ is a multiple of s. Note that, if $\mathrm{OPT}_{\mathrm{LP}} - b$ is not a multiple of s, then $\mathrm{OPT}_{\mathrm{LP}}$ is, and vice-versa, so at least one of the cases always holds, and both hold exactly if b is a multiple of s

We will show the following, see also Figure 2:

▶ **Theorem 2.** *Consider an instance of the resticted assignment problem with jobs of two sizes $s < b$ such that each configuration in the optimal fractional solution contains at most one big job.*

1. *If $\mathrm{OPT}_{\mathrm{LP}} - b$ is a multiple of s, then the integrality gap of the CLP is at most*
   - **(a)** $1 + \frac{(b-s)}{\mathrm{OPT}_{\mathrm{LP}}}$ *if* $\frac{2}{5} \leq \frac{s}{b} < \frac{1}{2}$ *and*
   - **(b)** $1 + \left\lceil \frac{2}{3}\left(\frac{b}{s} - 1\right)\right\rceil \frac{s}{\mathrm{OPT}_{\mathrm{LP}}}$ *otherwise.*
2. *If $\mathrm{OPT}_{\mathrm{LP}}$ is a multiple of s and $\frac{s}{b} \leq \frac{2}{5}$, then the integrality gap of the CLP is at most*
   - **(a)** $1 + \left(b - \left\lfloor \frac{b}{3s} + \frac{2}{3}\right\rfloor \cdot s\right) \cdot \frac{1}{\mathrm{OPT}_{\mathrm{LP}}}$.

An upper bound on the values given in Theorem 2 is $\frac{5}{3} + \frac{1}{3}\frac{s}{\mathrm{OPT}_{\mathrm{LP}}}$. Recall that Svenssons [9] bound generalizes to $\frac{5}{3} + \frac{s}{\mathrm{OPT}_{\mathrm{LP}}}$ for $b < \mathrm{OPT}_{\mathrm{LP}} < 2b$ [5]. To prove Theorem 2, we utilize the local search technique due to Svensson [9], but with an improved analysis.

For simplicity, we scale the processing times such that $\mathrm{OPT}_{\mathrm{LP}} = 1$ from now on. The high-level overview is as follows: we use a family $(A_R)_{R>0}$ of algorithms, where each member A_R takes a partial schedule σ, i.e. a feasible schedule for a subset $J' \subset J$ of the jobs, and a currently unscheduled job j_{new} as parameters. It should return a feasible schedule for $J' \cup \{j_{\mathrm{new}}\}$. In addition, A_R maintains the invariant that the makespan is at most $1 + R$. Given an instance of the problem for that CLP(1) is feasible, iteratively applying A_R to each job, starting with an empty schedule, yields a schedule for all jobs with makespan at most $1 + R$. Note that we cannot give a polynomial bound on the running time of this procedure, but the mere existence of the resultant schedule proves that the integrality gap is bounded by $1 + R$. The crucial point is indeed that the algorithm *successfully terminates at all*, and we can prove this if R meets certain requirements. A more detailed description of the algorithm is given in the next section.

2.1 Detailed Description of the Algorithm

The main idea is to move jobs from their current machine to another machine while maintaining the invariant that the makespan is at most $1 + R$. In the beginning, we wish to move only the unassigned job j_{new} to some machine. Suppose that all machines in $\mathrm{M}(j)$ have too much load, otherwise we are done. The algorithm then will try to reduce the load on some machine $i \in \mathrm{M}(j_{\mathrm{new}})$ by moving some job $j \in \sigma^{-1}(i)$ away from i. If such a move is again not immediately possible, the process repeats. Since we are trying to reduce the load on machine i, moving more jobs to i may be unhelpful, depending on the job's sizes. Thus the algorithm needs to store which jobs it currently tries to move and which machines it should not try to move jobs to, and it does so by the use of blockers. Whenever the algorithm decides that a move has the potential to be helpful but does not immediately lead to a schedule with makespan at most $1 + R$, a blocker is created. More formally, a move is a pair (j, i), where j is a job and $i \in \mathrm{M}(j) \setminus \{\sigma(j)\}$. We distinguish three types of moves: (j, i) is a small move if j is small, a big-to-small move if j is big and $\sigma^{-1}(i)$ contains only small jobs, and a big-to-big move if j is big and $\sigma^{-1}(i)$ contains a big job. If assigning $\sigma(j) = i$ yields a schedule of makespan at most $1 + R$, the move is called valid, otherwise it is invalid. When a potentially helpful move (j, i) is found to be invalid, a blocker B is created. B is a tuple

consisting of a machine $\mathrm{m}(B) = i$, the set $\mathrm{J}(B)$ of jobs we wish to move away from $\mathrm{m}(B)$, and the move $\mathrm{mv}(B) = (j, i)$ that caused the creation of B. If (j, i) is a big-to-big move, we call B a big blocker and set $\mathrm{J}(B) = \{j_{\mathrm{big}}\}$, where j_{big} is the single big job on i. Creating this big blocker will prevent all attempts to move another big job to i. The intuition is that the move (j, i) only can become valid if j_{big} is moved away from i and no other big job replaces it. Note that we use the fact that no two big jobs can be on one machine. If (j, i) is a small or big-to-small move, we set $\mathrm{J}(B) = \sigma^{-1}(i)$ and call B a small blocker. The algorithm will not try to move any job to machine i, increasing the likelihood that (j, i) becomes valid. All blockers are stored in a list $L = B_0, \ldots, B_t$ in order of creation.

We proceed to describe when the algorithm deems a move potentially helpful. Let $J(L) = \{j_{\mathrm{new}}\} \cup \bigcup_{k=0}^{t} \mathrm{J}(B_k)$ be the set of all jobs we wish to move. Define the set of machines that are contained in a big blocker by $M_{\mathrm{B}}(L)$, the set of machines that are contained in a small blocker by $M_{\mathrm{S}}(L)$, and $M(L) = M_{\mathrm{S}}(L) \cup M_{\mathrm{B}}(L)$. Furthermore denote the set of small jobs on machine i that cannot be moved to any other machine by $S_i \coloneqq \{j \in \sigma^{-1}(i) \mid j \text{ is small and } \mathrm{M}(j) \setminus \{i\} \subseteq M_{\mathrm{s}}(L)\}$. We now define the potential moves. A small move (j, i) is a potential move when $j \in J(L)$ and $i \notin M_{\mathrm{s}}(L)$. A big-to-small or big-to-big-move (j, i) is a potential move when $j \in J(L)$, $i \notin M(L)$, and $\mathrm{p}(S_i) \leq 1 - b + R$. In the presence of several potential moves, the algorithm chooses one with minimum lexicographical value, defined as

$$\mathrm{Val}(j, i) = \begin{cases} (0, 0) & \text{if } (j, i) \text{ is valid,} \\ (1, \mathrm{p}(\sigma^{-1}(i))) & \text{if } (j, i) \text{ is small move,} \\ (2, \mathrm{p}(\sigma^{-1}(i))) & \text{if } (j, i) \text{ is big-to-small move, and} \\ (3, 0) & \text{if } (j, i) \text{ is big-to-big move.} \end{cases} \tag{11}$$

The complete procedure is summarized in Algorithm 1.

Algorithm 1: $A_R(\sigma, j_{\mathrm{new}})$

```
Initialize L ← empty list
while σ(j_new) = ⊥ do
    Choose a potential move (j, i) of minimum lexicographic value
    if (j, i) is valid then
        Let B_k be the blocker in L = B_0, ..., B_t such that j ∈ J(B_k)
        Remove B_k and all blockers added after it from L: L ← B_0, ..., B_{k-1}
        Update Schedule: σ(j) ← i
    else if (j, i) is small or big-to-small then
        Create small blocker B with J(B) = σ^{-1}(i) \ J(L), m(B) = i, mv(B) = (j, i)
        Append B to L
    else # then (j, i) is a big-to-big move
        Let j_big be the big job in σ^{-1}(i)
        Create big blocker B with J(B) = {j_big}, m(B) = i, mv(B) = (j, i)
        Append B to L
return σ
```

2.2 Proof of Termination

As already mentioned, the crucial step is to prove that the algorithm is successful in creating a feasible schedule. This is of course equivalent to the termination of the algorithm. Svensson proved the termination for the special case $R = \frac{2}{3} + s$.

▶ **Lemma 3** ([9]). *Let $b = 1$. Then $A_{\frac{2}{3}+s}$ always terminates, unless* CLP(1) *is infeasible.*

We provide a stronger and more general variant of Lemma 3.

▶ **Lemma 4.** *Let $R \geq s$ and let each configuration in the optimal solution contain at most one big job. Define $k = \lfloor \frac{R}{s} \rfloor$ and $\delta = (k+1)s - R \in (0, s]$ and define $l = \lfloor \frac{R-b}{s} \rfloor$ and $\varepsilon = (l+1)s - (R-b) \in (0, s]$.*

1. *If $\mathrm{OPT}_{\mathrm{LP}} - b$ is a multiple of s and there exists $0 \leq c \leq b$ such that*

$$b + s - R - \delta \leq c \leq R + \delta \tag{12}$$
$$2R + 2\delta + c + 2(1-b) - s \geq 2 \tag{13}$$
$$2R + \delta - b - s \geq 0, \tag{14}$$

   then A_R always terminates, unless CLP(1) *is infeasible.*
2. *If $\mathrm{OPT}_{\mathrm{LP}}$ is a multiple of s and there exists $0 \leq c \leq b$ such that*

$$b + s - R - \varepsilon \leq c \leq R + \varepsilon \tag{15}$$
$$2R + 2\varepsilon + c + 2(1-b) - s \geq 2 \tag{16}$$
$$2R + \delta + \varepsilon - b - s \geq 0, \tag{17}$$

   then A_R always terminates, unless CLP(1) *is infeasible.*

Note that Lemma 3 can be derived from Lemma 4 by setting $b = 1$, $R = \frac{2}{3} + s$ and $c = \frac{2}{3}$, but none of the constraints are tight for these values. The improvement is mainly due to the introduction of ε and δ as the remainder of the division of $R - b$ respectively R by s: in several parts of the analysis, one can see that some multiple of s is larger than $R - b$ or R, so we know it is at least the next multiple of s, which we can now express as $R - b + \varepsilon$ and $R + \delta$. Since the value of ε and δ can be anywhere in the interval $(0, s]$, the gain varies greatly, and causes the discontinuities in the resulting bounds. For a more detailed proof of Lemma 4 we refer our readers to the full version of the paper.

Our second improvement is to not only consider $R = \frac{2}{3} + s$ and $c = \frac{2}{3}$, but to allow larger values for c, and subsequently, smaller values for R, depending on s and b. We have determined values for R (and c) that satisfy the prerequisites of Lemma 4. In the case that $\mathrm{OPT}_{\mathrm{LP}} - b$ is a multiple of s, we can prove that these values are best possible.

▶ **Lemma 5.**

1. *If $\mathrm{OPT}_{\mathrm{LP}} - b$ is a multiple of s, the following are the smallest values for R that satisfy the prerequisites of Lemma 4, case 1:*
   - **(a)** *$R = b - s$ if $\frac{2}{5} \leq \frac{s}{b} < \frac{1}{2}$ and*
   - **(b)** *$R = \lceil \frac{2}{3}(\frac{b}{s} - 1) \rceil s$ otherwise.*
2. *If $\mathrm{OPT}_{\mathrm{LP}}$ is a multiple of s and $\frac{s}{b} \leq \frac{2}{5}$, then $R = b - \lfloor \frac{b}{3s} + \frac{2}{3} \rfloor s$ satisfies the prerequisites of Lemma 4, case 2.*

We here show that the claimed values for case 1 satisfy the prerequisites of Lemma 4. We omit the proofs for case 2 and the optimality in case 1.

▶ Claim 1. In case (a), i.e. if $\mathrm{OPT}_{\mathrm{LP}} - b$ is a multiple of s and $\frac{2}{5} \leq \frac{s}{b} < \frac{1}{2}$, the value $R = b - s$ satisfies the prerequisites of Lemma 4, case 1.

Proof. Recall that $k = \lfloor \frac{R}{s} \rfloor$ and $\delta = (k+1)s - R$. Also note that $1 < \frac{R}{s} \leq \frac{3}{2}$ since $\frac{2}{5} \leq \frac{s}{b} < \frac{1}{2}$. Therefore $k = 1$ and $\delta = 3s - b$. Choose $c = R + \delta = 2s$. It is easily confirmed that $0 \leq c < b$ and conditions (12), (13), and (14) are satisfied. ◀

▶ Claim 2. In case (b), i.e. if $\mathrm{OPT}_{\mathrm{LP}} - b$ is a multiple of s and $\frac{s}{b} < \frac{2}{5}$ or $\frac{s}{b} \geq \frac{1}{2}$, the value $R = \lceil \frac{2}{3}(\frac{b}{s} - 1) \rceil s$ satisfies the prerequisites of Lemma 4, case 1.

Proof. Remember that processing times of the jobs satisfy $0 < s < b \leq 1$. Obviously, we have $k = \lceil \frac{2}{3}(\frac{b}{s} - 1) \rceil$ and $\delta = s$. Also note that

$$R = \left\lceil \frac{2}{3}\left(\frac{b}{s} - 1\right)\right\rceil s \geq \frac{2}{3}\left(\frac{b}{s} - 1\right)s = \frac{2}{3}(b - s). \tag{18}$$

Now set $c = \min\{b, R + \delta\}$. Then condition (12) is satisfied because

$$\begin{aligned} b - R + s - \delta = b - R &\leq b - \frac{2}{3}(b - s) = \frac{1}{3}b + \frac{2}{3}s < \frac{2}{3}b + \frac{1}{3}s \\ &= \frac{2}{3}(b - s) + s \leq R + s = R + \delta \end{aligned} \tag{19}$$

and $b - R + s - \delta = b - R \leq 1 - R < 1$.

Considering condition (13) we have

$$2R + 2\delta + c + 2(1 - b) - s \geq \frac{4}{3}(b - s) + s + c + 2 - 2b = -\frac{2}{3}b - \frac{1}{3}s + c + 2, \tag{20}$$

and the latter is at least 2 if $c \geq \frac{2}{3}b + \frac{1}{3}s$ holds. This is true since $b > \frac{2}{3}b + \frac{1}{3}s$ and $R + \delta \geq \frac{2}{3}(b - s) + s = \frac{2}{3}b + \frac{1}{3}s$.

To finally prove condition (14), we consider three cases. Note that (14) simplifies to $R \geq \frac{1}{2}b$ since $\delta = s$.

Case 1: $\frac{s}{b} < \frac{1}{4}$. We have $R \geq \frac{2}{3}(b - s) > \frac{2}{3}(b - \frac{1}{4}b) = \frac{1}{2}b$.

Case 2: $\frac{1}{4} \leq \frac{s}{b} < \frac{2}{5}$. By the bounds on $\frac{s}{b}$ we get $k = 2$, so $R = 2s \geq 2\frac{1}{4}b = \frac{1}{2}b$.

Case 3: $\frac{s}{b} \geq \frac{1}{2}$. In this case $k = 1$ and $R = s \geq \frac{1}{2}b$. ◀

3 Improving the Bound by Scaling

In this section we prove our main result, Theorem 1. Remember that the result by Lenstra, Shmoys, and Tardos [6] shows that the integrality gap is at most $\frac{3}{2}$ if $\mathrm{OPT}_{\mathrm{LP}} \geq 2b$. If $\frac{s}{b} \geq \frac{1}{3}$, we obtain that the integrality gap is at most $2 - \frac{s}{b} \leq \frac{5}{3}$ from section 4. Thus we can restrict our attention to instances I with $\frac{s}{b} < \frac{1}{3}$ and $\mathrm{OPT}_{\mathrm{LP}} < 2b$. To improve upon the bound from Theorem 2 for the remaining instances, we use a new scaling technique that considers several cases, depending on whether $\mathrm{OPT}_{\mathrm{LP}} - b$ or $\mathrm{OPT}_{\mathrm{LP}}$ is a multiple of s and the value of $\frac{s}{b}$. We denote the integrality gap of the configuration LP for an instance I by $IG(I)$ throughout this section. Remember that we scaled the processing times such that $\mathrm{OPT}_{\mathrm{LP}} = 1$.

3.1 Case 1: $\mathrm{OPT}_{\mathrm{LP}}(I) - b$ is a multiple of s

In this case, the longest configuration of the optimal fractional solution contains a big job, i.e. there is $x \in \mathbb{Z}_{\geq 0}$ with $\mathrm{OPT}_{\mathrm{LP}}(I) = b + xs$. Define $k = \lceil \frac{2}{3}(\frac{b}{s} - 1) \rceil$. Then Theorem 2 yields

$$IG(I) \leq 1 + ks. \tag{21}$$

We can prove another bound by scaling s. First, we describe the connection between k and $\frac{s}{b}$. Since $k \geq \frac{2}{3}(\frac{b}{s} - 1)$ we have $\frac{s}{b} \geq \frac{2}{3k+2}$. Similarly, $k - 1 < \frac{2}{3}(\frac{b}{s} - 1)$ and therefore $\frac{s}{b} < \frac{2}{3k-1}$. It follows that $\frac{s}{b} \in [\frac{2}{3k+2}, \frac{2}{3k-1})$. This interval actually corresponds to one of the continuous segments of our bound, see also Figure 2. The integrality gap increases as $\frac{s}{b}$

approaches $\frac{2}{3k-1}$ and jumps down again at $\frac{s}{b} = \frac{2}{3k-1}$. Therefore, if $\frac{s}{b}$ is slightly below $\frac{2}{3k-1}$, we can increase the processing time s of small jobs to s' such that $\frac{s'}{b} = \frac{2}{3k-1}$.

Define the instance I' identical to I, but with small jobs having processing time $s' = \frac{2b}{3k-1}$. Since $s \leq s'$, we have $\mathrm{OPT}(I) \leq \mathrm{OPT}(I')$. Define $\alpha = \frac{s'}{s} = \frac{2b}{(3k-1)s}$. We first prove that $\mathrm{OPT}_{\mathrm{LP}}(I') \leq \alpha \mathrm{OPT}_{\mathrm{LP}}(I)$. For this, let $T = \mathrm{OPT}_{\mathrm{LP}}(I)$ and consider a feasible solution x of $\mathrm{CLP}(T, I)$. Then x is also a feasible solution of $\mathrm{CLP}(\alpha T, I')$, since the processing time of a configuration increases at most by factor α in the modified instance I'. We therefore have

$$IG(I) = \frac{\mathrm{OPT}(I)}{\mathrm{OPT}_{\mathrm{LP}}(I)} \leq \frac{\mathrm{OPT}(I')}{\mathrm{OPT}_{\mathrm{LP}}(I')\frac{1}{\alpha}} \leq \alpha IG(I'). \tag{22}$$

Define

$$s_0 = -\frac{1}{2k} + \sqrt{\frac{1}{4k^2} + \frac{2b}{(3k-1)k} \cdot \left(1 + \frac{(k-1)2b}{3k-1}\right)}. \tag{23}$$

We will later show that scaling is beneficial if $s \geq s_0$. In order to apply Theorem 2 to I', we need to prove that the prerequisites hold. In particular, we show that $\mathrm{OPT}_{\mathrm{LP}}(I') - b$ is a multiple of s':

▶ **Lemma 6.** *Let I be an instance with small processing time $s \geq s_0$ and $\mathrm{OPT}_{\mathrm{LP}}(I) = b + xs$. Let I' be the modified instance with small processing time $s' = \frac{2b}{3k-1}$. Then $\mathrm{OPT}_{\mathrm{LP}}(I') = b + xs'$.*

Proof. We first claim without proof that $s_0 > \frac{2b}{3k}$. Now consider an optimal fractional solution to I. We show that this is a solution to I' with makespan at most $b + xs'$. Let C be any configuration that occurs in this solution. Denote by ℓ and ℓ' the length of C when small jobs have length s and s', respectively. Then $\ell \leq b + xs$. If C contains a big job, we have $\ell = b + ys$ for some $y \leq x$. It follows that $\ell' = b + ys' \leq b + xs'$. Otherwise, C only contains small jobs and $\ell = ys$ for some $y \in \mathbb{N}$. Define $z = y - x \in \mathbb{Z}$. Since $\mathrm{OPT}_{\mathrm{LP}}(I)$ is not a multiple of s, we have $zs + xs = \ell < \mathrm{OPT}_{\mathrm{LP}}(I) = b + xs$ and therefore $zs < b$. This implies $z < \frac{b}{s}$. Remember that $s \geq s_0 > \frac{2b}{3k}$, so $\frac{s}{b} > \frac{2}{3k}$. This implies $z < \frac{b}{s} < \frac{3k}{2}$. Since z is integral, we also have $z \leq \frac{3k-1}{2}$. It follows that

$$\ell' = ys' = zs' + xs' \leq \frac{3k-1}{2}\frac{2b}{3k-1} + xs' = b + xs'. \tag{24}$$

Therefore, $\mathrm{OPT}_{\mathrm{LP}}(I') \leq b + xs'$.

Now assume that there is a fractional solution for I' with makespan less than $b + xs'$. We will show that this implies that $\mathrm{OPT}_{\mathrm{LP}}(I) < b + xs$, a contradiction. Let C be any configuration occurring in the optimal solution and define ℓ and ℓ' as before. Then $\ell' < b + xs'$. If C contains a big job, we have $\ell = b + ys$ for some $y \in \mathbb{Z}_{\geq 0}$. We have $b + ys' = \ell' < b + xs'$, thus $y < x$ and $\ell = b + ys < b + xs$. Otherwise, C contains only small jobs and $\ell = ys$ for some $y \in \mathbb{N}$. Define again $z = y - x \in \mathbb{Z}$. Then $zs' + xs' = \ell' < b + xs'$, therefore $zs' < b$. This implies $\ell = ys = zs + xs < zs' + xs < b + xs$. ◀

To apply Theorem 2, we also have to scale I' by $\beta = \frac{1}{\mathrm{OPT}_{\mathrm{LP}}(I')} \leq 1$ to obtain the instance I'' with $\mathrm{OPT}_{\mathrm{LP}}(I'') = 1$. In I'' the processing times are $b'' = b \cdot \beta$ and $s'' = s' \cdot \beta \leq s'$. One can easily see that I' and I'' have the same integrality gap:

$$IG(I') = \frac{\mathrm{OPT}(I')}{\mathrm{OPT}_{\mathrm{LP}}(I')} = \frac{\beta \cdot \mathrm{OPT}(I')}{\beta \cdot \mathrm{OPT}_{\mathrm{LP}}(I')} = \frac{\mathrm{OPT}(I'')}{\mathrm{OPT}_{\mathrm{LP}}(I'')} = IG(I''). \tag{25}$$

Also $\frac{s}{b} < \frac{1}{3}$ implies $k \geq 2$ and thus $\frac{s'}{b} = \frac{2}{3k-1} \leq \frac{2}{5}$. This finally allows us to apply Theorem 2. From the definition $s' = \frac{2b}{3k-1}$ we can calculate $k - 1 = \frac{2}{3}(\frac{b}{s'} - 1)$. Since k is integral, we have $\lceil \frac{2}{3}(\frac{b}{s'} - 1) \rceil = k - 1$. The integrality gap of the original instance I is thus bounded by

$$IG(I) \leq \alpha IG(I')0 = \alpha IG(I'') \leq \alpha(1 + (k-1)s'') \leq \alpha(1 + (k-1)s'). \tag{26}$$

We will determine which of the two bounds (21) and (26) is better depending on the values of s and b.

▶ **Lemma 7.** *$1 + ks \geq \alpha(1 + (k-1)s')$ holds if and only if $s \geq s_0$.*

For a proof we refer our readers to the full version of our work.

It turns out that inqualities (26) and (21) can be combined if b is not too large.

▶ **Lemma 8.** *If $\frac{s}{b} < \frac{1}{3}$, $\mathrm{OPT}_{\mathrm{LP}}(I) - b$ is a multiple of s, and $b \leq \frac{80}{81}$, the integrality gap of the CLP is at most $\frac{5}{3}$.*

Proof. We consider two cases.
Case 1: $s \leq s_0$. Then the integrality gap is at most $1 + ks \leq 1 + ks_0$ by inequality (21).
Case 2: $s > s_0$. Remember that we scaled the processing time of small jobs to $s' = \alpha s = \frac{2b}{3k-1}$. One can easily see that the term

$$\alpha(1 + (k-1)s') = \frac{2b}{(3k-1)s}\left(1 + \frac{(k-1)2b}{3k-1}\right) = \frac{1}{s}\left(\frac{2b}{3k-1} + \frac{(k-1)4b^2}{(3k-1)^2}\right) \tag{27}$$

is monotonically decreasing with respect to s. From Lemma 7 and $\alpha = \frac{s'}{s}$ we also know that $\frac{s'}{s_0}(1 + (k-1)s') \leq 1 + ks_0$. We therefore have that the integrality gap is bounded by $\alpha(1 + (k-1)s') \leq 1 + ks_0$ for all $s > s_0$.

For both cases we can compute

$$\begin{aligned} 1 + ks_0 &= 1 - \frac{1}{2} + \sqrt{\frac{1}{4} + \frac{2bk}{(3k-1)} \cdot \left(1 + \frac{(k-1)2b}{3k-1}\right)} \\ &= \frac{1}{2} + \frac{1}{2(3k-1)} \cdot \sqrt{(3k-1)^2 + 8bk(3k-1+(k-1)2b)}. \end{aligned} \tag{28}$$

The last term attains its maximum $\frac{1}{4} \cdot (2 + \sqrt{18b+4})$ at $k = 2$. It is easy to verify that this is at most $\frac{5}{3}$ as long as $b \leq \frac{80}{81} \approx 0.988$. ◀

In the case that $b > \frac{80}{81}$, we scale the processing time of small jobs to $s' = \frac{2b}{3k-1}$ if $s > \frac{2b}{3k}$.

▶ **Lemma 9.** *If $\frac{s}{b} < \frac{1}{3}$, $\mathrm{OPT}_{\mathrm{LP}}(I) - b$ is a multiple of s, and $b > \frac{80}{81}$, the integrality gap of the CLP is at most $\frac{5}{3}$.*

Proof. In our analysis, we again distinguish the two cases whether small jobs were rounded or not. We use the same scaled instance I' from above, where we scaled s to $s' = \frac{2b}{3k-1}$. Remember that $\mathrm{OPT}_{\mathrm{LP}} = b + xs$ for $x \in \mathbb{N}$.
Case 1: $s \leq \frac{2b}{3k}$. We can directly apply Theorem 2 and obtain the bound

$$IG(I) \leq 1 + ks \leq 1 + k\frac{2b}{3k} \leq \frac{5}{3}. \tag{29}$$

Case 2: $s > \frac{2b}{3k}$. We claim without proof that $\mathrm{OPT}_{\mathrm{LP}}(I') \leq 1 + \frac{1}{79}s$.

Similar as in inequality (22), we find that $IG(I) \leq (1 + \frac{1}{79}s)IG(I')$. As above, we have to scale I' to I'' such that $\mathrm{OPT}_{\mathrm{LP}}(I'') = 1$. Applying Theorem 2 to I'' and using $s < \frac{2b}{3k-1}$ and $b \leq 1$, we obtain

$$\begin{aligned} IG(I) &\leq \left(1 + \frac{1}{79}s\right) IG(I') = \left(1 + \frac{1}{79}s\right) IG(I'') \\ &\leq \left(1 + \frac{1}{79}s\right)\left(1 + (k-1)\frac{s'}{\mathrm{OPT}_{\mathrm{LP}}(I')}\right) \\ &\leq \left(1 + \frac{1}{79}s\right)(1 + (k-1)s') \\ &< \left(1 + \frac{2}{79(3k-1)}\right)\left(1 + \frac{2(k-1)}{3k-1}\right) \\ &= \frac{15k^2 - \frac{1096}{79}k + \frac{231}{79}}{9k^2 - 6k + 1}. \end{aligned} \tag{30}$$

The last term can be seen to be monotonically increasing for $k \geq 1$ and has the limit $\frac{5}{3}$. ◀

3.2 Case 2: $\mathrm{OPT}_{\mathrm{LP}}(I)$ is a multiple of s

Using Theorem 2, case (b) with $\mathrm{OPT}_{\mathrm{LP}} = 1$ and $k = \lfloor \frac{1}{3}\frac{b}{s} + \frac{2}{3} \rfloor$ we have

$$IG(I) \leq 1 + R = 1 + b - ks. \tag{31}$$

Since $k \leq \frac{1}{3}\frac{b}{s} + \frac{2}{3}$, we have $\frac{s}{b} \leq \frac{1}{3k-2}$. Similarly, $k + 1 > \frac{1}{3}\frac{b}{s} + \frac{2}{3}$ and therefore $\frac{s}{b} > \frac{1}{3k+1}$. It follows that $\frac{s}{b} \in (\frac{1}{3k+1}, \frac{1}{3k-2}]$.

Now assume that $\frac{s}{b} \geq \frac{1}{3k}$. Then the integrality gap $IG(I)$ is bounded by

$$1 + b - ks \leq 1 + b - k\frac{b}{3k} = 1 + \frac{2}{3}b \leq \frac{5}{3}, \tag{32}$$

since $b \leq 1$.

Finally, we have the case that $\frac{s}{b} \in (\frac{1}{3k+1}, \frac{1}{3k})$. Our bound (31) increases when $\frac{s}{b}$ approaches $\frac{1}{3k+1}$ and jumps down when it reaches that value, see also Figure 2. So we create an instance I' by rounding the running time b of big jobs up to $b' = (3k+1)s$. Obviously, $\mathrm{OPT}(I') \geq \mathrm{OPT}(I)$. On the other hand, the largest configuration in the fractional optimum has the height xs for some $x \in \mathbb{Z}_{\geq 1}$, and any configuration that contains a big job has height $b + ys \leq xs$ for some $y \in \mathbb{Z}_{\geq 0}$. We assume that $b + ys < xs$, because otherwise b and therefore $\mathrm{OPT}_{\mathrm{LP}}(I) - b$ were also a multiples of s, and we could use the proof from section 3.1. Then, since $b \in (3ks, 3ks + s)$, we have

$$xs - (y + 3k)s > b + ys - ys - 3ks = b - 3ks > 0 \tag{33}$$

and therefore $xs - (y + 3k)s \geq s$. This implies

$$b' + ys = (3k+1)s + ys = (y + 3k)s + s \leq xs - s + s = xs, \tag{34}$$

i.e. $\mathrm{OPT}_{\mathrm{LP}}(I') = \mathrm{OPT}_{\mathrm{LP}}(I)$. In particular,

$$IG(I) = \frac{\mathrm{OPT}(I)}{\mathrm{OPT}_{\mathrm{LP}}(I)} \leq \frac{\mathrm{OPT}(I')}{\mathrm{OPT}_{\mathrm{LP}}(I')} = IG(I') \tag{35}$$

and $\mathrm{OPT}_{\mathrm{LP}}(I')$ is a multiple of s. Note that

$$\left\lfloor \frac{1}{3}\frac{b'}{s} + \frac{2}{3} \right\rfloor = \left\lfloor \frac{1}{3}(3k+1) + \frac{2}{3} \right\rfloor = \lfloor k + 1 \rfloor = k + 1. \tag{36}$$

Theorem 2, case (b) now yields

$$\begin{aligned} IG(I) &\leq IG(I') \leq 1 + R = 1 + b' - (k+1)s \\ &= 1 + (3k+1)s - (k+1)s = 1 + 2ks < 1 + 2k\frac{b}{3k} = 1 + \frac{2}{3}b \\ &\leq 1 + \frac{2}{3} = \frac{5}{3}. \end{aligned} \tag{37}$$

4 An $(\mathrm{OPT} + b - s)$-Approximation

In this section we present an algorithm for the restricted assignment problem with two different processing times, which also proves a bound on the integrality gap of the CLP. Here, we assume that the processing times $s < b$ are positive integers. Our algorithm depends on a result by Shmoys and Tardos [8] for a variant of unrelated scheduling with costs. Their algorithm is based on solving and rounding the assignment LP. Consider the assignment LP $\mathrm{ALP}(T)$ for a given makespan T. For $i \in \{1, \ldots, m\}$ and $q \in \{s, b\}$, let $a_{iq} = \sum_{j:p_{ij}=q} x_{ij}$ denote the fractional number of jobs of size q scheduled on machine i. We strengthen the LP relaxation by adding two classes of constraints

$$a_{ib} \leq \left\lfloor \frac{T}{b} \right\rfloor \quad \text{for each } i \in M \tag{38}$$

$$a_{is} \leq \left\lfloor \frac{T}{s} \right\rfloor - \left\lfloor \frac{b}{s} \right\rfloor a_{ib} \quad \text{for each } i \in M. \tag{39}$$

In the following, $\mathrm{OPT}_{\mathrm{LP}}$ denotes the optimal makespan of the augmented ALP. Our algorithm solves the augmented ALP and applies the rounding procedure of Shmoys and Tardos [8] to the solution.

▶ **Theorem 10.** *For the restricted assignment problem with two different processing times $s < b$ there is a polynomial time algorithm that produces a schedule of length at most $\min\{\mathrm{OPT}_{\mathrm{LP}} + b, \lfloor \mathrm{OPT}_{\mathrm{LP}}/s \rfloor s + \lfloor \mathrm{OPT}_{\mathrm{LP}}/b \rfloor (b-s)\}$, yielding a bound of $\mathrm{OPT}_{\mathrm{LP}} + (b-s)$ for the case $b \leq \mathrm{OPT}_{\mathrm{LP}} < 2b$. Furthermore the algorithm has a multiplicative performance guarantee of $(2 - \frac{s}{b})$.*

We omit the analysis of the algorithm due to space restrictions. As a corollary we can bound the integrality gap of the configuration LP.

▶ **Corollary 11.** *The described algorithm can be modified to work with the CLP yielding the same bounds. In particular, if an instance of the restricted assignment problem has only two different processing times $s < b$, the integrality gap of the CLP is at most $2 - \frac{s}{b}$.*

References

1 Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, (STOC 2006)*, pages 31–40, 2006.

2 Deeparnab Chakrabarty, Sanjeev Khanna, and Shi Li. On (1, epsilon)-restricted assignment makespan minimization. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 1087–1101, 2015.

3 Tomáš Ebenlendr, Marek Krčál, and Jiří Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. *Algorithmica*, 68(1):62–80, 2014.

4 Chien-Chung Huang and Sebastian Ott. A combinatorial approximation algorithm for graph balancing with light hyper edges. *CoRR*, abs/1507.07396, 2015.

5 Stavros G Kolliopoulos and Yannis Moysoglou. The 2-valued case of makespan minimization with assignment constraints. *Information Processing Letters*, 113(1):39–43, 2013.

6 Jan Karel Lenstra, David B Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1-3):259–271, 1990.

7 Evgeny V. Shchepin and Nodari Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, 33(2):127–133, 2005.

8 David B Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(1-3):461–474, 1993.

9 Ola Svensson. Santa claus schedules jobs on unrelated machines. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC 2011)*, pages 617–626, 2011.

A Plane 1.88-Spanner for Points in Convex Position*

Mahdi Amani[1], Ahmad Biniaz[2], Prosenjit Bose[3], Jean-Lou De Carufel[4], Anil Maheshwari[5], and Michiel Smid[6]

1 **Dipartimento di Informatica, Università di Pisa, Pisa, Italy**
m_amani@di.unipi.it
2 **School of Computer Science, Carleton University, Ottawa, Canada**
ahmad.biniaz@gmail.com
3 **School of Computer Science, Carleton University, Ottawa, Canada**
jit@scs.carleton.ca
4 **School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada**
jdecaruf@uottawa.ca
5 **School of Computer Science, Carleton University, Ottawa, Canada**
anil@scs.carleton.ca
6 **School of Computer Science, Carleton University, Ottawa, Canada**
michiel@scs.carleton.ca

Abstract

Let S be a set of n points in the plane that is in convex position. For a real number $t > 1$, we say that a point p in S is t-good if for every point q of S, the shortest-path distance between p and q along the boundary of the convex hull of S is at most t times the Euclidean distance between p and q. We prove that any point that is part of (an approximation to) the diameter of S is 1.88-good. Using this, we show how to compute a plane 1.88-spanner of S in $O(n)$ time, assuming that the points of S are given in sorted order along their convex hull. Previously, the best known stretch factor for plane spanners was 1.998 (which, in fact, holds for any point set, i.e., even if it is not in convex position).

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling, G.2.2 Graph Theory, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases points in convex position, plane spanner

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.25

1 Introduction

Let S be a set of n points in the plane. A *geometric graph* is a graph $G = (S, E)$ with vertex set S and edge set E consisting of line segments connecting pairs of vertices. The length (or weight) of any edge (p, q) in E is defined to be the Euclidean distance $|pq|$ between p and q. The length of any path in G is defined to be the sum of the lengths of the edges on this path. For any two vertices p and q of S, their shortest-path in G, denoted by $\delta^*_G(p, q)$, is a path in G between p and q that has the minimum length. We denote the length of $\delta^*_G(p, q)$ by $|\delta^*_G(p, q)|$. For a real number $t \geqslant 1$, the graph G is a *t-spanner* of S if for any two points p and q in S, $|\delta^*_G(p, q)| \leq t|pq|$. The smallest value of t for which G is a t-spanner is called

* Research supported by NSERC.

the *stretch factor* of G. A large number of algorithms have been proposed for constructing t-spanners for any given point set; see the book by Narasimhan and Smid [12].

In this paper, we consider *plane* spanners, i.e., spanners whose edges do not cross each other. Chew [4] was the first to prove that plane spanners exist; in fact, this was the first publication on geometric spanners. Chew proved that the L_1-Delaunay triangulation of a finite point set has stretch factor at most $\sqrt{10} \approx 3.16$ (observe that lengths in this graph are measured in the Euclidean metric). In the journal version [5], Chew proves that the Delaunay triangulation based on a convex distance function defined by an equilateral triangle is a 2-spanner.

Dobkin *et al.* [7] proved that the L_2-Delaunay triangulation is a t-spanner for $t = \pi(1+\sqrt{5})/2 \approx 5.08$. Keil and Gutwin [10] improved the upper bound on the stretch factor to $t = \frac{2\pi}{3\cos(\pi/6)} \approx 2.42$. This was subsequently improved by Cui *et al.* [6] to $t = 2.33$ for the case when the point set is in convex position. Currently, the best result is due to Xia [13], who proved that t is less than 1.998.

Thus, the current best upper bound on the stretch factor of plane spanners is 1.998. Regarding lower bounds, by considering the four vertices of a square, it is obvious that a plane t-spanner with $t < \sqrt{2}$ does not exist. Mulzer [11] has shown that every plane spanning graph of the vertices of a regular 21-gon has stretch factor at least 1.41611. Recently, Dumitrescu and Ghosh [8] improved the lower bound to 1.4308 for the vertices of a regular 23-gon.

1.1 Our Results

In this paper, we consider plane spanners for point sets that are in convex position. Currently, it is known that the stretch factor of any such spanner is less than 1.998. Moreover, the best lower bound is 1.4308. We improve the upper bound to 1.88. Our approach is as follows.

Let S be a finite and non-empty set of points in the plane and assume that S is in convex position. We denote the boundary of the convex hull of S by $CH(S)$. For any two points p and q in S, let $\delta^{cw}_{CH(S)}(p,q)$ and $\delta^{ccw}_{CH(S)}(p,q)$ denote the clockwise and counter-clockwise paths from p to q along $CH(S)$, respectively, and let $\delta^{*}_{CH(S)}(p,q)$ be the shorter one. Let $t \geqslant 1$ be a real number, and let p and q be two points of S. We say that p is *t-good* for q in S if $|\delta^{*}_{CH(S)}(p,q)| \leqslant t|pq|$. Observe that if p is t-good for q, then q is t-good for p. We say that the point $p \in S$ is *t-good* for S if p is t-good for all points of S. Define

$$t^* = \inf\{t : \text{ each finite and non-empty set of points in the plane that is in convex position has at least one } t\text{-good point}\}.$$

▶ **Theorem 1.** *Let S be a finite and non-empty set of points in the plane that is in convex position, and let $t > t^*$ be a real number. Then, there exists a plane t-spanner of S.*

Proof. Consider algorithm PlaneSpanner(S,t) and the graph $G = (S,E)$ that is returned by this algorithm. Initially, $B = S$. This graph G is obtained by iteratively cutting an ear of $CH(B)$. Therefore, G is a plane triangulation of $CH(S)$.

If $|B| \leqslant 3$, then E is the set of edges of the convex hull of S. Thus, G is 1-spanner. Assume $|B| > 3$. Consider one iteration of the **while** loop. Since $t > t^*$, there exists a t-good point in B; let p be such a point that is chosen in line 4 of algorithm PlaneSpanner(S,t). Let q and r be the two neighbors of p on $CH(B)$. We add the edge (q,r) to E, and remove the point p from B. See Figure 1(a). Since E contains the convex hull of B, it follows that for any point p' in B the shortest-path distance between p and p' in G is at most $|\delta^{*}_{CH(B)}(p,p')|$, which is at most $t|pp'|$. Therefore, the graph G is a t-spanner of S. ◀

In order to apply this result, we need an estimate on the value of t^*:

Algorithm 1 PLANESPANNER(S, t)

Input: A finite set S of points in the plane in convex position, and a real number $t > t^*$.
Output: A plane t-spanner of S.

```
E ← the set of edges of CH(S)
B ← S
while |B| ⩾ 4 do
    p ← a t-good point in B
    q, r ← the two neighbors of p on CH(B)
    E ← E ∪ {(q, r)}
    B ← B \ {p}
return G = (S, E)
```

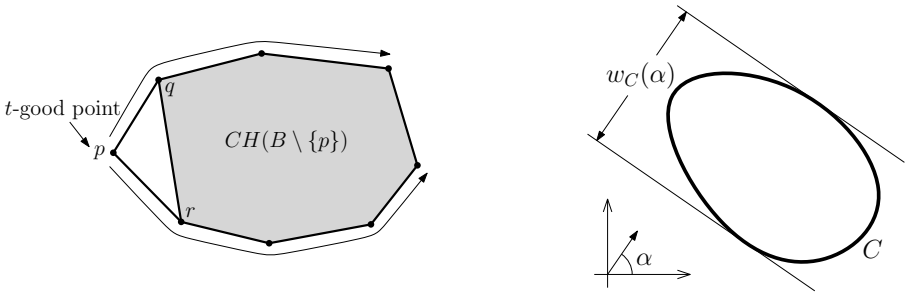


Figure 1 (a) The point p is t-good. The bold edges belong to G. (b) $w_C(\alpha)$ in direction α.

▶ **Problem.** *Is the value of t^* finite? If it is, determine upper and lower bounds on t^*.*

Our main result is a proof that $\sqrt{3} \leqslant t^* \leqslant 1.88$. In Section 2, we provide some preliminaries. In Section 3, we prove that any point of S that is an endpoint of diameter is 1.88-good. In Section 4, we consider an approximate diametral pair of S and prove that both points in this pair are 1.88-good. Based on this, in Section 5, we show how to construct a plane 1.88-spanner for S in $O(n)$ time, assuming that the points of S are given in sorted order along $CH(S)$. Some further results are given in Section 6. Concluding remarks and open problems are given in Section 7.

2 Preliminaries

For any two points p and q in the plane let pq denote the line segment between p and q, and let $R(p{\to}q)$ denote the ray emanating from p and passing through q. For a point p and a real number $\rho > 0$, let $C(p, \rho)$ be the closed disk of radius ρ that is centered at p. For any two points p and q in the plane let $L(p, q)$ denote the lune of p and q, which is the intersection of $C(p, |pq|)$ and $C(q, |pq|)$.

Let S be a finite and non-empty set of points in the plane. The *diameter* of S is the largest distance among the distances between all pairs of points of S. Any pair of points whose distance is equal to the diameter is called a *diametral pair*. Any point of any diametral pair of S is called a *diametral point*.

▶ **Observation 2.** *Let S be a finite set of at least two points in the plane, and let $\{p, q\}$ be any diametral pair of S. Then, the points of S lie in $L(p, q)$.*

The following theorem is a restatement of Theorem 7.11 in [1].

▶ **Theorem 3** (See [1]). *If C_1 and C_2 are convex polygonal regions with $C_1 \subseteq C_2$, then the length of the boundary of C_1 is at most the length of the boundary of C_2.*

We also restate the following two-dimensional version of Cauchy's surface-area formula. For a closed convex curve C in the plane let $w_C(\alpha)$ be the width of C in direction α; see Figure 1(b).

▶ **Theorem 4** (Cauchy [2]). *The length $|C|$ of the boundary of a closed convex curve C in the plane is given by*

$$|C| = \int_0^\pi w_C(\alpha)\,\mathrm{d}\alpha.$$

▶ **Lemma 5.** *Let S be a finite set of at least two points in the plane that is in convex position, and whose diameter is D. Then, for any two points p and q in S, $|\delta^*_{CH(S)}(p,q)| \leqslant \frac{D\pi}{2}$.*

Proof. Since $CH(S)$ is a closed convex polygonal curve and the width of $CH(S)$ in any direction is at most the diameter of S, i.e. D, we have, by Theorem 4,

$$|CH(S)| = \int_0^\pi w_{CH(S)}(\alpha)\,\mathrm{d}\alpha \leqslant \int_0^\pi D\,\mathrm{d}\alpha = D\pi.$$

Since p and q belong to $CH(S)$, there are two edge-disjoint paths between p and q along $CH(S)$. The length of the shorter one, i.e. $\delta^*_{CH(S)}(p,q)$, is at most $\frac{D\pi}{2}$. ◀

▶ **Lemma 6.** *Let $t \geqslant 1$ be a real number and let S be a finite set of at least two points in the plane that is in convex position and whose diameter is D. Let p and s be any pair of distinct points of S such that $|ps| \geqslant \frac{D\pi}{2t}$. Then $t \geqslant \frac{\pi}{2}$ and p is t-good for s.*

Proof. Since the diameter of S is D, we have $|ps| \leqslant D$. Thus $\frac{D\pi}{2t} \leqslant |ps| \leqslant D$, which implies $t \geqslant \frac{\pi}{2}$. By Lemma 5, we have $|\delta^*_{CH(S)}(p,s)| \leqslant \frac{D\pi}{2}$. Thus,

$$\frac{|\delta^*_{CH(S)}(p,s)|}{|ps|} \leqslant \frac{D\pi/2}{D\pi/2t} = t,$$

which implies that p is t-good for s. ◀

▶ **Lemma 7.** *Let a, b, and c be three points in the plane, let $\beta = \angle abc$, and let $t \geqslant 1$ be a real number. If $\beta \geqslant 2\arcsin(\frac{1}{t})$, then $\frac{|ab|+|bc|}{|ac|} \leqslant t$.*

Proof. Refer to Figure 2(a). Consider the triangle $\triangle abc$. Let ℓ be the bisector of β, and let d be the intersection point of ℓ and ac. Let a' (resp. c') be the point on ℓ that is closest to a (resp. c). We have $|ab| = |aa'|/\sin(\beta/2)$ and $|bc| = |cc'|/\sin(\beta/2)$. Thus,

$$\frac{|ab|+|bc|}{|ac|} = \frac{|aa'|+|cc'|}{|ac|\sin\left(\frac{\beta}{2}\right)} \leqslant \frac{|ad|+|dc|}{|ac|\sin\left(\frac{\beta}{2}\right)} = \frac{1}{\sin\left(\frac{\beta}{2}\right)} \leqslant \frac{1}{\sin\left(2\arcsin(\frac{1}{t})/2\right)} = t.$$ ◀

▶ **Theorem 8.** $t^* \geqslant \sqrt{3}$.

Proof. Let $S = \{p, q, r, p', q', r'\}$ be the set of six points in the plane and in convex position as shown in Figure 2(b). The points p, q, and r are the vertices of an equilateral triangle of side-length 1. The point p' is placed in the middle of qr; q' and r' are placed analogously. The

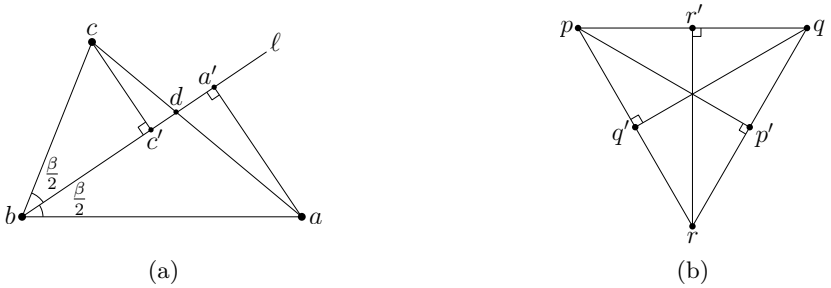


Figure 2 (a) Proof of Lemma 7. (b) Proof of Theorem 8.

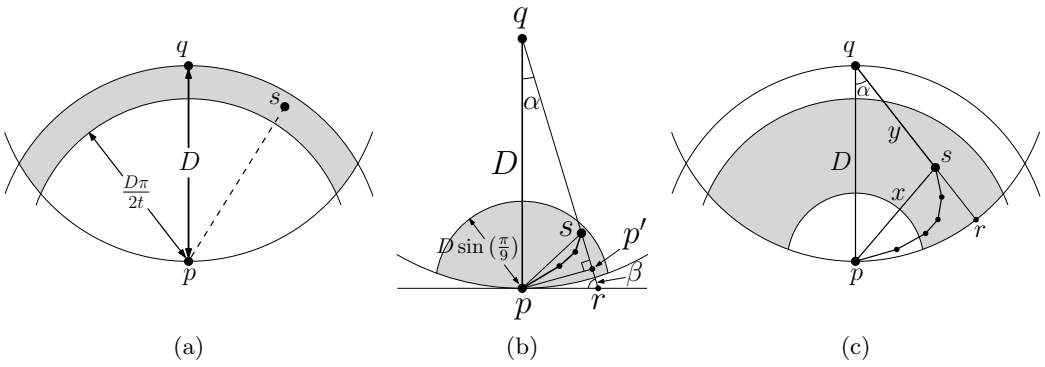


Figure 3 Illustration of the proof of Theorem 9.

two paths between p and p' along $CH(S)$ have lengths equal to $3/2$. Moreover, $|pp'| = \sqrt{3}/2$. Thus,

$$\frac{|\delta^*_{CH(S)}(p,p')|}{|pp'|} = \frac{3/2}{\sqrt{3}/2} = \sqrt{3}.$$

Therefore, for any $\varepsilon > 0$, p is not $(\sqrt{3}-\varepsilon)$-good for p', and vice versa. This implies that none of p, p', and similarly, none of q, q', r, r' is $(\sqrt{3}-\varepsilon)$-good for S. ◀

3 Diametral Points are Good

In this section we will prove the following theorem.

▶ **Theorem 9.** *Let S be a finite set of at least two points in the plane that is in convex position. Then any diametral point of S is 1.88-good for S.*

Throughout the rest of this section, let $t = 1.88$. Let D be the diameter of S, and let $\{p, q\}$ be any diametral pair of S, that is, $|pq| = D$. We are going to show that both p and q are t-good for S. Because of symmetry, it suffices to show that p is t-good. By Observation 2, all points of S are in the intersection of $C(p, D)$ and $C(q, D)$; see Figure 3.

Let s be any point of $S \setminus \{p\}$. We are going to show that p is t-good for s. If $s = q$, then as a consequence of Lemma 5, p is $\frac{\pi}{2}$-good for s and, thus, p is t-good for s. Assume $s \neq q$. Depending on $|ps|$ we differentiate between the following three cases:

- $|ps| > \frac{D\pi}{2t}$. By Lemma 6, p is t-good for s; see Figure 3(a).
- $|ps| < D \sin\left(\frac{\pi}{9}\right)$. Without loss of generality assume s is to the right of $R(p \to q)$. See Figure 3(b). Let r be the intersection point of $R(q \to s)$ with the line that is perpendicular

to pq and passes through p. Consider the path $\delta^{ccw}_{CH(S)}(p,s)$. Because of convexity, this path is to the right of $R(q{\to}s)$ and to the right of $R(p{\to}s)$. By Theorem 3, we have $|\delta^{ccw}_{CH(S)}(p,s)| \leqslant |pr|+|rs|$. Let $\alpha = \angle pqs$ and $\beta = \angle prs = \angle prq$. Let p' be the orthogonal projection of p onto $R(q{\to}s)$. Then $\sin\alpha = \frac{|pp'|}{|pq|} \leqslant \frac{|ps|}{|pq|} = \frac{|ps|}{D} < \sin\left(\frac{\pi}{9}\right)$ and, thus, $\alpha < \frac{\pi}{9}$. This implies that $\beta = \frac{\pi}{2} - \alpha > \frac{7\pi}{18}$. Since $t = 1.88$, we have $\beta > \frac{7\pi}{18} > 2\arcsin(\frac{1}{t})$. Thus, using Lemma 7, we have

$$\frac{|\delta^{*}_{CH(S)}(p,s)|}{|ps|} \leqslant \frac{|\delta^{ccw}_{CH(S)}(p,s)|}{|ps|} \leqslant \frac{|pr|+|rs|}{|ps|} \leqslant t,$$

which implies that p is t-good for s.

- $D\sin\left(\frac{\pi}{9}\right) \leqslant |ps| \leqslant \frac{D\pi}{2t}$. Refer to Figure 3(c). Observe that if s is on pq, then p is 1-good for s. Without loss of generality assume s is to the right of $R(p{\to}q)$. Let r be the intersection point of $R(q{\to}s)$ and the boundary of $C(q,D)$. Consider the path $\delta^{ccw}_{CH(S)}(p,s)$. Because of convexity, this path is to the right of $R(q{\to}s)$ and to the right of $R(p{\to}s)$. Note that $|\delta^{*}_{CH(S)}(p,s)| \leqslant |\delta^{ccw}_{CH(S)}(p,s)|$, and by Theorem 3 we have $|\delta^{ccw}_{CH(S)}(p,s)| \leqslant |rs|+|\widehat{pr}|$, where $|\widehat{pr}|$ denotes the length of the counter-clockwise arc on $C(q,D)$ from p to r. In order to prove that p is t-good for s it is sufficient to prove that

  $$\frac{|rs|+|\widehat{pr}|}{|ps|} \leqslant t,$$

  which is equivalent to

  $$t|ps| - |rs| - |\widehat{pr}| \geqslant 0. \tag{1}$$

  Let $x = |ps|$, $y = |qs|$, and $\alpha = \angle pqs$. Notice that $D\sin\left(\frac{\pi}{9}\right) \leqslant x \leqslant \frac{D\pi}{2t}$, $y \leqslant D$, and $0 \leqslant \alpha \leqslant \frac{\pi}{2}$. By the law of cosines we have $x^2 = D^2 + y^2 - 2Dy\cos\alpha$, which implies that

  $$y = D\cos\alpha \pm \sqrt{x^2 + D^2(\cos^2\alpha - 1)}.$$

  For a fixed value of α, x is minimum when $R(q{\to}s)$ is tangent to $C(p,x)$. This implies that $x \geqslant D\sin\alpha$, and consequently $\alpha \leqslant \arcsin\left(\frac{x}{D}\right)$. Note that $|rs| = D - y$ and $|\widehat{pr}| = D\alpha$. Thus, in view of Inequality (1) we have to show that

  $$tx - |rs| - |\widehat{pr}| = tx - \left(D - \left(D\cos\alpha \pm \sqrt{x^2 + D^2(\cos^2\alpha - 1)}\right)\right) - D\alpha \geqslant 0, \tag{2}$$

  for all $D\sin\left(\frac{\pi}{9}\right) \leqslant x \leqslant \frac{D\pi}{2t}$ and $0 \leqslant \alpha \leqslant \arcsin\left(\frac{x}{D}\right)$. Without loss of generality assume that $D = 1$. Observe that in the range for x and α, the radicand in $\sqrt{x^2 + \cos^2\alpha - 1}$ is non-negative. Also, it is sufficient to show that Inequality (2) holds for the minus sign in the $\pm$. That is, it is sufficient to show that

  $$tx - \alpha - 1 + \cos\alpha - \sqrt{x^2 + \cos^2\alpha - 1} \geqslant 0, \tag{3}$$

  for all $\sin\left(\frac{\pi}{9}\right) \leqslant x \leqslant \frac{\pi}{2t}$ and $0 \leqslant \alpha \leqslant \arcsin(x)$.
  In the full version of the paper we prove that Inequality (3) holds for $t \approx 1.879534$ and $t < 1.88$. This implies that p is 1.879534-good, and consequently 1.88-good for s. The sketch of the proof is given in Section 4. In fact, in Section 4 we will prove a slightly stronger result:

  $$tx - \alpha - (1 + 3 * 10^{-4}) + \cos\alpha - \sqrt{x^2 + \cos^2\alpha - 1} \geqslant 0$$

  holds for $t = 1.879534$ and all $\sin\left(\frac{\pi}{9}\right) \leqslant x \leqslant \frac{1.0001\pi}{2t}$ and $0 \leqslant \alpha \leqslant \arcsin(x)$.

We can show that Inequality (3) holds for $t = 1.879534$ and $0 \leqslant x \leqslant \frac{\pi}{2t}$. However, we considered $x = |ps| \leqslant D\sin\left(\frac{\pi}{9}\right)$ as a different case in order to unify the proof for Inequality (3) with the proof for Inequality (4) that we will see in Section 4.

4 Approximate-Diametral Points are Good

Let S be a finite set of at least two points in the plane that is in convex position. In Section 3 we proved that any diametral point of S is 1.88-good. In this section, we first present an algorithm that computes an approximate diametral pair of S; this algorithm is due to Janardan [9]. Then we show that the two points obtained by this algorithm are 1.88-good for S. In Section 5, we use this algorithm to compute a plane 1.88-spanner in linear time.

Let $c \geqslant 2$ be an integer-valued parameter. We use a family of coordinate systems, $\mathcal{C}_i$, $1 \leqslant i \leqslant c$, with orthogonal axes X_i and Y_i, respectively, where X_1 is horizontal and for $i = 2, ..., c$, X_i makes an angle of π/c with X_{i-1}. For each i we refer to the pair of points with minimum and maximum X_i-coordinates as the *extreme pair* in $\mathcal{C}_i$. To find an approximate diametral pair, we determine the Euclidean distance of the extreme pair in each $\mathcal{C}_i$ and report the pair that is farthest apart. The following lower bound on the distance of the reported extreme pair has been established by Janardan [9].

▶ **Lemma 10** (see Janardan [9]). *Let S be a finite set of at least two points in the plane that is in convex position, and whose diameter is D. Let p and q be the pair of points obtained by running the above algorithm on S. Then $|pq| \geqslant \sin\left(\frac{c-1}{c}\frac{\pi}{2}\right) D$.*

In the rest of this section we will prove the following theorem.

▶ **Theorem 11.** *Let S be a finite set of at least two points in the plane that is in convex position. Let p and q be the pair of points obtained by running the above algorithm on S with $c = 112$. Then both p and q are 1.88-good for S.*

Throughout the rest of this section, let $t = 1.88$. Because of symmetry, we prove Theorem 11 only for p. For each $i \in \{1, \ldots, 112\}$ and for each point $s \in S$, let $X_i(s)$ be the X_i-coordinate of s in the coordinate system $\mathcal{C}_i$. Moreover, let $l_i(s)$ be the line passing through s that is parallel to Y_i.

Let $\mathcal{C}_{pq}$ be the set of all coordinate systems in which p and q are the extreme pair. Note that $\mathcal{C}_{pq}$ is not empty, because p and q are the pair of points reported by the algorithm, and hence they are extreme pairs in at least one of the coordinate systems. Let $\mathcal{C}_{pq} = \{\mathcal{C}_{i_1}, \ldots, \mathcal{C}_{i_m}\}$, where $1 \leqslant m \leqslant 112$. Note that for each $j \in \{i_1, \ldots, i_m\}$ the points of S lie in the slab between the two parallel lines $l_j(p)$ and $l_j(q)$. For each $\mathcal{C}_j$, where $j \in \{i_1, \ldots, i_m\}$, let r_j be the point on $l_j(q)$ such that $\angle pr_jq = \frac{\pi}{2}$, and let $\alpha_j = \angle qpr_j$; observe that $\alpha_j \leqslant \frac{\pi}{2}$. See Figure 4.

Let k be an element of $\{i_1, \ldots, i_m\}$ for which α_k is minimum. Recall that all points of S are in the slab between $l_k(p)$ and $l_k(q)$.

▶ **Lemma 12.** $\alpha_k \leqslant \frac{\pi}{112}$.

Proof. The proof is by contradiction; thus, we assume that $\alpha_k > \frac{\pi}{112}$. Without loss of generality, assume $l_k(p)$, and consequently $l_k(q)$, are horizontal, p is below q, and q is to the right of $R(p \to r_k)$; see Figure 4. Let l_p and l_q be the lines that are perpendicular to pq and pass through p and q, respectively. Observe that each of l_p and l_q makes angle α_k with each of $l_k(p)$ and $l_k(q)$. Since $\alpha_k > \frac{\pi}{112}$, there is a coordinate system $\mathcal{C}_{k'} \in \{\mathcal{C}_1, \ldots, \mathcal{C}_{112}\}$ that is different from $\mathcal{C}_k$ and for which $l_{k'}(p)$ (resp. $l_{k'}(q)$) makes angle $\frac{\pi}{112}$ with $l_k(p)$ (resp. $l_k(q)$) and angle $\alpha_k - \frac{\pi}{112} > 0$ with l_p (resp. l_q). See Figure 4. We consider the following two cases.

- *All points of $S \setminus \{p, q\}$ are between $l_{k'}(p)$ and $l_{k'}(q)$.* Then all points of S lie in the shaded area in Figure 4(a). In this case p and q are the extreme pair in $\mathcal{C}_{k'}$. Thus $\mathcal{C}_{k'} \in \mathcal{C}_{pq}$ with $\alpha_{k'} = \alpha_k - \frac{\pi}{112}$. This contradicts our choice of k.

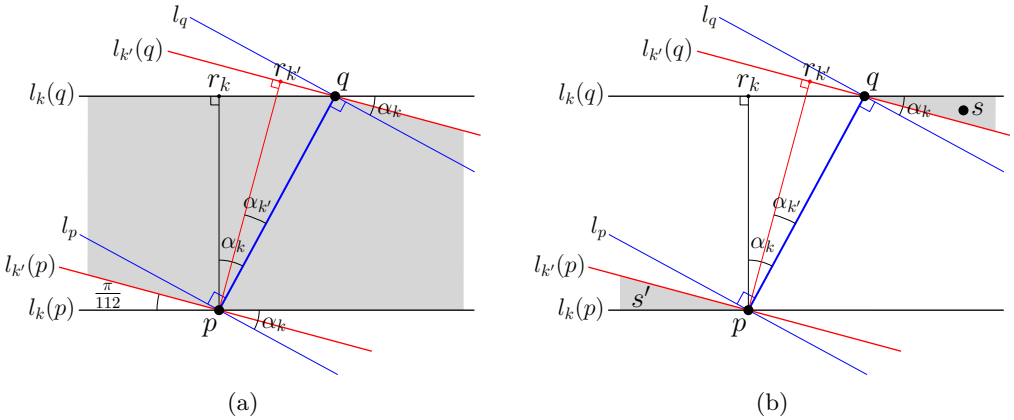


Figure 4 Proof of Lemma 12.

- *There is a point of $S \setminus \{p, q\}$ below $l_{k'}(p)$ or above $l_{k'}(q)$.* Without loss of generality assume there is a point of $S \setminus \{p, q\}$ that is above $l_{k'}(q)$. See Figure 4(b). In this case one of the extreme points of $C_{k'}$, say s, is above $l_{k'}(q)$ and its other extreme point, say s', is on or below $l_{k'}(p)$. Note that s is different from q while s' can be p. Observe that $|ss'| \geqslant |sp| > |pq|$. This contradicts the algorithm's choice of p and q as the farthest pair among the extreme pairs of all coordinate systems $\mathcal{C}_1, \ldots, \mathcal{C}_{112}$. ◀

Let D be the diameter of S. Recall that p and q are the pair of points that are returned by Janardan's algorithm. Let $|pq| = d$. By Lemma 10, we have

$$d \geqslant \sin\left(\frac{111\pi}{224}\right) D > 0.999901D,$$

and thus,

$$D < 1.0001d.$$

Note that all points of S are in the intersection of the two disks $C(p, D)$ and $C(q, D)$. See Figure 5. Let s be any point of S. We are going to show that p is t-good for s. Depending on $|ps|$ we consider the following three cases:

- $|ps| > \frac{D\pi}{2t}$. By Lemma 6, p is t-good for s.
- $|ps| < d \sin\left(\frac{\pi}{9}\right)$. Consider the coordinate system $\mathcal{C}_k$. Recall that $\mathcal{C}_k$ belongs to $\mathcal{C}_{pq}$, and by Lemma 12 we have $\alpha_k = \angle qpr_k \leqslant \frac{\pi}{112}$. Thus, q belongs to an interval $[q_1, q_2]$ on $l_k(q)$ such that $\angle q_1pr_k = \angle q_2pr_k = \frac{\pi}{112}$ and for each point $q' \in [q_1, q_2]$ we have $\angle q'pr_k \leqslant \frac{\pi}{112}$. Without loss of generality assume s is to the right of $R(p{\to}q)$. See Figure 5(a). Let r be the intersection point of $R(q{\to}s)$ with $l_k(p)$. Consider the path $\delta^{ccw}_{CH(S)}(p, s)$. Because of convexity, this path is to the right of $R(q{\to}s)$ and to the right of $R(p{\to}s)$. By Theorem 3, we have $|\delta^{ccw}_{CH(S)}(p, s)| \leqslant |pr| + |rs|$. Let $\alpha = \angle pqs$ and $\beta = \angle prs = \angle prq$. As in the proof of Theorem 9, we have $\sin\alpha \leqslant \frac{|ps|}{|pq|} = \frac{|ps|}{d} < \sin\left(\frac{\pi}{9}\right)$ and, thus, $\alpha < \frac{\pi}{9}$. Since $\angle qpr \leqslant \frac{\pi}{2} + \frac{\pi}{112}$, it follows that $\beta = \pi - \alpha - \angle qpr > \pi - \frac{\pi}{9} - \left(\frac{\pi}{2} + \frac{\pi}{112}\right) = \frac{383\pi}{1008}$. Since $t = 1.88$, we have $\beta > 2\arcsin(\frac{1}{t})$. Thus, using Lemma 7, we have

$$\frac{|\delta^*_{CH(S)}(p, s)|}{|ps|} \leqslant \frac{|\delta^{ccw}_{CH(S)}(p, s)|}{|ps|} \leqslant \frac{|pr| + |rs|}{|ps|} \leqslant t,$$

which implies that p is t-good for s.

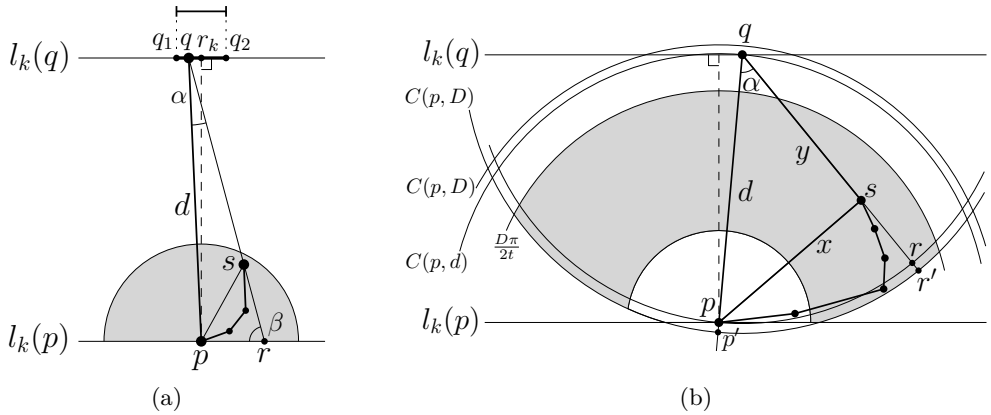


Figure 5 Proof of Theorem 11: (a) $|ps| < d \sin\left(\frac{\pi}{9}\right)$, and (b) $d \sin\left(\frac{\pi}{9}\right) \leqslant |ps| \leqslant \frac{D\pi}{2t}$.

$d \sin\left(\frac{\pi}{9}\right) \leqslant |ps| \leqslant \frac{D\pi}{2t}$. In this case s is in the shaded region of Figure 5(b). Consider $C(q, d)$ and $C(q, D)$; note that all points of S are in $C(q, D)$. Without loss of generality assume s is to the right of $R(p{\to}q)$. Let r and r' be the intersection points of $R(q{\to}s)$ with the boundaries of $C(q, d)$ and $C(q, D)$, respectively. Let p' be the intersection point of $R(q{\to}p)$ with the boundary of $C(q, D)$. Consider the path $\delta^{ccw}_{CH(S)}(p, s)$. Because of convexity, this path is to the right of $R(q{\to}s)$ and to the right of $R(p{\to}s)$. See Figure 5(b). Thus, Theorem 3 implies that $|\delta^{ccw}_{CH(S)}(p, s)| \leqslant |pp'| + \widehat{|p'r'|} + |r'r| + |rs|$, where $\widehat{|p'r'|}$ denotes the length of the counter-clockwise arc on $C(q, D)$ from p' to r'. Note that $|pp'| = |rr'| = D - d < 0.0001d$. Let $\alpha = \angle pqs$. Note that α is maximum when $R(q{\to}s)$ is tangent to $C\left(p, \frac{D\pi}{2t}\right)$. This implies that $\alpha \leqslant \arcsin\left(\frac{D\pi}{2td}\right) < \arcsin\left(\frac{1.0001\pi}{2t}\right) < 1$. Thus,

$$\widehat{|p'r'|} = D\alpha < 1.0001d\alpha = d\alpha + 0.0001d\alpha < \widehat{|pr|} + 0.0001d,$$

where $\widehat{|pr|}$ denotes the length of the counter-clockwise arc on $C(q, d)$ from p to r. Therefore, we have

$$|\delta^{*}_{CH(S)}(p, s)| \leqslant |\delta^{ccw}_{CH(S)}(p, s)| \leqslant |pp'| + \widehat{|p'r'|} + |r'r| + |rs| < |rs| + \widehat{|pr|} + 0.0003d.$$

In order to prove that p is t-good for s, it is sufficient to prove that

$$\frac{|rs| + \widehat{|pr|} + 0.0003d}{|ps|} \leqslant t,$$

or equivalently

$$t|ps| - |rs| - \widehat{|pr|} - 0.0003d \geqslant 0,$$

for all $d \sin\left(\frac{\pi}{9}\right) \leqslant |ps| \leqslant \frac{D\pi}{2t}$. Without loss of generality assume that $d = 1$, and thus, $D < 1.0001$. In view of the proof of Theorem 9 it turns out that we have to prove that

$$tx - \alpha - (1 + 3 * 10^{-4}) + \cos\alpha - \sqrt{x^2 + \cos^2\alpha - 1} \geqslant 0, \tag{4}$$

for all $\sin\left(\frac{\pi}{9}\right) \leqslant x \leqslant \frac{1.0001\pi}{2t}$ and $0 \leqslant \alpha \leqslant \arcsin(x)$.
In the full version of the paper we prove that Inequality (4) holds for $t \approx 1.879534$ and $t < 1.88$. This implies that p is 1.879534-good, and consequently 1.88-good for s.

To prove Inequality (4) we do the following. Let $\varepsilon = 10^{-4}$. The goal is to find the smallest value of t such that

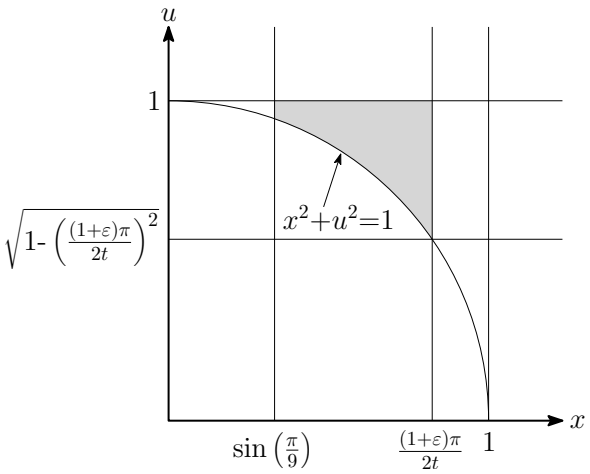
$$tx - \alpha - (1+3\varepsilon) + \cos(\alpha) - \sqrt{x^2 + \cos^2(\alpha) - 1} \geqslant 0,$$

for all $\sin\left(\frac{\pi}{9}\right) \leqslant x \leqslant \frac{(1+\varepsilon)\pi}{2t}$, $0 \leqslant \alpha \leqslant \arcsin(x)$. Note that for the left-hand side of the inequality to be well-defined, we need $x^2 + \cos^2(\alpha) \geqslant 1$. Since $x \leqslant \frac{(1+\varepsilon)\pi}{2t}$, we can re-write the constraints on α as $0 \leqslant \alpha \leqslant \arcsin\left(\frac{(1+\varepsilon)\pi}{2t}\right) = \arccos\left(\sqrt{1 - \left(\frac{(1+\varepsilon)\pi}{2t}\right)^2}\right)$.

Let $u = \cos(\alpha)$. This problem is equivalent to finding the smallest value of t for which

$$f(x,u) = tx - \arccos(u) - (1+3\varepsilon) + u - \sqrt{x^2 + u^2 - 1} \geqslant 0, \tag{5}$$

for all $\sin\left(\frac{\pi}{9}\right) \leqslant x \leqslant \frac{(1+\varepsilon)\pi}{2t}$, $\sqrt{1 - \left(\frac{(1+\varepsilon)\pi}{2t}\right)^2} \leqslant u \leqslant 1$ and $x^2 + u^2 \geqslant 1$.



Thus we want to verify the validity of Inequality (5) in the shaded region of the above figure. In the full version of the paper we show that for $t \approx 1.879534$, $f(x,u) \geqslant 0$ in the region defined by the constraints on x and u.

5 Algorithms

Let S be a set of n points in the plane that is in convex position. We assume that the points of S are given in sorted order along $CH(S)$. In this section, we describe how to construct a plane 1.88-spanner on S in $O(n)$ time.

By Theorem 9, any diametral point of S is 1.88-good for S. As discussed in the proof of Theorem 1, by running algorithm PlaneSpanner$(S, 1.88)$, a plane 1.88-spanner for S is obtained. Specifically, we obtain this spanner by choosing, in line 4 of the algorithm, a diametral point of S. Since the diameter of n points in convex position can be computed in $O(n)$ time, the algorithm runs in $O(n^2)$ time.

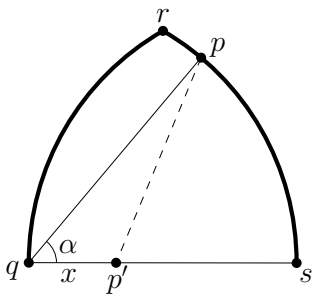
Note that in each iteration of the **while** loop in algorithm PlaneSpanner, we remove one point from S. Thus, any deletion-only data structure that maintains the diameter of S can be used here. In 2010, Chan [3] showed that the diameter of a fully dynamic point set in the plane can be maintained in $O(\log^8 n)$ expected amortized time. Based on that, algorithm PlaneSpanner can be implemented to run in $O(n \log^8 n)$ expected time.

Recall that in Section 4, we presented an algorithm that computes an approximate diametral pair of S. By Theorem 11, these diametral points are 1.88-good (assuming $c = 112$). Based on this algorithm, we present a deletion-only data structure that maintains an approximate diametral pair of S. For each i, $1 \leqslant i \leqslant c$, we store the points of $\mathcal{C}_i$ in a doubly connected linked list, L_i, in increasing order of their X_i-coordinates. The list L_i can be constructed in $O(n)$ time by merging the two convex chains of the points between the extreme pair in $\mathcal{C}_i$. The list L_i allows access to the extreme pair in $\mathcal{C}_i$ in $O(1)$ time, via explicitly-maintained pointers to the leftmost and rightmost nodes. For $i = 1, \ldots, c-1$ and for each point p in L_i, we store a cross pointer to the occurrence of p in L_{i+1}. Moreover, for any point p in L_c we store a cross pointer to the occurrence of p in L_1. To delete a point p from S, we delete p from each L_i, $1 \leqslant i \leqslant c$. If we are given a pointer to p's occurrence in one list L_i, then p can be deleted in $O(c)$ time by following the cross pointers. To answer a diameter query, we determine the Euclidean distance of the extreme pair in each L_i and report the pair that is farthest apart; this takes $O(c)$ time. We use this data structure, with $c = 112$, in line 4 of algorithm PlaneSpanner. Thus, each query takes $O(1)$ time and gives two pointers to the approximated diametral points. Using the cross pointers, the approximated diametral points can be deleted in $O(1)$ time. Thus, algorithm PlaneSpanner can be implemented to run in $O(n)$ time. Therefore, we have proved the following theorem.

▶ **Theorem 13.** *Let S be a set of n points in the plane that is in convex position. Assume that the points of S are given in sorted order along the boundary of the convex hull of S. Then a plane 1.88-spanner for S can be computed in $O(n)$ time.*

6 Remarks

1. *There exists a point set in the plane and in convex position such that some of its diametral points are not 1.868-good.*
   The figure below shows a point set S that contains the points p, q, r, s, p' and many points that are uniformly distributed on each of the arcs $\widehat{qr}$ and $\widehat{rs}$.



   The points q, r, and s are the vertices of an equilateral triangle of side length 1. The arc $\widehat{qr}$ (resp. $\widehat{rs}$) has radius 1 and is centered at s (resp. q). The point p' is placed on qs and at distance x from q. The point p is placed on $\widehat{rs}$ such that $\angle p'qp = \alpha$. Note that $0 < x < 1$ and $0 < \alpha < \pi/3$. We will compute the exact values of x and α later. Note that all points of S, except p', are diametral points. Moreover $|CH(S)| \approx 1 + \frac{2\pi}{3}$. We are going to place p and p' (or equivalently, choosing α and x) such that p is not 1.868-good for p', and hence it is not 1.868-good for S.
   We place p and p' such that the lengths of the two paths between p and p' on $CH(S)$ are equal to $|\delta^*_{CH(S)}(p, p')| \approx 1/2+\pi/3$ and $|pp'|$ is minimized. In this way, $|\delta^*_{CH(S)}(p, p')|/|pp'|$

is maximized. The length of the path $\delta^*_{CH(S)}(p,p')$ that is to the left of $R(p\to p')$ is $\alpha+1-x$. Thus, $\alpha+1-x=1/2+\pi/3$, which implies that $x=\alpha+1/2-\pi/3$. By the law of cosines we have

$$|pp'|=\sqrt{1+x^2-2x\cos\alpha}.$$

The value of α that minimizes $|pp'|$ is the solution of the equation

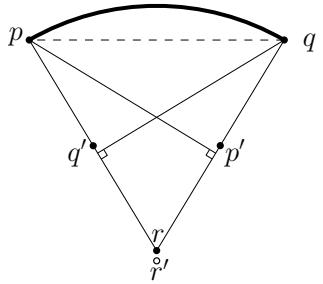
$$(6\alpha+3-2\pi)(1+\sin\alpha)-6\cos\alpha=0,$$

which is $\alpha\approx 0.897287$. Thus, we choose $\alpha=0.897287$ and $x=\alpha+1/2-\pi/3$. For these values of α and x we have $|pp'|\approx 0.828153$ and hence,

$$\frac{|\delta^*_{CH(S)}(p,p')|}{|pp'|}\approx 1.868.$$

Thus, the diametral point p is not 1.868-good for p', and hence is not 1.868-good for S.

2. *There exists a point set in the plane and in convex position such that none of its diametral points is* 1.75*-good.*
The figure below shows a point set S that contains the points p,q,r,p',q', and many points that are uniformly distributed on the arc $\widehat{pq}$.

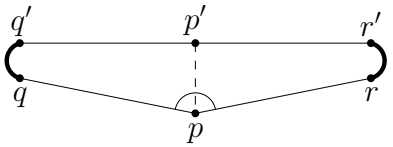


The points p, q, and r' are the vertices of an equilateral triangle of side-length 1; note that r' does not belong to S. The arc $\widehat{pq}$ is centered at r' and has radius 1. The point r is placed at distance $\varepsilon>0$ vertically above r'. Thus, p and q are the only diametral points in S. Moreover, $|CH(S)|\approx 2+\frac{\pi}{3}$. The point p' (resp. q') is placed on rq (resp. rp) and at distance $\frac{\pi}{6}$ from r. Thus $|\delta^*_{CH(S)}(p,p')|=|\delta^*_{CH(S)}(q,q')|\approx 1+\frac{\pi}{6}$. By the law of cosines we have $|pp'|=|qq'|\approx\frac{1}{6}\sqrt{36+\pi^2-6\pi}$. Thus,

$$\frac{|\delta^*_{CH(S)}(p,p')|}{|pp'|}=\frac{|\delta^*_{CH(S)}(q,q')|}{|qq'|}\approx 1.758.$$

This implies that p is not 1.75-good for p', and q is not 1.75-good for q'. Therefore, none of the diametral points of S is 1.75-good for S.

3. Intuitively, it seems that the point on the convex hull that has the smallest internal angle with its neighboring points is a suitable candidate to be a good point. But this is not true; the figure below shows a point set S that contains the points p,q,r,p',q',r', and many points that are uniformly distributed on each of the arcs $\widehat{qq'}$ and $\widehat{rr'}$.



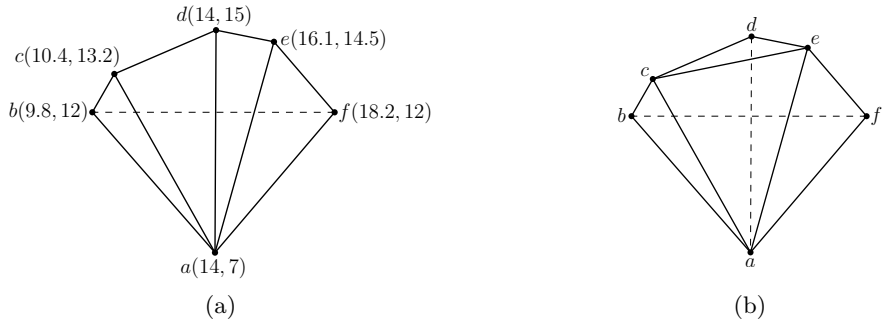


Figure 6 (a) Delaunay triangulation. (b) The graph computed by algorithm PlaneSpanner when it removes both points of a diametral pair in each iteration.

The point p is placed vertically below the midpoint of qr, and p' is placed on the midpoint of $q'r'$. Depending on the lengths of pr and pq, and on the distance between p and the midpoint of qr, the value of $|\delta^*_{CH(S)}(p,p')|/|pp'|$ can be arbitrary large. Thus, for any $t > 1$, we can select S such that p is not t-good for p', and hence it is not t-good for S.

4. There are point sets in the plane and in convex position for which the plane graph that is computed by algorithm PlaneSpanner has smaller stretch factor than the Delaunay triangulation of the same point set. Consider the set $S = \{a, b, c, d, e, f\}$ of six points in Figure 6. Figure 6(a) shows the Delaunay triangulation of S whose stretch factor is $(|bc| + |cd| + |de| + |ef|)/|bf| \approx 1.284$. Figure 6(b) shows the plane graph G obtained by algorithm PlaneSpanner when it removes both points of a diametral pair in each iteration. The points b and f are the only diametral pair in S, thus, in the first iteration ae and ac are added to G. In the next iteration a and d are the only diametral pairs, thus, the edge ec is added to G. The stretch factor of G is $(|ae| + |ed|)/|ad| \approx (|bc| + |ce| + |ef|)/|bf| \approx 1.244$. Note that there are point sets for which the Delaunay triangulation has a smaller stretch factor than the graph that is computed by algorithm PlaneSpanner.

5. The implementation of algorithm PlaneSpanner in Theorem 1 gives a simple (and surprising) $O(n)$–time algorithm for computing the closest pair in a set of n points in convex position: As discussed in Section 5, this algorithm computes a 1.88-spanner G in $O(n)$ time. It is well known that in any t-spanner, for any $t < 2$, the closest pair is connected by an edge. Thus, given G, the closest pair can be computed in $O(n)$ time.

7 Conclusions and Future Work

For a point set S in the plane and in convex position, we have shown that any approximate diametral point of S is 1.88-good. Based on this, we obtained a plane 1.88-spanner for S in $O(n)$ time. We have proved that $\sqrt{3} \leqslant t^* \leqslant 1.88$. By solving Inequality (3) directly, or by considering more coordinate systems in the approximate-diameter algorithm, we can show that any (approximate) diametral point of S is 1.8792-good. This implies that $t^* \leqslant 1.8792$. A natural problem is to improve any of the provided bounds. Another natural problem is to extend algorithm PlaneSpanner to point sets that are not in convex position.

References

1. R. V. Benson. *Euclidean geometry and convexity.* McGraw-Hill, 1966.
2. A. L. Cauchy. Note sur divers théorèmes relatifs á la rectification des courbes et á la quadrature des surfaces. *C. R. Acad. Sci. Paris*, 13:1060–1065, 1841.
3. T. M. Chan. A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. *J. ACM*, 57(3), 2010.
4. L. P. Chew. There is a planar graph almost as good as the complete graph. In *Proceedings of the 2nd ACM Symposium on Computational Geometry*, pages 169–177, 1986.
5. L. P. Chew. There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences*, 39:205–219, 1989.
6. S. Cui, I. A. Kanj, and G. Xia. On the stretch factor of Delaunay triangulations of points in convex position. *Computational Geometry: Theory and Applications*, 44:104–109, 2011.
7. D. P. Dobkin, S. J. Friedman, and K. J. Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete & Computational Geometry*, 5:399–407, 1990.
8. A. Dumitrescu and A. Ghosh. Lower bounds on the dilation of plane spanners. In *Proceedings of the 2nd International Conference on Algorithms and Discrete Applied Mathematics, CALDAM*, pages 139–151, 2016.
9. R. Janardan. On maintaining the width and diameter of a planar point-set online. *Int. J. Comput. Geometry Appl.*, 3(3):331–344, 1993.
10. J. M. Keil and C. A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete & Computational Geometry*, 7:13–28, 1992.
11. W. Mulzer. Minimum dilation triangulations for the regular n-gon. Master's thesis, Freie Universität Berlin, Germany, 2004.
12. G. Narasimhan and M. Smid. *Geometric Spanner Networks.* Cambridge University Press, Cambridge, UK, 2007.
13. G. Xia. The stretch factor of the Delaunay triangulation is less than 1.998. *SIAM Journal on Computing*, 42:1620–1659, 2013.

Approximating the Integral Fréchet Distance

Anil Maheshwari[1], Jörg-Rüdiger Sack[2], and Christian Scheffer[3]

1 **School of Computer Science, Carleton University, Ottawa, Canada**
anil@scs.carleton.ca

2 **School of Computer Science, Carleton University, Ottawa, Canada**
sack@scs.carleton.ca

3 **Department of Computer Science, Braunschweig University of Technology, Braunschweig, Germany**
scheffer@ibr.cs.tu-bs.de

Abstract

We present a pseudo-polynomial time $(1+\varepsilon)$-approximation algorithm for computing the integral and average Fréchet distance between two given polygonal curves T_1 and T_2. The running time is in $\mathcal{O}(\zeta^4 n^4/\varepsilon^2)$ where n is the complexity of T_1 and T_2 and ζ is the maximal ratio of the lengths of any pair of segments from T_1 and T_2.

Furthermore, we give relations between weighted shortest paths inside a single parameter cell C and the monotone free space axis of C. As a result we present a simple construction of weighted shortest paths inside a parameter cell. Additionally, such a shortest path provides an optimal solution for the partial Fréchet similarity of segments for all leash lengths. These two aspects are related to each other and are of independent interest.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Fréchet distance, partial Fréchet similarity, curve matching

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.26

1 Introduction

Measuring similarity between geometric objects is a fundamental problem in many areas of science and engineering. Applications arise e.g., when studying animal behaviour, human movement, traffic management, surveillance and security, military and battlefield, sports scene analysis, and movement in abstract spaces [9, 10, 11]. Due to its practical relevance, the resulting algorithmic problem of curve matching has become one of the well-studied problems in computational geometry. One of the prominent measures of similarities between curves is given by the *Fréchet distance* and its variants.

In the well-known dog-leash metaphor, the (standard) *Fréchet distance* is described as follows: suppose a person walks a dog, while both have to move from the starting point to the ending point on their respective curves T_1 and T_2. Each pair of walks induces a matching between T_1 and T_2. The *Fréchet* distance is the minimum leash length required over all possible pairs of walks, if neither person nor dog is allowed to move backwards.

In this paper, we study the integral and average Fréchet distance originally introduced by Buchin [4]. The *integral Fréchet distance* is defined as the minimal integral of the distances between points that are matched by a pair of walks. The *average Fréchet distance* is defined as the integral Fréchet distance divided by the sum of the lengths of T_1 and T_2.

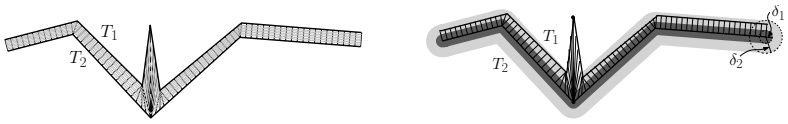


Figure 1 Left: A matching between T_1 and T_2. Right: The partial Fréchet similarity is unstable for distance thresholds between δ_1 and δ_2 where $\delta_1 \approx \delta_2$. In particular, for a leash length of δ_1, the partial Fréchet similarity of T_1 and T_2 is equal to zero, whereas for a leash length of δ_2, it is close to $|T_1| + |T_2|$. Furthermore, the traditional Fréchet distance is significantly enlarged by the peak of T_1.

1.1 Related Work

Alt and Godau [1] showed how to compute the Fréchet distance between two polygonal curves T_1 and T_2 in $\mathcal{O}(n^2 \log(n))$ time, where n is the complexity of T_1 and T_2. In the presence of outliers though, the Fréchet distance may not provide an appropriate result. This is due to the fact that the Fréchet distance measures the maximum of the matched distances. Thus, one large "peak" may substantially increase the Fréchet distance, see Figure 1 left.

To overcome the issue of outliers, Buchin et al. [3] introduced the *partial Fréchet similarity* and showed how to compute it in $\mathcal{O}(n^3 \log(n))$ time, where distances are measured w.r.t. the L_1 or L_∞ metric. The partial Fréchet similarity measures the cost of a matching as the lengths of the parts of T_1 and T_2 which are made up of matched point pairs whose distance to each other is upper-bounded by a given threshold $\delta \geq 0$, see Figure 1 right. De Carufel et al. [5] showed that the partial Fréchet similarity w.r.t. to the L_2 metric cannot be computed exactly over the rational numbers. Motivated by that, they gave a $(1 \pm \varepsilon)$-approximation algorithm guaranteeing a pseudo-polynomial running time. An alternative perspective on the partial Fréchet similarity is the partial Fréchet dissimilarity, i.e., the minimization of the portions on T_1 and T_2 which are involved in distances that are larger than δ.

Unfortunately, both the partial Fréchet similarity and dissimilarity are highly dependent on the choice of δ as provided by the user. As a function of δ, the partial Fréchet distance is unstable, i.e., arbitrarily small changes of δ can result in arbitrarily large changes of the partial Fréchet (dis)similarly, see Figure 1 right.

An approach related to the integral Fréchet distance is dynamic time warping (DTW), which arose in the context of speech recognition [12]. Here, a discrete version of the integral Fréchet distance is computed via dynamic programming. This is not suitable for general curve matching (see [7, p. 204]). Efrat et al. [7] worked out an extension of the idea of DTW to a continuous version. In particular, they compute shortest path distances on a combinatorial piecewise linear 2-manifold that is constructed by taking the Minkowski sum of T_1 and T_2. Furthermore, they gave two approaches dealing with that manifold. The first one does not yield an approximation of the integral Fréchet distance. The second one does not lead to theoretically provable guarantees.

1.2 Contributions

We present the first (pseudo-)polynomial time algorithm that approximates the integral Fréchet Distance, $\mathcal{F}_S(T_1, T_2)$, up to a multiplicative error of $(1 + \varepsilon)$.

As a by-product, we show that a shortest weighted path π_{ab} between two points a and b inside a parameter cell C can be computed in constant time. We also make the observation that π_{ab} provides an optimal matching for the partial Fréchet similarity for all leash length thresholds. This provides a natural extension of locally correct Fréchet matchings that were first introduced by Buchin et al. [2]. They suggest to: “restrict to the locally correct matching

that decreases the matched distance as quickly as possible."[2, p. 237]. The matching induced by π_{ab} fulfils this requirement.

2 Preliminaries

Let $T_1, T_2 : [0, n] \to \mathbb{R}^2$ be two polygonal curves. We denote the first derivative of a function f by f'. By, $||\cdot||_p$, we denote the p-norm and by $d_p(\cdot,\cdot)$ its induced L_p metric. The *lengths* $|T_1|$ *and* $|T_2|$ of T_1 and T_2 are defined as $\int_0^n ||(T_1)'(t)||_2\, dt$ and $\int_0^n ||(T_2)'(t)||_2\, dt$, respectively. To simplify the exposition, we assume that $|T_1| = |T_2| = n$ and that T_1 and T_2 each have n segments. A *reparametrization* is a continuous function $\alpha : [0, n] \to [0, n]$ with $\alpha(0) = 0$ and $\alpha(n) = n$. A reparameterization α is *monotone* if $\alpha(t_1) \leq \alpha(t_2)$ holds for all $0 \leq t_1 \leq t_2 \leq n$. A *(monotone) matching* is a pair of (monotone) reparametrizations (α_1, α_2). The *Fréchet distance* of T_1 and T_2 w.r.t. d_2 is defined as $\mathcal{D}(T_1, T_2) = \inf_{(\alpha_1,\alpha_2)} \max_{t\in[0,n]} d_2(T_1(\alpha_1(t)), T_2(\alpha_2(t)))$.

For a given leash length $\delta \geq 0$, Buchin et al. [3] define the *partial Fréchet similarity* $\mathcal{P}_{(\alpha_1,\alpha_2)}(T_1, T_2)$ *w.r.t. a matching* (α_1, α_2) as

$$\int_{d_2(T_1(\alpha_1(t)),T_2(\alpha_2(t)))\leq\delta} \left(||\left(T_1 \circ \alpha_1\right)'(t)||_2 + ||\left(T_2 \circ \alpha_2\right)'(t)||_2\right) dt$$

and the *partial Fréchet similarity* as $\mathcal{P}_\delta(T_1, T_2) = \sup_{\alpha_1,\alpha_2} \mathcal{P}_{(\alpha_1,\alpha_2)}(T_1, T_2)$.

Given a monotone matching (α_1, α_2), the *integral Fréchet distance* $\mathcal{F}_{\mathcal{S},(\alpha_1,\alpha_2)}(T_1, T_2)$ *of* T_1 *and* T_2 *w.r.t.* (α_1, α_2) is defined as:

$$\int_0^n d_2(T_1(\alpha_1(t)), T_2(\alpha_2(t)))\left(||\left(T_1 \circ \alpha_1\right)'(t)||_2 + ||\left(T_2 \circ \alpha_2\right)'(t)||_2\right) dt$$

and the *integral Fréchet distance* as $\mathcal{F}_\mathcal{S}(T_1, T_2) = \inf_{(\alpha_1,\alpha_2)} \mathcal{F}_{\mathcal{S},(\alpha_1,\alpha_2)}(T_1, T_2)$ [4]. Note that the derivatives of $(T_1 \circ \alpha_1)(\cdot)$ and $(T_2 \circ \alpha_2)(\cdot)$ are measured w.r.t. the L_2-norm because the lengths of T_1 and T_2 are measured in Euclidean space. Furthermore, $(T_1 \circ \alpha_1)'(t)$ and $(T_1 \circ \alpha_1)'(t)$ are well defined for all $t \in [0, n]$ because $(T_1 \circ \alpha_1)'(\cdot)$ and $(T_1 \circ \alpha_1)'(\cdot)$ are piecewise continuously differentiable. The *average Fréchet distance* is defined as $\mathcal{F}_S(T_1, T_2)/(|T_1| + |T_2|)$ [4].

The *parameter space* P of T_1 and T_2 is an axis aligned rectangle. The bottom-left corner $\mathfrak{s}$ and upper-right corner $\mathfrak{t}$ correspond to $(0, 0)$ and (n, n), respectively. We denote the x- and the y-coordinate of a point $a \in P$ by $a.x$ and $a.y$, respectively. A point $b \in P$ *dominates* a point $a \in P$, denoted by $a \leq_{xy} b$, if $a.x \leq b.x$ and $a.y \leq b.y$ hold. A path π is *(xy-) monotone* if $\pi(t_1) \leq \pi(t_2)$ holds for all $0 \leq t_1 \leq t_2 \leq n$. Thus, a monotone matching corresponds to a monotone path π with $\pi(0) = \mathfrak{s}$ and $\pi(n) = \mathfrak{t}$. By inserting $n+1$ vertical and $n+1$ horizontal *parameter lines*, we refine P into n rows and n columns such that the i-th row (column) has a height (resp., width) that corresponds to the length of the i-th segment on T_1 (resp., T_2). This induces a partitioning of P into cells, called *parameter cells*.

For $a, b \in P$ with $a \leq_{xy} b$, we have $||ab||_1 = \int_{a.x}^{b.x} ||(T_1)'(t)||_2\, dt + \int_{a.y}^{b.y} ||(T_2)'(t)||_2\, dt$. This is equal to the sum of the lengths of the subcurves between $T_1(a.x)$ and $T_1(b.x)$ and between $T_2(a.y)$ and $T_2(b.y)$. Thus, we define the *length* $|\pi|$ *of a path* $\pi : [0, n] \to P$ as $\int_0^n ||(\pi)'(t)||_1\, dt$. Note that for the paths inside the parameter space the 1-norm is applied, while the lengths of the curves in the Euclidean space are measured w.r.t. the 2-norm. As $\mathcal{F}_S(T_1, T_2)$ measures the length of T_1 and T_2 at which each $(T_1(\alpha_1(t)), T_2(\alpha_2(t)))$ is weighted by $d_2(T_1(\alpha_1(t)), T_2(\alpha_2(t)))$, we consider the *weighted length* of π defined as follows:

Let $w(\cdot) : P \to \mathbb{R}_{\geq 0}$ be defined as $w((x, y)) := d_2(T_1(x), T_2(y))$ for all $(x, y) \in P$. The weighted length $|\pi|_w$ of a path $\pi : [a, b] \to P$ is defined as $\int_a^b w(\pi(t))\, ||(\pi)'(t)||_1 dt$.

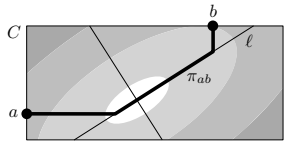


Figure 2 A weighted shortest xy-monotone path π_{ab} between two points $a, b \in C$, where $a \leq_{xy} b$.

▶ **Observation 1** ([4]). *Let π be a shortest weighted monotone path between $\mathfrak{s}$ and $\mathfrak{t}$ inside P. Then, we have $|\pi|_w = \mathcal{F}_{\mathcal{S}}(T_1, T_2)$.*

Motivated by Observation 1, we approximate $\mathcal{F}_S(T_1, T_2)$ by approximating the length of a shortest weighted monotone path $\pi \subset P$ connecting $\mathfrak{s}$ and $\mathfrak{t}$. Let $\delta \geq 0$ be chosen arbitrarily, but fixed. Inside each parameter cell C, the union of all points p with $w(p) \leq \delta$ is equal to the intersection of an ellipse $\mathcal{E}$ with C. Observe that $\mathcal{E}$ can be computed in constant time [1]. $\mathcal{E}$ is characterized by two focal points F_1 and F_2 and a radius r such that $\mathcal{E} = \{p \in \mathbb{R}^2 \mid d_2(p, F_1) + d_2(p, F_2) \leq r\}$. The two axes ℓ (monotone) and $\hbar$ (not monotone) of $\mathcal{E}$, called the *free space axes*, are defined as the line induced by F_1 and F_2 and the bisector between F_1 and F_2. If $\mathcal{E}$ is a disc, ℓ and $\hbar$ are the lines with gradients 1 and -1 and which cross each other in the middle of $\mathcal{E}$. Note that the axes are independent of the value of δ.

To approximate $|\pi|_w$ efficiently we make the following observation that is of independent interest: Let a, b be two parameter points that lie in the same parameter cell C such that $a \leq_{xy} b$. The shortest weighted monotone path π_{ab} between a and b (that induces an optimal solution for the integral Fréchet distance) is the monotone path between a and b that maximizes its subpaths that lie on ℓ (see Figure 2 and Lemma 7). Another interesting aspect of π_{ab} is that it also provides an optimal matching for the partial Fréchet similarity (between the corresponding (sub-)segments) for all leash lengths, as $\pi \cap \mathcal{E}_\delta$ has the maximal length for all $\delta \geq 0$, where $\mathcal{E}_\delta := \mathcal{E}$ for a specific $\delta \geq 0$. Next, we discuss our algorithms.

3 An Algorithm for Approximating Integral Fréchet Distance

We approximate the length of a shortest weighted monotone path between $\mathfrak{s}$ and $\mathfrak{t}$ as follows: We construct two weighted, directed, graphs $G_1 = (V_1, E_1, w_1)$ and $G_2 = (V_2, E_2, w_2)$ that lie embedded in P such that $\mathfrak{s}, \mathfrak{t} \in V_1$ and $\mathfrak{s}, \mathfrak{t} \in V_2$. Then, in parallel, we compute for G_1 and G_2 the lengths of the shortest weighted paths between $\mathfrak{s}$ and $\mathfrak{t}$. Finally, we output the minimum of both values as an approximation for $\mathcal{F}_S(T_1, T_2)$.

We introduce some additional terminology. A *geometric graph* $G = (V, E)$ is a graph where each $v \in V$ is assigned to a point $p_v \in P$, its *embedding*. The *embedding* of an edge $(u, v) \in E$ (into P) is $p_u p_v$. The *embedding of G (into P)* is $\bigcup_{(u,v)\in E} p_u p_v$. For $v \in V$ and $e \in E$, we denote simultaneously the vertex $v \in V$, the edge $e \in E$, and the graph (V, E) and their embeddings by v, e, and G, respectively. G is *monotone (directed)* if $p_u \leq_{xy} p_v$ holds for all $(u, v) \in E$. Let $R \subseteq P$ be an arbitrarily chosen axis aligned rectangle with height h and width b. The *grid (graph) of R with mesh size σ* is the geometric graph that is induced by the segments that are given as the intersections of R with the following lines: Let $h_1, \ldots, h_{k_1}$ be the $\lceil \frac{h}{\sigma} \rceil + 1$ equidistant horizontal lines and let $b_1, \ldots, b_{k_2}$ be the $\lceil \frac{b}{\sigma} \rceil + 1$ equidistant vertical lines such that $\partial R = R \cap (h_1 \cup h_{k_1} \cup b_1 \cup b_{k_2})$, where ∂R denotes the boundary of R.

3.1 Construction of G_1

Let μ be the length of a smallest segment from T_1 and T_2. We construct $G_1 = (V_1, E_1) \subset P$ as the monotone directed grid graph of P with a mesh size of $\frac{\varepsilon\mu^2}{40000(|T_1|+|T_2|)}$. Furthermore, we set $w_1((u,v)) := |uv|_w$ for all $(u,v) \in E_1$.

3.2 Construction of G_2

For $u \in P$ and $r \geq 0$, we consider the ball $B_r(u)$ with its center at u and a radius of r w.r.t. the L_∞ metric. For the construction of G_2 we need the free space axes of the parameter cells and so called grid balls:

▶ **Definition 2.** Let $u \in P$ and $r \geq 0$ be chosen arbitrarily. The *grid ball* $G_r(u)$ is defined as the grid of $B_r(u)$ that has a mesh size of $\frac{\varepsilon}{456}w(u)$. We say $G_r(u)$ *approximates* $B_r(u)$.

We define G_2 as the monotone directed graph that is induced by the arrangement that is made up of the following components restricted to P:

- (1) All monotone free space axes restricted to their corresponding parameter cell.
- (2) All grid balls $G_{62w(u)}(u)$ for $u := \arg\min_{p \in e} w(u)$ and any parameter edge e.
- (3) The segments $\mathfrak{s}c_\mathfrak{s}$ and $\mathfrak{t}c_\mathfrak{t}$ if the parameter cells $C_\mathfrak{s}$ and $C_\mathfrak{t}$ that contain $\mathfrak{s}$ and $\mathfrak{t}$ are intersected by their corresponding monotone free space axes $\ell_\mathfrak{s}$ and $\ell_\mathfrak{t}$, where $c_\mathfrak{s}$ and $c_\mathfrak{t}$ are defined as the bottom-leftmost and top-rightmost point of $\ell_\mathfrak{s} \cap C_\mathfrak{s}$ and $\ell_\mathfrak{t} \cap C_\mathfrak{t}$.

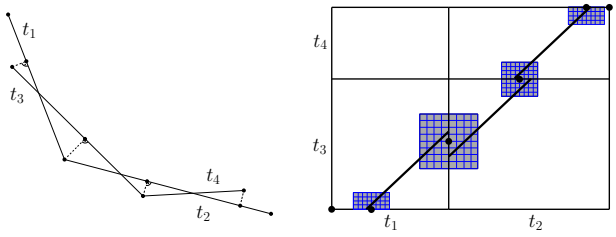


Figure 3 Exemplified construction of G_2 for two given polygonal curves T_1 and T_2. For simplicity, we only illustrate four grid balls (with reduced radii) and the corresponding point pairs from $T_1 \times T_2$.

Finally, we set $w_2((v_1,v_2)) := |v_1v_2|_w$ for all $(v_1,v_2) \in E_2$. For each edge $e \in G_2$, we choose the point $u \in e$ as the center of the corresponding grid ball because the free space axes of the parameters cells adjacent to e lie close to u.

We analyze our approach as follows: Since G_1 is monotone and each edge $(p_1,p_2) \in E_1$ is assigned to $|p_1p_2|_w$, we obtain that for each path it holds that $\widetilde{\pi} \subset G_1$ between $\mathfrak{s}$ and $\mathfrak{t}$ holds $|\pi|_w \leq |\widetilde{\pi}|_w$. The same argument applies to G_2. Hence, we still have to ensure that there is a path $\widetilde{\pi} \subset G_1$ or $\widetilde{\pi} \subset G_2$ such that $|\widetilde{\pi}|_w \leq (1+\varepsilon)|\pi|_w$. We say that a path $\pi \subset P$ is *low* if $w(p) \leq \frac{\mu}{100}$ holds for all $p \in \pi$. For our analysis, we show the following:

- Case A: There is a $\widetilde{\pi} \subset G_1$ with $|\widetilde{\pi}|_w \leq (1+\varepsilon)|\pi|_w$ if there is a shortest path $\pi \subset P$ that is not low (see Section 3.3).
- Case B: Otherwise, there is a $\widetilde{\pi} \subset G_2$ with $|\widetilde{\pi}|_w \leq (1+\varepsilon)|\pi|_w$ (see Section 3.4).

3.3 Analysis of Case A

In this section, we assume that there is a shortest path π between $\mathfrak{s}$ and $\mathfrak{t}$ that is not low. Furthermore, for any $o,p \in \pi$, we denote the subpath of π which is between o and p by π_{op}.

First, we prove a lower bound for $|\pi|_w$ (Lemma 5). This lower bound ensures that the approximation error that we make for a path in G_1 is upper-bounded by $\varepsilon|\pi|_w$ (Lemma 6).

A *cell* C *of* G_1 is the convex hull of four vertices $v_1, v_2, v_3, v_4 \in V_1$ such that $C \cap V_1 = \{v_1, v_2, v_3, v_4\}$. As the mesh size of G_1 is $\frac{\varepsilon\mu^2}{40000(|T_1|+|T_2|)}$, we have $d_1(p_1, p_2) \leq \frac{\varepsilon\mu^2}{20000(|T_1|+|T_2|)}$ for any two points p_1 and p_2 that lie in the same cell of G_1. The following property of $w(\cdot)$ is the key in the analysis of the weighted shortest path length of G_1:

▶ **Definition 3** ([8]). $f : P \to \mathbb{R}_{\geq 0}$ is 1-Lipschitz if $f(x) \leq f(y) + d_1(x, y)$ for all $x, y \in P$.

The requirement $|f(x) - f(y)| \leq d_1(x, y)$ is also occasionally used to define 1-Lipschitz continuity. Note that this alternative definition is equivalent to Definition 3.

▶ **Lemma 4.** *$w(\cdot)$ is 1-Lipschitz.*

Proof. Let $(a_1, a_2), (b_1, b_2) \in P$ be chosen arbitrarily. The subcurves $t_{T_1(a_1)T_1(b_1)} \subset T_1$ between $T_1(a_1)$ and $T_1(b_1)$ and $t_{T_2(a_2)T_2(b_2)} \subset T_2$ between $T_2(a_2)$ and $T_2(b_2)$ have lengths no larger than $|a_1 - b_2|$ and $|a_2 - b_2|$. Thus, $d_2(T_1(a_1), T_1(b_1)) \leq |a_1 - b_1|$ and $d_2(T_2(a_2), T_2(b_2)) \leq |a_2 - b_2|$. Furthermore, $w((a_1, a_2))$ is equal to $d_2(T_1(a_1), T_2(a_2))$. By triangle inequality, it follows that $w((b_1, b_2)) = d_2(T_1(b_1), T_2(b_2)) \leq d_2(T_2(b_2), T_2(a_2)) + d_2(T_2(a_2), T_1(a_1)) + d_2(T_1(a_1), T_1(b_1)) \leq d_1((a_1, a_2), (b_1, b_2)) + w((a_1, a_2))$, because $d_2(T_2(b_2), T_2(a_2)) = |b_2 - a_2|$, $d_2(T_2(a_2), T_1(a_1)) = w((a_1, a_2))$, $d_2(T_1(a_1), T_1(b_1)) = |b_1 - a_1|$, and $d_1((a_1, a_2), (b_1, b_2)) = |b_1 - a_1| + |b_2 - a_2|$. ◀

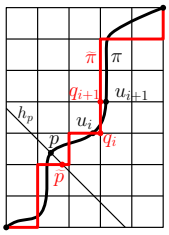
Lemma 4 allows us to prove the following lower bound for the weighted length of π.

▶ **Lemma 5.** $|\pi|_w \geq \frac{\mu^2}{20000}$.

Proof. Let $p \in \pi$ such that $w(p) \geq \frac{\mu}{100}$. Let $\psi := \pi \cap B_{\frac{\mu}{100}}(p)$. We have $|\psi|_w \geq \frac{\mu^2}{20000}$ because $w(\cdot)$ is 1-Lipschitz. Furthermore, $\psi \subset \pi$ implies $|\psi|_w \leq |\pi|_w$ which yields $\frac{\mu^2}{20000} \leq |\pi|_w$. ◀

▶ **Lemma 6.** *There is a path $\widetilde{\pi} \subset G_1$ that connects $\mathfrak{s}$ and $\mathfrak{t}$ such that $|\widetilde{\pi}|_w \leq (1 + \varepsilon)|\pi|_w$.*

Proof. Starting from $\mathfrak{s}$, we construct $\widetilde{\pi}$ inductively as follows: If π crosses a vertical (horizontal) parameter line next, $\widetilde{\pi}$ goes one step to the right (top). For $p \in \pi$ let h_p be the line with gradient -1 such that $p \in h_p$ (see the figure on the right). As π and $\widetilde{\pi}$ are monotone, the point $\widetilde{p} := h_p \cap \widetilde{p}$ is unique and well defined. For all p, p and $\widetilde{p}$ lie in the same cell of G_1 and thus, $w(\widetilde{p}) \leq w(p) + \frac{\varepsilon\mu^2}{20000(|T_1|+|T_2|)}$. This implies $|\widetilde{\pi}|_w \leq (1 + \varepsilon)|\pi|_w$ because $|\widetilde{\pi}| = |\pi|$. To be more precise, we consider $\widetilde{\pi}, \pi : [0, 1] \to P$ to be parametrized such that $d_1(\mathfrak{s}, \pi(t)) = d_1(\mathfrak{s}, \widetilde{\pi}(t)) = t d_1(\mathfrak{s}, \mathfrak{t})$. We obtain, $||(\widetilde{\pi})'(t)||_1 = d_1(\mathfrak{s}, \mathfrak{t}) = ||(\pi)'(t)||_1$ for all $t \in [0, 1]$.



Furthermore, the above implies $w(\widetilde{\pi}(t)) \leq w(\pi(t)) + \frac{\varepsilon\mu^2}{20000(|T_1|+|T_2|)}$ $(\star)$. Thus:

$$\begin{aligned}|\widetilde{\pi}|_w &= \int_0^1 w(\widetilde{\pi}(t))||(\widetilde{\pi})'(t)||_1 \, dt \overset{(\star)}{\leq} \int_0^1 \left(w(\pi(t)) + \frac{\varepsilon\mu^2}{20000(|T_1|+|T_2|)}\right) ||(\pi)'(t)||_1 \, dt \\ &= \int_0^1 w(\pi(t))||(\pi)'(t)||_1 \, dt + \frac{\varepsilon\mu^2 \int_0^1 1 \; ||(\pi)'(t)||_1 \, dt}{20000(|T_1|+|T_2|)} \\ &= |\pi|_w + \frac{\varepsilon\mu^2}{20000} \overset{\text{Lemma 5}}{\leq} |\pi|_w + \varepsilon|\pi|_w = (1+\varepsilon)|\pi|_w.\end{aligned}$$

◀

The proof of Lemma 6 is omitted due to space constraints. All proofs that are omitted or just sketched can be found in the Appendix.

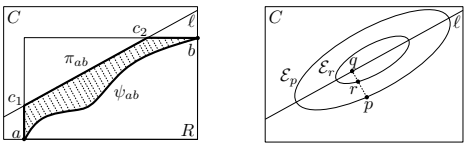
3.4 Analysis of Case B

In this section, we assume that there is a shortest monotone low path π between $\mathfrak{s}$ and $\mathfrak{t}$. First, we make a key observation that is also of independent interest.

▶ **Lemma 7.** *Let C be an arbitrarily chosen parameter cell and $a, b \in C$ such that $a \leq_{xy} b$. Furthermore, let ℓ be the monotone free space axis of C and R the rectangle that is induced by a and b. The shortest path $\pi_{ab} \subset C$ between a and b is given as:*

- *$ac_1 \cup c_1c_2 \cup c_2b$, if ℓ intersects R in c_1 and c_2 such that $c_1 <_{xy} c_2$ and as*
- *$ac \cup cb$, otherwise, where c is defined as the closest point from R to ℓ.*

Proof. Let $\psi_{ab} \subset C$ by an arbitrary monotone path that connects a and b. In the following, we show that $|\pi_{ab}|_w \leq |\psi_{ab}|_w$. For this, we prove the following: Let $p \in C$ be chosen arbitrarily and q be its orthogonal projection onto ℓ (see the figures right). We show $w(r) \leq w(p)$ for $r \in pq$. This implies that there is an injective, continuous function $\perp : \psi_{ab} \to \pi_{ab}$ with $w(\perp(p)) \leq w(p)$ for all $p \in \psi$. In particular, $\perp(p)$ is defined as the intersection point of π_{ab} and the line d that lies perpendicular to ℓ such that $p \in d$. The function $\perp(\cdot)$ is well defined and injective as both ψ_{ab} and π_{ab} are monotone paths that connect a and b. Similarly, as in the proof of Lemma 6, this implies $|\pi_{ab}|_w \leq |\psi_{ab}|_w$ because $|\pi_{ab}| = |\psi_{ab}|$.

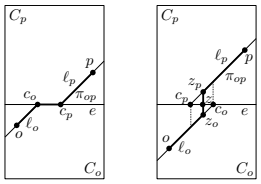


To be more precise, consider $\psi, \pi : [0,1] \to C$ to be parametrized such that $d_1(a, \psi(t)) = d_1(a, \pi(t)) = td_1(a, b)$. This implies $||(\psi)'(t)||_1 = d_1(a, b) = ||(\pi)'(t)||_1$ for all $t \in [0, 1]$. Thus:

$$|\psi_{ab}|_w \quad = \quad \int_0^1 w(\psi_{ab}(t))||(\psi_{ab})'(t)||_1 \, dt \geq \int_0^1 w(\pi_{ab}(t))||(\pi_{ab})'(t)||_1 \, dt = |\pi_{ab}|_w.$$

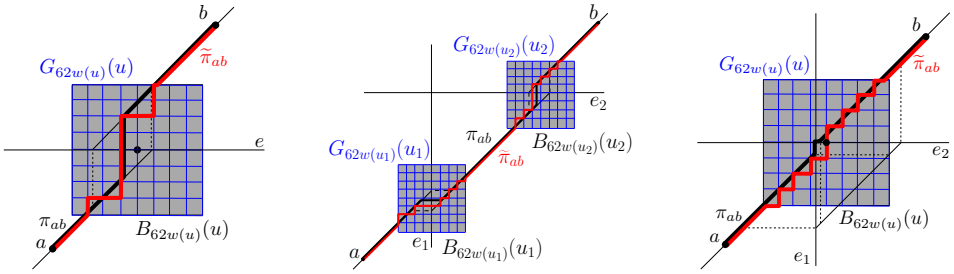
Finally, we show: $w(r) \leq w(p)$, for $r \in pq$. Note that $w(r)$ and $w(p)$ are the leash lengths for r and p that lie on the boundary of the white space inside C, i.e., on the boundary of the ellipses $\mathcal{E}_r$ and $\mathcal{E}_p$, respectively. Since $r \in pq$ we get $\mathcal{E}_r \subseteq \mathcal{E}_p$, which implies $w(r) \leq w(p)$. ◀

We call a point $p \in C$ *canonical* if $p \in \ell$. Let C_o and C_p be two parameter cells that share a parameter edge e. Furthermore, let $o \in \ell_o \subset C_o$ and $p \in \ell_p \subset C_p$ be two canonical parameter points such that $o \leq_{xy} p$ where ℓ_o and ℓ_p are the monotone free space axis of C_o and C_p, respectively. Let c_o be the top-right end point of ℓ_o and c_p the bottom-left end point of ℓ_p. The following corollary to Lemma 7 characterizes how a shortest path passes through the parameter edges.



(a) $o \leq_{xy} p$ (b) $o \not\leq_{xy} p$

Figure 4 Configurations of Corollary 8.



(a) Case (1.) (b) Case (2.1.) (c) Case (2.2.)

Figure 5 Different subcases how π_{ab} is approximated by free space axes and grid balls.

▶ **Corollary 8.** *If $c_o, c_p \in e$ and $c_o \leq_{xy} c_p$, π_{op} is equal to the concatenation of the segments oc_o, c_oc_p, and c_pp (see Figure 4(a) on right). Otherwise, there is a $z \in e$ such that π_{op} is equal to the concatenation of the segments oz_o, z_oz_p, and z_pp, where $z_o \in \ell_{C_o}$ and $z_p \in C_p$ such that z is the orthogonal projection of z_o and z_p onto e (see Figure 4(b)).*

3.4.1 Outline of the analysis of Case B

In the following, we apply Lemma 7 and Corollary 8 to subpaths π_{ab} of π in order to ensure that π_{ab} is a subset of the union of a constant number of balls (that are approximated by grid balls in our approach) and monotone free space axes. In particular, we construct a discrete sequence of points from π which lie on the free space axes, see Section 3.4.2. For each induced subpath π_{ab}, we ensure that π_{ab} crosses one or two perpendicular parameter edges. For the analysis we distinguish between the two cases which we consider separately:

- Case 1: π_{ab} crosses one parameter edge and
- Case 2: π_{ab} crosses two parameter edges.

For Case 1, we show that, if π_{ab} crosses one edge (e) then π_{ab} is a subset of the union of the two monotone free space axes of the parameter cells that share e and the ball $B_{62w(u)}(u)$ for $u := \arg\min_{p \in e} w(u)$ (see Figure 5(a) and Section 3.4.3).

For Case 2, (see Section 3.4.4), we consider the case that π_{ab} crosses two parameter edges e_1 and e_2. In particular, π_{ab} runs through three parameter cells C_q, C_r, and C_s, where C_q and C_r share e_1 and C_r and C_s share e_2.

We further distinguish further between two subcases. For this, let $u_1 := \arg\min_{p \in e_1} w(p)$ and $u_2 := \arg\min_{p \in e_2} w(p)$.

- Case 2.1: We show that, if $d_1(u_1, u_2) \geq 6 \max\{w(u_1), w(u_2)\}$, then π_{ab} is a subset of the union of the balls $B_{62w(u_1)}(u_1)$ and $B_{62w(u_2)}(u_2)$ and the monotone free space axes of C_q, C_r, and C_s (see Figure 5(b) and Lemma 13).
- Case 2.2: We show that, if $d_1(u_1, u_2) \leq 6 \max\{w(u_1), w(u_2)\}$, then π_{ab} is a subset of the union of the ball $B_{62w(u)}(u)$ and the monotone free space axes of C_q and C_s for $u \in \{u_1, u_2\}$ (see Figure 5(c) and Lemma 17).

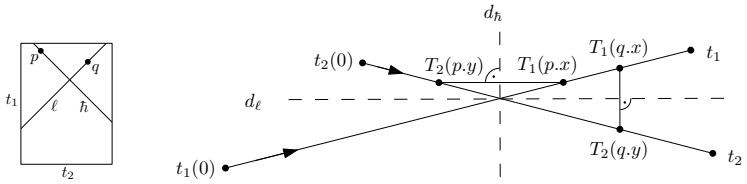


Figure 6 Duality of parameter points from ℓ ($\hbar$) and leashes that lie perpendicular to d_ℓ ($d_\hbar$).

For the analysis of the length of a shortest path $\widetilde{\pi} \subset G_2$ that lies between $\mathfrak{s}$ and $\mathfrak{t}$, we construct for $\pi_{ab} \subset \pi$ a path $\widetilde{\pi}_{ab} \subset G_2$ between a and b such that $|\widetilde{\pi}_{ab}|_w \leq (1+\varepsilon)|\pi_{ab}|_w$. In particular, $\widetilde{\pi}_{ab}$ is a subset of the grid balls that approximate the above considered balls and the free space axes that are involved in the individual (sub-)case for π_{ab} (see, Figure 5). Finally, we define $\widetilde{\pi} \subset G_2$ as the concatenation of the approximations $\widetilde{\pi}_{ab}$ for all π_{ab}.

3.4.2 Separation of a shortest path

In the following, we determine a discrete sequence of canonical points $\mathfrak{s} = p_1, ..., p_k = \mathfrak{t} \in \pi$ such that $\pi_{p_i p_{i+1}}$ crosses at most two parameter lines for each $i \in \{1, ..., k-1\}$. First, we need the following supporting lemma:

▶ **Lemma 9.** *For all $q_1, q_2 \in \pi$ that lie in the same parameter cell with $q_1 \leq_{xy} q_2$ we have $q_2.y - q_1.y - \frac{\mu}{50} \leq q_2.x - q_1.x \leq q_2.y - q_1.y + \frac{\mu}{50}$.*

Proof. By triangle inequality we obtain:
$d_2(T_2(q_2.y), T_2(q_1.y)) \leq d_2(T_2(q_2.y), T_1(q_2.x)) + d_2(T_1(q_2.x), T_1(q_1.x)) + d_2(T_1(q_1.x), T_2(q_1.y))$.
This implies $d_2(T_2(q_2.y), T_2(q_1.y)) - \frac{\mu}{50} \leq d_2(T_1(q_2.x), T_1(q_1.x))$, because $d_2(T_2(q_2.y), T_1(q_2.x)), d_2(T_1(q_1.x), T_2(q_1.x)) \leq \frac{\mu}{100}$. Furthermore, $d_2(T_2(q_2.y), T_2(q_1.y)) = q_2.y - q_1.y$ and $d_2(T_1(q_2.x), T_1(q_1.x)) = q_2.x - q_1.x$ because q_1 and q_2 lie in the same cell. This implies $q_2.y - q_1.y - \frac{\mu}{50} \leq q_2.x - q_1.x$. A corresponding argument yields that $q_2.x - q_1.x \leq q_2.y - q_1.y + \frac{\mu}{50}$. ◀

▶ **Lemma 10.** *There are canonical points $\mathfrak{s} = p_1, \ldots p_k = \mathfrak{t} \in \pi$ such that for all $i \in \{1, \ldots, k-1\}$ the following holds: (P1) $\pi_{p_i p_{i+1}}$ crosses at most one vertical and at most one horizontal parameter line which are both not part of ∂P and (P2) the distance of p_i to a parameter line is lower-bounded by $\frac{\mu}{6}$ for all $i \in \{2, \ldots, k-1\}$.*

3.4.3 Analysis of subpaths that cross one parameter edge

We need to show that those parts of π that do not lie on the free space axes are covered by the balls $B_{62w(u)}$. For this, we use the following geometrical interpretation of the free space axes ℓ and $\hbar$ of a parameter cell C. Let $t_1 \in T_1$ and $t_2 \in T_2$ be the segments that correspond to C. We denote the angular bisectors of t_1 and t_2 by d_ℓ and $d_\hbar$ such that the start points $t_1(0)$ and $t_2(0)$ of t_1 and t_2 lie on different sides w.r.t. $d\ell$, see Figure 6 right. If t_1 and t_2 are parallel, then d_ℓ denotes the line between t_1 and t_2 and we declare $d_\hbar$ as undefined. We observe:

▶ **Observation 11.** *$q \in \ell \Leftrightarrow T_1(q.x)T_2(q.y) \perp d_\ell$ and $p \in \hbar \Leftrightarrow T_1(p.x)T_2(p.y) \perp d_\hbar$.*

From now on, let $o, p \in \pi$ be two consecutive, canonical points that are given via Lemma 10 such that $o \leq_{xy} p$. Furthermore, let ℓ_o and ℓ_p be the free space axes of the parameter cells C_o and C_p such that $o \in \ell_o \subset C_o$ and $p \in \ell_p \subset C_p$.

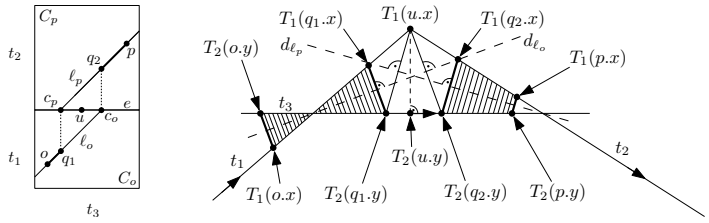


Figure 7 Configuration of the Lemmas 12 and 13: The length of the subpath of π_{op} that does not necessarily lie on $\ell \cup \hbar$ is related to $w(u)$.

▶ **Lemma 12.** *If π_{op} crosses one parameter edge e, points $c_o, c_p \in e$ exist and we have $d_\infty(c_o, c_p) \leq \frac{w(u)}{2}$ where $u = \arg\min_{p \in e} w(p)$.*

Proof. W.l.o.g., we assume that e is horizontal. Let $t_1, t_2 \in T_1$ and $t_3 \in T_2$ be the segments that induce parameter cells C_o and C_p. Below, we show $\angle(t_1, t_3), \angle(t_2, t_3) \leq 7°$ and, then, that $d_1(c_0, c_p) \leq w(u)$. Let $q_1 \in \ell_o$ and $q_2 \in \ell_p$ such that $q_1.x = c_p$ and $q_2.x = c_o$, see Figure 7 left. $\angle(t_1, t_3) \leq 7°$ implies $\angle(T_1(u.x)T_2(u.y), T_1(u.x)T_2(q_2.y)) \leq 3.5°$. Furthermore, $c_p = e \cap \ell_p$ implies: c_p corresponds to a leash $l_p = (T_1(c_p.x), T_2(c_p.y))$ such that $T_1(c_p.x) = T_1(u.x)$ and $T_1(c_p.x), T_2(c_p.y) \perp d_{\ell_o}$, see Figure 7 right. Thus, $d_2(T_2(q_2.y), T_2(u.y))$ is upper-bounded by $d_2(T_2(u.y), T_2(q_2.y)) \leq d_2(T_1(u.x), T_2(u.y)) \tan(3.5°) \leq 0.065 w(u) < \frac{w(u)}{2}$.

Finally, we show that $\angle(t_1, t_3), \angle(t_2, t_3) \leq 7°$. We know that $d_2(T_1(o.x), T_2(o.y))$ and $d_2(T_1(u.x), T_2(u.x))$ are upper-bounded by $\frac{\mu}{100}$ because π is low. Lemma 10 implies $d_2(T_1(o.x), T_1(u.x)), d_2(T_2(o.y), T_2(u.y)) \geq \frac{\mu}{6}$. Thus, $\angle(t_1, t_3) \leq \arcsin \frac{6}{50} \leq 7°$. A similar argument implies that $\angle(t_2, t_3) \leq \arcsin \frac{6}{50} \leq 7°$ ◀

▶ **Lemma 13.** *$\pi_{op} \subset \ell_o \cup B_{w(u)}(u) \cup \ell_p$ (see Figure 5(a)).*

Proof. We combine Corollary 8 and Lemma 12. Corollary 8 implies that π_{op} orthogonally crosses e at a point z that lies between c_o and c_p such that $z \in z_o z_p \subset \pi_{op}$. Lemma 12 implies $d_1(c_o, c_p) \leq \frac{w(u)}{2}$. Thus, $z_o z_p \subset B_{w(u)}(u)$. Furthermore, $o z_o \subset \ell_o$ and $z_p p \subset \ell_p$. This implies $\pi_{op} \subset \ell_o \cup B_{w(u)}(u) \cup \ell_p$ because $\pi_{op} = o z_o \cup z_o z_p \cup z_p p$. ◀

▶ **Lemma 14.** *There is a path $\widetilde{\pi}_{op} \subset G_2$ between o and p such that $|\widetilde{\pi}_{op}|_w \leq (1+\varepsilon)|\pi_{op}|_w$.*

Proof (Sketch). By Lemma 13, the following two intersection points are well defined: Let z_o be the intersection point of ℓ_o and $\partial B_{62w(u)}(u)$ that lies on the left or bottom edge of $\partial B_{62w(u)}(u)$. Analogously, let z_p be the intersection point of ℓ_p and $\partial B_{62w(u)}(u)$ that lies on the right or top edge of $\partial B_{62w(u)}(u)$. By Lemma 13, we can subdivide π_{op} into the three pieces $o z_o \subset \ell_o$, $\pi_{z_o z_p}$, and $z_p p \subset \ell$. As $o z_o, z_p p \subset G_2$, we just have to construct a path $\widetilde{\pi}_{z_o z_p} \subset G_2$ between z_o and z_p such that $|\pi_{z_o z_p}|_w \leq (1+\varepsilon)|\widetilde{\pi}_{z_o z_p}|_w$.

We construct $\widetilde{\pi}_{z_o z_p}$ by applying the same approach as used in the proof of Lemma 6 (see Figure 5(a)). To upper-bound $|\widetilde{\pi}_{z_o z_p}|_w$ by $(1+\varepsilon)|\pi_{z_o z_p}|_w$, we first lower-bound $|\pi_{z_o z_p}|_w$ by $\frac{1}{2}w^2(u)$. By using a similar approach as in the proof of Lemma 6, we can conclude the proof. Further details are provided in the Appendix. Let $\psi := \pi_{z_o z_p} \cap B_{w(u)}(u)$. As $|\psi| \geq w(u)$ and $w(\cdot)$ is 1-Lipschitz, we obtain $|\psi|_w \geq \frac{1}{2}w^2(u)$. Thus, $|\pi_{z_o z_p}|_w \geq \frac{1}{2}w^2(u)$ as $\psi \subset \pi_{z_o z_p}$. ◀

3.4.4 Analysis of subpaths that cross two parameter edges

Let q and s be two consecutive parameter points from $\{p_2, \ldots, p_{k-1}\}$ such that π_{qs} crosses two parameter edges e_1 and e_2. By Lemma 10, e_1 and e_2 are perpendicular to each other and are adjacent at a point c. Let C_r be the parameter cell such that e_1 and e_2 are part of

the boundary of C_r. Furthermore, let C_q and C_s be the parameter cells such that $q \in C_q$ and $s \in C_s$. We denote the monotone free space axis of C_q, C_r, and C_s by ℓ_q, ℓ_r, and ℓ_s, respectively. Let $u_1 := \arg\min_{a \in e_1} w(a)$ and $u_2 := \arg\min_{a \in e_2} w(a)$.

▶ **Lemma 15.** *If $d_1(u_1, u_2) \geq 6\max\{w(u_1), w(u_2)\}$, there is another canonical parameter point $r \in \ell_r$ such that $\pi_{qs} \subset \ell_q \cup B_{w(u_1)}(u_1) \cup \ell_r \cup B_{w(u_2)}(u_2) \cup \ell_s$.*

The proof of Lemma 15 is similar to the proof of Lemma 12.

▶ **Lemma 16.** *If $d_1(u_1, u_2) \geq 6\max\{w(u_1), w(u_2)\}$, then there is a path $\widetilde{\pi}_{qs} \subset G_2$ between q and s such that $|\widetilde{\pi}_{qs}|_w \leq (1+\varepsilon)|\pi_{qs}|_w$.*

Proof. Lemma 15 implies that the following constructions are unique and well defined: Let z_1 (z_2) be the intersection point of $\partial B_{w(u_1)}(u_1)$ and ℓ_q (ℓ_r) that lies on the left or bottom (respectively, right or top) edge of $\partial B_{w(u_1)}(u_1)$. Analogously, let z_3 (z_4) be the intersection point of $\partial B_{w(u_2)}(u_2)$ and ℓ_r (ℓ_s) that lies on the left or bottom (respectively, right or top) edge of $\partial B_{w(u_2)}(u_2)$. By applying the approach of Lemma 14, for $\pi_{z_1 z_2}$ and $\pi_{z_3 z_4}$, we obtain a path $\widetilde{\pi}_{z_1 z_2} \subset G_2$ between z_1 and z_2 and a path $\widetilde{\pi}_{z_3 z_4} \subset G_2$ between z_3 and z_4 such that $|\widetilde{\pi}_{z_1 z_2}|_w \leq (1+\varepsilon)|\pi_{z_1 z_2}|_w$ and $|\widetilde{\pi}_{z_3 z_4}|_w \leq (1+\varepsilon)|\pi_{z_3 z_4}|_w$. This concludes the proof because $qz_1, z_2 z_3, z_4 s \subset G_2$. ◀

▶ **Lemma 17.** *If $d_1(u_1, u_2) \leq 6\max\{w(u_1), w(u_2)\}$, then $\pi_{qs} \subset \ell_q \cup B_{62w(u)\}}(u) \cup \ell_s$ where $u := \arg\max_{u \in \{u_1, u_2\}}\{w(u_1), w(u_2)\}$.*

Lemma 17 implies that the approach taken in the proof of Lemma 14 yields that there is a path $\widetilde{\pi}_{qs} \subset G_2$ between q and s such that $|\widetilde{\pi}_{qs}|_w \leq (1+\varepsilon)|\pi_{qs}|_w$ If $d_1(u_1, u_2) < 6\max\{w(u_1), w(u_2)\}$. Combining this with Lemmas 14 and 16 yields the following corollary:

▶ **Corollary 18.** *Let $\widetilde{\pi} \subset G_2$ be a shortest path. We have $|\pi|_w \leq |\widetilde{\pi}|_w \leq (1+\varepsilon)|\pi|_w$.*

3.5 "Bringing it all together"

In Sections 3.3 and 3.4, we proved that in Cases A and B, the minimum of the shortest path lengths in G_1 and G_2 is no larger than $(1+\varepsilon)|\pi|_w$, where π_w is a shortest path in P.

Next, we discuss that our algorithm has a running time of $\mathcal{O}(\frac{\zeta^4 n^4}{\varepsilon})$. Graph G_1 is given by the arrangement that is induced by $\Theta(\frac{\zeta^2 n^2}{\varepsilon})$ horizontal and $\Theta(\frac{\zeta^2 n^2}{\varepsilon})$ vertical lines because the corresponding grid has a mesh of size $\frac{\varepsilon \mu^2}{40000(|T_1|+|T_2|)}$. Thus, $|E_1| \in \Theta(\frac{\zeta^4 n^4}{\varepsilon^2})$. Graph G_2 is given by the arrangement that is induced by $\mathcal{O}(n^2)$ free space axis and $\Theta(n^2)$ grid balls. Each grid ball has a complexity of $\Theta(\frac{1}{\varepsilon})$. Thus, $|E_2| \in \mathcal{O}(\frac{n^4}{\varepsilon^2})$. Applying Dijkstra's shortest path algorithm on G_1 and G_2 takes time proportional to $\mathcal{O}(|E_1|)$ and $\mathcal{O}(|E_2|)$. As $|E_1| \in \Theta(\frac{\zeta^4 n^4}{\varepsilon^2})$ and $|E_2| \in \mathcal{O}(\frac{n^4}{\varepsilon^2})$ we have to ensure that each edge of $E_1 \cup E_2$ can be computed in constant time to guarantee an overall running time of $\mathcal{O}(\frac{\zeta^4 n^4}{\varepsilon^2})$.

▶ **Lemma 19.** *All edges of G_1 and G_2 can be computed in $\mathcal{O}(1)$ time.*

This leads to our main result.

▶ **Theorem 20.** *We can compute an $(1+\varepsilon)$-approximation of $\mathcal{F}_{\mathcal{S}}(T_1, T_2)$ in $\mathcal{O}(\frac{\zeta^4 n^4}{\varepsilon^2})$ time.*

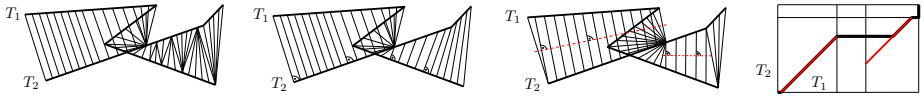


Figure 8 First: The definition of local correctness still allowes "unnatural" matchings. Second: A lexicographic matching. Third: A locally optimal matching that is also optimal w.r.t. integral Fréchet distance. Fourth: The shortest path $\pi \subset P$ (black) that corresponds to the third matching.

4 Locally optimal Fréchet matchings

In this section, we discuss an application of our observations regarding free space axes to so-called *locally correct (Fréchet) matchings* (α_1, α_2) as introduced by Buchin et al. [2]. For $i \in \{1, 2\}$ and $0 \leq a \leq b \leq n$, we denote the subcurve between $T_i(a)$ and $T_i(b)$ by $T_i[a, b]$.

▶ **Definition 21** ([2]). (α_1, α_2) is *locally correct* if $\mathcal{D}\left(T_1[\alpha_1(a), \alpha_2(b)], T_2[\alpha_1(a), \alpha_2(b)]\right) = \max_{t \in [\alpha_1(a), \alpha_2(b)]} d_2(T_1(t), T_2(t))$, for all $0 \leq a \leq b \leq n$.

Buchin et al. [2] suggested to extend the definition of locally correct matchings to "locally optimal" matchings as a future work. "The idea is to restrict to the locally correct matching that decreases the matched distance as quickly as possible."[2, p. 237].

Rote [13] proposed such an extension in terms of the *profile* of a matching. Roughly speaking, the profile of a matching measures, for each threshold $\delta \geq 0$, the amount of time that $d_2(T_1(\alpha_1), T_2(\alpha_2))$ is at least δ. Based on matchings' profiles, Rote defined an order of matchings by applying the *lexicographic order* of their profiles. Without any further restrictions, the lexicographic order of matchings does not make sense because "otherwise we could simply traverse the two curves at a larger speed and accordingly scale down the profile of the considered matching."[13]. Thus, Rote assumes additionally that the "speed at which the curves are traversed by parametrizations is bounded by 1"[13].

Rote [13] gives an algorithm to compute a lexicographic matching in $\mathcal{O}\left(n^3 \log n\right)$ time.

In contrast to [13], we do not measure time w.r.t. the integral of parameter values but w.r.t. the length of traversed subcurves. This has the advantage that we do not need an additional restriction to the considered matchings because the lengths of traversed subcurves is invariant w.r.t. the speed in that they are traversed. In the following, we give a corresponding definition of simply computable locally optimal matchings.

Let (α_1, α_2) be a locally correct matching. As the function $f : t \mapsto d_2(T_1(\alpha_1(t)), T_2(\alpha_2(t)))$ is in general not monotone, we ask for a matching that locally increases and decreases the leash length between two maxima "as fast as possible". In particular, we measure speed in terms of the lengths of subcurves being traversed to achieve a required leash length.

More formally, $t \in [0, n]$ is the *parameter of a local maxima* of f if there is a $\delta_t > 0$ such that for all $0 \leq \delta \leq \delta_t : f(t \pm \delta) \leq f(t)$ and $f(t + \delta) < f(t)$ or $f(t - \delta) < f(t)$. For any $t_1, t_2 \in [0, n]$ and $i \in \{1, 2\}$, we denote the restriction of α_i to $[t_1, t_2]$ as $\alpha_i[t_1, t_2]$.

▶ **Definition 22.** (α_1, α_2) is locally optimal if $\mathcal{P}_\delta(T_1[\alpha_1(t_1), \alpha_1(t_2)], T_2[\alpha_2(t_1), \alpha_2(t_2)]) = \mathcal{P}_{(\alpha_1[t_1,t_2], \alpha_2[t_1,t_2])}(T_1, T_2)$ for all $\delta \geq 0$ and for all parameters of local maxima $t_1, t_2 \in [0, n]$ such that $[t_1, t_2]$ does not contain any further parameter of a local maximum.

By applying a similar approach as in the proof of Lemma 7 we obtain the following:

▶ **Lemma 23.** *Let C be an arbitrarily chosen parameter cell and $a, b \in C$ such that $a \leq_{xy} b$ and π_{ab} the path induced by Lemma 7. Then, $\mathcal{P}_\delta(T_1[a.x, b.x], T_2[a.y, b.y]) = |\mathcal{E}_\delta \cap \pi_{ab}|$ for all $\delta \geq 0$, where $\mathcal{E}_\delta$ is the free space ellipse of C for the distance threshold δ.*

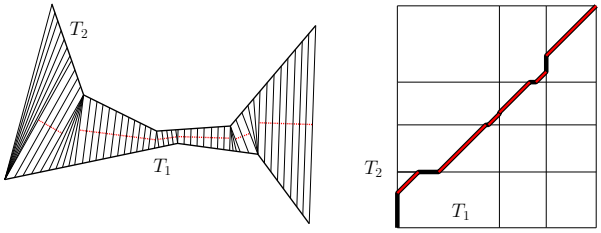


Figure 9 Left: A locally optimal matching that is also global optimal w.r.t. integral Fréchet distance and partial Fréchet similarity for any $\delta \geq 0$. Right: The shortest path that corresponds to the matching to the left. Following completely the free space axes is allowed because the end points of the free space axes can be ordered w.r.t. xy-monotonicity.

Lemma 23 implies the following:

▶ **Corollary 24.** *A locally correct matching can be transformed into a locally optimal Fréchet matching in $\mathcal{O}(n)$. Generally, a locally optimal matching can be computed in $\mathcal{O}(n^3 \log n)$.*

Proof. Let $\pi \subset P$ be the path that corresponds to the locally correct matching. Furthermore, let $p_1, \ldots, p_{2n} \in \pi$ be the intersection points of π with the parameter grid. For each $i \in \{1, ..., 2n-1\}$ we substitute the subpath $\pi_{p_i p_{i+1}}$ by the path between p_i and p_{i+1} which is induced by Lemma 7. The algorithm from [2] computes a locally correct matching in $\mathcal{O}(n^3 \log n)$ time. Thus, a locally optimal matching can be computed in $\mathcal{O}(n^3 \log n)$ time. ◀

5 Conclusion

We presented a pseudo-polynomial $(1+\varepsilon)$-approximation algorithm for the integral and average Fréchet distance which has a running time of $\mathcal{O}(\frac{\zeta^4 n^4}{\varepsilon^2})$. In particular, in our approach we compute two geometric graphs and their weighted shortest path lengths in parallel. It remains open if one can reduce the complexity of G_1 to polynomial with respect to the input parameters such that using $G_1 \cup G_2$ still ensures an $(1+\varepsilon)$-approximation.

As a byproduct we developed techniques to determine the local nature of an optimal matching (α_1, α_2) (without any further restrictions to (α_1, α_2)) w.r.t. different Fréchet measures. It remains open how these techniques can be extended such that not only local, but global optimal matchings can be computed. See Figure 9 for an example. We are currently investigating this extension.

References

1 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl.*, 5:75–91, 1995. doi:10.1142/S0218195995000064.

2 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Bettina Speckmann. Locally correct Fréchet matchings. In Leah Epstein and Paolo Ferragina, editors, *ESA*, volume 7501 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2012. doi:10.1007/978-3-642-33090-2_21.

3 Kevin Buchin, Maike Buchin, and Yusu Wang. Exact algorithms for partial curve matching via the Fréchet distance. In Claire Mathieu, editor, *SODA*, pages 645–654. SIAM, 2009. doi:10.1137/1.9781611973068.

4 Maike Buchin. On the computability of the Fréchet distance between triangulated surfaces. Ph.D. thesis, Dept. of Comput. Sci., Freie Universität, Berlin, 2007.

5 Jean-Lou De Carufel, Amin Gheibi, Anil Maheshwari, Jörg-Rüdiger Sack, and Christian Scheffer. Similarity of polygonal curves in the presence of outliers. *Comput. Geom.*, 47(5):625–641, 2014.

6 Isabel F. Cruz, Craig A. Knoblock, Peer Kröger, Egemen Tanin, and Peter Widmayer, editors. *SIGSPATIAL 2012 International Conference on Advances in Geographic Information Systems (formerly known as GIS), SIGSPATIAL'12, Redondo Beach, CA, USA, November 7-9, 2012*. ACM, 2012.

7 Alon Efrat, Quanfu Fan, and Suresh Venkatasubramanian. Curve matching, time warping, and light fields: New algorithms for computing similarity between curves. *Journal of Mathematical Imaging and Vision*, 27(3):203–216, 2007. doi:10.1007/s10851-006-0647-0.

8 Stefan Funke and Edgar A. Ramos. Smooth-surface reconstruction in near-linear time. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.*, pages 781–790, 2002.

9 Joachim Gudmundsson, Patrick Laube, and Thomas Wolle. Movement patterns in spatio-temporal data. In Shashi Shekhar and Hui Xiong, editors, *Encyclopedia of GIS*, pages 726–732. Springer, 2008. doi:10.1007/978-0-387-35973-1_823.

10 Joachim Gudmundsson and Nacho Valladares. A GPU approach to subtrajectory clustering using the Fréchet distance. In Cruz et al. [6], pages 259–268. doi:10.1145/2424321.2424355.

11 Joachim Gudmundsson and Thomas Wolle. Football analysis using spatio-temporal tools. In Cruz et al. [6], pages 566–569. doi:10.1145/2424321.2424417.

12 Lawrence R. Rabiner and Biing-Hwang Juang. *Fundamentals of speech recognition*. Prentice Hall Signal Processing series. Prentice Hall, 1993.

13 Günter Rote. Lexicographic fréechet matchings. In *Proceedings of the 32st European Workshop on Computational Geometry*, pages 101–104, 2014.

Minimizing the Continuous Diameter when Augmenting Paths and Cycles with Shortcuts*

Jean-Lou De Carufel[1], Carsten Grimm[2], Anil Maheshwari[3], and Michiel Smid[4]

1 **School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada**
2 **School of Computer Science, Carleton University, Ottawa, Canada and Institut für Simulation und Graphik, Otto-von-Guericke-Universität Magdeburg, Magdeburg, Germany**
carsten.grimm@ovgu.de
3 **School of Computer Science, Carleton University, Ottawa, Canada**
4 **School of Computer Science, Carleton University, Ottawa, Canada**

Abstract

We seek to augment a geometric network in the Euclidean plane with shortcuts to minimize its continuous diameter, i.e., the largest network distance between any two points on the augmented network. Unlike in the discrete setting where a shortcut connects two vertices and the diameter is measured between vertices, we take all points along the edges of the network into account when placing a shortcut and when measuring distances in the augmented network.

We study this network augmentation problem for paths and cycles. For paths, we determine an optimal shortcut in linear time. For cycles, we show that a single shortcut never decreases the continuous diameter and that two shortcuts always suffice to reduce the continuous diameter. Furthermore, we characterize optimal pairs of shortcuts for convex and non-convex cycles. Finally, we develop a linear time algorithm that produces an optimal pair of shortcuts for convex cycles. Apart from the algorithms, our results extend to rectifiable curves.

Our work reveals some of the underlying challenges that must be overcome when addressing the discrete version of this network augmentation problem, where we minimize the discrete diameter of a network with shortcuts that connect only vertices.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory, I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Network Augmentation, Shortcuts, Diameter, Paths, Cycles

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.27

1 Introduction

The minimum-diameter network augmentation problem is concerned with minimizing the largest distance between two vertices of an edge-weighted graph by introducing new edges as shortcuts. We study this problem from a new perspective in a continuous and geometric setting where the network is a geometric graph embedded into the Euclidean plane, the weight of a shortcut is the Euclidean distance of its endpoints, and shortcuts can be introduced between any two points along the network that may be vertices or points along edges.

* This work was partially supported by NSERC.

As a sample application, consider a network of highways where we measure the distance between two locations in terms of the travel time. An urban engineer might want to improve the worst-case travel time along a highway or along a ring road by introducing shortcuts. Our work advises where these shortcuts should be built. For example, we show where to find the best shortcut for a highway and we show that a ring road requires two shortcuts.

1.1 Preliminaries

A *network* is an undirected graph that is embedded into the Euclidean plane and whose edges are weighted with their Euclidean length. For our algorithms we focus on networks with straight line edges, whereas the remaining results require rectifiable curves as edges. We say a point p lies on a network G and write $p \in G$ when there is an edge e of G such that p is a point along the embedding of e. A point p on an edge e of length l subdivides e into two sub-edges lengths $(1 - \lambda) \cdot l$ and $\lambda \cdot l$ for some value $\lambda \in [0, 1]$. We represent the points on G in terms of their relative position (expressed by λ) along their containing edge.

The *network distance* between two points p and q on a network G is the length of a weighted shortest path from p to q in G. We denote the network distance between p and q by $d_G(p, q)$ and we omit the subscript when the network is understood. The largest network distance between any two points on G is the *continuous diameter* of G, denoted by $\mathrm{diam}(G)$, i.e., $\mathrm{diam}(G) = \max_{p,q \in G} d_G(p, q)$. The term *continuous* distinguishes this notion from the *discrete diameter* that measures the largest network distance between any two vertices.

We denote the Euclidean distance between p and q by $|pq|$. A line segment pq, with $p, q \in G$ is a *shortcut* for G when $|pq| < d_G(p, q)$. We augment a network G with a shortcut pq as follows. We introduce new vertices at p and at q in G, subdividing their containing edges, and we add an edge from p to q of length $|pq|$. We do not introduce any vertices at any crossings between pq and other edges of G. The resulting network is denoted by $G + pq$. We seek to minimize the continuous diameter of a network by introducing shortcuts.

When G is a path or cycle, $|G|$ denotes its length. A cycle C is convex, when C forms a convex polygon with non-empty interior, i.e., a convex cycle cannot be confined to a line.

1.2 Related Work

In the discrete abstract setting, we consider an abstract graph G with unit weights and ask whether we can decrease the discrete diameter of G to at most D by adding at most k edges. For any fixed $D \geq 2$, this problem is NP-hard [2, 7, 9], has parametric complexity W[2]-hard [4, 5], and remains NP-hard even if G is a tree [2]. For an overview of the approximation algorithms in terms of both D and k refer, for instance, to Frati *et al.* [4].

In the discrete geometric setting, we consider a geometric graph, where a shortcut connects two vertices. Große *et al.* [6] are the first to consider diameter minimization in this setting. They determine a shortcut that minimizes the discrete diameter of a path with n vertices in $O(n \log^3 n)$ time. The spanning ratio of a geometric network, i.e., the largest ratio between the network distance and the Euclidean distance of any two points, has been considered as target function for edge augmentation, as well. For instance, Farshi *et al.* [3] compute a shortcut that minimizes the spanning ratio in $O(n^4)$ time while Luo and Wulff-Nilsen [8] compute a shortcut that maximizes the spanning ratio in $O(n^3)$ time.

1.3 Structure and Results

Our results concern networks that are paths, cycles, and convex cycles. Figures 1 and 2 illustrate examples of optimal shortcuts for paths and cycles. In Section 2, we develop an

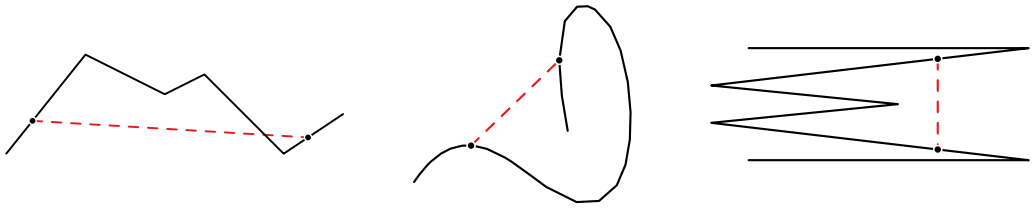


Figure 1 Examples for paths with an optimal shortcut.

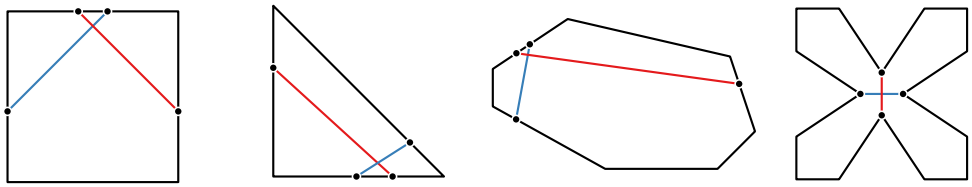


Figure 2 Examples for cycles with optimal pairs of shortcuts.

algorithm that produces an optimal shortcut for a path with n vertices in $O(n)$ time. In Section 3, we show that for cycles a single shortcut never suffices to reduce the diameter and that two shortcuts always suffice. We characterize pairs of optimal shortcuts for convex and non-convex cycles. Based on this characterization, we develop an algorithm in Section 4 that determines an optimal pair of shortcuts for a convex cycle with n vertices in $O(n)$ time.

The full version of this paper [1] contains all proofs that were omitted in this version.

2 Shortcuts for Paths

Consider a polygonal path P in the plane with n vertices. We seek a shortcut pq for a path P that minimizes the continuous diameter of the augmented path $P + pq$, i.e.,

$$\operatorname{diam}(P + pq) = \min_{a,b \in P} \operatorname{diam}(P + ab) = \min_{a,b \in P} \max_{u,v \in P+ab} d_{P+ab}(u, v) \ .$$

For this section, let s and e be the endpoints of P and let p be closer to s than q along P, i.e., $d(s, p) < d(s, q)$, as illustrated in Figure 3. For $a, b \in P$, let $P[a, b]$ denote the sub-path from a to b along P, and let $C(p, q)$ be the simple cycle in $P + pq$.

▶ **Lemma 1.** *Let pq be a shortcut for P. Every continuous diametral path in $P + pq$ contains an endpoint of P, except when the shortcut connects the endpoints of P.*

According to Lemma 1, we have the following three candidates for continuous diametral paths in the augmented network $P + pq$, two of which are illustrated in Figure 4.

1. The path $U(p, q)$ from s to e via the shortcut pq,
2. the path $S(p, q)$ from s to the farthest point from s on $C(p, q)$, and
3. the path $E(p, q)$ from e to the farthest point from e on $C(p, q)$.

Let $\bar{p}$ be the farthest point from p on $C(p, q)$, and let $\bar{q}$ be the farthest point from q on $C(p, q)$. Furthermore, let $\delta(p, q) := \frac{d(p,q)-|pq|}{2}$ denote the slack between p and $\bar{q}$ (and symmetrically between $\bar{p}$ and q) along $C(p, q)$. With this notation, we have

$$d(p, \bar{p}) = d(q, \bar{q}) = \frac{|C(p, q)|}{2} = \frac{d(p, q) + |pq|}{2} = \frac{d(p, q) - |pq|}{2} + |pq| = \delta(p, q) + |pq| \ ,$$

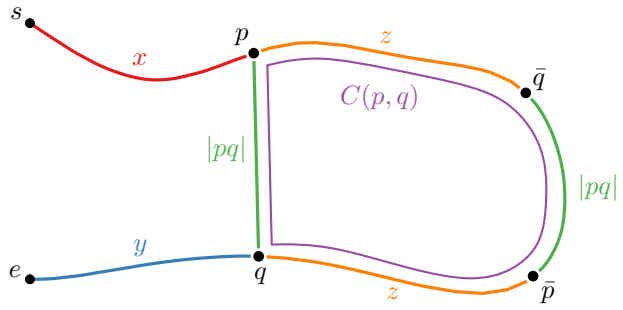
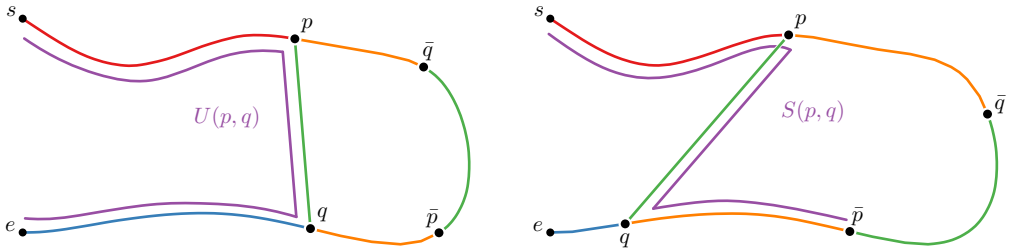


Figure 3 Augmenting a path P with a shortcut pq. The shortcut creates a cycle $C(p,q)$ with the sub-path from p to q along P. The farthest point from p on this cycle is $\bar{p}$ and $\bar{q}$ is farthest from q on $C(p,q)$. The distance $d(\bar{q},\bar{p})$ between $\bar{q}$ and $\bar{p}$ along P matches the Euclidean distance between p and q, because of the following. When we move a point g from p to q along the shortcut pq, then the farthest point $\bar{g}$ from g along $C(p,q)$ moves from $\bar{p}$ to $\bar{q}$ traveling the same distance as g, i.e., $|pq|$.



(a) The path $U(p,q)$ from s to e.

(b) The path $S(p,q)$ from s to the cycle.

Figure 4 Two candidate diametral paths in $P+pq$, namely the shortest path connecting s and e in a and a path from s via p to the farthest point from p on the cycle $C(p,q)$ in b. For the latter, there is a second path $S'(p,q)$ of the same length traversing $C(p,q)$ in the other direction.

and we can express the lengths of $U(p,q)$, $S(p,q)$, and $E(p,q)$ as follows.

$$|U(p,q)| = d(s,p) + |pq| + d(q,e)$$
$$|S(p,q)| = d(s,p) + d(p,\bar{p}) = d(s,p) + |pq| + \delta(p,q)$$
$$|E(p,q)| = d(e,q) + d(q,\bar{q}) = d(e,q) + |pq| + \delta(p,q)$$

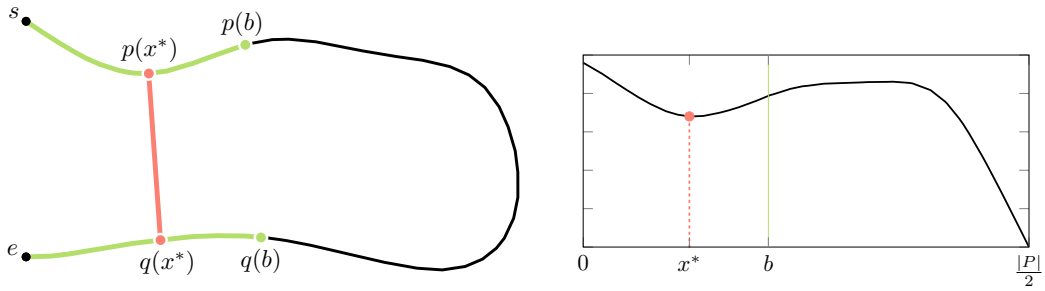
The following lemma characterizes which of the paths $U(p,q)$, $S(p,q)$, and $E(p,q)$ determine the diameter of $P+pq$. Notice that these cases overlap, for instance, $E(p,q)$ and $S(p,q)$ are both continuous diametral when $d(s,p) = d(e,q) \leq \delta(p,q)$.

▶ **Lemma 2.** *Let pq be a shortcut for a path P. Let $x = d(s,p)$, $y = d(e,q)$, and $z = \delta(p,q)$.*

- *The path $U(p,q)$ is continuous diametral if and only if $z = \min(x,y,z)$.*
- *The path $S(p,q)$ is continuous diametral if and only if $y = \min(x,y,z)$.*
- *The path $E(p,q)$ is continuous diametral if and only if $x = \min(x,y,z)$.*

▶ **Lemma 3.** *For every path P, there is an optimal shortcut pq such that $S(p,q)$ and $E(p,q)$ are continuous diametral paths in $P+pq$, i.e.,* $\operatorname{diam}(P+pq) = |S(p,q)| = |E(p,q)|$.

According to Lemmas 2 and 3, we can restrict our search for an optimal shortcut to those shortcuts satisfying $d(s,p) = d(e,q) \leq \delta(p,q)$. For $x \in [0, |P|/2]$, let $p(x)$ and $q(x)$ be the points on P such that $x = d(s,p(x))$ and $x = d(e,q(x))$, and let $D(x) = |p(x)q(x)|$. Notice



(a) A path P with its optimal shortcut $p(x^*)q(x^*)$. **(b)** A plot of $D(x) = |p(x)q(x)|$.

Figure 5 The optimal shortcut for the path in a with the function $D(x)$ plotted in b.

that $d(p(x), q(x)) = |P| - 2x$. Using this notation, we phrase our problem as

$$\begin{aligned} \min\ & x + \frac{d(p(x),q(x)) + |p(x)q(x)|}{2} = \min\ x + \frac{|P| - 2x + |p(x)q(x)|}{2} = \min\ \frac{|P| + D(x)}{2} \\ \text{s.t. } & x \le \delta(p(x), q(x)) = \frac{d(p(x),q(x)) - |p(x)q(x)|}{2} = \frac{|P| - 2x - D(x)}{2}\ , \end{aligned}$$

which simplifies to minimizing $D(x)$ such that $4x + D(x) \le |P|$.

▶ **Lemma 4.** *The function $B(x) = 4x + D(x)$ is strictly increasing on $[0, |P|/2]$.*

The following theorem describes an optimal shortcut and is illustrated in Figure 5.

▶ **Theorem 5.** *Let P be a path and let b be the unique value in $[0, |P|/2]$ with $B(b) = |P|$. Suppose D has a global minimum in the interval $[0, b]$ at x^*, i.e., $D(x^*) = \min_{x\in[0,b]} D(x)$. Then the shortcut $p(x^*)q(x^*)$ achieves the minimum continuous diameter for P.*

▶ **Lemma 6.** *Let P be a path with n vertices. Then $D^2(x)$ is a continuous function whose graph consists of at most n parabolic arcs or line segments.*

▶ **Corollary 7.** *Given a path P with n vertices, we can compute a shortcut for P achieving the minimal continuous diameter in $O(n)$ time.*

Proof. Let $x_{\pi(1)} \le x_{\pi(2)} \le \cdots \le x_{\pi(n)}$ be the values in $[0, |P|/2]$ where $p(x_{\pi(i)})$ or $q(x_{\pi(i)})$ coincides with the i-th vertex of P for each $i = 1, 2, \ldots, n$.

We compute the minimum of the parabolic arc of D^2 on each interval $[x_{\pi(i)}, x_{\pi(i+1)}]$ for $i = 1, 2, \ldots, n$ until we arrive at k with $B(x_{\pi(k)}) < |P|$ and $B(x_{\pi(k+1)}) \ge |P|$. We then compute b by solving the quadratic equation $D^2(b) = (|P| - 4b)^2$ and, finally, compute the minimum of $D^2(x)$ on $[x_{\pi(k)}, b]$. The lowest minima of the encountered parabolic arcs is the global minimum of D on $[0, b]$, which reveals the position of an optimal shortcut according to Theorem 5. Altogether, the running time is $\Theta(k + 1) = O(n)$, since we obtain $x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(k+1)}$ by merging the vertices by their distances from s or from e. ◀

▶ Remark. Our result on the location of an optimal shortcut from Theorem 5 also holds for rectifiable curves in the plane. However, obtaining an optimal shortcut for such curves depends on our ability to calculate b and a global minimum of $D(x)$ in the interval $[0, b]$.

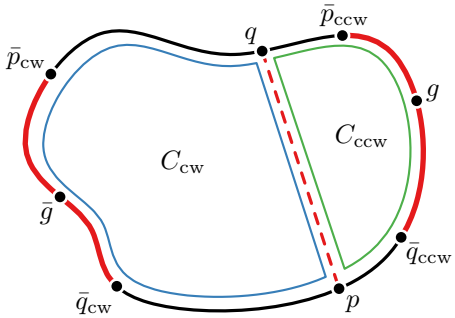


Figure 6 The unaffected regions (solid red) for a shortcut pq (dashed red) to a cycle. The point $\bar{x}_{\text{y}}$ denotes the farthest points from x along the cycle C_{y} for $x \in \{p, q\}$ and $\text{y} \in \{\text{cw}, \text{ccw}\}$. Any point g along the clockwise path from $\bar{p}_{\text{ccw}}$ to $\bar{q}_{\text{ccw}}$ has their farthest point $\bar{g}$ on the clockwise path from $\bar{q}_{\text{cw}}$ to $\bar{p}_{\text{cw}}$ and vice versa. The distance between g and $\bar{g}$ is unaffected by the addition of pq to C.

3 Shortcuts for Cycles

Consider a polygonal cycle C in the plane that may have crossings. For any two points p and q along C that may be vertices or points along edges of C, let $d_{\text{ccw}}(p, q)$ and $d_{\text{cw}}(p, q)$ be their counter-clockwise and clockwise distance along C, respectively. Let $d(p, q) = \min(d_{\text{ccw}}(p, q), d_{\text{cw}}(p, q))$ denote the network distance between p and q along C. We seek to minimize the continuous diameter by augmenting C with shortcuts.

▶ **Lemma 8.** *Adding a single shortcut pq to a polygonal cycle C never decreases the continuous diameter, i.e.,* $\text{diam}(C) = \text{diam}(C + pq)$ *for all* $p, q \in C$.

Proof Sketch. Consider any shortcut pq to a cycle C. Let C_{ccw} be the cycle consisting of pq and the counter-clockwise path from p to q along C, as illustrated in Figure 6. Let $\bar{p}_{\text{ccw}}$ and $\bar{q}_{\text{ccw}}$ be the farthest points from p and from q on C_{ccw}, respectively. Since $\bar{p}_{\text{ccw}}$ and $\bar{q}_{\text{ccw}}$ are antipodal from p and q in C_{ccw}, we have $d(\bar{q}_{\text{ccw}}, \bar{p}_{\text{ccw}}) = |pq|$ and $d(\bar{p}_{\text{ccw}}, q) = d(p, \bar{q}_{\text{ccw}})$.

Consider a point g along the clockwise path from $\bar{p}_{\text{ccw}}$ to $\bar{q}_{\text{ccw}}$ and let $\bar{g} \in C$ be the farthest point from g with respect to C. We can show that $d_C(g, \bar{g}) = d_{C+pq}(g, \bar{g})$. ◀

According to Lemma 8, some points preserve their farthest distance in C when adding a single shortcut pq to C. The points that are unaffected by pq in this sense form the *unaffected region* of pq that consists of the clockwise path from $\bar{p}_{\text{ccw}}$ to $\bar{q}_{\text{ccw}}$ and the clockwise path from $\bar{q}_{\text{cw}}$ to $\bar{p}_{\text{cw}}$, as illustrated in Figure 6. Conversely, every point on C outside of the unaffected region uses pq as a shortcut to their farthest point in $C + pq$.

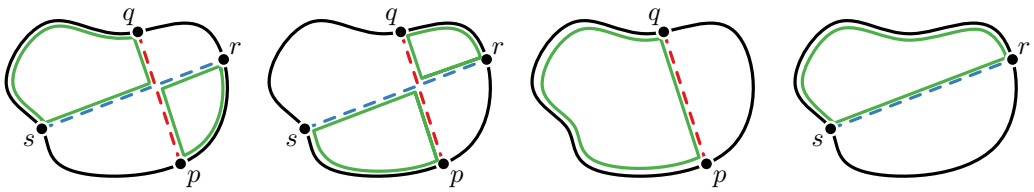
Consequently, we have to add at least two shortcuts pq and rs in order to decrease the continuous diameter of the augmented cycle $C + pq + rs$. Figure 2 illustrates examples of optimal pairs of shortcuts for cycles. We call a pair of shortcuts pq and rs *useful* when $\text{diam}(C) > \text{diam}(C + rs + pq)$, and we call pq and rs *useless*, otherwise. A pair of shortcuts pq and rs is useful if and only if their unaffected regions are disjoint.

We call a polygonal cycle C *degenerate* when it consists of two congruent line segments of length $|C|/2$. Any number of shortcuts cannot decrease the diameter of a degenerate cycle, since the endpoints of its line segment remain at the same distance.

▶ **Theorem 9.** *For every non-degenerate cycle C, there exists a pair of shortcuts pq and rs that decrease the continuous diameter, i.e.,* $\text{diam}(C) > \text{diam}(C + pq + rs)$.

(a) Alternating Configuration **(b)** Consecutive Configuration

Figure 7 The two cases for adding two shortcuts pq and rs to a cycle C. The endpoints of the shortcuts appear in alternating cyclic order p, r, q, and s, as shown in a, or in consecutive cyclic order p, q, r, and s, as shown in b. The two cases overlap when q coincides with r.



(a) The bowtie ($\bowtie$). **(b)** The hourglass ($\boxtimes$). **(c)** The red split ($\oslash$). **(d)** The blue split ($\oslash$).

Figure 8 The candidate diametral cycles, except C, for shortcuts in the alternating configuration.

Proof Sketch. Suppose there exist three points p, q, and s on C with $d(p,q) = d(q,s) = |C|/4$ such that pq and qs are shortcuts, i.e., $|pq| < d(p,q)$ and $|qs| < d(q,s)$. We can argue that pq and qs are useful. Conversely, we can show that C is degenerate when at least one of pq and qs is not a shortcut for every three points p, q, and s on C with $d(p,q) = d(q,s) = |C|/4$. ◀

3.1 Alternating vs. Consecutive

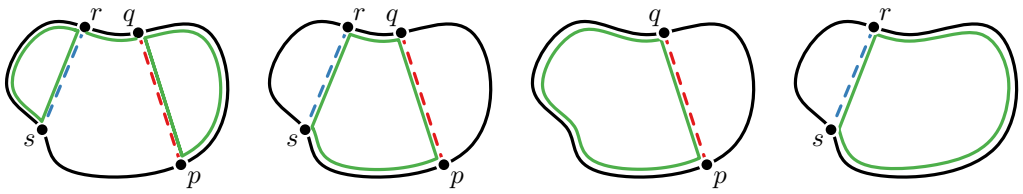
When placing two shortcuts pq and rs on a cycle C, we distinguish whether their endpoints appear in alternating order or in consecutive order along the cycle, as illustrated in Figure 7.

We show that there is always an optimal pair of shortcuts in the alternating configuration by studying the cycles created by the insertion of the shortcuts. A cycle in $C + pq + rs$ is *diametral* when it contains a diametral pair. Each configuration has five candidates for diametral cycles: two that use both shortcuts, two that use one of the shortcuts, and one (C) that does not use any shortcut. Figures 8 and 9 illustrate the candidates for diametral cycles in each configuration, except for C itself. To distinguish the cycles using one shortcut, we color pq red and rs blue and we refer to the longer cycle in $C + pq + rs$ using the red shortcut pq as the *red split* and we refer to the longer cycle using the blue shortcut rs as the *blue split*. If C happens to be diametral in $C + pq + rs$, then our pair of shortcuts is useless.

▶ **Lemma 10.** *Two shortcuts pq and rs in alternating configuration are useful if and only if $|pq| + |rs| < d_{ccw}(r,q) + d_{ccw}(s,p)$ and $|pq| + |rs| < d_{ccw}(p,r) + d_{ccw}(q,s)$.*

Proof. Suppose pq and rs are useless, i.e., the unaffected regions of pq and rs overlap. In the alternating configuration, this overlap occurs along the bowtie or along the hourglass. Since these cases are symmetric, we consider only the former in the following.

An overlap on the bowtie manifests along the clockwise path from $\bar{r}_{\text{ccw}}$ to $\bar{p}_{\text{cw}}$ with a mirrored overlap along the clockwise path from $\bar{s}_{\text{cw}}$ to $\bar{q}_{\text{ccw}}$, as illustrated in Figure 10. This means the sum of the lengths of the counter-clockwise paths from r to $\bar{r}_{\text{ccw}}$ and from $\bar{p}_{\text{cw}}$ to p is



(a) The handset. **(b)** The base station. **(c)** The red split. **(d)** The blue split.

Figure 9 The candidate diametral cycles, except C, for shortcuts in the consecutive configuration, depicted for $d(q,r) \leq d(s,p)$. Even though the handset a is no simple cycle, it might still contain a diametral pair. Observe that the base station b is only listed for the sake of completeness: by the triangle inequality, this cycle is never longer than the split cycles and, therefore, never diametral.

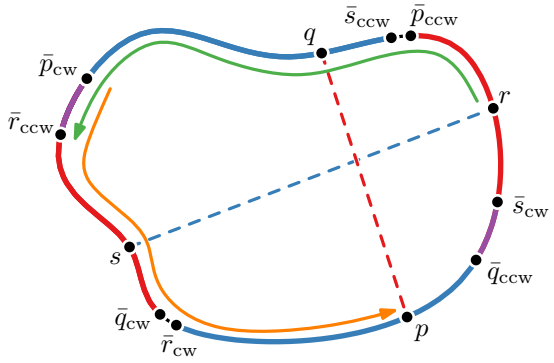


Figure 10 A pair of useless shortcuts whose unaffected regions have an overlap (purple) along the bowtie, i.e., the points s, $\bar{r}_{\text{ccw}}$, $\bar{p}_{\text{cw}}$, and q appear clockwise in this order along the cycle.

at least the length of the counter-clockwise path from r to p, i.e., $d_{\text{ccw}}(\bar{p}_{\text{cw}}, p) + d_{\text{ccw}}(r, \bar{r}_{\text{ccw}}) \geq d_{\text{ccw}}(r,p)$. This is equivalent to $|pq| + |rs| \geq d_{\text{ccw}}(r,q) + d_{\text{ccw}}(s,p)$, since

$$\begin{aligned} d_{\text{ccw}}(q,p) + |pq| + d_{\text{ccw}}(r,s) + |rs| &= 2d_{\text{ccw}}(\bar{p}_{\text{cw}}, p) + 2d_{\text{ccw}}(r, \bar{r}_{\text{ccw}}) \geq 2d_{\text{ccw}}(r,p) \\ \iff |pq| + |rs| &\geq \underbrace{d_{\text{ccw}}(r,p) - d_{\text{ccw}}(q,p)}_{=d_{\text{ccw}}(r,q)} + \underbrace{d_{\text{ccw}}(r,p) - d_{\text{ccw}}(r,s)}_{=d_{\text{ccw}}(s,p)} . \end{aligned}$$

Analogously, we derive that $|pq| + |rs| \geq d_{\text{ccw}}(p,r) + d_{\text{ccw}}(q,s)$ holds if and only if there is an overlap along the hourglass. Consequently, the shortcuts pq and rs are useful if and only if $|pq| + |rs| < d_{\text{ccw}}(r,q) + d_{\text{ccw}}(s,p)$ and $|pq| + |rs| < d_{\text{ccw}}(p,r) + d_{\text{ccw}}(q,s)$. ◀

▶ **Lemma 11.** *Consider two consecutive shortcuts pq and rs with $d_{ccw}(q,r) \leq d_{ccw}(s,p)$. Then pq and rs are useful if and only if $|pq| + |rs| < d_{ccw}(s,p) - d_{ccw}(q,r)$.*

▶ **Theorem 12.** *Let pq and rs be a pair of shortcuts for a cycle C in consecutive configuration. There exists a pair $p'q'$ and $r's'$ of shortcuts in the alternating configuration that are at least as good as pq and rs, i.e.,* $\text{diam}(C + p'q' + r's') \leq \text{diam}(C + pq + rs)$.

Proof Sketch. Suppose pq and rs are useful shortcuts in the consecutive configuration. Assume, without loss of generality, $d_{\text{ccw}}(q,r) \leq d_{\text{ccw}}(s,p)$ and $d_{\text{ccw}}(p,q) \leq d_{\text{ccw}}(r,s)$. We consider the shortcuts $p'q' = pr$ and $r's' = rs$, which are illustrated in Figure 11 and lie in the intersection of the alternating and consecutive case. We argue that pr and rs are useful shortcuts and that each candidate diametral cycle in $C + pq + rs$ has a one-to-one correspondence to a candidate diametral cycle in $C + pr + rs$ of smaller or equal length. ◀

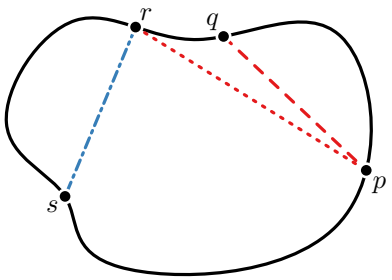


Figure 11 Replacing consecutive shortcuts pq and rs with alternating $p'q' = pr$ and $r's' = rs$.

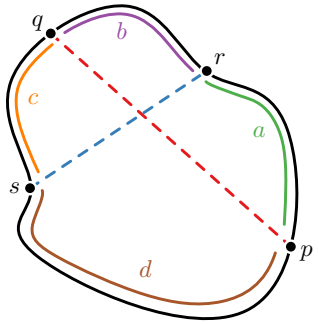


Figure 12 The sections of a cycle with a pair of alternating shortcuts.

3.2 Balancing Diametral Cycles

We show that every cycle has an optimal pair of alternating shortcuts where the bowtie and the hourglass are both diametral and we show that every convex cycle has an optimal pair of shortcuts where both split cycles are diametral, as well. We obtain these results by applying a sequence of operations – some of which are shown in Figure 14 – that each slide the shortcuts along the cycle in a way that reduces or maintains the continuous diameter and brings the candidate diametral cycles closer to the desired balance. The last two operations only reduce the diameter for convex cycles as the shortcuts might get stuck at reflex vertices, which leads to our characterization of optimal shortcuts for convex and non-convex cycles.

Let pq and rs be two alternating shortcuts and let $a = d_{\text{ccw}}(p, r)$, $b = d_{\text{ccw}}(r, q)$, $c = d_{\text{ccw}}(q, s)$, and $d = d_{\text{ccw}}(s, p)$. We assume that the red split contains s and the blue split contains p, i.e., $a + b \leq c + d$ and $b + c \leq a + d$, as in Figure 12. We abbreviate the lengths of the bowtie ($\bowtie$), the hourglass ($\boxtimes$), the red split ($\oslash$), and the blue split ($\oslash$) as follows.

$$\bowtie := a + c + |pq| + |rs| \quad \oslash := c + d + |pq| \quad \boxtimes := b + d + |pq| + |rs| \quad \oslash := a + d + |rs|$$

▶ **Lemma 13.** *For each relation $\sim \in \{<, =, >\}$, we have*

$$\bowtie \sim \boxtimes \iff a + c \sim b + d \qquad \oslash \sim \oslash \iff c + |pq| \sim a + |rs|$$
$$\bowtie \sim \oslash \iff a + |rs| \sim d \qquad \boxtimes \sim \oslash \iff b + |rs| \sim c$$
$$\bowtie \sim \oslash \iff c + |pq| \sim d \qquad \boxtimes \sim \oslash \iff b + |pq| \sim a$$

and pq and rs are useful if and only if $|pq| + |rs| < a + c$ and $|pq| + |rs| < b + d$.

Proof. The claims follow from the definitions of $\bowtie$, $\boxtimes$, $\oslash$, and $\oslash$. ◀

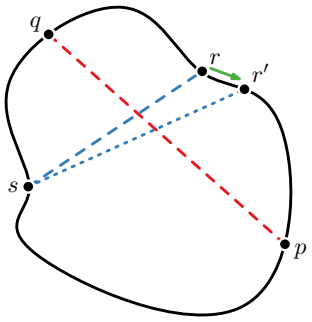


Figure 13 Shrinking the blue split.

▶ **Lemma 14.** *Consider a pair of useful alternating shortcuts where one of the split cycles evenly divides the cycle. Then this split cycle must have length at most* $\bowtie$ *or at most* $⧖$.

Proof. Assume that we have a pair of useful shortcuts where $⦸$ divides the cycle evenly, i.e., $a + b = c + d$, and where $\bowtie < ⦸$ and $⧖ < ⦸$. Then $a + |rs| < d$ and $b + |rs| < c$, by Lemma 13, which yields the contradiction $|rs| < 0$, as $a + |rs| < d = a + b - c < a - |rs|$. ◀

▶ **Lemma 15.** *There exists an optimal pair of shortcuts in alternating configuration such that none of the split cycles is the only diametral cycle.*

Proof Sketch. Suppose pq and rs are useful and $⊘$ is the only diametral cycle in $C + pq + rs$, i.e., $\bowtie, ⧖, ⦸ < ⊘$. Then we have $b + |pq| < a$ and $c + |pq| - |rs| < a$. We move r clockwise until we arrive at some r' where $b' + |pq| = a'$ or $c + |pq| - |r's| = a'$, as in Figure 13.

Since the blue split is diametral, it cannot divide the cycle evenly, by Lemma 14. Therefore, changing r to r' changes the candidate diametral cycles as follows.

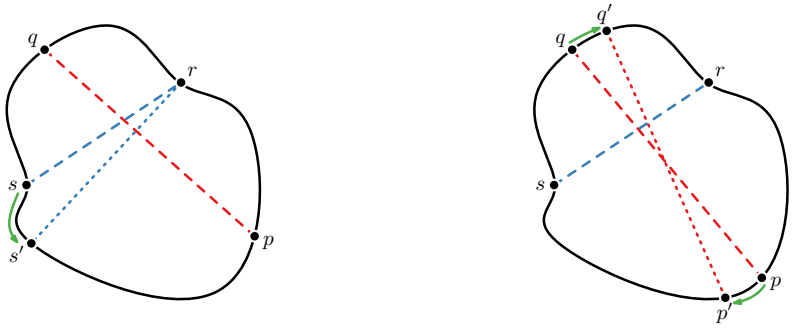
- The blue split shrinks or remains the same, i.e., $⊘ \geq ⊘'$.
- The red split remains the same, i.e., $⦸ = ⦸'$.
- The bowtie changes as the blue split, i.e., $\bowtie \geq \bowtie'$ and $\bowtie' < ⊘'$.
- The hourglass remains the same or increases, i.e., $⧖ \leq ⧖'$.
- The hourglass increases when the blue split remains the same, i.e, $⊘ - ⧖ > ⊘' - ⧖'$.

Consequently, $\operatorname{diam}(C + pq + r's) \leq \operatorname{diam}(C + pq + rs)$ and $⊘' = ⧖'$ or $⊘' = ⦸'$, which implies our claim, provided that pq and $r's$ are useful shortcuts. ◀

▶ **Lemma 16.** *There exists a pair of optimal shortcuts with* $\bowtie = ⧖$.

Proof Sketch. Suppose pq and rs are useful shortcuts with $\bowtie \neq ⧖$. We balance $\bowtie$ and $⧖$ using the following operations, as illustrated in Figure 14. They maintain or decrease the continuous diameter while decreasing the difference between bowtie and hourglass.

1. As long as neither split cycle divides the cycle C evenly, we shrink the larger split cycle in a way that decreases the difference of bowtie and hourglass:
   - **a.** When $\bowtie < ⧖$ and $⦸ \leq ⊘$, we move s counter-clockwise.
   - **b.** When $\bowtie < ⧖$ and $⦸ > ⊘$, we move p clockwise.
   - **c.** When $\bowtie > ⧖$ and $⦸ \leq ⊘$, we move r clockwise.
   - **d.** When $\bowtie > ⧖$ and $⦸ > ⊘$, we move q counter-clockwise.
2. Once a split cycle evenly divides the cycle, we move the endpoints of the corresponding shortcut in the direction that decreases the difference between bowtie and hourglass:



(a) Operation 1.a. **(b)** Operation 2.a.

Figure 14 Some of the operations that are used to balance the candidate diametral cycles.

a. When $\bowtie < \between$ and $\oslash$ evenly divides the cycle, we move p and q clockwise.
b. When $\bowtie > \between$ and $\oslash$ evenly divides the cycle, we move p and q counter-clockwise.
c. When $\bowtie < \between$ and $\oslash$ evenly divides the cycle, we move s and r counter-clockwise.
d. When $\bowtie > \between$ and $\oslash$ evenly divides the cycle, we move s and r clockwise.

For each operation, we argue that pq and rs remain useful shortcuts and that the diameter never increases while the difference between hourglass and bowtie always decreases. ◀

▶ **Corollary 17.** *There exists a pair of optimal shortcuts that is in the alternating configuration such that none of the split cycles is the only diametral cycle and such that the bowtie and the hourglass have the same length.*

Proof. By Lemma 15, we have a pair of optimal shortcuts pq and rs for a cycle C where none of the splits is the only diametral cycle. The Operations 1.a to 1.d from Lemma 16 shrink the larger split cycle at the same rate as they shrink the larger of bowtie and hourglass. Thus, we do not create a sole diametral cycle by applying these operations. Furthermore, the Operations 2.a to 2.d rotate an even split that cannot become diametral by Lemma 14. By applying Lemma 16, we obtain a pair of optimal shortcuts $p'q'$ and $r's'$ with at least two diametral cycles and where bowtie and hourglass have the same length. ◀

▶ **Theorem 18.** *For every non-degenerate cycle, there exists an optimal pair of shortcuts such that the hourglass and the bowtie are both diametral.*

Proof. Let C be a non-degenerate cycle. By Corollary 17, there is a pair of optimal shortcuts pq and rs where neither split cycle is the only diametral cycle and where $\bowtie = \between$.

Suppose that $\bowtie$ and $\between$ are not diametral. The cycle C cannot be diametral, since pq and rs are useful. This means a split is diametral, i.e., $\oslash > \bowtie = \between$ or $\oslash > \bowtie = \between$. Since neither $\oslash$ nor $\oslash$ is the only diametral cycle, we have $\oslash = \oslash > \bowtie = \between$.

We shrink the splits by simultaneously moving p and r clockwise while moving q and s counter-clockwise, as illustrated in Figure 15. By moving pq and rs at appropriate speeds, we ensure that this operation maintains both the balance between the split cycles and the balance between bowtie and hourglass, i.e., $\oslash' = \oslash'$ and $\bowtie' = \between'$. This decreases the continuous diameter, provided that the line segments $p'q'$ and $r's'$ remain useful shortcuts.

Assume, for the sake of a contradiction, that $p'q'$ is not a shortcut, i.e., $|p'q'| = d(p'q')$. By Lemma 14 we cannot pass through an even red split during our operation. Thus, we have $d(p', q') = d_{\text{ccw}}(p', q')$, i.e., the line segment $p'q'$ contains pq contradicting the choice of pq as shortcut. Therefore, $p'q'$ is a shortcut. Symmetrically, we can argue that $r's'$ is a shortcut.

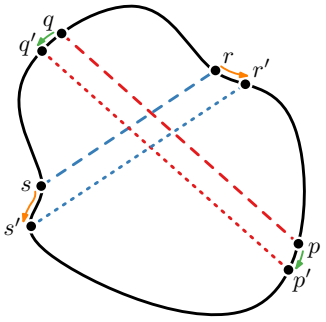


Figure 15 Shifting the shortcuts to shrink the split cycles while maintaining $\oslash = \oslash$ and $\bowtie = \between$.

We argue that $p'q'$ and $r's'$ remain useful. From $\bowtie' = \between' \leq \oslash' = \oslash'$, we obtain $|p'q'| \leq d' - c'$, $|p'q'| \leq a' - b'$, and $|r's'| \leq c' - b'$, by Lemma 13. Together with $b' > 0$ and $a' + c' = b' + d'$, we derive that $p'q'$ and $r's'$ are useful, because $|p'q'| + |r's'| \leq d' - c' + c' - b' = d' - b' < d' + b' = a' + c'$. This means $p'q'$ and $r's'$ are useful shortcuts with $\bowtie' = \between' = \oslash' = \oslash'$ and $\operatorname{diam}(C + pq + rs) > \operatorname{diam}(C + p'q' + r's')$ contradicting the optimality of pq and rs. Therefore, there are optimal shortcuts where the hourglass and the bowtie are diametral. ◀

▶ **Theorem 19.** *For every convex cycle, there exists an optimal pair of alternating shortcuts such that the hourglass, the bowtie, and the splits are diametral, i.e.,* $\bowtie = \between = \oslash = \oslash$.

Proof Sketch. According to Theorem 18, there are optimal shortcuts pq and rs with $\bowtie = \between \geq \oslash$ and $\bowtie = \between \geq \oslash$. Suppose we have $\bowtie = \between > \oslash$ or $\bowtie = \between > \oslash$.

First, since C is convex we can increase each split in a way that shrinks its shortcut. Second, we grow the smaller split until both splits are equal. Third, we grow both splits at the same rate until they are equal to bowtie and hourglass. ◀

▶ **Corollary 20.** *For every non-degenerate cycle, there exists an optimal pair of shortcuts such that the hourglass and the bowtie are diametral and such that each split cycle is diametral or the shortcut of the split has at least one endpoint at a reflex vertex.*

▶ **Corollary 21.** *For every convex cycle, there exists an optimal pair of shortcuts with* $a + b \leq c + d$ *and* $b + c \leq a + d$ *such that the following holds.*

$$d = \frac{|C|}{4} + \frac{|pq| + |rs|}{2} = \operatorname{diam}(C + pq + rs) \qquad a = \frac{|C|}{4} + \frac{|pq| - |rs|}{2}$$

$$b = \frac{|C|}{4} - \frac{|pq| + |rs|}{2} = \operatorname{diam}(C) - \operatorname{diam}(C + pq + rs) \qquad c = \frac{|C|}{4} + \frac{|rs| - |pq|}{2}$$

Surprisingly, this means we can read the new continuous diameter of $C + pq + rs$ from d and we can read the benefit of adding the shortcuts pq and rs to C from b.

4 A Linear-Time Algorithm for Convex Cycles

For convex cycles, we restrict our search to the pairs of shortcuts satisfying $\bowtie = \between = \oslash = \oslash$, due to Theorem 19. We proceed as follows. First, we pick some point p on the cycle C and compute three points q, r, and s such that $\bowtie = \between = \oslash = \oslash$ – regardless of whether pq and rs are shortcuts. We show that the points q, r, and s exist and are unique for every point p along C. Once we have balanced p, q, r, and s, we slide p along C maintaining the balance by moving q, r, and s appropriately. We show that q, r, and s move in the same direction as

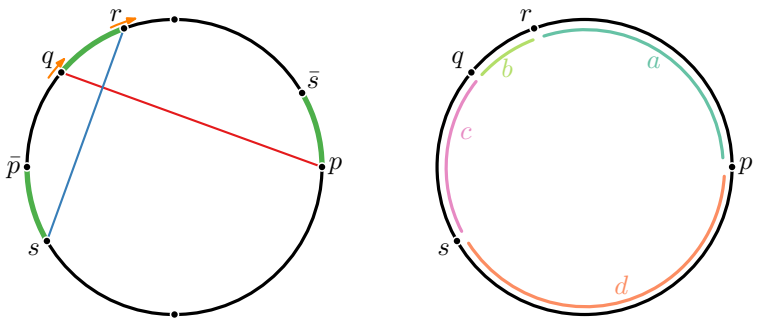


Figure 16 Locating r and s to balanced the splits for fixed p and s.

p while preserving their order along C. Thus, each endpoint traverses each edge of the n edges of C at most once throughout this process, which therefore takes $O(n)$ time.

For the remainder of this section, we only focus on convex cycles with non-empty interior. Consider a cycle C and a fixed point p on C. We say a triple of points q, r, and s is *in balanced configuration with* p when the points p, r, q, and s appear counter-clockwise in this order along C, $d_{\text{ccw}}(p,q) \leq |C|/2$, $d_{\text{ccw}}(r,s) \leq |C|/2$, and $\bowtie = \between = \circledcirc = \oslash$.

▶ **Theorem 22.** *Consider a convex cycle C and a point p on C. There exists a unique triple q, r, s of points on C that are in balanced configuration with p.*

Proof Sketch. Suppose we place s at some position on C with $|C|/4 \leq d_{\text{ccw}}(s,p) \leq |C|/2$. The points r and q must have fixed distance $d(r,q) = |C|/2 - d_{\text{ccw}}(s,p)$ to ensure $\bowtie = \between$. Suppose we slide q and r along the clockwise path from $\bar{p}$ to $\bar{s}$ while maintaining $d(q,r) = d(\bar{p},s)$, as in Figure 16. When q and r are close to s, we have $\circledcirc < \oslash$ and when q and r are close to p, we have $\circledcirc > \oslash$. By the intermediate value theorem, there exist positions for q and r such that $\bowtie = \between$ and $\circledcirc = \oslash$ and these positions are unique, since C is convex.

Suppose we slide s from $\bar{p}$ towards p while maintaining $\bowtie = \between$ and $\circledcirc = \oslash$. When $d(s,p) = |C|/2$, we end up with $\bowtie = \between < \circledcirc = \oslash$ and when $d(s,p) = |C|/4$, we end up with $\bowtie = \between > \circledcirc = \oslash$. By the intermediate value theorem, there exist positions for s, q, and r such that $\bowtie = \between = \circledcirc = \oslash$. We find these positions with two nested binary searches. ◀

▶ **Lemma 23.** *Consider a convex cycle C. Suppose p moves counter-clockwise along C. Then any three points in balanced configuration with p are moving counter-clockwise, as well.*

▶ **Theorem 24.** *Consider a convex cycle C with n vertices. We can compute an optimal pair of shortcuts for C in $O(n)$ time.*

Proof. We pick an arbitrary point p along some edge e_p of C and identify the edges e_q, e_r, and e_s containing the points q, r, and s that form a balanced configuration with p, as described in the proof sketch of Theorem 22. We find a (locally) optimal pair of shortcuts p^*q^* and r^*s^* with whose endpoints lie on the edges e_p, e_q, e_r, e_s by minimizing $d = \text{diam}(C + pq + rs)$ subject to $a + b \leq |C|/2$ and $b + c \leq |C|/2$, and the constraints stated in Corollary 21 that ensure $\bowtie = \between = \circledcirc = \oslash$. Then, we identify the four edges that would host p, q, r, and s next, if p were to move counter-clockwise: for each endpoint $x \in \{p,q,r,s\}$, we calculate how far the other endpoints would move under the assumption that x is the first point to hit a vertex. Theorem 22 guarantees that this calculation has a unique solution. Since all points move in the same direction as p, an edge e will never host an endpoint x in any subsequent step, once x has left e. Therefore, the entire process takes $O(n)$ time. Since we encounter every four points in balanced configuration, we also encounter an optimal pair of shortcuts. ◀

5 Conclusion and Future Work

Our work reveals some of the underlying challenges that must be overcome when addressing the discrete version of the network augmentation problem, where we minimize the discrete diameter of a network with shortcuts that connect only vertices. We shall investigate to what extend we can translate to the discrete setting. For instance, we would like to know when and how well the optimal continuous shortcuts approximate the optimal discrete shortcuts.

By Corollary 20, we can determine an optimal pair of shortcuts for non-convex cycles with r reflex vertices in $O(rn^3)$ time: we compute the best shortcuts satisfying $\bowtie = \between = \oslash = \oslash$ and we check all possible triples of edges that might contain the other endpoints when one shortcut is stuck at a reflex vertex. We seek to improve this naïve approach by generalizing our sliding-sweep algorithm for convex cycles to non-convex cycles. In addition the shortcuts with $\bowtie = \between = \oslash = \oslash$ (which may be non-optimal), we would have to keep track of each locally optimal shortcuts with one endpoint at a reflex vertex. However, some properties, such as the uniqueness of shortcuts in balance, break down for non-convex cycles.

As the next natural step after paths and cycles, we shall study minimizing the continuous diameter of trees, uni-cyclic networks, and so forth by introducing shortcuts.

References

1 Jean-Lou De Carufel, Carsten Grimm, Anil Maheshwari, and Michiel Smid. Minimizing the continuous diameter when augmenting paths and cycles with shortcuts. This is the full version of this paper., 2015. arXiv:1512.02257.

2 Victor Chepoi and Yann Vaxès. Augmenting trees to meet biconnectivity and diameter constraints. *Algorithmica*, 33(2):243–262, 2002. doi:10.1007/s00453-001-0113-8.

3 Mohammad Farshi, Panos Giannopoulos, and Joachim Gudmundsson. Improving the stretch factor of a geometric network by edge augmentation. *SIAM Journal on Computing*, 38(1):226–240, 2008. doi:10.1137/050635675.

4 Fabrizio Frati, Serge Gaspers, Joachim Gudmundsson, and Luke Mathieson. Augmenting graphs to minimize the diameter. *Algorithmica*, 72(4):995–1010, 2015. doi:10.1007/s00453-014-9886-4.

5 Yong Gao, Donovan R. Hare, and James Nastos. The parametric complexity of graph diameter augmentation. *Discrete Applied Mathematics*, 161(10-11):1626–1631, 2013. doi:10.1016/j.dam.2013.01.016.

6 Ulrike Große, Joachim Gudmundsson, Christian Knauer, Michiel Smid, and Fabian Stehn. Fast algorithms for diameter-optimally augmenting paths. In *42nd International Colloquium on Automata, Languages, and Programming (ICALP 2015)*, pages 678–688, 2015. doi:10.1007/978-3-662-47672-7_55.

7 Chung-Lun Li, S. Thomas McCormick, and David Simchi-Levi. On the minimum-cardinality-bounded-diameter and the bounded-cardinality-minimum-diameter edge addition problems. *Operations Research Letters*, 11(5):303–308, 1992. doi:10.1016/0167-6377(92)90007-P.

8 Jun Luo and Christian Wulff-Nilsen. Computing best and worst shortcuts of graphs embedded in metric spaces. In *19th International Symposium on Algorithms and Computation (ISAAC 2008)*, pages 764–775, 2008. doi:10.1007/978-3-540-92182-0_67.

9 Anneke A. Schoone, Hans L. Bodlaender, and Jan van Leeuwen. Diameter increase caused by edge deletion. *Journal of Graph Theory*, 11(3):409–427, 1987. doi:10.1002/jgt.3190110315.

A Clustering-Based Approach to Kinetic Closest Pair

Timothy M. Chan[1] and Zahed Rahmati[2]

1 **Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada**
tmchan@uwaterloo.ca

2 **School of Electrical and Computer Engineering, University of Tehran, Tehran, Iran**
rahmati@ece.ut.ac.ir

Abstract

Given a set P of n moving points in fixed dimension d, where the trajectory of each point is a polynomial of degree bounded by some constant, we present a kinetic data structure (KDS) for maintenance of the closest pair on P. Assuming the closest pair distance is between 1 and Δ over time, our KDS uses $O(n \log \Delta)$ space and processes $O(n^2 \beta \log \Delta \log n + n^2 \beta \log \Delta \log \log \Delta))$ events, each in worst-case time $O(\log^2 n + \log^2 \log \Delta)$. Here, β is an extremely slow-growing function. The locality of the KDS is $O(\log n + \log \log \Delta)$. Our closest pair KDS supports insertions and deletions of points. An insertion or deletion takes *worst-case* time $O(\log \Delta \log^2 n + \log \Delta \log^2 \log \Delta)$.

Also, we use a similar approach to provide a KDS for the all ε-nearest neighbors in $\mathbb{R}^d$.

The complexities of the previous KDSs, for both closest pair and all ε-nearest neighbors, have polylogarithmic factor, where the number of `log`s depends on dimension d. Assuming Δ is polynomial in n, our KDSs obtain improvements on the previous KDSs.

Our solutions are based on a kinetic clustering on P. Though we use ideas from the previous clustering KDS by Hershberger, we simplify and improve his work.

1998 ACM Subject Classification F.2.2 [Nonnumerical Algorithms and Problems] Geometrical problems and computations

Keywords and phrases Kinetic Data Structure, Clustering, Closest Pair, All Nearest Neighbors

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.28

1 Introduction

Let P be a set of of points in $\mathbb{R}^d$. The *closest pair* problem is a fundamental, well-studied proximity problem in computational geometry, which is to find a pair of points in P with minimum separation distance. A decision version of the closest pair problem, called the *closest pair decision* problem, is to decide whether the closest pair distance is less than or equal to a given r. In many applications, *e.g.*, collision detection, the closest pair decision problem is more important than the closest pair problem. A general version of the closest pair problem is finding the nearest neighbor in P for each point in P, which is called the *all nearest neighbors* problem. The *all ε-nearest neighbors* problem is to find a point $p \in P$ to each point $q \in P$ such that $d(p, q) \leq (1 + \varepsilon) \cdot d(p^*, q)$, where $p^* \in P$ is the nearest neighbor of q, and $d(.,.)$ denotes the Euclidean distance between two points; p is called an ε-nearest neighbor to q.

The *unit disk covering* problem is to find the minimum cardinality set $\mathcal{S}$ of unit disks such that each point in P is covered by some disk in $\mathcal{S}$. The problem is well-motivated from

many applications, *e.g.*, VLSI design and facility location. The unit disk covering problem is NP-hard in the L_2 and L_∞ metrics [7]. There exist polynomial time approximation solutions, of constant factor, to the unit disk covering problem in the L_2 and L_∞ metrics [4, 6, 9, 11].

Consideration of the problems on a set of moving objects has been studied extensively in different communities (*e.g.*, computational geometry, robotics, and computer graphics); see [12] and references therein. In this paper, we focus on the kinetic problems for a set P of n moving points in a *fixed* dimension d. Next, we formally state the kinetic problems.

KDS framework

Basch, Guibas, and Hershberger [2] first introduced the *kinetic data structure* (KDS) framework to maintain an attribute (*e.g.*, closest pair) of a set P of moving points. In the KDS framework, it is assumed that the trajectory of each point in P is given by a polynomial of degree bounded by some constant $\bar{s}$. A set of data structures and algorithms, namely a *kinetic data structure* (KDS), is built to maintain the attribute of interest. A KDS includes a set of *certificates* (boolean functions) that attests the attribute of interest is valid over time, except at some *discrete moments* (failure times of the certificates); when a certificate fails we say an *event* occurs. To track the next event after the current time, we define a *priority queue* of the failure times of the certificates. Note that any change to the events in the priority queue requires $O(\log n)$ for the update, and also note that the response time to an event in the KDS does not include this update time. An important criterion in a KDS is the *locality* of the KDS, which is the number of certificates associated with a particular point at any fixed time. If the locality of a KDS is polylogarithmic in n (or maximum nearest neighbor distance), the KDS is called *local*. A local KDS ensures that when a point changes its trajectory only a small number of changes is needed in the KDS.

Statements of kinetic problems

The *kinetic closest pair* problem is to maintain the closest pair in P over time. The *kinetic closest pair decision problem* is defined as follows: Given a parameter r, build a KDS to determine at any time whether the closest pair distance is less than or equal to r. Maintaining the nearest neighbor in P to each point in P is called the *kinetic all nearest neighbors* problem. The *kinetic all ε-nearest neighbors* problem is to maintain some ε-nearest neighbor $p \in P$ to each point $q \in P$ such that $d(p, q) \leq (1 + \varepsilon) \cdot d(p^*, q)$, where $p^* \in P$ is the nearest neighbor of q.

The *kinetic clustering* problem is to build a KDS that maintains a set S of clusters on the moving points in P, such that each cluster can be covered by a (unit) disk, and such that the cardinality $|S|$ of S is within a small factor of $|\mathcal{S}|$, the minimum possible by the optimal covering $\mathcal{S}$.

Related work

Basch, Guibas, and Hershberger [2] gave the first KDS for the closest pair on a set of n moving points, where the trajectory of each point is a polynomial bounded by some constant $\bar{s}$. Let $s = 2\bar{s} + 2$. Their KDS uses $O(n)$ space and processes $O(n^2 \beta_s(n) \log n)$ events, each in time $O(\log^2 n)$; their KDS is local. Here, $\beta_s(n)$ is an extremely slow-growing function, *i.e.*, $\beta_s(n) = \lambda_s(n)/n$, where $\lambda_s(n)$ is the maximum length of Davenport-Schinzel sequences of order s on n symbols. Their KDS was later simplified and extended to higher dimensions d, using multidimensional range trees, by Basch, Guibas, and Zhang [3]. The KDS of [3] uses $O(n \log^{d-1} n)$ space, processes $O(n^2 \beta_s(n) \log n)$ events, each in time $O(\log^d n)$, and it is local.

Table 1 The previous (and new) kinetic results for the attribute closest pair (CP). The attribute CP(r) is to decide whether the closest pair distance is at most r.

attribute	dim.	space	#events	proc. time	local
CP [2]	2	$O(n)$	$O(n^2\beta_s(n)\log n)$	$O(\log^2 n)$ /event	Yes
CP [3]	d	$O(n\log^{d-1} n)$	$O(n^2\beta_s(n)\log n)$	$O(\log^d n)$ /event	Yes
CP [1]	d	$O(n\log^{d-1} n)$	$O(n^2\beta_s(n)\log n)$	$O(\log^d n)$ /event	Yes
CP [14]	2	$O(n)$	$O(n^2\beta_s^2(n)\log n)$	$O(n^2\beta_s^2(n)\log^2 n)$	No
CP [here]	d	$O(n\log\Delta)$	$O((n^2\beta_s(n\log\Delta)\log\Delta)\cdot$ $(\log n+\log\log\Delta))$	$O(\log^2 n+$ $\log^2\log\Delta)$ /event	Yes
CP(r) [here]	d	$O(n)$	$O(n^2)$	$O(1)$ /event	Yes

Agarwal *et al.* [1] used multidimensional range trees to provide KDSs for maintenance of the closest pair and all the nearest neighbors in $\mathbb{R}^d$. Their closest pair KDS, which has the same approach and complexity as that of [3], supports insertions and deletions of points, where each operation takes *amortized* time $O(\log^{d+1} n)$. For maintenance of all the nearest neighbors, they implemented multidimensional range trees by randomized search trees (treaps). Their all nearest neighbors KDS uses $O(n\log^d n)$ space and handles $O(n^2\beta_s(n)\log^{d+1} n)$ events, with total processing time $O(n^2\beta_s(n)\log^{d+2} n)$. Each insertion or deletion in this KDS takes *expected* time $O(n)$. Rahmati *et al.* [14] used the kinetic semi-Yao graph (*i.e.*, theta graph) as a supergraph of the nearest neighbor graph to present a simple method for maintenance of the closest pair and all nearest neighbors. Their kinetic approach, which in fact maintains two Delaunay triangulations in $\mathbb{R}^2$, uses linear space and processes $O(n^2\beta_s^2(n)\log n)$ events, with total cost $O(n^2\beta_s^2(n)\log^2 n)$. By taking advantage of multidimensional range trees, the approach of [14] was later extended to higher dimensions to maintain all the nearest neighbors and all the ε-nearest neighbors [13]. None of the KDSs for maintenance of all the *exact* nearest neighbors is local.

Tables 1 and 2 summarize the complexities of the previous KDSs for maintenance of the closest pair, all the nearest neighbors, and all the ε-nearest neighbors. Here, "dim.", "#events", and "proc." stand for "dimension", "number of events", and "processing", respectively. There is also a different track, instead of maintaining an attribute over time, one would be interested in finding a time value for which the attribute is minimized or maximized. For a set of linearly moving points, in fixed dimension d, Chan and Rahmati [5] provided an approach to approximate the *minimum closest pair distance* and *minimum nearest neighbor distances* over time. For any constant $\varepsilon > 0$, their approach computes a $(1+\varepsilon)$-factor approximation to the minimum closest pair distance in time $\tilde{O}(n^{5/3})$. The notation $\tilde{O}$ hides polylogarithmic factors. Assuming $n \le m \le n^5$, their approach builds a data structure, which uses $\tilde{O}(m)$ preprocessing time and space, for answering queries: For any linearly moving query point q, their structure computes in time $\tilde{O}(\frac{n}{m^{1/5}})$ a $(1+\varepsilon)$-factor approximation to the minimum nearest neighbor distance to q over time.

Gao *et al.* [8] presented a *randomized* algorithm to maintain a clustering of moving points in $\mathbb{R}^2$, where each cluster can be covered by a unit square such that the centers of the squares are located at the points of P. The number of squares in their approach is on the order of $10^6\cdot|\mathcal{S}|$. Their KDS uses $O(n\log n\log\log n)$ space, and processes $O(n^2\log\log n)$ events, each in expected time $O(\log^{3.6} n)$. The locality of their KDS is $O(\log\log n)$. They proved that the number of changes of the optimal covering is $\Theta(n^3)$, and any approximate covering with constant factor undergoes $\Omega(n^2)$ changes.

Hershberger [10] gave a *deterministic* solution to the kinetic clustering problem in fixed dimension d in the L_∞ metric, where the number of axis-aligned boxes is at most $3^d\cdot|\mathcal{S}|$.

Table 2 The previous (and new) kinetic results for maintenance of all nearest neighbors (NNs) and all ε-nearest neighbors (ε-NNs).

attribute	dim.	space	#events	proc. time	local
all NNs [1]	d	$O(n \log^d n)$	$O(n^2 \beta_s(n) \log^{d+1} n)$	$O(n^2 \beta_s(n) \log^{d+2} n)$	No
all NNs [14]	2	$O(n)$	$O(n^2 \beta_s^2(n) \log n)$	$O(n^2 \beta_s^2(n) \log^2 n)$	No
all NNs [13]	d	$O(n \log^d n)$	$O(n^2 \beta_s^2(n) \log n)$	$O(n^2 \beta_s(n) \log^{d+1} n)$	No
all ε-NNs [13]	d	$O(n \log^d n)$	$O(n^2 \log^d n)$	$O(\log^d n \log\log n)$ /event	Yes
all ε-NNs [here]	d	$O(n \log \Delta')$	$O((n^2 \beta_s(\log \Delta')) \cdot (\log \Delta' \log\log \Delta'))$	$O(\log^2 \log \Delta')$ /event	Yes

His KDS uses linear space, and processes $O(n^2)$ events, each in $O(\log^2 n)$ time. The locality of the KDS is $O(\log n)$. His approach uses a dimensional reduction technique: It partitions the points into 1-dimensional clusters, covered by strips (of width at most one) perpendicular to x_1-axis, then partitions the points *in each of these clusters* into 2-dimensional clusters, covered by strips (of width at most one) perpendicular to x_2-axis, and so on. Each event at one level of this hierarchy creates $O(1)$ dynamic changes to the clusters at next level of the hierarchy. Handling an event in his approach requires dynamic maintenance, which in fact involves checking many *complicated* cases. For each strip at each level of the hierarchy, his approach uses two *dynamic and kinetic tournament trees* to track the leftmost point and rightmost point of the strip. He posed the problem of providing a smooth kinetic maintenance for clustering on P *without dimension reduction*.

Main contributions

For a set P of n moving points, in fixed dimension d, where the trajectory of each point is a polynomial of degree bounded by some constant $\bar{s}$, we provide clustering-based solutions to the kinetic closest pair decision problem and kinetic closest pair problem. Our kinetic clustering approach in $\mathbb{R}^d$ uses the kinetic 1-dimensional clustering by Hershberger.

Given a parameter r, we present a KDS for deciding in time $O(1)$ whether the closest pair distance is less than or equal to r. This KDS uses $O(n)$ space and processes $O(n^2)$ events, each in $O(1)$ time. The KDS can support insertions and deletions of points, where each operation can be performed in worst-case time $O(\log n)$.

To solve the optimization problem of maintaining the closest pair, we assume the closest pair distances is between 1 and Δ. This assumption is related to the assumption that the ratio between the maximum closest pair distance and the minimum closest pair distance is bounded by some parameter Δ. In many applications, the maximum closest pair distance over time is small, which makes our assumption and results reasonable. However, we can use our kinetic solution for the closest pair decision problem to detect if the closest pair distance is less than 1 or greater than Δ.

Our KDS for maintenance of the closest pair in $\mathbb{R}^d$ uses $O(n \log \Delta)$ space and processes $O(n^2 \log \Delta \log n \beta_s(n \log \Delta) + n^2 \log \Delta \log\log \Delta \beta_s(n \log \Delta))$ events, each in time $O(\log^2 n + \log^2 \log \Delta)$. We can dynamize our closest pair KDS such that each insertion or deletion takes *worst-case* time $O(\log \Delta \log^2 n + \log \Delta \log^2 \log \Delta)$. Note that, the space and processing time of each event in both previous closest pair KDSs by Basch *et al.* [3] and Agarwal *et al.* [1] are $O(n \log^{d-1} n)$ and $O(\log^d n)$, respectively, and also each insertion or deletion in the closest pair KDS by Agarwal *et al.* takes *amortized* time $O(\log^{d+1} n)$, where the number of `log`s is dependent on dimension d, whereas in our KDS it does not grow with d.

In addition, assuming the nearest neighbor distance to each point is between 1 and Δ', we provide a KDS (similar to that of the closest pair) for maintenance of all the ε-nearest neigh-

bors in $\mathbb{R}^d$. This KDS uses $O(n \log \Delta')$ space and handles $O(n^2 \log \Delta' \log\log \Delta' \beta_s(\log \Delta'))$ events, each in *worst-case* time $O(\log^2 \log \Delta')$. The locality of the KDS is $O(\log\log \Delta')$. Our KDS for all ε-nearest neighbors supports insertions and deletions of points, where each operation takes *worst-case* time $O(\log \Delta' \log n + \log \Delta' \log^2 \log \Delta')$. In the previous KDS for maintenance of all the exact nearest neighbors by Agarwal *et al.* [1], an insertion or deletion takes *expected* time $O(n)$.

Tables 1 and 2 give the comparisons between our results and the previous results.

Our approach is based on a clustering on moving points. We use the 1-dimensional clustering KDS by Hershberger to provide a d-dimensional clustering KDS. Our KDS uses $O(n)$ space and processes $O(n^2)$ events, each in $O(1)$ time. Each point participates in $O(1)$ certificates. At any time, each cluster can be covered by a d-dimensional axis-aligned box of maximum side-length one, and the number of boxes is $|S| \leq 3^d \cdot |\mathcal{S}|$. For the $\mathbb{R}^d$ case, our approach is simpler than the approach by Hershberger: We in fact do the future work stated in his paper; we solve the problem *without dimension reduction*, which is a need to his KDS. Our KDS uses d *kinetic sorted lists*, but his KDS uses (order of) $2 \cdot d \cdot 3^d \cdot |\mathcal{S}|$ dynamic and kinetic tournament trees. Also, we obtain improvements on his KDS: Processing an event in our KDS takes constant time, but the processing time in his KDS is $O(\log^2 n)$. The locality of our KDS is $O(1)$, but it is $O(\log n)$ in his KDS.

2 Kinetic Clustering

Section 2.1 provides a kinetic 1-dimensional clustering for a set P of moving points in $\mathbb{R}^d$, where the trajectory of each point is a polynomial of degree bounded by some constant. In Section 2.2, we give a simple generalization that allows us to maintain a d-dimensional clustering, where each cluster can be covered by a d-dimensional axis-aligned box of maximum side-length one.

2.1 The 1-d Case

Hershberger [10] provided a kinetic approach for clustering a set P of moving points by strips, perpendicular to x-axis, of width at most one. We call an *x-cluster* the set of points in P covered by some strip B. Denote by $lpt(C)/rpt(C)$ the leftmost/rightmost point of the x-cluster C. The *diameter* of an x-cluster C is $x(rpt(C)) - x(lpt(C))$. Let $lb(C)/rb(C)$ denote the x-coordinate of the left/right boundary of B, the strip corresponding to C. Let C_ℓ/C_r denote the next x-cluster on the left/right side of C.

Hershberger's kinetic approach uses three types of x-clusters (*right set*, *left set*, and *gap set*) with the following properties to obtain a smooth kinetic maintenance of x-clusters.

$$(lb(C), rb(C)) = \begin{cases} (x(lpt(C)), x(lpt(C)) + 1) & \text{when } C \text{ is a right set,} \\ (lb(C_r) - 1, lb(C_r)) & \text{when } C \text{ is a left set,} \\ (x(lpt(C)), x(rpt(C))) & \text{when } C \text{ is a gap set.} \end{cases}$$

His approach maintains the following invariants over time, where each of them can be considered as a KDS certificate, called an *invariant certificate*. An invariant certificate fails when the distance between two points is zero, one, or two.

(I1) If $p \in C$, then either $lb(C) \leq x(p) < rb(C)$, or $x(p) = rb(C)$ and C is a gap set.
(I2) For all C, $rb(C) \leq lb(C_r)$.
(I3) If C is a gap (resp. left) set, then C_r is not a gap (resp. left) set.
(I4) If C is a gap set, then $lb(C_r) - lb(C) < 1$.
(I5) If C is a gap set, and C_r and C_ℓ are right sets, then $lb(C_r) - rb(C_\ell) < 1$.

▶ **Lemma 1** ([10]). *Each point in P participates in $O(1)$ invariant certificates. When an invariant fails, the corresponding certificates can be updated in $O(1)$ time, by a constant number of x-cluster type changes, point transfers between the x-clusters, and singleton x-cluster creations. The number of x-clusters, at any time, is within a factor of 3 of the minimum possible by the optimal covering.*

Let L be a *kinetic sorted list* of the points in P, in increasing order according to their x-coordinates. For any two consecutive points in L, an *ordering certificate* is defined that attests the order of the two points along the x-axis; an ordering event occurs when two consecutive points in L exchange their order. We use the kinetic sorted list L to maintain $lpt(C)$ and $rpt(C)$ of all the x-clusters C.[1]

▶ **Lemma 2.** *Each point participates in two ordering certificates. When an ordering event occurs, the corresponding certificates can be updated in $O(1)$ time.*

Note that, for some x-cluster C, an update to $lpt(C)/rpt(C)$ implies $O(1)$ updates to the invariant certificates. Also note that updating some invariant certificate may create $O(1)$ changes to $lpt(\cdot)/rpt(\cdot)$ of the x-clusters. From this, together with Lemmas 1 and 2, we conclude:

▶ **Lemma 3.** *There exists a KDS that maintains a set S of x-clusters, such that each x-cluster can be covered by a strip of width at most one, where $|S| \leq 3 \cdot |\mathcal{S}|$. The KDS uses $O(n)$ space and handles $O(n^2)$ events, each in $O(1)$ time. The locality of the KDS is $O(1)$.*

2.2 The General Case: Any Fixed d

Denote the d coordinate axes by x_j, $j = 1, \ldots, d$. Hershberger's approach uses a dimension reduction approach: It first creates the x_1-clusters, then for each x_1-cluster it creates the x_2-clusters, and so on. This approach needs to extend the smooth maintenance of Lemma 1 to support insertions and deletions. The dynamic maintenance of his approach considers many complicated cases to update the clusters; each event at one level of the hierarchy creates dynamic changes to the clusters at next level of the hierarchy. Here, we show how simply we can maintain a set S of d-dimensional clusters on a point set P, without the dimension reduction and without the need of dynamic maintenance used by Hershberger.

Notation

Denote by $i_1, \ldots, i_d$ the indices that we use to refer to the strips perpendicular to the $x_1, \ldots, x_d$-axes, respectively. Let $B(i_j)$ denote some strip perpendicular to the x_j-axis, and let $C(i_j) = P \cap B(i_j)$ be the corresponding x_j-cluster for $B(i_j)$. Denote by $C(i_j + k)$ (resp. $C(i_j - k)$) the kth x_j-cluster right after (resp. before) the x_j-cluster $C(i_j)$. Let $B(i_1, \ldots, i_d)$ denote the d-dimensional axis-aligned box which is formed by the intersection of the strips $B(i_1), \ldots, B(i_d)$; let $C(i_1, \ldots, i_d) = P \cap B(i_1, \ldots, i_d)$. We denote by S the set of *non-empty* clusters $C(i_1, \ldots, i_d)$, for all $i_1, \ldots, i_d$. Figure 1 shows the strips of x_1-clusters and x_2-clusters of a set of points in $\mathbb{R}^2$; the nine non-empty boxes give a covering of the point set.

By application of Lemma 3, corresponding to each x_j-axis, we maintain a set of x_j-clusters. When an event associated with some x_j-cluster occurs, $O(1)$ points transfer between

[1] The KDS of Hershberger uses two *kinetic tournament trees* to maintain $lpt(C)$ and $rpt(C)$ for each cluster C. Thus his KDS includes a set of tournament certificates, where each point participates in $O(\log n)$ such certificates.

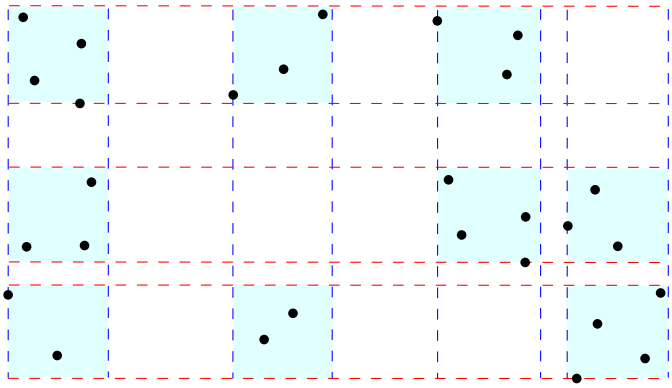


Figure 1 A set of 2-dimensional clusters of a point set in $\mathbb{R}^2$.

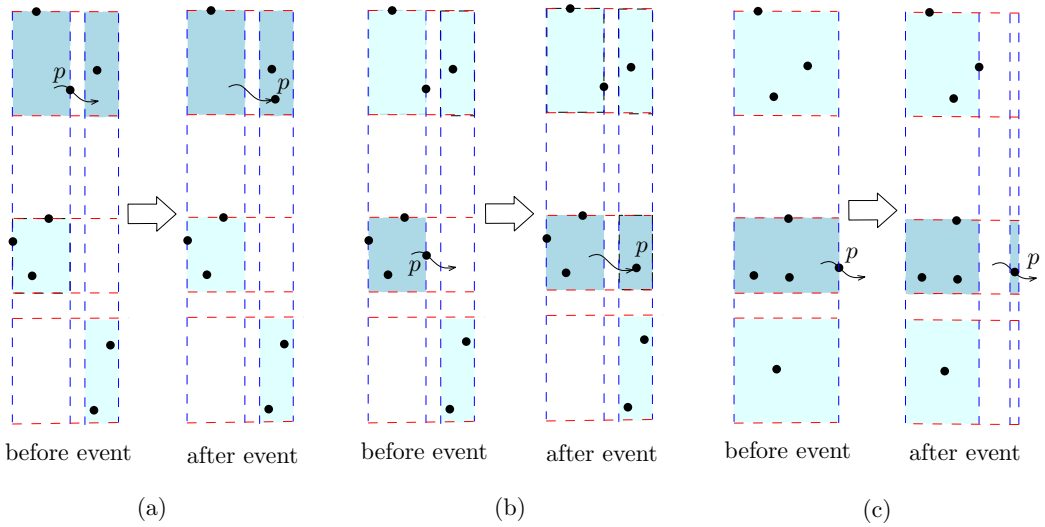


Figure 2 Updating the 2-dimensional clusters. (a) and (b) When p moves to an existing x_1-cluster. (c) When p moves to a new x_1-cluster.

the x_j-clusters, and $O(1)$ singleton x_j-clusters are created (from Lemma 1). Fix some $j \in \{1, \ldots, d\}$. Assume p is in some x_j-cluster $C(i_j)$, before an event. We update the set S of the d-dimensional clusters as follows.

- If p moves to an *existing* x_j-cluster, after the event: We delete p from the previous d-dimensional cluster and add to an existing/new d-dimensional cluster. For example, in Figures 2(a), we add p to an *existing* 2-dimensional cluster; in Figure 2(b), we create a *new* singleton 2-dimensional cluster for p.
- If p moves to a *new* x_j-cluster, after the event: We delete p from the previous d-dimensional cluster and add to a new singleton d-dimensional cluster; see Figure 2(c).

Consider $B(i_1, \ldots, i_j, \ldots, i_d)$, the axis-aligned box of $C(i_1, \ldots, i_j, \ldots, i_d)$, which contains p. (For simplicity, we use the notation C and B instead of $C(i_1, \ldots, i_j, \ldots, i_d)$ and $B(i_1, \ldots, i_j, \ldots, i_d)$, respectively.) The left/right boundary of B along the x_j-axis follows the left/right boundary of the strip $B(i_j)$ of $C(i_j)$, where $p \in C(i_j)$; *i.e.*, $lb(C) = lb(C(i_j))$ and $rb(C) = rb(C(i_j))$. Note that the left and right boundaries of B along other x_ℓ-axes

($\ell \neq j$) are the same as those of the box to which p belonged before the event. Also note that we delete a cluster C when its cardinality becomes equal to zero.

From the above discussion, there are $O(1)$ creations of new d-dimensional clusters of constant size in S, and $O(1)$ point insertions into or deletions from the clusters of S. Therefore, together with Lemma 3, we obtain:

▶ **Theorem 4.** *For a set P of moving points in fixed dimension d, where the trajectory of each point is a polynomial of degree bounded by some constant, there exists a KDS that maintains a set S of d-dimensional clusters, such that each cluster can be covered by an axis-aligned box of maximum side-length one, where $|S| \leq 3^d \cdot |\mathcal{S}|$. The KDS uses $O(n)$ space and handles $O(n^2)$ events, each in $O(1)$ time* (plus $O(\log n)$ time to update the priority queue)*. The locality of the KDS is $O(1)$.*

3 KDS for Closest Pair

In Section 3.1, we first build a KDS to solve the *closest pair* decision problem. Then, in Section 3.2, we solve the optimization problem maintaining the closest pair over time. Finally, we dynamize the KDSs in Section 3.3.

3.1 Kinetic Closest Pair Decision Problem

Consider the following decision problem:

▶ **Decision Problem 1.** *Given a parameter r, determine at any time whether the closest pair distance is less than or equal to r.*

By application of Theorem 4, build a kinetic data structure $\mathcal{D}(r)$ for maintaining a set S of clusters on the moving points in P in $\mathbb{R}^d$, such that the maximum side-length of the axis-aligned boxes corresponding to the clusters is $r/\sqrt{d}$. Let $C'(i_j)$ be some x_j-cluster in the neighborhood of the x_j-cluster $C(i_j)$. We call $C'(i_1, \ldots, i_d)$ (or C' for short) a *neighbor* cluster to $C(i_1, \ldots, i_d)$ (or C for short) if $C'(i_j)$ is between $C(i_j - \lceil 2\sqrt{d} \rceil - 1)$ and $C(i_j + \lceil 2\sqrt{d} \rceil + 1)$ for all j, $1 \leq j \leq d$, *i.e.*,

▶ **Condition 1.** $C'(i_j) \in \{C(i_j - \lceil 2\sqrt{d} \rceil - 1), \ldots, C(i_j + \lceil 2\sqrt{d} \rceil + 1)\}$*, for all j ($1 \leq j \leq d$).*

If there exist two points of P in the same cluster $C \in S$, then the closest pair distance is less than or equal to r. Otherwise, for each singleton cluster $C = \{p\}$, we need to find the points q in the neighborhood, and check the possible candidate pairs (p, q) for the closest pair. In other words:

▶ **Lemma 5.** *The answer to Decision Problem 1 is yes iff the following disjunction is true:*

$$\left(\bigvee_{c} A_c \right) \vee \left(\bigvee_{c,c'} E_{c,c'} \right), \tag{1}$$

where

$$A_c = \begin{cases} true & \text{if } |C| \geq 2, \\ false & \text{if } |C| = 1, \end{cases} \qquad E_{c,c'} = \begin{cases} true & \text{if } d(p,q) \leq r, \text{ where } p \in C \text{ and } q \in C', \\ false & \text{if } d(p,q) > r, \text{ where } p \in C \text{ and } q \in C'. \end{cases}$$

Let κ be the number of true expressions among A_c and $E_{c,c'}$ in the disjunction of (1). We do the following updates during the changes to the clusters.

(U1) When a new cluster C is created, that in fact contains a single point, we define a new expression A_c with value false. Then we update the neighbors C'' of C' (neighbors of neighbors for C) as C might violate them, and define the corresponding edges (p, q) and expressions $E_{i',i''}$ between singleton clusters C' and C''. We set the value of $E_{i',i''}$ to true (resp. false) if $d(p, q) \leq r$ (resp. $d(p, q) > r$), where $C' = \{p\}$ and $C'' = \{q\}$.

(U2) When the cardinality of some $C \in S$ becomes equal to one, we find all the singleton clusters $C' \in S$ which satisfy Condition 1, and define the edges (p, q) and their corresponding expressions $E_{c,c'}$ with a valid value true/false, where $C_i = \{p\}$ and $C' = \{q\}$.

(U3) When the cardinality of some cluster C becomes bigger than one, the value of A_c becomes true, which implies that the disjunction of (1) is true.

We can easily track the value of κ over time: We increase (resp. decrease) κ by one if the cardinality of some $C \in S$ gets > 1 (resp. $= 1$). Also, we increase (resp. decrease) κ by one if $d(p, q) \leq r$ (resp. $d(p, q) > r$), where $C = \{p\}$ and $C' = \{q\}$; we can define a boolean function for each edge (p, q) attesting its length is less than or equal to r. Note that when $|C|$ gets bigger than one, since we do not need to track the values of $E_{c,c'}$ for all neighbors C', we delete all the expressions $E_{c,c'}$ and the edges (p, q), and decrease κ by the number of edges (p, q) such that $d(p, q) \leq r$. Now, we can conclude:

▶ **Theorem 6.** *Let r be a positive real parameter. For a set P of moving points in fixed dimension d, where the trajectory of each point is a polynomial of degree bounded by some constant, there exists a KDS $\mathcal{D}(r)$ that decides in $O(1)$ time whether the closest pair distance is less than or equal to r. $\mathcal{D}(r)$ uses $O(n)$ space and handles $O(n^2)$ events, each in $O(1)$ time* (plus $O(\log n)$ time to update the priority queue). *The locality of $\mathcal{D}(r)$ is $O(1)$.*

Proof. By the invariant certificates (I1)-(I5), for each x_j-axis, $rb(C(i_j + \lceil 2\sqrt{d} \rceil + 1)) - lb(C(i_j + 1)) > r$, which implies that (if exists) we can find a pair (p, q) in our KDS such that $d(p, q) \leq r$.

Condition 1 of the definition of a *neighbor* cluster insures that we check only a constant number of neighbor clusters. Thus the updates (U1)-(U3) can be done in time $O(1)$. At any time, deciding whether $\kappa > 1$ is equivalent to deciding whether the disjunction of (1) is true. From this together with Theorem 4, the proof obtains. ◀

3.2 Kinetic Closest Pair Problem

Assume the Euclidean distance between any two points in P at any time is at least 1 and at most Δ. Let $r_\ell = 2^\ell$, $0 \leq \ell \leq \log \Delta$.

Fix some $\ell \in \{0, \ldots, \log \Delta\}$. In a similar way to that of Section 3.1, we build a kinetic data structure $\mathcal{D}(r_\ell)$. Let E_ℓ denote the set of edges (p, q) between the clusters $C = \{p\}$ and the neighbor clusters $C' = \{q\}$ satisfying Condition 1. Let e_ℓ be the edge with minimum length in E_ℓ. At any time, the edge with minimum length among all e_ℓ, $\ell = 0, \ldots, \log \Delta$, gives the closest pair, which can be maintained over time using a *dynamic and kinetic tournament tree* $\mathcal{T}$ over the $O(n \log \Delta)$ edges in $\cup_\ell E_\ell$. Next, we summarize the main result of this section.

▶ **Theorem 7.** *For a set P of moving points in fixed dimension d, where the trajectory of each point is a polynomial of degree bounded by some constant $\bar{s}$, our closest pair KDS uses $O(n \log \Delta)$ space and handles $O((n^2 \log \Delta \beta_s(n \log \Delta)) \cdot (\log n + \log \log \Delta))$ events, each in worst-case time $O(\log^2 n + \log^2 \log \Delta)$. Here, $s = 2\bar{s} + 2$. The total processing time of all events and the locality of the KDS are $O((n^2 \log \Delta \beta_s(n \log \Delta)) \cdot (\log^2 n + \log^2 \log \Delta))$ and $O(\log n + \log \log \Delta)$, respectively.*

Proof. By Theorem 3.1 of [1], for a sequence of m insertions/deletions into $\mathcal{T}$ whose maximum size at any time is $\tilde{n}$ ($m \geq \tilde{n}$), $\mathcal{T}$ handles $O(m\beta_{2\bar{s}+2}(\tilde{n}) \log \tilde{n})$ events. The total processing time for handling all the events is $O(m\beta_{2\bar{s}+2}(\tilde{n}) \log^2 \tilde{n})$, each event can be handled in worst-case time $O(\log^2 \tilde{n})$, and each point participates in $O(\log \tilde{n})$ tournament certificates.

From Theorem 6, and the fact that $\tilde{n} = |\cup_\ell E_\ell| = O(n \log \Delta)$ and the number of events is $m = O(n^2 \log \Delta)$ for all the levels ℓ, $0 \leq \ell \leq \log \Delta$, the proof obtains. ◀

3.3 Dynamizing the KDSs

Here, we present that how our KDSs in Sections 3.1 and 3.2 support insertions and deletions of points.

Hershberger showed that the smooth kinetic maintenance of x_j-clusters (Lemma 1) can support insertions and deletions of points: When a point p is inserted into or deleted from P, the invariant certificates (I1)-(I5) can be updated by a constant number of x_j-cluster type changes and point transfers between the x_j-clusters. In Section 2.1, we use kinetic sorted lists L_j on the point set P, in increasing order along the x_j-axes, in order to track the $lpt(\cdot)/rpt(\cdot)$ of the x_j-clusters. We dynamize the kinetic sorted lists L_j to support point insertions and deletions; each operation can be handled in time $O(\log n)$. This implies that our KDS (of Section 2.2) for maintenance of a set S of d-dimensional clusters can easily support insertions and deletions of points.

Given the dynamic and kinetic clustering S in $\mathbb{R}^d$, we can perform the updates (U1)-(U3), after each cluster change, to decide whether the closest pair distance is less than or equal to r; each update can be done in time $O(1)$. This implies:

▶ **Lemma 8.** *Our KDS $\mathcal{D}(r)$ of Theorem 6* (for deciding, at any time, whether the closest pair distance in $\mathbb{R}^d$ is less than or equal to r) *supports insertions and deletions of points. Each operation can be performed in worst-case time $O(\log n)$.*

Assume that the Euclidean distance between the *inserted point* q and any other point $p \in P$, at any time, is at least 1 and at most Δ. When q is inserted into (resp. deleted from) P, we insert q into (resp. delete q from) the $\log \Delta + 1$ levels of our closest pair KDS of Section 3.2. Since we can dynamize each $\mathcal{D}(r_\ell)$, $0 \leq \ell \leq \log \Delta$ (by Lemma 8), and each insertion into or deletion from $\mathcal{T}$ can be done in $O(\log^2(n \log \Delta))$, we obtain the following.

▶ **Lemma 9.** *Our KDS of Theorem 7* (for maintenance of the closest pair in $\mathbb{R}^d$) *supports insertions and deletions of points. Each operation can be performed in worst-case time $O(\log \Delta \log^2 n + \log \Delta \log^2 \log \Delta))$.*

4 KDS for All ε-Nearest Neighbors

Here, we first consider a decision version of the all ε-nearest neighbors problem, and then provide a KDS for maintenance of some ε-nearest neighbor to each point in P.

4.1 Kinetic All ε-Nearest Neighbors Decision Problem

Consider the following decision problem:

▶ **Decision Problem 2.** *Given parameters ε and r, (for each point $q \in P$) determine at any time whether there exists some point $p \in P$ such that its distance to q is less than or equal to $(1 + \varepsilon) \cdot r$.*

In a similar way to that of Section 3.1, build a kinetic data structure $\mathcal{D}(\varepsilon r)$ for maintaining a set S of clusters, such that the maximum side-length of the boxes corresponding to the clusters is $\varepsilon r/\sqrt{d}$. For each cluster $C \in S$, we maintain some point z_c in C as the *representative point* of C. The distance between z_c and any other point in C is at most εr.

Recall that $C(i_j + k)$ (resp. $C(i_j - k)$) denote the kth x_j-cluster after (resp. before) the x_j-cluster $C(i_j)$, along the x_j-axis. Here, we define a new condition for a *neighbor* cluster $C'(i_1, \ldots, i_d)$ to $C(i_1, \ldots, i_d)$. We say C' is the neighbor cluster of C if:

▶ **Condition 2.** *$C'(i_j) \in \{C(i_j - \lceil 2\sqrt{d}/\varepsilon \rceil - 1), \ldots, C(i_j + \lceil 2\sqrt{d}/\varepsilon \rceil + 1)\}$, for all x_j-axes, $j = 1, \ldots, d$.*

Fix some point $q \in P$, and assume $q \in C$. Let $E(q)$ be the set of edges $(q, z_{c'})$, where $z_{c'}$ are the representative points of the neighbor clusters C' satisfying Condition 2.

▶ **Lemma 10.** *The answer to Decision Problem 2 (for each $q \in P$) is yes iff the following disjunction is true:*

$$D(q) = A_c(q) \vee \left(\bigvee_{c'} E_{c,c'}(q) \right), \tag{2}$$

where (assuming $q \in C$)

$$A_c(q) = \begin{cases} true & \text{if } |C| \geq 2, \\ false & \text{if } |C| = 1, \end{cases} \qquad E_{c,c'}(q) = \begin{cases} true & \text{if } d(q, z_{c'}) \leq (1+\varepsilon) \cdot r, \\ false & \text{if } d(q, z_{c'}) > (1+\varepsilon) \cdot r. \end{cases}$$

Let $\mathcal{T}(q)$ be a dynamic and kinetic tournament tree over the edges in $E(q)$, which maintains the edge $e(q)$ with minimum length in $E(q)$. From Lemma 10, if $|C| \geq 2$, then the value of $D(q)$ would be true; otherwise, the value of $D(q)$ is the answer to whether $\|e(q)\| \leq (1+\varepsilon) \cdot r$. For each $q \in P$, we define $\mathcal{T}(q)$, and maintain the values of $D(q)$. We do the following updates to $D(q)$, during the changes to the clusters in S.

(U1) When p is deleted from some C such that $z_c = p$, we select a point v in $C - \{p\}$ as the new representative point. If after the event $C = \{v\}$, we first build $\mathcal{T}(v)$ to determine the value of $D(v)$. Then we find all the singleton neighbor clusters $C' = \{q\}$, and in $\mathcal{T}(q)$ we replace the edge (q, p) with (q, v). Note that, if after the event $C = \emptyset$, we update the neighbors C'' of neighbors C', for C; for the singleton neighbor clusters $C' = \{q\}$, we update $\mathcal{T}(q)$ if the neighbors C'' of C' change.

(U2) When a point p is inserted into some C, we ignore $\mathcal{T}(p)$ and set the value of $D(p)$ to true. Note that, if before the event C is a singleton cluster (say $C = \{q\}$), we also delete $\mathcal{T}(q)$ and set the value of $D(q)$ to true.

(U3) When a new cluster C is created, that contains a single point (say $C = \{p\}$), we build $\mathcal{T}(p)$. We then update the neighbors C'' of neighbors C', for C, and also for the singleton neighbor clusters $C' = \{q\}$, we apply the required changes to $\mathcal{T}(q)$ if the neighbors of C'' of C' change.

▶ **Lemma 11.** *Let ε and r be positive real parameters, and let P be a set of moving points in fixed dimension d, where the trajectory of each point is a polynomial of degree bounded by some constant. There exists a KDS that decides, for any point $q \in P$ at any time, in time $O(1)$ whether there is a point $p \in P$ such that $d(p, q) \leq (1+\varepsilon) \cdot r$. The KDS uses $O(n)$ space and handles $O(n^2)$ events, each in $O(1)$ time* (plus $O(\log n)$ time to update the priority queue)*. The locality of the KDS is $O(1)$.*

Proof. From the invariant certificates (I1)-(I5), for the x_j-axis, $rb(C(i_j + \lceil 2\sqrt{d}/\varepsilon \rceil + 1)) - lb(C(i_j + 1)) > (1 + \varepsilon) \cdot r$. This implies that, for each q, we can find a point p in a neighbor cluster such that $d(p, q) \leq (1 + \varepsilon) \cdot r$, if any such p exists.

Assuming ε and d are constants, Condition 2 implies that the number of neighbor clusters for each cluster in S is $O(1)$. Therefore, each of the updates ($\overline{\text{U}}1$)-($\overline{\text{U}}3$) can be done in time $O(1)$. The number of changes to the representative points, which is on the order of the number of changes to the clusters in S, is $O(n^2)$. This implies that the number of all events for all the constant size tournament trees $\mathcal{T}(q)$, for all $q \in P$, is $O(n^2)$. From this, together with the complexity of a $\mathcal{D}(\varepsilon r)$, the proof obtains. ◀

4.2 Kinetic All ε-Nearest Neighbors Problem

Assume the nearest neighbor distance to any point $q \in P$ is at least 1 and at most Δ'.

Let $r_\ell = 2^\ell$, $0 \leq \ell \leq \log \Delta'$. Fix some $\ell \in \{0, \ldots, \log \Delta'\}$. In a similar way to that of Section 4.1, we build $\mathcal{D}(\varepsilon r_\ell)$. Let $E_\ell(q)$ denote the set of edges $(q, z_{c'})$ between $C = \{q\}$ and its neighbor clusters C' satisfying Condition 2. We build a dynamic and kinetic tournament tree over the edges in $\cup_\ell E_\ell(q)$ to maintain the edge with minimum length in $\cup_\ell E_\ell(q)$, which in fact gives some ε-nearest neighbor to q.

▶ **Theorem 12.** *For a set P of moving points in fixed dimension d such that the trajectory of each point is a polynomial of degree bounded by some constant $\bar{s}$, our KDS for maintenance of all the ε-nearest neighbors uses $O(n \log \Delta')$ space and handles $O(n^2 \log \Delta' \beta_s(\log \Delta') \log\log \Delta')$ events, each in worst-case time $O(\log^2 \log \Delta')$. The total processing time for all the events and the locality of the KDS are $O(n^2 \log \Delta' \beta_s(\log \Delta') \log^2 \log \Delta')$ and $O(\log\log \Delta')$, respectively. Here, $s = 2\bar{s} + 2$.*

Proof. The proof is similar to the proof of Theorem 7. The dynamic and kinetic tournament tree corresponding to the point $q \in P$ handles $O(m_q \beta_{2\bar{s}+2}(\log \Delta') \log\log \Delta')$ events, each in worst-case time $O(\log^2 \log \Delta')$. Here, m_q is the number of insertions and deletions performed on the tournament tree of q. The total processing time of the events (associated with the tournament tree of q) and the locality are $O(m_q \beta_{2\bar{s}+2}(\log \Delta') \log^2 \log \Delta')$ and $O(\log\log \Delta')$, respectively. Since the number of insertions/deletions to all the tournament trees, for all the points in P, is $\sum_q m_q = O(n^2 \log \Delta')$, the proof obtains. ◀

▶ Remark. Our KDS of Theorem 12 *(for maintenance of all the ε-nearest neighbors in $\mathbb{R}^d$)* supports insertions and deletions of points. When a point q is inserted into (resp. deleted from) the point set P, we insert q into (resp. delete q from) the $\log \Delta' + 1$ levels of the kinetic data structures $\mathcal{D}(\varepsilon r_\ell)$. At each level, there exist $O(1)$ changes to the dynamic and kinetic sorted lists (where each one takes time $O(\log n)$; see Section 3.3), and $O(1)$ changes to the dynamic and kinetic tournament trees of the points (where each one takes time $O(\log^2 \log \Delta')$; see Theorem 12). Therefore, each operation can be performed in worst-case time $O(\log \Delta' \log n + \log \Delta' \log^2 \log \Delta')$.

▶ Remark. Our approach of Theorem 12 works to solve the bichromatic version of the problem. Given a set B of blue points and a set G of green points, for each green point $g \in G$, we want to maintain some blue point $b \in B$ as the bichromatic ε-nearest neighbor to g. Using a similar approach to that of Section 4.1, for each cluster C at each level ℓ ($0 \leq \ell \leq \log \Delta'$), we maintain a blue representative point. Then, for each green point g, where $g \in C$, we track some blue representative point in the neighbor clusters C' satisfying Condition 2.

References

1. Pankaj K. Agarwal, Haim Kaplan, and Micha Sharir. Kinetic and dynamic data structures for closest pair and all nearest neighbors. *ACM Transactions on Algorithms*, 5:4:1–37, 2008.
2. Julien Basch, Leonidas J. Guibas, and John Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31:1–19, 1999.
3. Julien Basch, Leonidas J. Guibas, and Li Zhang. Proximity problems on moving points. In *Proceedings of the 13th Annual Symposium on Computational Geometry (SoCG'97)*, pages 344–351, New York, NY, USA, 1997. ACM.
4. H. Brönnimann and M.T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete & Computational Geometry*, 14(1):463–479, 1995.
5. Timothy M. Chan and Zahed Rahmati. Approximating the minimum closest pair distance and nearest neighbor distances of linearly moving points. *Computational Geometry*, 2016.
6. Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC'88)*, pages 434–444, New York, NY, USA, 1988. ACM.
7. Robert J. Fowler, Michael S. Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information Processing Letters*, 12(3):133–137, 1981.
8. Jie Gao, Leonidas Guibas, John Hershberger, Li Zhang, and An Zhu. Discrete mobile centers. *Discrete & Computational Geometry*, 30(1):45–63, 2003.
9. Teofilo F. Gonzalez. Covering a set of points in multidimensional space. *Information Processing Letters*, 40(4):181–188, 1991.
10. John Hershberger. Smooth kinetic maintenance of clusters. *Computational Geometry*, 31(1–2):3–30, 2005.
11. Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32(1):130–136, 1985.
12. Zahed Rahmati. *Simple, Faster Kinetic Data Structures*. PhD thesis, University of Victoria, 2014.
13. Zahed Rahmati, Mohammad Ali Abam, Valerie King, and Sue Whitesides. Kinetic k-semi-Yao graph and its applications. *Computational Geometry*, 2016.
14. Zahed Rahmati, Mohammad Ali Abam, Valerie King, Sue Whitesides, and Alireza Zarei. A simple, faster method for kinetic proximity problems. *Computational Geometry*, 48(4):342–359, 2015.

Constrained Geodesic Centers of a Simple Polygon*

Eunjin Oh[1], Wanbin Son[2], and Hee-Kap Ahn[3]

1 **Department of Computer Science and Engineering, POSTECH, Pohang, Korea**
jin9082@postech.ac.kr
2 **Center for Geometry and Its Applications (GAIA), POSTECH, Pohang, Korea**
minibiny@gmail.com
3 **Department of Computer Science and Engineering, POSTECH, Pohang, Korea**
heekap@postech.ac.kr

Abstract

For any two points in a simple polygon P, the geodesic distance between them is the length of the shortest path contained in P that connects them. A geodesic center of a set S of sites (points) with respect to P is a point in P that minimizes the geodesic distance to its farthest site. In many realistic facility location problems, however, the facilities are constrained to lie in feasible regions. In this paper, we show how to compute the geodesic centers constrained to a set of line segments or simple polygonal regions contained in P. Our results provide substantial improvements over previous algorithms.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Geodesic distance, simple polygons, constrained center, facility location, farthest-point Voronoi diagram

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.29

1 Introduction

For a simple polygon P with n vertices in the plane, the geodesic path, denoted by $\pi(x, y)$, between any two points x and y in P is the shortest path between x and y contained in P, and the geodesic distance between x and y, denoted by $d(x, y)$, is the length of $\pi(x, y)$, that is, the sum of the Euclidean lengths of each segment in $\pi(x, y)$.

Let S be a set of k sites (points) in P. For any point x in P, a *geodesic farthest site* of x, denoted by $f_S(x)$, is a site of S that is farthest from x among all sites of S with respect to the geodesic distance. A point x in P that minimizes $d(x, f_S(x))$ among all points in P is called the *geodesic center* of S with respect to P. The geodesic center is unique and can be computed in $O(n + k)$ time if S consists of points on the boundary of P [1]. For S consisting of arbitrary points lying in P, the geodesic center can be computed in $O(n + k(\log n + \log k))$ time by constructing the geodesic convex hull $\mathsf{CH}_P(S)$ of S [13] and the geodesic center of $\mathsf{CH}_P(S)$.

For a subset Q of P, a geodesic center of S *constrained to* Q with respect to P is a point $q \in Q$ that minimizes $d(q, f_S(q))$ among all points in Q. Such a set Q is called a *constraint* or *feasible region* for facilities to be located in many realistic facility location problems. If the unconstrained geodesic center c coincides with a point $q \in Q$, then the geodesic center of

* This work was supported by the NRF grant 2011-0030044 (SRC-GAIA) funded by the government of Korea.

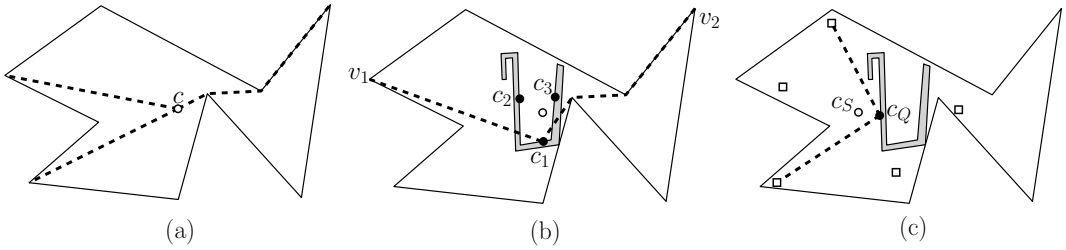


Figure 1 (a) The point c is the unconstrained geodesic center of the polygon. (b) The points c_1, c_2, and c_3 are the geodesic centers of the polygon constrained to the (gray) polygonal region. Here, $f(c_2)$ is v_2 and $f(c_3)$ is v_1, while c_1 has two farthest sites, v_1 and v_2. (c) For the sites (squares) lying in the polygon, c_S is the geodesic center of the sites and c_Q is the geodesic center constrained to the (gray) polygonal region.

S constrained to Q with respect to P is unique which is q. The geodesic center $c = q$ can be computed in $O(n + m + k(\log n + \log k))$ time, where m is the complexity of Q. If the unconstrained geodesic center c lies in $P \setminus Q$, a constrained geodesic center of S is a point q on the boundary of Q that minimizes $d(q, f_S(q))$ among all boundary points of Q. See Figure 1. Contrary to the unconstrained case, there might be more than one constrained geodesic centers in Q, but the geodesic distance from any constrained geodesic center q to its farthest point $f_S(q)$ is the same. We call the distance *the radius of the geodesic centers constrained to* Q and denote it by r_Q.

In this paper, we consider the problem of computing the geodesic centers of S with respect to a simple polygon P that are constrained to a subset Q of P consisting of line segments or simple polygonal regions.

Related works. Asano and Toussaint [3] studied the geodesic problem in which Q is the polygon P and S is the vertex set of P, and gave the first algorithm for computing the unconstrained geodesic center of P with n vertices which runs in $O(n^4 \log n)$ time. Afterwards, Pollack et al. [20] improved it to $O(n \log n)$ time. Finally, Ahn et al. [1] settled the problem by presenting a linear time algorithm for the problem.

To the extent of our knowledge, there is no known result for computing the geodesic center constrained to lie in a subset of P, except the one by Bose and Toussaint [7]. Their algorithm computes the geodesic center of P constrained to lie in a polygonal region $Q \subset P$ with m vertices in $O(n(n + \ell))$ time, where ℓ is the number of intersections of the geodesic farthest-point Voronoi diagram of the vertices of P with Q, and therefore $\ell = \Theta(nm)$ in the worst case.

The constrained center problem has been studied extensively under the Euclidean metric in the plane. Here P is the whole plane and S is a set of k points in the plane that we want to enclose. This problem is known as the constrained minimum enclosing circle problem or the constrained 1-center facility location problem. Megiddo [18] presented an $O(k)$-time algorithm for the problem in which the constraint Q is a line, and Hurtado et al. [15] presented an $O(k + m)$-time algorithm for the problem in which the constraint Q is a convex m-gon. Bose and Toussaint [7] considered the problem in which the center of the enclosing circle is constrained to lie in a simple polygon Q with m vertices and presented an $O((k+m) \log k + k \log m + \ell)$-time algorithm, where ℓ is the number of intersections of Q with the farthest-point Voronoi diagram of S. Later, Bose and Wang [8] removed the dependency on ℓ from the running time. Bose et al. [6] showed that the minimum enclosing circle whose center is constrained to lie on a query line segment can be reported in $O(\log k)$ time after

Table 1 Our results for constrained geodesic centers. $T(n,k) = O(n \log\log n + k\log(n+k))$ time [19] and $T(n,k) = \Omega(n+k)$. When S is the vertex set of P, $T(n,k) = O(n\log\log n)$ [19].

Constraints	Total running time
line segments	$O(m\log(n+k+m)) + T(n,k)$ or $O((m+k)\log(n+k+m) + mk\log n + n)$
disjoint line segments, disjoint polygonal regions	$O(m\log(n+k)) + T(n,k)$ or $O((m+k)\log(n+k) + mk\log n + n)$
geodesic convex polygon, disjoint geodesic pseudo polygons	$O(m) + T(n,k)$
disjoint polygonal regions with vertices on ∂P	$O(n+m+k\log(n+k))$

computing the farthest-point Voronoi diagram of S. Barba [4] presented an algorithm that reports the minimum enclosing circle with center on a given disk in $O(\log k)$ time after computing the farthest-point Voronoi diagram of S. Recently, Barba et al. [5] proposed algorithms that return the constrained center for constraint either a set of points or a set of segments in expected $\Theta((k+m)\log\min\{k,m\})$ time. For a constraint of a simple polygon Q, the expected running time becomes $\Theta(m + k\log k)$.

Our results. We begin with a set of (possibly crossing) line segments as the constraint Q and present an efficient algorithm that returns all constrained geodesic centers in $O(n+k+m\log(n+k+m))$ time after constructing the farthest-point geodesic Voronoi diagram FVD of S with respect to P in $T(n,k) = O(n\log\log n + k\log(n+k))$ time [19]. The algorithm also works in $O((m+k)\log(n+k+m) + mk\log n + n)$ time if we do not construct FVD of S.

Then we show that the running time can be improved slightly to $O(n+k+m\log(n+k)+T(n,k))$ if Q is a set of m disjoint line segments or disjoint polygonal regions with m vertices in total. When S is the vertex set of P, the running time becomes $O(n\log\log n + m\log n)$, which improves the $O(n(n+\ell))$-time result by Bose and Toussaint [7]. The algorithm also works in $O((m+k)\log(n+k) + mk\log n + n)$ time if we do not construct FVD of S.

Finally, we show that the running time can be further improved to linear if Q belongs to one of a few special polygon types. If Q is a geodesic convex polygon or a set of disjoint geodesic pseudo polygons (to be defined later) with m vertices in total, we can solve the problem in $O(n+m+k)$ time once FVD is computed. We can solve the problem in $O(n+m+k\log(n+k))$ time, for a set Q of disjoint polygonal regions with m vertices in total whose vertices are on the boundary of P without computing FVD. Our results are summarized in Table 1.

Recently we notice that the algorithm of Bose et al. [6] that computes the smallest enclosing circle whose center is constrained to lie on a line segment in the plane can be extended for the problem to find the geodesic centers constrained to line segments. By a similar argument of the algorithm of Bose et al., the geodesic centers of S constrained to Q with respect to P can be computed in $O(n+k+m\log(n+k))$, once FVD of S with respect to P is computed.

The algorithm of Bose et al. [6] relies on a property that the farthest-point Voronoi diagram of points in the plane is a tree, so it is hard to extend the algorithm for problems that do not satisfy this property. One representative example is the constrained weighted minimum enclosing circle problem. If we replace P with the whole plane and assign a positive weight for each point in S, the problem becomes the constrained weighted minimum enclosing circle problem in the plane. In this problem, the distance between two points $x \in P$ and

$y \in S$ is $w(y) \cdot d(x, y)$, where $w(y)$ is the weight of y. The algorithm of Bose et al. may not work since the farthest-point Voronoi diagram for weighted points is not necessarily a tree [17]. However our algorithms work for the constrained weighted minimum enclosing circle problem since most of the properties in this paper still hold for the problem.

2 Preliminaries

For a set A of points, let ∂A and $\mathrm{int}(A)$ denote the boundary and the interior of A, respectively. A subset A of a simple polygon P is *geodesically convex* if $\pi(x, y) \subseteq A$ for any two points $x, y \in A$. Let $\mathsf{hub}(r)$ be the set of points $x \in P$ such that $d(x, f_S(x))$ is at most r. Clearly, $\mathsf{hub}(r_S)$ with $r_S = \min_{p \in P} d(p, f_S(p))$ consists of a single point which is the geodesic center of S with respect to P. For any $r \geq \max_{p \in P} d(p, f_S(p))$, $\mathsf{hub}(r)$ is P itself.

▶ **Lemma 1.** *The set* $\mathsf{hub}(r)$ *is geodesically convex and monotone, that is,* $\mathsf{hub}(r') \subseteq \mathsf{hub}(r)$ *for any* $0 \leq r' \leq r$.

Proof. The monotonicity follows from the definition of $\mathsf{hub}(r)$ directly. For $0 < r < r_S$, $\mathsf{hub}(r) = \emptyset$. For $r \geq r_S$, $\mathsf{hub}(r)$ is the common intersection of the geodesic disks of radius r, each centered at a site of S. Since a geodesic disk is connected and geodesically convex, their nonempty common intersection is also geodesically convex. ◀

By the definition of constrained geodesic centers and Lemma 1, the lemma below holds.

▶ **Lemma 2.** *Every geodesic center constrained to* Q *lies on the boundary of* $\mathsf{hub}(r_Q)$*, where* r_Q *is the geodesic distance from any constrained geodesic center* c *to* $f_S(c)$*. Moreover, no point in* Q *lies in the interior of* $\mathsf{hub}(r_Q)$.

2.1 The Refined Farthest-point Geodesic Voronoi diagram

The *farthest-point geodesic Voronoi diagram* of S with respect to P is the subdivision of P such that each cell consists of the points with the common farthest site. Aronov et al. [2] showed that a farthest-point geodesic Voronoi diagram has linear complexity.

We consider a *refined* version of the farthest-point geodesic Voronoi diagram. Let C be a cell in the farthest-point geodesic Voronoi diagram FVD and let $f \in S$ be the common farthest site of the points in C. The shortest path map of f, which can be obtained by extending all edges of the shortest path tree [12], subdivides C into smaller cells, which we call *refined cells* of C (and of FVD). The *refined farthest-point geodesic Voronoi diagram* of FVD is the set $\mathrm{int}(P) \setminus \bigcup_{C \in \mathcal{C}} \mathrm{int}(C)$, where $\mathcal{C}$ is the collection of all refined cells of FVD. Here, a cell in the shortest path map of f contains at least one (hyperbolic or straight) arc of the boundary of C. This implies that the complexity of a refined farthest-point geodesic Voronoi diagram is still linear. Moreover, a refined cell has the following property, which comes directly from its definition.

▶ **Lemma 3.** *Every refined cell has exactly one boundary line segment that lies on the boundary of* P.

Given a simple polygon P with n vertices and a set S of k sites, Aronov et al. [2] gave an $O((n + k) \log(n + k))$ algorithm to compute the farthest-point geodesic Voronoi diagram of S with respect to P. Recently, Oh et al. [19] gave an $O(n \log \log n)$-time algorithm to compute the farthest-point geodesic Voronoi diagram for the special case that the sites are the vertices of P. The algorithm in [19] can be generalized to the case that S is a set of

arbitrary points in P. To this end, we first compute the geodesic convex hull of S with respect to P in $O(n + k \log(n + k))$ time [13]. Then we can compute the farthest-point geodesic Voronoi diagram inside the geodesic convex hull in $O((k + n) \log \log n)$ time by [19]. For the region outside the geodesic convex hull, we can apply a technique similar to [19] and compute the diagram in $O((k + n) \log \log n)$ time. Therefore, we can compute the diagram in total $O(n \log \log n + k \log(n + k))$ time.

Both algorithms in [2] and [19] can compute also the refined farthest-point geodesic Voronoi diagram without increasing their running times. In the following, we assume that we already have the refined farthest-point geodesic Voronoi diagram of S with respect to P.

3 Overlay of FVD and Curves in Geodesic Convex Position

In this section, we consider a simple constraint, *curves in geodesic convex position*. We say that curves are in *geodesic convex position* if the curves are contained in the boundary of the geodesic convex hull of themselves. We give a combinatorial property of the overlay between the farthest-point geodesic Voronoi diagram of sites with respect to P and curves in geodesic convex position. Specifically, we will show that the overlay has complexity linear to the number of sites and the complexity of the curves.

Since Euclidean farthest-point Voronoi diagrams have straight line segments as their arcs, the overlay of a diagram and curves in convex position has complexity linear to sum of their complexities - each line segment of the diagram intersects the curves at most twice and the complexity of the Euclidean Voronoi diagram is linear to the number of sites. However, a farthest-point geodesic Voronoi diagram defined in a simple polygon P might have hyperbolic arcs which intersect a convex curve contained in P more than a constant number of times, and therefore the argument for the Euclidean case does not work for the geodesic case.

To bound the complexity for the geodesic case, we consider a polygonal subdivision of P with respect to the diagram as follows. Let S be a set of sites contained in P and C be a refined cell of the geodesic farthest-point Voronoi diagram FVD of S. The boundary of C consists of (possibly empty) line segments and (possibly empty) hyperbolic arcs. Every point $x \in C$ has the same farthest neighbor $f_S(x)$ and has the same combinatorial structure of $\pi(f_S(x), x)$. Moreover, exactly one line segment of ∂C lies on ∂P by Lemma 3.

For a point $x \in C$, we call the last vertex that the path $\pi(f_S(x), x)$ from $f_S(x)$ goes through before x the *anchor of* x and denote it by $\mathsf{anchor}(x)$. For a hyperbolic arc α bounding C with endpoints a and b, let a' be the first point of ∂P hit the ray from a in the direction opposite to $\mathsf{anchor}(a)$ with respect to the site of C. See Figure 2(a). The point b' is defined analogously. We claim that the two line segments aa' and bb' subdivide C into at most three disjoint regions. To show this claim, we need the following lemma, which can be proved by the triangle inequality.

▶ **Lemma 4** ([19]). *Let x be a point in a refined cell C of a farthest-point geodesic Voronoi diagram of P. Then the line segment connecting x and y is contained in C, where y is the point on the boundary of P hit by the ray from $\mathsf{anchor}(x)$ towards x.*

The above lemma implies that aa' and bb' intersect ∂C only at their endpoints unless they are completely contained in ∂C; If there is another point $x \in \partial C$ in the interior of aa', aa' touches ∂C at x and there must be a point $x' \in \partial C \setminus \{a, a', x\}$ such that the segment $x'y'$ crosses ∂C at a point in the interior of $x'y'$, where y' is the point on the boundary of P hit by the ray from $\mathsf{anchor}(x')$ towards x'. Therefore, aa' and bb' subdivide C into at most three disjoint regions.

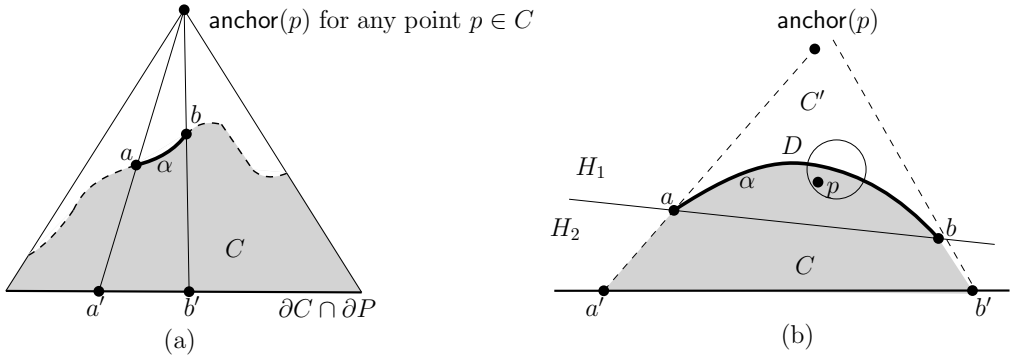


Figure 2 (a) A refined cell C is subdivided into at most three disjoint regions by aa' and bb'. (b) A hyperbolic arc α is incident to two refined cells, C and C'. If the anchor q of p lies on H_2, the ray from p in the direction opposite to q passes through α, which is a contradiction. Thus $\mathsf{anchor}(p)$ lies on H_1 and ab is contained in the region bounded by aa', bb' and α.

Let M be the subdivision of P with respect to FVD by introducing for each refined cell C of FVD, the line segments of ∂C and three line segments aa', bb', and ab for every hyperbolic arc α with endpoints a and b on ∂C. Each hyperbolic arc of ∂C is completely contained in a cell, and each cell of M contains at most one hyperbolic arc of ∂C as shown in the following lemma.

▶ **Lemma 5.** *Let α be a hyperbolic arc of a farthest-point geodesic Voronoi diagram* FVD*. The line segment connecting the two endpoints of α does not intersect in its interior any arc of* FVD *or ∂P.*

Proof. The arc α is incident to exactly two refined cells of FVD, say C and C'. Without loss of generality, we assume that α is locally convex with respect to C. Let a and b be the two endpoints of α. Let H_1 be the half plane that is bounded by the line through a and b and contains α, and let H_2 be the other half plane. In the following, we show that the anchor of a point x in α defined by the geodesic path from the site of C to x is contained in H_1, and the anchor of x defined by the geodesic path from the site of C' to x is contained in H_2.

There exists a disk D of sufficiently small radius such that $D \subset C \cup C'$ and α subdivides D into two pieces, one contained in the region R bounded by α and the segment ab, and the other contained in C'. See Figure 2(b). Let p be a point lying in the interior of $D \cap R$. Then, $\mathsf{anchor}(p)$ must be contained in H_1. If $\mathsf{anchor}(p)$ is in H_2, the ray from p in the direction opposite to $\mathsf{anchor}(p)$ intersects the interior of α, and then it intersects C', which contradicts to Lemma 4.

Then the region of C bounded by aa', bb', α, and $a'b'$ contains ab, where a' (and b') is the first point of ∂P hit by the ray from a (and from b) in the direction opposite to $\mathsf{anchor}(a)$ (and opposite to $\mathsf{anchor}(b)$) with respect to the site of C. Thus no arc of FVD intersects ab in its interior.

To show that ∂P does not intersect ab, we use Lemma 3 that exactly one line segment of ∂C lies on ∂P. If the line segment lying on $\partial P \cap \partial C$ intersects ab, then ∂P also intersects α. However, α is contained in P, which is a contradiction.

Therefore, ab does not intersect in its interior any arc of FVD or ∂P. ◀

Clearly, the subdivision M has complexity $O(k+n)$ because it is constructed by overlaying $O(k+n)$ line segments in the plane which are pairwise-disjoint in their interiors, where k the number of sites in S.

▶ **Lemma 6.** *The subdivision M consists of $O(n+k)$ cells.*

Now, consider curves on geodesic convex position with m vertices in total. Since M consists of $O(n+k)$ edges (line segments), the overlay of the curves and M has complexity $O(n+m+k)$. Since a hyperbolic arc α is contained in exactly one cell Δ of M and a hyperbolic arc intersects a line segment at most twice, α intersects the curves at most $2m_\Delta$ times, where m_Δ is the complexity of parts of the curves lying in Δ. Since the cells of M are disjoint each other, the total complexity of parts of the curves lying in Δ over all cells of M is $O(n+m+k)$, which implies that the curves intersects the hyperbolic arcs of FVD $O(n+m+k)$ times in total. This section is summarized as follows.

▶ **Lemma 7.** *The overlay of* FVD *and curves in geodesic convex position with m vertices has complexity $O(n+m+k)$.*

4 Geodesic Centers Constrained to Line Segments

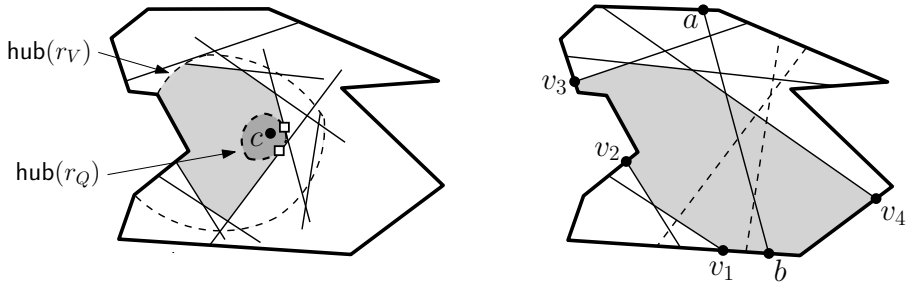
In the following, we assume that S is the vertex set of P. We will show how to handle the general case that S is a set of arbitrary points in P at the end of this section. We use $f(x)$ to denote the farthest site in the vertex set of P of x. In this section, we give an algorithm to compute the geodesic centers of P constrained to Q in the case that Q is a set of m (possibly crossing) line segments.

Once we have the overlay of Q and the farthest-point geodesic Voronoi diagram FVD of the vertices of P, we can compute the geodesic centers constrained to Q in time linear to the complexity of this overlay. Indeed, each line segment of Q is partitioned into smaller pieces in the overlay. We find the points c that minimize $d(c, f(c))$ in each smaller piece in constant time since each piece is contained in a refined cell of FVD. However, the number of these smaller pieces, that is, the complexity of the overlay might be quadratic.

Therefore, we avoid computing the overlay. Instead, we construct the cell Γ in the overlay of Q with $\mathsf{hub}(r)$ for a certain value r such that Γ contains the unconstrained geodesic center c. We will show that every geodesic center constrained to Q lies on the boundary of Γ. Once we find Γ, we can compute all geodesic centers constrained to Q in $O(n+m)$ time.

One might think that instead of computing a hub, considering the arrangement of Q inside P alone without overlaying it with $\mathsf{hub}(r)$ makes the algorithm simpler and easier. However, since Q is a set of line segments, the cell containing the unconstrained geodesic center in the arrangement of Q inside P has $O((n+m)\alpha(n+m))$ complexity [11], where $\alpha(n)$ is the inverse Ackermann function of n. Moreover, the best known algorithm for computing the cell takes $O((n+m)\alpha(n+m)\log(n+m))$ time [11]. Even worse, the cell is not necessarily convex, so the overlay of FVD and the boundary of the cell might be still quadratic.

Algorithm. Our algorithm works as follows. In the first step, we compute the farthest-point geodesic Voronoi diagram FVD of the vertices of P. Let Q_V be the set of the endpoints of the line segments in Q. For each $q \in Q_V$, we find the cell of FVD containing q. We preprocess FVD in $O(n)$ time to support an $O(\log n)$-time point-location query for a connected polygonal subdivision [9, 16]. In our case, some arcs of FVD might be hyperbolic while others are straight. To apply their point-location query structure to our case, we make use of the subdivision M of P with respect to FVD which we defined in Section 3. The subdivision M is a connected polygonal subdivision of $O(n)$ complexity. (See Lemma 6.) To find the cell of FVD containing a point q, we first find the cell of M containing q in $O(\log n)$ time. Recall that the interior of a cell of M intersects at most two cells of FVD. Thus, the cell containing



(a) (b)

Figure 3 (a) The points with squares are the geodesic centers constrained to Q. (b) The gray region is the intersection we computed before considering ab. The last chain in the gray region connecting v_3 and v_4 is intersected by ab. Dashed line segments are the line segments lying after ab.

a point $q \in Q_V$ can be found in $O(\log n)$ time and the point-location queries can be done in $O(m \log n)$ time for all points $q \in Q_V$.

Now, we have $f(q)$ for every $q \in Q_V$. Let r_V denote the minimum distance $d(q, f(q))$ among all points q of Q_V. Note that the combinatorial structure of $\pi(p, f(p))$ is the same for any point p in the same refined cell of FVD. Thus, we can compute $d(q, f(q))$ in constant time once we have the refined cell of FVD containing q. We compute r_V in $O(m)$ time.

By Lemma 2, hub(r_V) contains no point of Q_V in its interior. But it contains some points in Q on its boundary, thus we have $r_V \geq r_Q$. Consider the case that $r_V = r_Q$. Then, the points in Q lying on the boundary of hub(r_V) are the geodesic centers of P constrained to Q by Lemma 2. If $r_V > r_Q$, there are some line segments of Q that cross hub(r_V). Moreover, the geodesic centers constrained to Q are contained in such line segments. See Figure 3(a).

We compute hub(r_V). For each refined cell of FVD, we can compute part of hub(r_V) contained in the refined cell in time linear to the complexity of the refined cell, because we already have the farthest site and the anchor of the refined cell. This can be done in $O(n)$ time for all refined cells once we construct the refined cells of FVD.

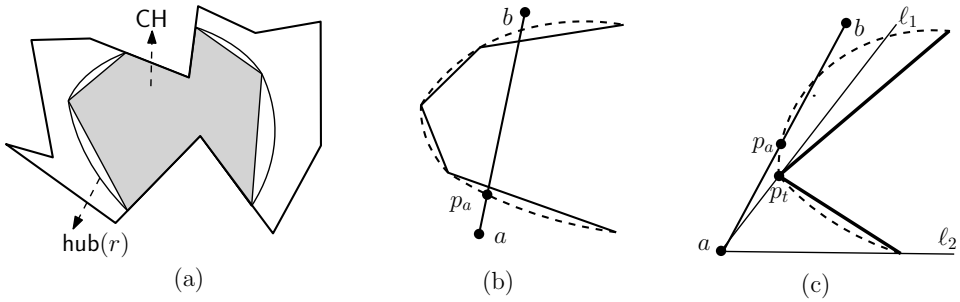
Then, for each line segment of Q, we check whether it crosses hub(r_V). If so, we additionally find the circular arcs of hub(r_V) crossed by the line segment. We do this for all line segments in $O(m \log n)$ time. The detailed procedure will be described in Section 4.1.

The line segments of Q crossing hub(r_V) subdivide hub(r_V) into $O(m^2)$ geodesic convex regions. Note that we do not need to construct the whole subdivision. We construct only the cell containing the unconstrained geodesic center c of P in the subdivision. This is because the geodesic centers constrained to Q are on the boundary of the cell containing c in this subdivision by Lemma 1 and 2, and the fact that hub(r) contains c for any $r \geq d(c, f(c))$. We find the cell Γ containing c in $O(n + m \log(n + m))$ time, which will be explained in Section 4.2.

Finally, we compute the overlay of the boundary of Γ and FVD in time linear to their total complexity. This can be done by traversing the boundary of Γ and the refined cells in FVD. Since Γ is geodesically convex, the complexity of the overlay is linear to their total complexity by Lemma 7. Then we can find the geodesic centers of P constrained to Q in the overlay in the same time.

4.1 Finding the Circular Arcs Intersecting a Line Segment

We are given hub(r) for some $r \in \mathbb{R}$ and a set Q of m line segments contained in P. Let c be the unconstrained geodesic center of P, which can be computed in $O(n)$ time [1]. In this



(a) (b) (c)

Figure 4 (a) The gray region is the geodesic convex hull of the endpoints of the circular arcs of the hub. (b) If ab crosses the convex chain, it crosses also the boundary of the hub. (c) If ab crosses the hub but does not cross the convex chain, then ab crosses the arc one of whose endpoint is p_t, where p_t is the point where a line passing through a is tangent to the convex chain.

section, we compute the intersection points of $\mathsf{hub}(r)$ and line segments in Q.

The boundary of $\mathsf{hub}(r)$ consists of (possibly empty) circular arcs and (possibly empty) polygonal chains which are from the boundary of P. Let ab be a line segment contained in P. Since $\mathsf{hub}(r)$ is geodesically convex by Lemma 1, ab intersects at most two circular arcs of $\mathsf{hub}(r)$. Moreover, one intersection point is closer to a than b, and the other one is closer to b than a. We first show how to compute the intersection point closer to a. The other intersection point can be computed analogously.

Let p_a be the intersection point closer to a. Consider the geodesic convex hull CH of the endpoints of circular arcs of $\mathsf{hub}(r)$. See Figure 4(a). Since we have the boundary of $\mathsf{hub}(r)$, we can compute CH in $O(n)$ time. We find the connected component R of $P \setminus \mathsf{CH}$ containing a in $O(\log n)$ time [16]. The connected region R contains a convex chain H of CH on its boundary. If the ray from a towards b hits H at some point in an edge e of H, then p_a is contained in the circular arc of $\mathsf{hub}(r)$ whose endpoints are the endpoints of e. See Figure 4(b). Thus, we can compute p_a in $O(\log n)$ time.

However, it is possible that p_a exists but the ray from a towards b does not hit H. See Figure 4(c). In this case, we consider the two lines ℓ_1 and ℓ_2 passing through a and tangent to H, which can be computed in $O(\log n)$ time. The point p_a lies in a circular arc of $\mathsf{hub}(r)$ one of whose endpoints is a point where ℓ_1 or ℓ_2 is tangent to H. Thus, in any case, we can compute p_a in $O(\log n)$ time.

▶ **Lemma 8.** *Given* $\mathsf{hub}(r)$ *with* $r \in \mathbb{R}$ *and a line segment* ab *contained in* P*, the circular arcs of* $\mathsf{hub}(r)$ *intersected by* ab *can be computed in* $O(\log n)$ *time after linear-time preprocessing for* $\mathsf{hub}(r)$*.*

4.2 Finding the Cell Containing the Geodesic Center

Let Q be a set of m line segments whose endpoints lie on the boundary of $\mathsf{hub}(r)$. In this section, we compute the cell Γ in the arrangement of Q inside $\mathsf{hub}(r)$ containing the unconstrained geodesic center c in $O(n + m\log(n + m))$ time.

For each line segment in Q, we extend the line segment in both directions until the two endpoints hit the boundary of P in $O(\log n)$ time [10]. Then a line segment ℓ in Q partitions P into two subpolygons one of which contains c. Let ℓ^+ be the subpolygon bounded by ℓ and containing c. We first compute the intersection I of all subpolygons ℓ^+ for all line segments ℓ in $O(m \log m)$ time as follows.

We sort the line segments in Q by the order of their first endpoints along the boundary of P, and then handle them one by one in order as follows. Initially we set P to I. While we

handle the line segments, we update I to the intersection of ℓ^+ for all line segments ℓ which are handled so far. The intersection I is bounded by polygonal chains from ∂P and parts of line segments of Q. Moreover, parts of line segments of Q lying on ∂I form a number of convex chains. To maintain I, we store each convex chain using a binary search tree. The first line segment ℓ of Q subdivides I into two subpolygons, and we update I to the subpolygon containing c. Here, ℓ is the only one element stored in a binary search tree. As we handle more line segments, we create more binary search trees. For the next line segment ℓ', if both endpoints of ℓ' lies after the most clockwise point in the convex chain stored in the last binary search tree, we create a new binary search tree containing only one element ℓ'. See Figure 3(b). Otherwise, ℓ' may cross $\partial I \setminus \partial P$ in at most two points. To find this, it is sufficient to check the first and the last binary search trees. Thus, this takes $O(\log m)$ time.

By definition, Γ is the intersection of I and $\mathsf{hub}(r)$. So, we compute the intersection of I and $\mathsf{hub}(r)$ by traversing the boundary of I starting from a line segment on ∂I in clockwise order as follows. When we reach an endpoint of some line segment, we find the endpoint next to it. Then we connect these two endpoints by the boundary of $\mathsf{hub}(r)$. In this procedure, we traverse the boundary of I and the boundary of $\mathsf{hub}(r)$ once. Thus, we can compute Γ in $O(n + m\log(n + m))$ time.

▶ **Lemma 9.** *Given* $\mathsf{hub}(r)$ *and a set of* m *line segments crossing the hub, the cell containing the geodesic center of* P *in the arrangement of* $\mathsf{hub}(r)$ *and the line segments can be computed in* $O(n + m\log(n + m))$ *time.*

Until now, we assumed that S coincides with the vertex set of P. However, once the farthest-point geodesic Voronoi diagram of S is computed, the algorithm in this section works also for the case where the points of S are allowed to lie in the interior of P. The arguments in this section prove the following theorem.

▶ **Theorem 10.** *Let* P *be a simple* n*-gon and let* Q *be a set of* m *line segments the lie in* P*. For a set* S *of* k *sites (points) in* P*, the geodesic centers of* S *constrained to* Q *with respect to* P *can be computed in* $O(n + k + m\log(n + k + m))$ *time, once the farthest-point geodesic Voronoi diagram of* S *with respect to* P *is computed.*

For small m and k, the running time to compute FVD dominates the time complexity of our algorithm. The running time of our algorithm can be improved slightly for the case by avoiding to compute FVD explicitly. Recall that our algorithm uses FVD to compute r_V, $\mathsf{hub}(r_V)$ and the overlay of the boundary of Γ and FVD. We can compute them without constructing FVD of S as follows. The geodesic distance between two points can be computed in $O(\log n)$ time [13] after $O(n)$ preprocessing, so r_V can be computed in $O(mk\log n)$ time by finding $f(q)$ for all $q \in Q_V$. We compute $\mathsf{hub}(r_V)$ in $O(k)$ time, once the geodesic convex hull of S is computed in $O(n+k\log(n+k))$ time by applying a technique similar to Theorem 6 in [19] which shows how to compute FVD of points on the boundary of a simple k-gon in $O(k)$ time. The overlay of the boundary of Γ and FVD can also be computed similarly.

▶ **Theorem 11.** *Let* P *be a simple* n*-gon and let* Q *be a set of* m *line segments the lie in* P*. For a set* S *of* k *sites (points) in* P*, the geodesic centers of* S *constrained to* Q *with respect to* P *can be computed in* $O((m + k)\log(n + k + m) + n)$ *time.*

4.3 Geodesic Centers Constrained to Disjoint Line Segments

In this section, we give an algorithm to compute the geodesic centers of S with respect to P constrained to a set Q of m disjoint line segments. For ease of explanation, we assume that S is the vertex set of P.

Recall that once FVD is computed, the algorithm for the general case of crossing line segments takes $O(n + m \log n)$ time except the last step, which finds the cell Γ in the arrangement of Q and $\mathsf{hub}(r)$ containing the unconstrained geodesic center c. We show how to compute the cell Γ in $O(n + m \log n)$ time for the case of disjoint line segments, which improves the running time slightly to $O(n + m \log n)$.

We have $\mathsf{hub}(r)$ and a set of line segments crossing the hub. Moreover, we know the intersection points of the boundary of the hub and each line segment but they are not sorted. Instead of sorting the intersection points along the boundary of the hub, which takes $O(m \log m)$ time, we give an $O(n + m)$-time algorithm to compute the cell containing the geodesic center of P in the arrangement of the line segments and $\mathsf{hub}(r)$.

For a circular arc β of the hub, we have the line segments intersecting β. Without loss of generality, we assume that the two endpoints of β are on the x-axis. There are at most two line segments that contribute to the boundary of Γ among the line segments intersecting β: one is the line segment s_L that is closest geodesically to c among the line segments that are to the left of c, and the other is the line segment s_R that is closest geodesically to c among the line segments that are to the right of c. We find two line segments s_L and s_R, if they exist, for every circular arc β of the hub in $O(n + m)$ time. After doing this, we have $O(n)$ line segments which are sorted along the boundary of the hub, and we can compute the cell containing the geodesic center in $O(n)$ time.

▶ **Lemma 12.** *Given* $\mathsf{hub}(r)$ *and a set of* m *disjoint line segments, the cell containing the unconstrained geodesic center of* P *in the arrangement of the hub and the line segments can be computed in* $O(n + m \log n)$ *time.*

For the case that Q is a set of disjoint polygonal regions contained in P with m vertices in total, the geodesic centers of P constrained to Q lie on the boundary of Q unless they coincide with the unconstrained geodesic center of P. Thus we can use the algorithm in this section to compute the geodesic centers constrained to a set of disjoint polygonal regions.

▶ **Theorem 13.** *Let* P *be a simple* n*-gon and let* Q *be a set of* m *disjoint line segments or disjoint polygonal regions with* m *vertices in total that lie in* P*. For a set* S *of* k *sites (points) in* P*, the geodesic centers of* S *constrained to* Q *with respect to* P *can be computed in* $O(n + k + m \log(n + k))$ *time, once the farthest-point geodesic Voronoi diagram of* S *with respect to* P *is computed.*

The algorithm also works in $O((m + k) \log(n + k) + mk \log n + n)$ time without constructing FVD of S as similar to Theorem 11.

▶ **Theorem 14.** *Let* P *be a simple* n*-gon and let* Q *be a set of* m *disjoint line segments or disjoint polygonal regions with* m *vertices in total that lie in* P*. For a set* S *of* k *sites (points) in* P*, the geodesic centers of* S *constrained to* Q *with respect to* P *can be computed in* $O((m + k) \log(n + k) + mk \log n + n)$ *time.*

5 Geodesic Centers Constrained to a Polygon of Special Types

In this section, we consider a few special types of polygons. When Q is a geodesic convex polygon or a set of disjoint geodesic pseudo polygons, which will be defined, we can compute the constrained geodesic centers in linear time once we have the farthest-point geodesic Voronoi diagram of S with respect to P. In addition, when all the vertices of Q lie on ∂P, we can compute the constrained geodesic centers efficiently without computing a farthest point geodesic Voronoi diagram. We assume that S is the vertex set of P unless stated otherwise.

5.1 Geodesic Convex Polygons and Geodesic Pseudo Polygons

In this subsection, we assume that the farthest-point geodesic Voronoi diagram FVD of S with respect to P is already computed.

Let Q be a geodesic convex polygon. By Lemma 7, the complexity of the overlay of FVD and Q is linear to the complexity of FVD and Q. Thus, we compute the overlay of FVD and Q in linear time by traversing the cells of FVD and the edges of Q. Then, we choose the points which minimize the geodesic distance to their farthest sites in linear time.

We call a polygon contained in P a *geodesic pseudo polygon* if its boundary consists of (possibly empty) polygonal chains from ∂P and (possibly empty) concave chains lying in the interior of P. Let Q be a set of disjoint geodesic pseudo polygons contained in P. Note that the region of P lying outside of the polygons of Q may not be connected. If the unconstrained geodesic center of P is contained in Q, then it is also the unique geodesic center of P constrained to Q. Thus, we are done. Otherwise, we find the connected component R of the region of P lying outside of the polygons of Q containing the unconstrained geodesic center in linear time. Then, by Lemma 2 and the geodesic convexity of a hub, all constrained geodesic centers lie on the boundary of the concave chains of Q shared by R. Thus, we compute the overlay of FVD and the concave chains in linear time by Lemma 7 and return the answer.

▶ **Theorem 15.** *Let P be a simple n-gon and let Q be a geodesic convex polygon or a set of disjoint geodesic pseudo polygons with m vertices in total. For a set S of k sites (points) in P, the geodesic centers of S constrained to Q with respect to P can be computed in $O(n+m+k)$ time once the farthest-point geodesic Voronoi diagram of S with respect to P is computed.*

5.2 Polygons with Vertices on the boundary of P

In this subsection, we consider a set Q of disjoint polygonal regions whose vertices are on the boundary of P and show how to compute the geodesic centers constrained to Q efficiently without computing the whole FVD.

We assume that Q does not contain the unconstrained geodesic center. As we did in the previous case, we compute the connected component R of $P \setminus Q$ containing the unconstrained geodesic center in linear time. Then, we have the set Q' of the edges of the regions in Q that lie on ∂R. By Lemma 2 and the geodesic convexity of a hub, all constrained geodesic centers lie on line segments in Q'.

We compute the overlay of a line segment $\ell \in Q'$ and FVD as follows. The line segment ℓ subdivides P into two parts exactly one of which contains R. Let R' be the part of P which does not contain R. Let S_1 be the set of sites of S contained in R', and S_2 be the set of sites in S whose refined cells of FVD intersect the boundary of R' excluding ℓ. Then we consider the farthest-point geodesic Voronoi diagram of S_1 restricted to ℓ, which we denote by FVD_1, and the farthest-point geodesic Voronoi diagram of S_2 restricted to ℓ, which we denote by FVD_2. Once we have FVD_1 and FVD_2, we can compute the overlay of FVD and ℓ in time linear to the total complexity of FVD_1 and FVD_2.

If all sites are on the boundary of P, we can compute FVD_1 and FVD_2 in linear time for all line segments $\ell \in Q'$ [19]. Otherwise, we compute FVD_1 and FVD_2 in $O(n + k\log(n+k))$ time for all line segments $\ell \in Q'$ combining the results by [2, 14, 19].

▶ **Theorem 16.** *Let P be a simple n-gon and let Q be a set of disjoint polygonal regions with m vertices in total whose vertices are on the boundary of P. For a set S of k sites (points) in P, the geodesic centers of S constrained to Q with respect to P can be computed in $O(n + m + k\log(n+k))$ time.*

References

1 Hee-Kap Ahn, Luis Barba, Prosenjit Bose, Jean-Lou De Carufel, Matias Korman, and Eunjin Oh. A linear-time algorithm for the geodesic center of a simple polygon. In *Proc. 31st Int'l Symposium on Computational Geometry (SoCG 2015)*, pages 209–223, 2015.

2 Boris Aronov, Steven Fortune, and Gordon Wilfong. The furthest-site geodesic Voronoi diagram. *Discrete & Computational Geometry*, 9(1):217–255, 1993.

3 Tetsuo Asano and Godfried Toussaint. Computing the geodesic center of a simple polygon. Technical Report SOCS-85.32, McGill University, 1985.

4 Luis Barba. Disk constrained 1-center queries. In *Proc. 24th Canadian Conference on Computational Geometry (CCCG 2012)*, pages 15–19, 2012.

5 Luis Barba, Prosenjit Bose, and Stefan Langerman. Optimal algorithms for constrained 1-center problems. In *Proc. 11th Latin American Theoretical Informatics Symposium (LATIN 2014)*, pages 84–95, 2014.

6 Prosenjit Bose, Stefan Langerman, and Sasanka Roy. Smallest enclosing circle centered on a query line segment. In *Proc. 20th Canadian Conference on Computational Geometry (CCCG 2008)*, pages 167–170, 2008.

7 Prosenjit Bose and Godfried Toussaint. Computing the constrained Euclidean geodesic and link center of a simple polygon with applications. In *Proc. 14th Computer Graphics International (CGI 1996)*, pages 102–110, 1996.

8 Prosenjit Bose and Qingda Wang. Facility location constrained to a polygonal domain. In *Proc. 5th Latin American Theoretical Informatics Symp. (LATIN'02)*, pages 153–164, 2002.

9 Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(3):485–524, 1991.

10 Bernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas Guibas, John Hershberger, Micha Sharir, and Jack Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

11 Bernard Chazelle, Herbert Edelsbrunner, Leonidas Guibas, Micha Sharir, and Jack Snoeyink. Computing a face in an arrangement of line segments and related problems. *SIAM Journal on Computing*, 22(6):1286–1302, 1993.

12 Leonidas Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1):209–233, 1987.

13 Leonidas J. Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989.

14 John Hershberger and Subhash Suri. Matrix searching with the shortest-path metric. *SIAM Journal on Computing*, 26(6):1612–1634, 1997.

15 Ferran Hurtado, Vera Sacristán, and Godfried Toussaint. Some constrained minimax and maximin location problems. *Studies in Locational Analysis*, 15:17–35, 2000.

16 David Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

17 D.T. Lee and V.B. Wu. Multiplicative weighted farthest neighbor Voronoi diagrams in the plane. In *Proc. International Workshop on Discrete Mathematics and Algorithms*, pages 154–168, 1993.

18 Nimrod Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983.

19 Eunjin Oh, Luis Barba, and Hee-Kap Ahn. The farthest-point geodesic Voronoi diagram of points on the boundary of a simple polygon. To appear in Proc. 32nd International Symposium on Computational Geometry (SoCG 2016), 2016.

20 Richard Pollack, Micha Sharir, and Günter Rote. Computing the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 4(6):611–626, 1989.

Time-Space Trade-offs for Triangulating a Simple Polygon*

Boris Aronov[1], Matias Korman[2], Simon Pratt[3], André van Renssen[4], and Marcel Roeloffzen[5]

1 **Tandon School of Engineering, New York University, New York, USA**
boris.aronov@nyu.edu
2 **Tohoku University, Sendai, Japan**
mati@dais.is.tohoku.ac.jp
3 **Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada**
Simon.Pratt@uwaterloo.ca
4 **National Institute of Informatics (NII), Tokyo, Japan; and JST, ERATO, Kawarabayashi Large Graph Project, Tokyo, Japan**
andre@nii.ac.jp
5 **National Institute of Informatics (NII), Tokyo, Japan; and JST, ERATO, Kawarabayashi Large Graph Project, Tokyo, Japan**
marcel@nii.ac.jp

Abstract

An s-workspace algorithm is an algorithm that has read-only access to the values of the input, write-only access to the output, and only uses $O(s)$ additional words of space. We give a randomized s-workspace algorithm for triangulating a simple polygon P of n vertices, for any $s \in O(n)$. The algorithm runs in $O(n^2/s + n(\log s)\log^5(n/s))$ expected time using $O(s)$ variables, for any $s \in O(n)$. In particular, when $s \in O(\frac{n}{\log n \log^5 \log n})$ the algorithm runs in $O(n^2/s)$ expected time.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases simple polygon, triangulation, shortest path, time-space trade-off

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.30

1 Introduction

Triangulation of a simple polygon, often used as a preprocessing step in computer graphics, is performed in a wide range of settings including on embedded systems like the Raspberry Pi or mobile phones. Such systems often run read-only filesystems for security reasons and have very limited working memory. An ideal triangulation algorithm for such an environment would allow for a trade-off in performance in time versus working space.

Computer science and specifically the field of Algorithms generally has two optimization goals; running time and memory size. In the 70's there was a strong focus on algorithms that required low memory as it was expensive. As memory became cheaper and more widely available this focus shifted towards optimizing algorithms for their running time, with memory mainly as a secondary constraint.

* Work on this paper by B. A. was supported in part by NSF Grants CCF-11-17336 and CCF-12-18791. M. K. was supported in part by the ELC project (MEXT KAKENHI No. 12H00855 and 15H02665). S. P. was supported in part by the Ontario Graduate Scholarship and The Natural Sciences and Engineering Research Council of Canada.

Nowadays, even though memory is cheap, there are other constraints that limit memory usage. First, there is a vast number of embedded devices that operate on batteries and have to remain small, which means they simply cannot contain a large memory. Second, some data may be read-only, due to hardware constraints (i.e., DVD/CDs can be written only once), or concurrency issues (i.e., to allow many processes to access the database at once).

These memory constraints can all be described in a simple way by the so-called *constrained-workspace* model (see Section 2 for details). Our input is read-only and potentially much larger than our working space, and the output we produce must we written to write-only memory. More precisely, we assume we have a read-only dataset of size n and a working space of size $O(s)$, for some user-specified parameter s. In this model, the aim is to design an algorithm whose running time decreases as s grows. Such algorithms are called *time-space trade-off* algorithms [11].

Previous Work

Several models of computation that consider space constraints have been studied in the past (we refer the interested reader to [9] for an overview). In the following we discuss the results related to triangulations. The concept of memory-constrained algorithms attracted renewed attention within the computational geometry community by the work of Asano *et al.* [4]. One of the algorithms presented in [4] was for triangulating a set of n points in the plane in $O(n^2)$ time using $O(1)$ variables. More recently, Korman *et al.* [10] introduced two different time-space trade-off algorithms for triangulating a point set: the first one computes an arbitrary triangulation in $O(n^2/s + n(\log n)\log s)$ time using $O(s)$ variables. The second is a randomized algorithm that computes the Delaunay triangulation of the given point set in expected $O((n^2/s)\log s + n(\log s)\log^* s)$ time within the same space bounds.

The above results address triangulating discrete point sets in the plane. The first algorithm for triangulating simple polygons was due to Asano *et al.* [2] (in fact, the algorithm works for slightly more general inputs: plane straight-line graphs). It runs in $O(n^2)$ time using $O(1)$ variables. The first time-space trade-off for triangulating polygons was provided by Barba *et al.* [5]. In their work, they describe a general time-space trade-off algorithm that in particular could be used to triangulate monotone polygons. An even faster algorithm (still for monotone polygons) was afterwards found by Asano and Kirkpatrick [3]: $O(n \log_s n)$ time using $O(s)$ variables. Despite extensive research on the problem, there was no known time-space trade-off algorithm for general simple polygons. It is worth noting that no lower bounds on the time-space trade-off are known for this problem either.

Results

This paper is structured as follows: In Section 2 we define our model, as well as the problems we study. Our main result on triangulating a simple polygon P with n vertices using only a limited amount of memory can be found in Section 3. Our algorithm achieves expected running time of $O(n^2/s + n(\log s)\log^5(n/s))$ using $O(s)$ variables, for any $s \in \Omega(\log n) \cap O(n)$. Note that for most values of s (i.e., when $s \in O(\frac{n}{(\log n)\log^5 \log n})$) the algorithm runs in $O(n^2/s)$ expected time.

Our approach uses a recent result by Har-Peled [8] as a tool for subdividing P into smaller pieces and solving them recursively. A similar approach can be used for other problems. Indeed, in an extended version of this paper [1] we show how a similar approach can be used to compute the *shortest-path map* or *shortest-path tree* from any point $p \in P$, or simply to split P by $\Theta(s)$ pairwise disjoint diagonals into smaller subpolygons, each with $\Theta(n/s)$ vertices.

2 Preliminaries

In this paper, we utilize the *s-workspace* model of computation that is frequently used in the literature (see for example [2, 5, 6, 8]). In this model the input data is given in a read-only array or some similar structure. In our case, the input is a simple polygon P; let $v_1, v_2, \ldots, v_n$ be the vertices of P in clockwise order along the boundary of P. We assume that, given an index i, in constant time we can access the coordinates of the vertex v_i. We also assume that the usual word RAM operations (say, given i, j, k, finding the intersection point of the line passing through vertices v_i and v_j and the horizontal line passing through v_k) can be performed in constant time.

In addition to the read-only data, an s-workspace algorithm can use $O(s)$ variables during its execution, for some parameter s determined by the user. Implicit memory consumption (such as the stack space needed in recursive algorithms) must be taken into account when determining the size of a workspace. We assume that each variable or pointer is stored in a data word of $\Theta(\log n)$ bits. Thus, equivalently, we can say that an s-workspace algorithm uses $O(s \log n)$ bits of storage.

In this model we study the problem of computing a triangulation of a simple polygon P. A *triangulation* of P is a maximal crossing-free straight-line graph whose vertices are the vertices of P and whose edges lie inside P. Unless s is very large, the triangulation cannot be stored explicitly. Thus, the goal is to report a triangulation of P in a write-only data structure. Once an output value is reported it cannot be afterwards accessed or modified.

In other memory-constrained triangulation algorithms [2, 3] the output is reported as a list of edges in no particular order (with no information on neighboring edges or faces). Moreover, it is not clear how to modify these algorithms to obtain such information. Our approach has the advantage that, in addition to the list of edges, we can report adjacency information as well. For example, we could report the triangulation in a doubly connected edge list (or any other similar format). More details on how we can report the triangulation are given in Section 3.4.

A vertex of a polygon is *reflex* if its interior angle is larger than 180°. Given two points $p, q \in P$, the *geodesic* (or *shortest path*) between them is the path of minimum length that connects p and q and that stays within P (viewing P as a closed set). The length of that path is the *geodesic distance* from p to q. It is well known that, for any two points of P, their geodesic π always exists and is unique. Such a path is a polygonal chain whose vertices (other than p and q) are reflex vertices of P. Thus, we often identify π with the ordered sequence of reflex vertices traversed by the path from p to q. When that sequence is empty (i.e., the geodesic consists of the straight segment pq) we say that p *sees* q (and vice versa).

Our algorithm relies in a recent result by Har-Peled [8] for computing geodesics under memory constraints. Specifically, it computes the geodesic between any two points in a simple polygon of n vertices in expected $O(n^2/s + n \log s \log^4(n/s))$ time using $O(s)$ words of space. Note that this path might not fit in memory, so the edges of the geodesic are reported one by one in order.

3 Algorithm

Let π be the geodesic connecting v_1 and $v_{\lfloor n/2 \rfloor}$. From a high-level perspective, the algorithm uses the approach of Har-Peled [8] to compute π. We will use the computed edges to subdivide P into smaller problems that can be solved recursively.

We start by introducing some definitions that will help in storing which portion of the polygon has already been triangulated. Vertices v_1 and $v_{\lfloor n/2 \rfloor}$ split the boundary of P into

two chains. We say v_i is a *top* vertex if $1 < i < \lfloor n/2 \rfloor$ and a *bottom* vertex if $\lfloor n/2 \rfloor < i \leq n$. Top/bottom is the *type* of a vertex and all vertices (except for v_1 and $v_{\lfloor n/2 \rfloor}$) have exactly one type. A diagonal c is *alternating* if it connects a top and a bottom vertex (or one of its endpoints is either v_1 or $v_{\lfloor n/2 \rfloor}$), and *non-alternating* otherwise.

We will use diagonals to partition P into two parts. For simplicity of the exposition, given a diagonal d, we regard both components of $P \setminus d$ as closed (i.e., the diagonal belongs to both of them). Since any two consecutive vertices of P can see each other, the partition an edge of P is trivial, in the sense that one subpolygon is P and the other one is a line segment.

▶ **Observation 1.** *Let c be a diagonal of P not incident to v_1 or $v_{\lfloor n/2 \rfloor}$. Vertices v_1 and $v_{\lfloor n/2 \rfloor}$ belong to different components of $P \setminus c$ if and only if c is an alternating diagonal.*

▶ **Corollary 2.** *Let c be a non-alternating diagonal of P. The component of $P \setminus c$ that contains neither v_1 nor $v_{\lfloor n/2 \rfloor}$ has at most $\lceil n/2 \rceil$ vertices.*

While triangulating the polygon, an alternating diagonal a_c records the part of the polygon has already been triangulated. More specically, we maintain the following invariant: the connected component of $P \setminus a_c$ not containing $v_{\lfloor n/2 \rfloor}$ has already been triangulated.

Ideally, a_c would be a segment of π (the geodesic connecting v_1 and $v_{\lfloor n/2 \rfloor}$), but this is not always possible. Instead, we guarantee that at least one of the endpoints of a_c is a vertex of π that has already been computed in the execution of the shortest-path algorithm.

With these definitions in place, we can give an intuitive description of our algorithm: we start by setting a_c as the degenerate diagonal from v_1 to v_1. We then use the shortest-path computation approach of Har-Peled. Our aim is to walk along π until we find a new alternating diagonal a_{new}. At that moment we pause the execution of the shortest-path algorithm, triangulate the subpolygons of P that have been created (and contain neither v_1 nor $v_{\lfloor n/2 \rfloor}$) recursively, update a_c to the newly found alternating diagonal, and then continue with the execution of the shortest-path algorithm.

Although our approach is intuitively simple, there are several technical difficulties that must be carefully considered. Ideally, the number of vertices we walked along π before finding an alternating diagonal is small and thus they can be stored explicitly. But if we do not find an alternating diagonal on π in just a few steps (indeed, it could even be that there is no alternating diagonal in π), we need to use other diagonals. We also need to make sure that the complexity of each recursive subproblem is reduced by a constant fraction, that we never exceed space bounds, and that no part of the triangulation is reported more than once.

Let v_c denote the endpoint of a_c that is on π and that is closest to $v_{\lfloor n/2 \rfloor}$. Recall that the subpolygon defined by a_c containing v_1 has already been triangulated. Let $w_0, \ldots, w_k$ be the portion of π up to the next alternating diagonal. That is, path π is of the form $\pi = (v_1, \ldots, v_c = w_0, w_1, \ldots, w_k, \ldots, v_{\lfloor n/2 \rfloor})$ where $w_1, \ldots, w_{k-1}$ are of the same type as v_c, and w_k is of different type (or $w_k = v_{\lfloor n/2 \rfloor}$ if all vertices between v_c and $v_{\lfloor n/2 \rfloor}$ are of the same type).

Consider the partition of P induced by a_c and this portion of π, see Figure 1. Let P_1 be the subpolygon induced by a_c that does not contain $v_{\lfloor n/2 \rfloor}$. Similarly, let $P_{\lfloor n/2 \rfloor}$ be the subpolygon that is induced by the alternating diagonal $w_{k-1}w_k$ and does not contain v_1[1].

[1] For simplicity of the exposition, the definition of P_1 assumes that $v_{\lfloor n/2 \rfloor}$ is not an endpoint of a_c (similarly, v_1 not an endpoint of $w_{k-1}w_k$ for the definition of $P_{\lfloor n/2 \rfloor}$). Each of these conditions is not satisfied once (i.e., with the first and last diagonals of π), and in those cases the polygons P_1 and $P_{\lfloor n/2 \rfloor}$ are not properly defined. Whenever this happens we have $k = 1$ and a single diagonal that splits P in

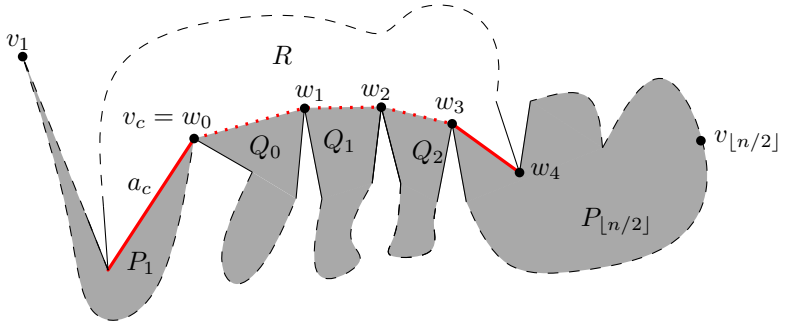


Figure 1 Partitioning P into subpolygons P_1, $P_{\lfloor n/2 \rfloor}$, R, Q_1, …, Q_{k-2}. The two alternating diagonals are marked by thick red lines.

For any $i < k-1$ we define Q_i as the subpolygon induced by the non-alternating diagonal w_iw_{i+1} that contains neither v_1 nor $v_{\lfloor n/2 \rfloor}$. Finally, let R be the remaining component of P. Note that some of these subpolygons may be degenerate and consist only of a line segment (for example, when w_iw_{i+1} is an edge of P).

▶ **Lemma 3.** *Each of the subpolygons R, Q_1, Q_2, …, Q_{k-2} has at most $\lceil n/2 \rceil + k$ vertices. Moreover, if $w_k = v_{\lfloor n/2 \rfloor}$, then the subpolygon $P_{\lfloor n/2 \rfloor}$ also has at most $\lceil n/2 \rceil$ vertices.*

Proof. Subpolygons Q_i are induced by non-alternating diagonals and cannot have more than $\lceil n/2 \rceil$ vertices, by Corollary 2. The proof for R follows by definition: the boundary of R (other than vertices $w_0, \ldots, w_k$) is defined by a contiguous portion of P consisting of only top vertices or only bottom vertices. Recall that there are at most $\lceil n/2 \rceil$ of them. Similarly, if $w_k = v_{\lfloor n/2 \rfloor}$, subpolygon $P_{\lfloor n/2 \rfloor}$ can only have vertices of one type (either only top or only bottom vertices), and thus the bound holds. This completes the proof of the Lemma. ◀

This result allows us to treat the easy case of our algorithm. When k is small (say, a constant number of vertices), we can pause the shortest-path computation algorithm, explicitly store all vertices w_i, recursively triangulate R as well as the subpolygons Q_i (for all $i \leq k-2$), update a_c to the edge $w_{k-1}w_k$ and continue with the shortest-path algorithm.

Handling the case of large k is more involved. Note that we do not know the value of k until we find the next alternating diagonal, but we need not compute it directly. Given a parameter τ related to the workspace allowed for our algorithms, we say that the path is *long* when $k > \tau$. Initially we set $\tau = s$ but the value of this parameter will change as we descend the recursion tree. We say that the distance between two alternating diagonals is *long* whenever we have computed τ vertices of π besides v_c and they are all of the same type as v_c. That is, path π is of the form $\pi = (v_1, \ldots, v_c = w_0, w_1, \ldots, w_\tau, \ldots v_{\lfloor n/2 \rfloor})$ and vertices $w_0, w_1, \ldots w_\tau$ are all of the same type. In particular, the vertices $w_0, \ldots, w_\tau$ must form a convex chain (see Figure 1). Rather than continue walking along π, we look for a vertex u of P that together with w_τ forms an alternating diagonal. Once we have found this diagonal, we have at most $\tau + 2$ diagonals ($a_c, w_0w_1, w_1w_2, \ldots, w_{\tau-1}w_\tau$, and uw_τ) partitioning P into at most $\tau + 3$ subpolygons once again: P_1 is the part induced by a_c which does not contain $v_{\lfloor n/2 \rfloor}$, $P_{\lfloor n/2 \rfloor}$ is the part induced by uw_τ which does not contain v_1, Q_i is the part induced by w_iw_{i+1}, which contains neither v_1 nor $v_{\lfloor n/2 \rfloor}$, and R is the remaining component.

two. Thus, if $v_{\lfloor n/2 \rfloor} \in a_c$ (and thus P_1 is undefined), we simply define P_1 as the complement $P_{\lfloor n/2 \rfloor}$ (similarly, if $v_1 \in w_{k-1}w_k$, we define $P_{\lfloor n/2 \rfloor}$ as complement of P_1. If both subpolygons are undefined simultaneously we assign them arbitrarily.

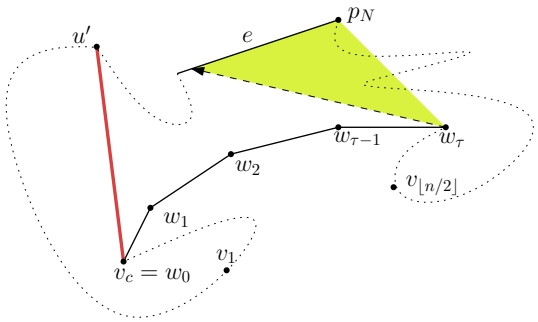


Figure 2 After we have walked τ steps of π we can find an alternating diagonal by shooting a ray from w_τ either towards u' or $w_{\tau-1}$ (whichever is higher). The upper endpoint p_N of the first edge e hit might not be visible. But in this case the reflex vertex of smallest angle inside the triangular zone must be visible.

▶ **Lemma 4.** *We can find a vertex u that together with w_τ forms an alternating diagonal in $O(n)$ time using $O(1)$ space. Moreover, each of the subpolygons R, Q_1, Q_2, …, $Q_{\tau-2}$ has at most $\lceil n/2 \rceil + \tau$ vertices.*

Proof. Proofs for the size of the subpolygons are identical to those of Lemma 3. Thus, we focus on how to compute u efficiently. Without loss of generality, we may assume that the edge $w_{\tau-1}w_\tau$ is horizontal. Recall that the chain $w_0, \ldots, w_\tau$ is in convex position, thus all of these vertices must lie on one side of the line $\ell_{\tau,\tau-1}$ through w_τ and $w_{\tau-1}$. Without loss of generality, we may assume that they all lie below $\ell_{\tau,\tau-1}$. Let u' be the endpoint of a_c other than v_c. If u' also lies below $\ell_{\tau,\tau-1}$, we shoot a ray from w_τ towards $w_{\tau-1}$. Otherwise, we shoot a ray from w_τ towards u'. Let e be the first edge that is properly intersected by the ray and let p_N be the endpoint of e of highest y-coordinate. Observe that p_N must be on or above $\ell_{\tau,\tau-1}$, see Figure 2.

Ideally, we would like to report p_N as the vertex u. However, point p_N need not be visible even when some portion of e is. Whenever this happens we can use the visibility properties of simple polygons: since e is partially visible, we know that the portion of P that obstructs visibility between w_τ and p_N must cross the segment from w_τ to p_N. In particular, there must be one or more reflex vertices in the triangle formed by w_τ, p_N, and the visible point of e (shaded region of Figure 2). Among those vertices, we know that the vertex r that maximizes the angle $\angle p_N w_\tau r$ must be visible (see Lemma 1 of [6]). Further note that r must be a top vertex: otherwise π would need to traverse through r to reach $v_{\lfloor n/2 \rfloor}$, and this would force π to do a reflex turn, which is impossible in a geodesic.

As described in Lemma 1 of [6], in order to find such a reflex vertex we need to scan the input polygon at most three times, each time storing a constant amount of information: once for finding the edge e and point p_N, once more to determine if p_N is visible, and a third time to find r if p_N is not visible. ◀

At high level, our algorithm walks from v_1 to $v_{\lfloor n/2 \rfloor}$. We stop after walking τ steps or when we find an alternating diagonal (whichever comes first). This generates several subproblems of smaller complexity that are solved recursively. Once the recursion is done we update a_c (to keep track of the portion of P that has been triangulated), and continue walking along π. The walking process ends when it reaches $v_{\lfloor n/2 \rfloor}$. In this case, in addition to triangulating R and the Q_i subpolygons as usual, we must also triangulate $P_{\lfloor n/2 \rfloor}$.

The algorithm in the deeper levels of recursion is almost identical. There are only three minor changes that need to be introduced. We need some base cases to end the recursion. Recall that τ denotes the amount of space available to the current level of recursion. Thus, if τ is comparable to n (say, $10\tau \geq n$), then the whole polygon fits into memory and can be triangulated in linear time [7]. Similarly, if τ is small (say $\tau \leq 1$, we have run out of space and thus we triangulate P using a constant workspace algorithm [2]. In all other cases we continue with the recursive algorithm as usual.

For ease in handling the subproblems, at each step we also indicate the vertex that fulfils the role of v_1 (i.e., one of the vertices from which the geodesic must be computed). Recall that we have random access to the vertices of the input. Thus, once we know which vertex plays the role of v_1, we can find the vertex that satisfies the role of $v_{\lfloor n/2 \rfloor}$ in constant time as well.

In order to avoid exceeding the space bounds, at each level of the recursion we decrease the value of τ by a factor of $\kappa < 1$. The exact value of κ will be determined below. Pseudocode of the recursive algorithm can be found in the Appendix (Algorithm 1).

▶ **Theorem 5.** *Let P be a simple polygon of n vertices. We can compute a triangulation of P in $O(n^2/s + n(\log s)\log^5(n/s))$ expected time using $O(s)$ variables (for any $s \in O(n)$). In particular, when $s \in O(\frac{n}{\log n \log^5 \log n})$ the algorithm runs in $O(n^2/s)$ expected time.*

In the remainder of the section we prove correctness and both the time and space bounds for our algorithm.

3.1 Correctness

The current diagonal a_c properly records what portion of the polygon has already been triangulated. Thus, we never report an edge of the triangulation more than once. Hence, in order to show correctness of the algorithm, we must show that the recursion eventually terminates.

During the execution of the algorithm, we invoke recursion for polygons Q_i, R, and $P_{\lfloor n/2 \rfloor}$ (the latter one only when we have reached $v_{\lfloor n/2 \rfloor}$). By Lemma 3 all of these polygons have size at most $n/2 + \tau$. Since we only enter this level of recursion whenever $\tau \leq n/10$ (see lines 1-3 of Algorithm 1), overall the size of the problem decreases by a factor of 6/10. That is, at each level of recursion the problem instances are reduced by a constant fraction. In particular, after $O(\log n)$ steps the subpolygons will be of constant size and will be solved without recursion.

At each level of recursion we use the shortest-path algorithm of Har-Peled. This algorithm needs random access in constant time to the vertices of the polygon. Thus, we must make sure that this property is preserved at all levels of recursion. A simple way to do so would be to explicitly store the polygon in memory at every recursive call, but this may exceed the space bounds of the algorithm.

Instead, we make sure that the subpolygon is described by $O(\tau)$ words. By construction, each subpolygon consists of a single chain of contiguous input vertices of P and at most τ additional *cut* vertices (vertices from the geodesics at higher levels). We can represent the portion of P by the indices of the first and last vertex of the chain and explicitly store the indices of all cut vertices. By an appropriate renaming of the indices within the subpolygon, we can make the vertices of the chain appear first, followed by the cut vertices. Thus, when we need to access the ith vertex of the subpolygon, we can check if i corresponds to a vertex of the chain or one of the cut vertices and identify the desired vertex in constant time, in either case.

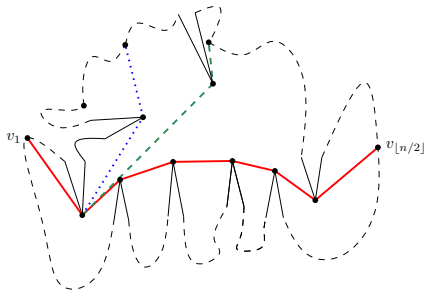


Figure 3 At different level of recursion the subproblems are formed by a consecutive chain of the input and a list of $O(s)$ cut vertices. The geodesics used to split the problem at first, second and third level are depicted in solid red, dashed green, and dotted blue, respectively.

Now, we must show that each recursive call satisfies this property. Clearly this holds for the top level of recursion, where the input polygon is simply P and no cut vertices are needed. At the next level of recursion each subproblem has up to τ cut vertices and a chain of contiguous input vertices. The way we make sure that this property is satisfied at lower levels of recursion is by a correct choice of v_1 (the vertex from which we start the path): at each level of recursion we build the next geodesic starting from either the first or last cut vertex. This might create additional cut vertices, but their position is immediately after or before the already existing cut vertices (see Figure 3). This way we certify that random access to the input polygon is possible at all levels of recursion.

3.2 Time Bounds

We use a two-parameter function $T(\eta, \tau)$ to bound the expected running time of the algorithm at all levels of recursion. The first parameter η represents the size of the problem. Specifically, for a polygon of n vertices we set $\eta = n - 2$, namely, the number of triangles to be reported. The second parameter τ gives the space bound for the algorithm. Initially, we have $\tau = s$, but this value decreases by a factor of κ at each level of recursion. Recall that τ is also the workspace limit for the shortest-path algorithm of Har-Peled that we invoke as part of our algorithm. In addition, τ is also used as the limit on the length of the geodesic we explore looking for an alternating diagonal.

When τ becomes really small (say $\tau \leq 10$) we have run out of allotted space. Thus, we triangulate the polygon using the constant workspace method of Asano *et al.* [2] that runs in $O(n^2)$ time. Similarly, if the space is large when compared to the instance size (say, $10\tau \geq \eta$) the polygon fits in the allowed workspace, hence we use Chazelle's algorithm [7] for triangulating it. In both cases we have $T(\eta, \tau) \leq c_\Delta \eta^2/\tau$ (for some constant $c_\Delta > 0$).

In other situations, we must partition the problem and solve it recursively. First we bound the time needed to compute the partitions. The main tool we use is computing the geodesic between v_1 and $v_{\lfloor n/2 \rfloor}$. This is done by the algorithm of Har-Peled [8] which takes $O(\eta^2/\tau + \eta(\log \tau) \log^4(\eta/\tau))$ expected time and uses $O(\tau)$ space. Recall that we pause and continue it often during the execution of our algorithm, but overall we only execute it once. Thus, the total time spent in shortest-path computation at one level is unchanged.

Another operation that we execute is FindAlternatingDiagonal (i.e., Lemma 4) which takes $O(\eta)$ time and $O(1)$ space. In the worst case, this operation is invoked once for every τ vertices of π. Since π cannot have more than n vertices, the overall time spent in this operation is bounded by $O(\eta^2/\tau)$. Thus, ignoring the time spent in recursion, the expected running time of the algorithm is $c_{\text{HP}}(\eta^2/\tau + \eta(\log \tau) \log^4(\eta/\tau))$ for some constant

c_{HP}, which without loss of generality we assume to be at least c_Δ.

That is, for any value of η and τ we never spend more than $c_{\text{HP}}(\eta^2/\tau + \eta(\log\tau)\log^4(\eta/\tau))$ time. To this value we must add the time spent in recursion. At each level we launch several subproblems, giving a recurrence of the form

$$T(\eta,\tau) \le c_{\text{HP}}(\eta^2/\tau + \eta(\log\tau)\log^4(\eta/\tau)) + \sum_j T(\eta_j, \kappa\tau).$$

Recall that the values η_j cannot be very large when compared to η. Indeed, each subproblem can have at most a constant fraction c of vertices of the original one (i.e., the way in which lines 1–4 of Algorithm 1 have been set, we have $c = 6/10$). Thus, each η_j satisfies $\eta_j \le c(\eta+2) - 2 \le c\eta$. Since every edge is reported exactly once, we also have $\sum_j \eta_j = \eta$.

We claim that for any $\tau, \eta > 0$ there exists a constant c_R so that $T(\eta,\tau) \le c_R(\eta^2/\tau + \eta(\log\tau)\log^5(\eta/\tau))$. Indeed, when $\tau \le 10$ or $10\tau \ge \eta$ we have $T(\eta,\tau) \le c_\Delta\eta^2 \le c_{\text{HP}}\eta^2$. Otherwise, we use induction and obtain

$$\begin{aligned}
T(\eta,\tau) &\le c_{\text{HP}}(\eta^2/\tau + \eta(\log\tau)\log^4(\eta/\tau)) + \sum_j T(\eta_j, \tau\kappa)\\
&\le c_{\text{HP}}(\eta^2/\tau + \eta(\log\tau)\log^4(\eta/\tau)) + \frac{c_R}{\tau\kappa}\sum_j \eta_j^2 + c_R\sum_j \eta_j(\log\kappa\tau)\log^5(\frac{\eta_j}{\tau\kappa})\\
&\le (c_{\text{HP}}\frac{\eta^2}{\tau} + \frac{c_R}{\tau\kappa}\sum_j \eta_j^2) + c_{\text{HP}}\eta(\log\tau)\log^4(\eta/\tau) + c_R\sum_j \eta_j(\log\tau)\log^5(\frac{\eta_j}{\tau\kappa})\\
&\le (c_{\text{HP}}\frac{\eta^2}{\tau} + \frac{c_R}{\tau\kappa}\sum_j \eta_j^2) + c_{\text{HP}}\eta(\log\tau)\log^4(\eta/\tau) + c_R\sum_j \eta_j(\log\tau)\log^5(\frac{c\eta}{\tau\kappa})\\
&\le (c_{\text{HP}}\frac{\eta^2}{\tau} + \frac{c_R}{\tau\kappa}\sum_j \eta_j^2) + c_{\text{HP}}\eta(\log\tau)\log^4(\eta/\tau) + c_R\eta(\log\tau)\log^5(\frac{c\eta}{\tau\kappa}).
\end{aligned}$$

The sum $\sum_j \eta_j^2$ is at most $\frac{n}{c\eta}(c\eta)^2 = c\eta^2$, since $\eta_j \le c\eta$, yielding

$$\begin{aligned}
T(\eta,\tau) &\le (c_{\text{HP}}\frac{\eta^2}{\tau} + \frac{c_R c}{\kappa}\frac{\eta^2}{\tau}) + c_{\text{HP}}\eta(\log\tau)\log^4(\eta/\tau) + c_R\eta(\log\tau)\log^5(\frac{c\eta}{\tau\kappa})\\
&\le \frac{c_R\eta^2}{\tau} + c_{\text{HP}}\eta(\log\tau)\log^4(\eta/\tau) + c_R\eta(\log\tau)\log^5(\frac{c\eta}{\tau\kappa}),
\end{aligned}$$

where the inequality $c_{\text{HP}} + \frac{c}{\kappa}c_R \le c_R$ holds for sufficiently large values of c_R and $\kappa < 1$ (say, $c_R = 10c_{\text{HP}}$ and $\kappa = 9/10$). Now we focus on the second term of the inequation. We upper bound $\log^5(\frac{c\eta}{\tau\kappa})$ by $\log^4(\frac{\eta}{\tau})\log(\frac{c\eta}{\tau\kappa}) = \log^4(\frac{\eta}{\tau})(\log(\frac{\eta}{\tau}) - \log(\frac{\kappa}{c}))$ and substitute to obtain:

$$\begin{aligned}
T(\eta,\tau) &\le \frac{c_R\eta^2}{\tau} + c_{\text{HP}}\eta(\log\tau)\log^4(\eta/\tau) + c_R\eta(\log\tau)\log^4(\frac{\eta}{\tau})(\log(\frac{\eta}{\tau}) - \log(\frac{\kappa}{c}))\\
&\le \frac{c_R\eta^2}{\tau} + (\eta(\log\tau)\log^4(\frac{\eta}{\tau}))(c_{\text{HP}} + c_R\log(\frac{\eta}{\tau}) - c_R\log(\frac{\kappa}{c}))\\
&\le \frac{c_R\eta^2}{\tau} + c_R(\eta(\log\tau)\log^5(\frac{\eta}{\tau})) = c_R(\eta^2/\tau + \eta(\log\tau)\log^5(\frac{\eta}{\tau})).
\end{aligned}$$

Again, the $c_{\text{HP}} - c_R\log(\frac{\kappa}{c}) \le 0$ inequality holds for sufficiently large values of c_R, that depend on c_{HP}, κ and c.

3.3 Space Bounds

We now show that the space bound holds. Recall that we stop recursion whenever the problem instance fits into memory or $\tau \le 1$. Since the value of τ decreases by a constant

factor at each level of recursion, we will never recurse for more than $\log_\kappa s = O(\log s)$ levels. Thus, the implicit memory consumption used in recursion does not exceed the space bounds.

Now we bound the size of the workspace needed by the algorithm at level i of the recursion (with the main algorithm invocation being level 0) by $O(s \cdot \kappa^i)$. Indeed, this is the threshold of space we receive as input (recall that initially we set $\tau = s$ and that at each level we reduce this value by a factor of κ). This threshold value is the amount of space for the shortest-path computation algorithm invoked at the current level, as well as limit on the number of vertices of π that are stored explicitly before invoking procedure FindAlternatingDiagional. Once we have found the new alternating diagonal, the vertices of π that were stored explicitly are used to generate the subproblems for the recursive calls.

The space used for storing the intermediate points can be reused after the recursive executions are finished, so overall we conclude that at the i-th level of recursion the algorithm never uses more than $O(s \cdot \kappa^i)$ space. Since we never have two simultaneously executing recursive calls at the same level, and $\kappa < 1$, the total amount of space used in the execution of the algorithm is bounded by

$$O(s) + O(s \cdot \kappa) + O(s \cdot \kappa^2) + \ldots = O(s).$$

3.4 Considerations on the Output

For simplicity in the explanation we assumed that in order to report the triangulation, reporting the edges suffices. However, we note that we can also report the triangulation in any other format, such as a list of adjacencies. That is, we can report the triangles generated, and for each one we additionally report the three boundary edges and the three triangles adjacent to it). Most of this is easy to do, since the triangles are reported at the bottom level of the recursion where the subpolygons fit in memory. Thus, for each triangle we can report their adjacencies as usual (using for example the indices of the vertices to identify the triangles). The only difficulty arises around the edges used to split the polygon into subpolygons. When we create the triangle on one side of such an edge we do not yet know which triangle will be created on the other side as this triangle resides in a different subpolygon. Hence, this triangle cannot report its adjacencies yet.

Instead we delay reporting triangles along these splitting edges until both triangles have been constructed. For this purpose we must slightly alter the triangulation invariant associated to a_c: subpolygon P_1 has been triangulated and all triangles have been reported *except* the triangle whose boundary is a_c. This triangle (along with its two neighbors in P_1) is stored explicitly in memory.

The algorithm proceeds, partitioning into subproblems $Q_1, \ldots Q_{k-2}$ and R as usual. Each subproblem Q_i returns a triangle that has not been reported yet along with its two adjacencies (or nothing if the corresponding subpolygon Q_i is empty). The neighbors of these triangles are in the subproblem R, so they are given to the recursive procedure of R. As soon as the missing neighbor is computed, we can report the stored triangle delete it from memory. Once R has finished we need to update a_c and v_c as usual. In addition, we must now store (and do not yet report) the triangle that is adjacent to a_c. The bottommost level of recursion triangulates as usual and stores the single triangle that has not been reported so it can be reported when processing R.

Overall, at each level of recursion we need to store as many triangles as subproblems generated. Moreover, once R has been recursively triangulated, this information need not be stored anymore. Recall that the number of subproblems generated is at most the space threshold. Thus, we conclude that the storage bounds are asymptotically unaffected.

Acknowledgements. The authors would like to thank Jean-François Baffier, Man-Kwun Chiu, Wolfgang Mulzer and Takeshi Tokuyama for valuable discussion in the creation of this paper.

References

1. B. Aronov, M. Korman, S. Pratt, A. van Renssen, and M. Roeloffzen. Time-space tradeoffs for triangulating a simple polygon. *CoRR*, abs/1509.07669, 2015. URL: `http://arxiv.org/abs/1509.07669`.
2. T. Asano, K. Buchin, M. Buchin, M. Korman, W. Mulzer, G. Rote, and A. Schulz. Memory-constrained algorithms for simple polygons. *Computational Geometry: Theory and Applications*, 46(8):959–969, 2013.
3. T. Asano and D. Kirkpatrick. Time-space tradeoffs for all-nearest-larger-neighbors problems. In *Proc. 13th Int. Conf. Algorithms and Data Structures (WADS)*, pages 61–72, 2013.
4. T. Asano, W. Mulzer, G. Rote, and Y. Wang. Constant-work-space algorithms for geometric problems. *Journal of Computational Geometry*, 2(1):46–68, 2011.
5. L. Barba, M. Korman, S. Langerman, K. Sadakane, and R. I. Silveira. Space–time trade-offs for stack-based algorithms. *Algorithmica*, 72(4):1097–1129, 2015. `doi:10.1007/s00453-014-9893-5`.
6. L. Barba, M. Korman, S. Langerman, and R. I. Silveira. Computing the visibility polygon using few variables. *Computational Geometry: Theory and Applications*, 47(9):918–926, 2013.
7. B. Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6:485–524, 1991. `doi:10.1007/BF02574703`.
8. S. Har-Peled. Shortest path in a polygon using sublinear space. In *Proceedings of the 31st International Symposium on Compututational Geometry (SoCG)*, pages 111–125, 2015. `doi:10.4230/LIPIcs.SOCG.2015.111`.
9. M. Korman. Memory-constrained algorithms. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 1–7. Springer Berlin Heidelberg, 2015. `doi:10.1007/978-3-642-27848-8_586-1`.
10. M. Korman, W. Mulzer, M. Roeloffzen, A. v. Renssen, P. Seiferth, and Y. Stein. Time-space trade-offs for triangulations and voronoi diagrams. In *Proc. 14th Int. Conf. Algorithms and Data Structures (WADS)*, pages 482–494, 2015.
11. J. E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley, 1998.

A Algorithm Pseudocode

Algorithm 1: Pseudocode for Triangulate(P, v_1, τ) that, given a simple polygon P, a vertex v of P, and workspace capacity τ, computes a triangulation of P in $O(n^2/\tau)$ time using $O(\tau)$ variables.

if $10\tau \geq n$ **then** (* The polygon fits into memory. *)
  Triangulate P using Chazelle's algorithm [7]
else if $\tau \leq 10$ **then** (* We ran out of recursion space. *)
  Triangulate P using the constant workspace algorithm [2]
else (* P is large, we will use recursion. *)
  $a_c \leftarrow v_1 v_1$
  $v_c \leftarrow v_1$
  walked $\leftarrow v_1$ (* Variable to keep track of how far we have walked on π. *)
  while walked $\neq v_{\lfloor n/2 \rfloor}$ **do**
    $i \leftarrow 0$ (* i counts the number of steps before finding an alternation edge *)
    repeat
      $i \leftarrow i + 1$
      $w_i \leftarrow$ next vertex of π
    until $i = \tau$ **or** type$(v_c) \neq$ type(w_i)
    if type$(v_c) \neq$ type(w_i) **then**
      $u' \leftarrow w_{i-1}$
      $a_{\text{new}} \leftarrow w_i w_{i-1}$
    else (* We walked too much. Use Lemma 4 to partition the problem. *)
      $u' \leftarrow$ FindAlternatingDiagional$(P, a_c, v_c, w_1, \ldots, w_\tau)$
      $a_{\text{new}} \leftarrow u' w_i$
    end if
    (* Now we triangulate the subpolygons. *)
    Triangulate$(R, u', \tau \cdot \kappa)$
    for j=0 **to** i-2 **do**
      Triangulate$(Q_j, w_j, \tau \cdot \kappa)$
    end for
    $a_c \leftarrow a_{\text{new}}$
    $v_c \leftarrow w_\tau$
    walked $\leftarrow w_\tau$
  end while
  (* We reached $v_{\lfloor n/2 \rfloor}$. All parts except $P_{\lfloor n/2 \rfloor}$ have been triangulated. *)
  Triangulate$(P_{\lfloor n/2 \rfloor}, w_i, \tau \cdot \kappa)$
end if

Excluded Grid Theorem: Improved and Simplified

Julia Chuzhoy

Toyota Technological Institute, Chicago, USA
cjulia@ttic.edu

Abstract

One of the key results in Robertson and Seymour's seminal work on graph minors is the Excluded Grid Theorem. The theorem states that there is a function f, such that for every positive integer g, every graph whose treewidth is at least $f(g)$ contains the $(g \times g)$-grid as a minor. This theorem has found many applications in graph theory and algorithms. An important open question is establishing tight bounds on $f(g)$ for which the theorem holds. Robertson and Seymour showed that $f(g) \geq \Omega(g^2 \log g)$, and this remains the best current lower bound on $f(g)$. Until recently, the best upper bound was super-exponential in g. In this talk, we will give an overview of a recent sequence of results, that has lead to the best current upper bound of $f(g) = O(g^{19} \operatorname{poly} \log(g))$. We will also survey some connections to algorithms for graph routing problems.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Graph Minor Theory, Excluded Grid Theorem, Graph Routing

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.31

Category Invited Talk

15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016).
Editor: Rasmus Pagh; Article No. 31; pp. 31:1–31:1

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The Complexity Landscape of Fixed-Parameter Directed Steiner Network Problems

Dániel Marx

Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary
dmarx@cs.bme.hu

Abstract

Given a directed graph G and a list $(s_1, t_1), \ldots, (s_k, t_k)$ of terminal pairs, the Directed Steiner Network problem asks for a minimum-cost subgraph of G that contains a directed $s_i \to t_i$ path for every $1 \leq i \leq k$. Feldman and Ruhl presented an $n^{O(k)}$ time algorithm for the problem, which shows that it is polynomial-time solvable for every fixed number k of demands. There are special cases of the problem that can be solved much more efficiently: for example, the special case Directed Steiner Tree (when we ask for paths from a root r to terminals t_1, $\ldots$, t_k) is known to be fixed-parameter tractable parameterized by the number of terminals, that is, algorithms with running time of the form $f(k) \cdot n^{O(1)}$ exist for the problem. On the other hand, the special case Strongly Connected Steiner Subgraph (when we ask for a path from every t_i to every other t_j) is known to be W[1]-hard parameterized by the number of terminals, hence it is unlikely to be fixed-parameter tractable. In the talk, we survey results on parameterized algorithms for special cases of Directed Steiner Network, including a recent complete classification result (joint work with Andreas Feldmann) that systematically explores the complexity landscape of directed Steiner problems to fully understand which special cases are FPT or W[1]-hard.

1998 ACM Subject Classification F2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases Directed Steiner Tree, Directed Steiner Network, fixed-parameter tractability, dichotomy

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.32

Category Invited Talk

15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016).
Editor: Rasmus Pagh; Article No. 32; pp. 32:1–32:1

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Computation as a Scientific Weltanschauung

Christos H. Papadimitriou

University of California, Berkeley, USA
christos@berkeley.edu

Abstract

Computation as a mechanical reality is young – almost exactly seventy years of age – and yet the spirit of computation can be traced several millennia back. Any moderately advanced civilization depends on calculation (for inventory, taxation, navigation, land partition, among many others) – our civilization is the first one that is conscious of this reliance.

Computation has also been central to science for centuries. This is most immediately apparent in the case of mathematics: the idea of the algorithm as a mathematical object of some significance was pioneered by Euclid in the 4^{th} century BC, and advanced by Archimedes a century later. But computation plays an important role in virtually all sciences: natural, life, or social. Implicit algorithmic processes are present in the great objects of scientific inquiry – the cell, the universe, the market, the brain – as well as in the models developed by scientists over the centuries for studying them. This brings about a very recent – merely a few decades old – mode of scientific inquiry, which is sometime referred to as the *lens of computation*: When students of computation revisit central problems in science from the computational viewpoint, often unexpected progress results. This has happened in statistical physics through the study of phase transitions in terms of the convergence of Markov chain-Monte Carlo algorithms, and in quantum mechanics through quantum computing.

This talk will focus on three other manifestations of this phenomenon. Almost a decade ago, ideas and methodologies from computational complexity revealed a subtle conceptual flaw in the solution concept of Nash equilibrium, which lies at the foundations of modern economic thought. In the study of evolution, a new understanding of century-old questions has been achieved through surprisingly algorithmic ideas. Finally, current work in theoretical neuroscience suggests that the algorithmic point of view may be invaluable in the central scientific question of our era, namely understanding how behavior and cognition emerge from the structure and activity of neurons and synapses.

1998 ACM Subject Classification F. Theory of Computation

Keywords and phrases Lens of computation, Nash equilibrium, neuroscience

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.33

Category Invited Talk

15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016).
Editor: Rasmus Pagh; Article No. 33; pp. 33:1–33:1

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany